aws

Entwicklerhandbuch

Amazon DynamoDB

API-Version 2012-08-10

Copyright © 2026 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

Amazon DynamoDB: Entwicklerhandbuch

Copyright © 2026 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

Die Handelsmarken und Handelsaufmachung von Amazon dürfen nicht in einer Weise in Verbindung mit nicht von Amazon stammenden Produkten oder Services verwendet werden, durch die Kunden irregeführt werden könnten oder Amazon in schlechtem Licht dargestellt oder diskreditiert werden könnte. Alle anderen Marken, die nicht im Besitz von Amazon sind, gehören den jeweiligen Besitzern, die möglicherweise mit Amazon verbunden sind oder von Amazon gesponsert werden.

Table of Contents

Was ist Amazon DynamoDB? 1
Merkmale 2
Serverless 2
NoSQL 2
Vollständig verwaltet 2
Leistung im einstelligen Millisekundenbereich 3
Anwendungsfälle 3
Capabilities 4
Multiaktive Replikation mit globalen Tabellen 4
ACID-Transaktionen 5
Erfassung von Datenänderungen 5
Sekundäre Indizes 5
Service-Integrationen 5
Serverless-Integrationen 5
Importieren und Exportieren von Daten in Amazon S3 6
Null-ETL-Integration 6
Caching 6
Sicherheit 7
Ausfallsicherheit 7
Globale Tabellen 8
Kontinuierliche Backups und Wiederherstellung point-in-time 8
On-Demand-Backup und Wiederherstellung 8
Zugreifen auf DynamoDB 9
Preisgestaltung 9
Erste Schritte 9
Erste Schritte mit DynamoDB 11
Ressourcen für Erstbenutzer 12
Zusätzliche bewährte Methoden für Amazon DynamoDB für Erstbenutzer 13
AWS CLI Ressourcen 14
Programmierressourcen 14
Zugreifen auf DynamoDB 14
Verwenden der Konsole 15
Mit dem AWS CLI 15
Verwenden der API 18

Verwenden von NoSQL-Workbench 19
IP-Adressbereiche 20
Dual-Stack-Endpunkte für Internet Protocol Version 6 () IPv6 20
Voraussetzungen 21
Einrichten von DynamoDB 21
Einrichten von DynamoDB (Webservice) 22
Lokale Einrichtung von DynamoDB (herunterladbare Version) 26
Schritt 1: Erstellen einer Tabelle 54
Schritt 2: Schreiben von Daten 107
Schritt 3: Lesen von Daten 140
Schritt 4: Aktualisieren von Daten 167
Schritt 5: Abfragen von Daten 200
Schritt 6: (optional) Aufräumen 239
Nächste Schritte 257
Console-to-Code 258
Funktionsweise 258
Vorteile der Verwendung Console-to-Code mit DynamoDB 259
Beispielanwendungsfälle 259
Erste Schritte 260
Funktionsweise 261
Spickzettel 261
Erstes Einrichten von 261
SDK oder CLI 262
Grundlegende Aktionen 262
Benennungsregeln 263
Grundlegende Informationen zu Service Quotas 264
Weitere Informationen 266
Kernkomponenten 266
Tabellen, Elemente und Attribute 267
Primärschlüssel 271
Sekundäre Indizes 272
DynamoDB Streams 276
DynamoDB API 278
Steuerebene 278
Datenebene 279
DynamoDB Streams 281

Transaktionen ... 281
Unterstützte Datentypen und Benennungsregeln ... 282
Benennungsregeln ... 282
Datentypen ... 283
Datentypbeschreibungen ... 288
DynamoDB-Tabellenklassen ... 289
Partitionen und Datenverteilung in DynamoDB ... 290
Datenverteilung: Partitionsschlüssel ... 290
Datenverteilung: Partitions- und Sortierschlüssel ... 292
Erfahren Sie, wie Sie von SQL zu NoSQL wechseln. ... 294
Relational oder NoSQL? ... 295
Zugriff und Authentifizierung ... 298
Erstellen einer Tabelle ... 302
Abrufen von Informationen zu einer Tabelle ... 304
Schreiben von Daten in eine Tabelle ... 306
Lesen von Daten aus einer Tabelle ... 311
Verwalten von Indexen ... 320
Ändern von Daten in einer Tabelle ... 326
Löschen von Daten aus einer Tabelle ... 330
Entfernen einer Tabelle ... 332
Lernressourcen und Tools für Amazon DynamoDB ... 333
Tools für die Codierung und Visualisierung ... 333
Prescriptive Guidance ... 334
Wissenscenter ... 335
Blogbeiträge, Repositorys und Leitfäden ... 336
Datenmodellierung und Designmuster ... 336
Schulungskurse ... 337
Lese- und Schreibvorgänge ... 338
DynamoDB-Lesekonsistenz ... 338
Letzendlich konsistente Lesevorgänge ... 338
Strikt konsistente Lesevorgänge ... 339
Lesekonsistenz in globalen Tabellen ... 339
Lese- und Schreibvorgänge ... 339
Verbrauch der Leseoperationen ... 340
Verbrauch der Kapazitätseinheiten ... 342
DynamoDB-Durchsatzkapazität ... 344

On-Demand-Modus ... 344
Modus bereitgestellter Kapazität ... 345
On-Demand-Kapazitätsmodus von DynamoDB ... 345
Leseanforderungseinheiten und Schreibanforderungseinheiten ... 347
Anfänglicher Durchsatz und Skalierungseigenschaften ... 347
Maximaler DynamoDB-Durchsatz für On-Demand-Tabellen ... 348
Modus bereitgestellter Kapazität ... 350
Lese- und Schreibkapazitätseinheiten ... 352
Auswählen der ersten Durchsatzeinstellungen ... 352
DynamoDB Auto Scaling ... 354
Verwaltung der Durchsatzkapazität mit Auto Scaling ... 355
Reservierte Kapazität ... 386
Warmdurchsatz ... 387
Überprüfen des Warmdurchsatzes Ihrer Tabelle ... 388
Erhöhen des Warmdurchsatzes für Ihre Tabelle ... 391
Erstellen einer Tabelle mit einem höheren Warmdurchsatz ... 399
Szenarien für den Warmdurchsatz ... 410
Burst- und adaptive Kapazität ... 412
Burst-Kapazität ... 412
Adaptive Kapazität ... 412
Umstellen der Kapazitätsmodi ... 414
Vom Modus bereitgestellter Kapazität zum On-Demand-Modus ... 414
Vom On-Demand-Modus zum Modus bereitgestellter Kapazität ... 416
Programmieren mit DynamoDB ... 418
Überblick über die AWS SDK-Unterstützung für DynamoDB ... 418
SDK-Unterstützung für AWS kontobasierte Endgeräte ... 420
Programmierschnittstellen, die mit DynamoDB funktionieren ... 421
Higher-Level-Programmierschnittstellen ... 428
Ausführen der Codebeispiele ... 493
Low-Level-API ... 501
Programmieren mit Python ... 507
Informationen zu Boto ... 508
Boto-Dokumentation ... 509
Client- und Ressourcenebenen ... 509
Verwenden von batch_writer ... 513
Weitere Codebeispiele ... 513

Sicherheit bei Sitzungen und Threads ... 514
Config ... 514
Fehlerbehandlung ... 520
Protokollierung ... 522
Ereignis-Hooks ... 523
Paginierung und der Paginator ... 524
Waiter ... 527
Programmieren mit JavaScript ... 527
Über AWS SDK für JavaScript ... 528
AWS SDK für JavaScript V3 ... 528
JavaScript Dokumentation ... 528
Abstraktionsebenen ... 529
Marshall-Hilfsfunktion ... 531
Lesen von Elementen ... 532
Bedingte Schreibvorgänge ... 534
Paginierung ... 535
Config ... 537
Waiter ... 540
Fehlerbehandlung ... 541
Protokollierung ... 543
Überlegungen ... 544
Programmieren mit dem AWS SDK for Java 2.x ... 545
Über die AWS SDK for Java 2.x ... 545
Erste Schritte ... 546
SDK for Java-2.x-Dokumentation ... 556
Unterstützte Schnittstellen ... 556
Weitere Codebeispiele ... 571
Synchrone und asynchrone Programmierung ... 572
HTTP-Clients ... 572
Config ... 574
Fehlerbehandlung ... 581
AWS Anforderungs-ID ... 582
Protokollierung ... 583
Paginierung ... 585
Annotationen für Datenklassen ... 587
Fehlerbehandlung ... 587

Fehlerkomponenten ... 588
Transaktionsfehler ... 588
Fehlermeldungen und Codes ... 589
Fehlerbehandlung in Ihrer Anwendung ... 594
Wiederholversuche bei Fehlern und exponentielles Backoff ... 595
Batchoperationen und Fehlerbehandlung ... 596
Arbeitet mit AWS SDKs ... 597
Arbeiten mit DynamoDB ... 599
Arbeiten mit Tabellen ... 599
Grundlegende Operationen für Tabellen ... 600
Überlegungen bei der Auswahl einer Tabellenklasse in DynamoDB ... 609
Tags und Labels ... 610
Arbeiten mit globalen Tabellen ... 616
Konsistenzmodi ... 617
Kontokonfigurationen ... 617
Schlüsselkonzepte ... 619
Globale Tabelle für dasselbe Konto ... 624
Globale Tabellen mit mehreren Konten ... 687
Fakturierung für globale Tabellen ... 710
Versionen der globalen Tabellen ... 713
Bewährte Methoden für globale Tabellen ... 726
Arbeiten mit Elementen ... 728
Elementgrößen und -formate ... 729
Lesen eines Elements ... 730
Schreiben eines Elements ... 731
Rückgabewerte ... 734
Batch-Vorgänge ... 735
Unteilbare Zähler ... 738
Bedingte Schreibvorgänge ... 739
Verwenden von Ausdrücken ... 745
Time to Live (TTL) ... 789
Abfragen von Tabellen ... 819
Scannen von Tabellen ... 829
PartiQL-Abfragesprache ... 838
Arbeiten mit Elementen: Java ... 889
Arbeiten mit Elementen: .NET ... 922

Arbeiten mit Indizes ... 956
Globale sekundäre Indizes ... 961
Lokale sekundäre Indizes ... 1061
Arbeiten mit Transaktionen ... 1115
Funktionsweise ... 1116
Verwenden von IAM mit Transaktionen ... 1126
Beispiel-Code ... 1129
Arbeiten mit Datenströmen ... 1133
Optionen ... 1134
Arbeiten mit Kinesis Data Streams ... 1136
Arbeiten mit DynamoDB Streams ... 1155
In-Memory-Beschleunigung mit DAX ... 1240
Anwendungsfälle für DAX ... 1241
Nutzungshinweise für DAX ... 1242
Funktionsweise ... 1243
Wie DAX-Anforderungen verarbeitet ... 1245
Element-Cache ... 1247
Abfrage-Cache ... 1248
DAX-Cluster-Komponenten ... 1249
Knoten ... 1249
Cluster ... 1250
Regionen und Availability Zones ... 1251
Parametergruppen ... 1252
Sicherheitsgruppen ... 1252
Cluster-ARN ... 1253
Cluster-Endpunkt ... 1253
Knotenendpunkte ... 1253
Subnetzgruppen ... 1254
--Ereignisse ... 1254
Wartungsfenster ... 1254
Erstellen eines DAX-Clusters ... 1256
Erstellen einer IAM-Servicerolle für DAX für den Zugriff auf DynamoDB ... 1256
DAX und IPv6 ... 1258
Unter Verwendung des AWS CLI ... 1261
Verwenden der Konsole ... 1267
Konsistenzmodelle ... 1273

Konsistenz zwischen DAX-Cluster-Knoten ... 1273
Verhalten des DAX-Element-Caches ... 1274
Verhalten des DAX-Abfrage-Caches ... 1278
Strongly Consistent- und Transactional-Lesevorgänge ... 1279
Negative Cache-Speicherung ... 1279
Strategien für Schreibvorgänge ... 1280
Entwicklung mit dem DAX-Client ... 1283
Lernprogramm: Ausführen einer Beispielanwendung ... 1284
Ändern einer vorhandenen Anwendung für die Verwendung von DAX ... 1351
Verwalten von DAX-Clustern ... 1352
IAM-Berechtigungen zum Verwalten eines DAX-Clusters ... 1353
Skalieren eines DAX-Clusters ... 1356
Anpassen der DAX-Cluster-Einstellungen ... 1357
Konfigurieren der TTL-Einstellungen ... 1358
Unterstützung von Markierungen für DAX ... 1360
AWS CloudTrail Integration ... 1361
Löschen eines DAX-Clusters ... 1362
Überwachen von DynamoDB Accelerator ... 1362
DAX-Überwachungstools ... 1362
Überwachung mit CloudWatch ... 1364
Protokollierung von DAX-Vorgängen mit AWS CloudTrail ... 1391
DAX-T3/T2-Instances mit Spitzenlastleistung ... 1392
DAX-T2-Instance-Familie ... 1392
DAX-T3-Instance-Familie ... 1392
DAX-Zugriffskontrolle ... 1393
IAM-Servicerolle für DAX ... 1394
IAM-Richtlinie, um DAX-Cluster-Zugriff zu gewähren ... 1396
Fallstudie: Zugreifen auf DynamoDB und DAX ... 1397
Zugriff auf DynamoDB, aber kein Zugriff mit DAX ... 1399
Zugriff auf DynamoDB und DAX ... 1401
Zugriff auf DynamoDB via DAX, aber kein direkter Zugriff auf DynamoDB ... 1407
DAX-Verschlüsselung im Ruhezustand ... 1409
Aktivierung der Verschlüsselung im Ruhezustand mit dem AWS-Managementkonsole ... 1412
DAX-Verschlüsselung während der Übertragung ... 1413
Verwenden von serviceverknüpften Rollen für DAX ... 1414
Berechtigungen von serviceverknüpften Rollen für DAX ... 1415

Erstellen einer serviceverknüpften Rolle für DAX 1416
Bearbeiten einer serviceverknüpften Rolle für DAX 1417
Löschen einer serviceverknüpften Rolle für DAX 1417
AWS Kontenübergreifender Zugriff auf DAX 1419
IAM-einrichten 1419
Einrichten einer VPC 1422
Ändern des DAX-Clients, um den kontoübergreifenden Zugriff zu erlauben 1424
DAX-Clustergrößenleitfaden 1429
-Übersicht 1430
Schätzung des Datenverkehrs 1430
Lasttest 1431
Datenmodellierung 1434
Arbeiten mit Elementauflistungen 1436
Beschleunigen von Abfragen durch Organisieren der Daten mithilfe von Elementauflistungen 1437
Grundlagen der Datenmodellierung 1437
Design mit einer einzelnen Tabelle 1438
Design mit mehreren Tabellen 1441
Bausteine der Datenmodellierung 1443
Zusammengesetzter Sortierschlüssel 1443
Mehrmandantenfähigkeit 1445
Sparse Index 1446
Time to Live 1447
Time to Live für die Archivierung 1449
Vertikale Partitionierung 1450
Schreib-Sharding 1453
Pakete für das Schemadesign für die Datenmodellierung 1455
Voraussetzungen 1455
Soziales Netzwerk 1456
Gaming-Profil 1466
System zur Beschwerdeverwaltung 1475
Wiederkehrende Zahlungen 1494
Gerätestatusaktualisierungen 1499
Online-Shop 1513
Relationale Modellierung 1537
Traditionelle relationale Datenbankmodelle 1538

So macht DynamoDB JOIN-Operationen überflüssig 1540
So verringern DynamoDB-Transaktionen den Aufwand für den Schreibprozess 1541
Erste Schritte 1542
Beispiel 1545
Migrieren zu DynamoDB 1558
Gründe für die Migration 1558
Überlegungen zur Migration 1560
Funktionsweise 1562
Migrationstools 1563
Auswählen einer Migrationsstrategie 1564
Offline-Migration 1567
Hybridmigration 1569
Online – 1:1-Migration aller Tabellen 1571
Online – Migration mit einer benutzerdefinierten Staging-Tabelle 1572
NoSQL-Workbench 1575
Download 1576
Data Modeler 1577
Erstellen eines neuen Modells 1578
Importieren eines vorhandenen Modells 1580
Bearbeiten eines vorhandenen Modells 1580
Hinzufügen von Beispieldaten 1581
Zugriffsmuster hinzufügen und validieren 1582
Importieren aus CSV 1583
Facets 1584
Aggregierte Ansicht 1585
Exportieren eines Modells 1585
Übergeben eines Datenmodells 1586
Operation Builder 1587
Herstellen einer Verbindung zu Datensätzen 1588
Erstellen von Operationen 1589
Klonen von Tabellen 1602
Exportieren in CSV 1603
Beispieldatenmodelle 1604
Mitarbeiterdatenmodell 1605
Datenmodell des Diskussionsforums 1605
Datenmodell der Musikbibliothek 1606

Datenmodell für Skigebiet ... 1606
Datenmodell für Kreditkartenangebote ... 1607
Datenmodell für Lesezeichen ... 1607
Versionsverlauf ... 1608
Backup und Wiederherstellung ... 1615
Zeitpunktbezogene Backups ... 1617
Bevor Sie beginnen ... 1617
Aktivieren Sie die Wiederherstellung point-in-time ... 1618
On-Demand-Sicherungen ... 1623
Funktionsweise ... 1623
Backup einer Tabelle ... 1627
Wiederherstellen einer Tabelle ... 1630
Löschen eines Tabellen-Backups ... 1636
Verwenden von IAM ... 1638
Fakturierung für Backups ... 1645
Funktionsweise ... 1645
Beispiel für die Fakturierung von DynamoDB-Backups ... 1646
Wiederherstellen ... 1651
Eine Tabelle mithilfe der Wiederherstellung wiederherstellen point-in-time ... 1651
Wiederherstellen einer DynamoDB-Tabelle auf einen bestimmten Zeitpunkt ... 1653
AWS-Backup verwenden ... 1659
Funktionsweise ... 1661
Erstellen von Backups ... 1664
Kopieren eines Backups ... 1666
Wiederherstellen einer Tabelle ... 1667
Löschen eines Backups ... 1668
On-Demand-Backups, verwaltet von AWS Backup versus DynamoDB ... 1669
Codebeispiele ... 1671
Grundlagen ... 1675
Hallo DynamoDB ... 1676
Kennenlernen der Grundlagen ... 1686
Aktionen ... 1843
Szenarien ... 2252
Beschleunigen von Lesevorgängen mit DAX ... 2255
Erweiterte Szenarien für globale sekundäre Indizes ... 2263
Erstellen Sie eine App zum Senden von Daten an eine DynamoDB-Tabelle ... 2266

Vergleichen mehrerer Werte mit einem einzigen Attribut 2268
Bedingtes Aktualisieren der TTL eines Elements 2290
Herstellen einer Verbindung mit einer lokalen Instance 2297
Zählen von Ausdrucksoperatoren 2299
Erstellen einer REST-API zur Verfolgung von COVID-19-Daten 2331
Erstellen einer Messenger-Anwendung 2332
Erstellen einer Serverless-Anwendung zur Verwaltung von Fotos 2333
Erstellen einer Tabelle mit einem globalen sekundären Index 2337
Erstellen einer Tabelle mit aktiviertem Warmdurchsatz 2342
Erstellen einer Webanwendung zur Verfolgung von DynamoDB-Daten 2352
Erstellen einer Websocket-Chat-Anwendung 2354
Erstellen eines Elements mit einer TTL 2354
Erstellen und Verwalten von globalen MRSC-Tabellen 2361
Erstellen und Verwalten globaler Tabellen zur Demonstration von MREC 2381
Löschen von Daten mit PartiQL DELETE 2391
Erkennen von PSA in Bildern 2397
Einfügen von Daten mithilfe von PartiQL INSERT 2398
Aufrufen einer Lambda-Funktion von einem Browser aus 2402
Verwalten globaler sekundärer Indizes 2403
Verwalten ressourcenbasierter Richtlinien 2406
Überwachen der Leistung von Amazon DynamoDB 2409
Ausführen erweiterter Abfrageoperationen 2409
Ausführen von Listenoperationen 2445
Ausführen von Map-Operationen 2477
Ausführen von Set-Operationen 2501
Abfragen einer Tabelle mithilfe von Stapeln von PartiQL-Anweisungen 2532
Abfragen einer Tabelle mit PartiQL 2594
Abfragen einer Tabelle mit einem globalen sekundären Index 2649
Abfragen einer Tabelle mithilfe einer „begins_with“-Bedingung 2658
Abfragen einer Tabelle mit einem Datumsbereich 2664
Abfragen einer Tabelle mit einem komplexen Filterausdruck 2673
Abfragen einer Tabelle mit einem dynamischen Filterausdruck 2681
Abfragen einer Tabelle mit einem Filterausdruck und einem Limit 2690
Abfragen einer Tabelle mit verschachtelten Attributen 2695
Abfragen einer Tabelle mit Paginierung 2704
Abfragen einer Tabelle mit strikt konsistenten Lesevorgängen 2715

Abfragen von Daten mit PartiQL SELECT ... 2719
Abfragen nach TTL-Elementen ... 2725
Abfragen von Tabellen mithilfe von Datums- und Uhrzeitmustern ... 2729
EXIF- und andere Bildinformationen speichern ... 2744
Einrichten der attributbasierten Zugriffskontrolle ... 2745
Verstehen der Reihenfolge der Aktualisierungsausdrücke ... 2748
Aktualisieren der Einstellung für den Warmdurchsatz einer Tabelle ... 2779
Aktualisieren der TTL eines Elements ... 2784
Aktualisieren von Daten mit PartiQL UPDATE ... 2789
Verwenden von API Gateway zum Aufrufen einer Lambda-Funktion ... 2795
Verwenden von Step Functions, um Lambda-Funktionen aufzurufen ... 2798
Verwenden eines Dokumentmodells ... 2798
Verwenden eines übergeordneten Object-Persistence-Modells ... 2814
Verwenden von Operationen mit unteilbaren Zählern ... 2824
Verwenden von bedingten Operationen ... 2847
Verwenden von Ausdrucksattributnamen ... 2877
Verwendung geplanter Ereignisse zum Aufrufen einer Lambda-Funktion ... 2905
Arbeiten mit lokalen sekundären Indizes ... 2907
Arbeite mit Streams und Time-to-Live ... 2909
Arbeiten mit globalen Tabellen und letztendlicher multiregionaler Konsistenz (Multi-Region Eventual Consistency, MREC) ... 2913
Arbeiten mit Ressourcen-Tagging ... 2917
Arbeiten mit der Tabellenverschlüsselung ... 2920
Serverless-Beispiele ... 2922
Aufrufen einer Lambda-Funktion über einen DynamoDB-Auslöser ... 2922
Melden von Batch-Elementfehlern für Lambda-Funktionen mit einem DynamoDB-Auslöser ... 2931
AWS Beiträge der Community ... 2942
Erstellen und Testen einer Serverless-Anwendung ... 2943
Sicherheit ... 2945
AWS verwaltete Richtlinien ... 2946
AWS verwaltete Richtlinien ... 2946
AWS verwaltete Richtlinie: Access_v2 AmazonDynamo DBFull ... 2947
AmazonDynamoDBReadOnlyAccess ... 2948
DynamoDB-Updates für verwaltete Richtlinien AWS ... 2949
Ressourcenbasierte Richtlinien ... 2951

Tabelle erstellen ... 2952
Anhängen einer ressourcenbasierten Richtlinie ... 2958
Anhängen einer Richtlinie an einen Stream ... 2963
Entfernen einer ressourcenbasierten Richtlinie ... 2966
Kontoübergreifender Zugriff ... 2967
Blockieren des öffentlichen Zugriffs ... 2970
API-Operationen ... 2973
IAM-Autorisierung ... 2982
Beispiele ... 2983
Überlegungen ... 2990
Best Practices ... 2992
Attributbasierte Zugriffskontrolle ... 2993
Weshalb sollte ich ABAC verwenden? ... 2995
Bedingungsschlüssel ... 2996
Überlegungen ... 2996
Aktivieren von ABAC in DynamoDB ... 2997
Verwenden von ABAC ... 3000
Beispielanwendungsfälle ... 3002
Fehlerbehebung ... 3015
Datenschutz ... 3017
Verschlüsselung im Ruhezustand ... 3017
Sichern von DynamoDB-Verbindungen ... 3046
IAM ... 3048
Identitäts- und Zugriffsverwaltung ... 3048
Verwenden von Bedingungen ... 3081
Compliance-Validierung ... 3111
Ausfallsicherheit ... 3111
Sicherheit der Infrastruktur ... 3112
Verwenden eines VPC-Endpunkts ... 3113
AWS PrivateLink für DynamoDB ... 3123
Arten von Amazon-VPC-Endpunkten ... 3124
Überlegungen bei der Verwendung AWS PrivateLink für Amazon DynamoDB ... 3125
Erstellen eines Amazon VPC-Endpunkts ... 3125
Zugreifen auf Schnittstellenendpunkte in Amazon DynamoDB ... 3126
Zugreifen auf DynamoDB-Tabellen und -Control-API-Operationen über DynamoDB-Schnittstellenendpunkte ... 3126

Aktualisieren einer lokalen DNS-Konfiguration 3128
Erstellen einer Amazon-VPC-Endpunktrichtlinie 3131
Verwenden von DynamoDB-Endpunkten mit privatem Zugriff AWS-Managementkonsole .. 3132
AWS PrivateLink für DynamoDB Streams 3132
AWS PrivateLink für DAX 3139
Konfigurations- und Schwachstellenanalyse 3142
Bewährte Methoden für die Gewährleistung der Sicherheit 3142
Bewährte Methoden für vorbeugende Sicherheitsmaßnahmen 3143
Bewährte Methoden für aufdeckende Sicherheitsmaßnahmen 3146
Überwachung und Protokollierung 3150
Überwachungsplan 3150
Leistungsbasislinie 3150
Integrierte Services 3151
Automatisierte Überwachungstools 3151
Überwachung von Kennzahlen 3152
Wie verwende ich DynamoDB-Metriken? 3153
Metriken in der Konsole anzeigen CloudWatch 3154
Metriken anzeigen im AWS CLI 3154
Metriken und Dimensionen 3155
CloudWatch Alarme in DynamoDB erstellen 3187
Protokollieren von Operationen 3191
DynamoDB-Informationen in CloudTrail 3192
Grundlagen zu DynamoDB-Protokolldateieinträgen 3196
Contributor Insights 3215
Funktionsweise 3215
Erste Schritte 3227
Verwenden von IAM 3237
Best Practices 3244
NoSQL-Design 3245
NoSQL gegenüber RDBMS 3245
Zwei Schlüsselkonzepte 3246
Allgemeiner Ansatz 3246
NoSQL-Workbench 3247
DynamoDB Well-Architected Lens 3248
Kostenoptimierung 3248

Durchführung der Überprüfung unter Verwendung der Amazon DynamoDB Well-Architected Lens ... 3301
Die Säulen der Amazon DynamoDB Well-Architected Lens ... 3301
Entwerfen von Partitionsschlüsseln ... 3304
Verteilung von Workloads ... 3305
Schreib-Sharding ... 3307
Hochladen von Daten ... 3308
Sortierschlüsselentwurf ... 3310
Versionskontrolle ... 3311
Sekundäre Indizes ... 3312
Allgemeine Richtlinien ... 3313
Sparse Indexes ... 3316
Aggregation ... 3319
GSI-Überladung ... 3326
GSI-Sharding ... 3327
Erstellen eines Replikats ... 3332
Große Elemente ... 3334
Komprimierung ... 3334
Vertikale Partitionierung ... 3335
Verwenden von Amazon S3 ... 3335
Zeitreihendaten ... 3336
Entwurfsmuster für Zeitreihendaten ... 3336
Beispiele für Zeitreihentabellen ... 3337
Many-to-many Beziehungen ... 3338
Adjazenzlisten ... 3338
Materialisierte Diagramme ... 3340
Abfragen und Scannen ... 3345
Scan-Leistung ... 3345
Vermeidung von Spitzen ... 3346
Parallele Scans ... 3349
Tabellendesign ... 3350
Verwenden von globalen Tabellen ... 3350
Wichtige Fakten ... 3351
Wichtige Fakten zu MREC ... 3352
Wichtige Fakten zu MRSC ... 3354
Anwendungsfälle ... 3356

Schreibmodi ... 3357
Routing-Strategien in DynamoDB ... 3363
Evakuierungsprozesse ... 3370
Planung der Durchsatzkapazität ... 3373
Checkliste für die Vorbereitung ... 3375
Fazit und Ressourcen ... 3384
Steuerebene ... 3386
Massendatenoperationen ... 3386
Bedingte Massenaktualisierung ... 3387
Effiziente Massenoperationen ... 3392
Umgang mit gleichzeitigen Updates ... 3394
Optimistische Sperre ... 3395
Pessimistisches Sperren bei Transaktionen ... 3398
Verteiltes Sperren ... 3400
Wählen Sie eine Strategie zur Kontrolle der Parallelität ... 3404
Fakturierungs- und Nutzungsberichte ... 3405
Durchsatzkapazität ... 3408
Streams ... 3413
Speicher ... 3414
Backup und Backup ... 3415
Datenübertragung ... 3419
CloudWatch ... 3420
DAX ... 3421
Migrieren einer DynamoDB-Tabelle zwischen zwei Konten ... 3423
Migrieren einer Tabelle mit AWS Backup für kontoübergreifende Backups und Wiederherstellungen ... 3424
Migrieren einer Tabelle mithilfe des Exports nach S3 und des Imports aus S3 ... 3426
Vorgeschriebene Anleitung für DAX ... 3429
Bewerten der Eignung von DAX ... 3429
Konfigurieren Ihres DAX-Clients ... 3432
Konfigurieren Ihres DAX-Clusters ... 3433
Dimensionieren Ihres DAX-Clusters ... 3441
Bereitstellen eines Clusters ... 3448
Cluster-Operationen ... 3450
Überwachen von DAX ... 3453
DynamoDB mit anderen Diensten verwenden AWS ... 3456

Integrieren mit Amazon Cognito ... 3456
Integrieren mit Amazon Redshift ... 3459
Überlegungen zu einer kontoübergreifenden Integration in CMK ... 3459
Null-ETL-Integration in Amazon Redshift ... 3463
Laden von Daten aus DynamoDB in Amazon Redshift mit COPY ... 3472
Integration in Amazon EMR ... 3474
-Übersicht ... 3474
Tutorial: Arbeiten mit Amazon DynamoDB und Apache Hive ... 3475
Erstellen einer externen Tabelle in Hive ... 3484
Verarbeiten von HiveQL-Anweisungen ... 3488
Abfragen von Daten in DynamoDB ... 3489
Kopieren von Daten in und aus Amazon DynamoDB ... 3492
Leistungsoptimierung ... 3506
Integration mit S3 ... 3512
Importieren aus Amazon S3 ... 3513
Exportieren zu Amazon S3 ... 3534
Integration mit Amazon SageMaker Lakehouse ... 3562
Zero-ETL-Integration mit Amazon Lakehouse SageMaker ... 3562
Integration mit Amazon OpenSearch Service ... 3564
Funktionsweise ... 3565
Erstellen einer Integration ... 3566
Nächste Schritte ... 3566
Umgang mit grundlegenden Änderungen ... 3566
Zero-ETL-Integration mit Service OpenSearch ... 3571
Integration mit Amazon EventBridge ... 3573
Funktionsweise ... 3574
Erstellen einer Integration über die Konsole ... 3575
Nächste Schritte ... 3578
Integrieren in Amazon MSK ... 3579
Funktionsweise ... 3579
Beispielintegration ... 3580
Nächste Schritte ... 3587
Bewährte Methoden für die Integration ... 3587
Erstellen eines Snapshots ... 3587
Erfassung von Datenänderungen ... 3588
Verwenden von generative KI mit DynamoDB ... 3589

Anwendungsfälle für die Nutzung generativer KI in DynamoDB ... 3589
Generative KI-Blogs für DynamoDB ... 3590
Nutzung der DynamoDB-Zero-ETL-Integration mit Service OpenSearch ... 3590
Kontingente und Beschränkungen ... 3592
Ausführen von Aufgaben zur Quotenverwaltung ... 3592
Zugreifen auf DynamoDB-Kontingente ... 3592
Anzeigen Ihrer aktuellen Kontingente in der Konsole ... 3593
Anzeigen der aktuellen Kontingente mithilfe der AWS CLI ... 3594
Beantragen einer Kontingenterhöhung ... 3596
Kontingente ... 3596
Lese- und Schreibdurchsatz ... 3597
Reservierte Kapazität ... 386
Tabellen ... 3601
Globale Tabellen ... 3601
Sekundäre Indizes ... 3603
Projizierte sekundäre Indexattribute ... 3603
DynamoDB Streams ... 3603
Importieren aus Amazon S3 ... 3604
Exportieren von Tabellen zu Amazon S3 ... 3604
Backup und Wiederherstellung ... 3604
Contributor Insights ... 3604
Beschränkungen ... 3604
Lese-/Schreibkapazitätsmodus ... 3605
Sekundäre Indizes ... 3603
Partitions- und Sortierschlüssel ... 3607
Benennungsregeln ... 3607
Datentypen ... 3608
Items ... 3609
Attribute ... 3610
Ausdrucksparameter ... 3610
DynamoDB-Transaktionen ... 3611
DynamoDB Streams ... 3612
DynamoDB Accelerator (DAX) ... 3612
API-spezifische Einschränkungen ... 3613
Ruhende DynamoDB-Verschlüsselung ... 3616
API-Referenz ... 3617

Fehlerbehebung 3618
Interne Serverfehler 3618
Untersuchen interner Serverfehler 3619
Minimieren der Auswirkungen interner Serverfehler 3620
Verbessern der operativen Einblicke 3620
Latenz 3624
Drosselung 3628
Drosselungsdiagnose 3630
Lösungsleitfaden 3637
GSI-Schreibdrosselung und Gegendruck 3665
CloudWatch-Drosselungsmetriken 3667
Anhang 3670
Behebung von Problemen SSL/TLS beim Verbindungsaufbau mit DynamoDB 3670
Testen Ihrer Anwendung oder Ihres Services 3670
Testen des Client-Browsers 3671
Aktualisieren des Software-Anwendungsclients 3671
Aktualisieren Ihres Clientbrowsers 3672
Manuelles Aktualisieren Ihres Zertifikatpakets 3672
Beispieltabellen und -daten zur Verwendung in DynamoDB 3673
Beispieldatendateien 3674
Erstellen von Beispieltabellen und Hochladen von Daten 3687
Erstellen von Beispieltabellen und Hochladen von Daten – Java 3687
Erstellen von Beispieltabellen und Hochladen von Daten – .NET 3697
Beispielanwendung mit AWS SDK für Python (Boto3) 3709
Schritt 1: Lokales Bereitstellen und Testen 3711
Schritt 2: Überprüfen des Datenmodells und der Implementierungsdetails 3715
Schritt 3: Bereitstellen in Produktion 3726
Schritt 4: Bereinigen von Ressourcen 3736
Reservierte Wörter in DynamoDB 3736
AWS SDK-Beispiele für Java 1.x 3750
DAX und Java SDK v1 3750
Verwenden einer vorhandenen SDK for Java 1.x Anwendung zur Nutzung von DAX 3762
Abfragen von globalen sekundären Indizes mit SDK for Java 1.x 3767
AWS Beispiele für SDK for Go 1.x 3771
Go und DAX 3772
AWS Beispiele für SDK für Node.js 2.x 3774

Node.js und DAX 3774
Dokumentverlauf 3785
Frühere Aktualisierungen 3825
Legacy-Features 3861
Version 2017.11.29 (Legacy) der globalen Tabellen 3861
Funktionsweise 3862
Anforderungen und bewährte Methoden 3868
Eine globale Tabelle erstellen (Version 2017.11.29) 3872
Überwachen globaler Tabellen 3877
Verwenden von IAM mit globalen Tabellen 3878
Vorherige Version der Low-Level-API für DynamoDB (2011-12-05) 3882
BatchGetItem 3883
BatchWriteItem 3891
CreateTable 3898
DeleteItem 3908
DeleteTable 3915
DescribeTables 3919
GetItem 3924
ListTables 3928
PutItem 3931
Abfrage 3939
Scan 3956
UpdateItem 3977
UpdateTable 3987
Bedingte Legacy-Parameter für DynamoDB 3992
AttributesToGet 3994
AttributeUpdates 3995
ConditionalOperator 3998
Expected 3999
KeyConditions 4005
QueryFilter 4008
ScanFilter 4011
Schreiben von Bedingungen mit Legacy-Parametern 4013
..... mmmmxxiii

Was ist Amazon DynamoDB?

Amazon DynamoDB ist eine serverlose, vollständig verwaltete, verteilte NoSQL-Datenbank mit einer Leistung im einstelligen Millisekundenbereich in jeder Größenordnung.

Mit DynamoDB lassen sich die komplexen Anforderungen bezüglich Skalierung und Betrieb von relationalen Datenbanken überwinden. DynamoDB wurde speziell für betriebliche Workloads entwickelt und optimiert, die eine konsistente Leistung in jeder Größenordnung erfordern. DynamoDB bietet beispielsweise eine konsistente Leistung im einstelligen Millisekundenbereich für einen Einkaufswagen-Anwendungsfall, unabhängig davon, ob Sie 10 oder 100 Millionen Benutzer haben. Seit seiner Einführung 2012 unterstützt DynamoDB Anwender dabei, sich von relationalen Datenbanken zu verabschieden und gleichzeitig Kosten zu senken und die Leistung spürbar zu verbessern.

Kunden aller Größen, Branchen und Regionen verwenden DynamoDB, um moderne Serverless-Anwendungen zu entwickeln. Diese können von einer kleinen Ausgangsgröße global skaliert werden. Durch seine Skalierbarkeit kann DynamoDB Tabellen praktisch jeder Größe unterstützen. Gleichzeitig bietet es eine konsistente Leistung im einstelligen Millisekundenbereich und hohe Verfügbarkeit.

Bei Ereignissen wie dem Amazon Prime Day unterstützt DynamoDB mehrere Amazon-Standorte und -Systeme mit hohem Datenverkehr, darunter Alexa, Amazon.com-Websites und alle Amazon-Versandzentren. Für solche Ereignisse APIs hat DynamoDB Billionen von Aufrufen von Amazon-Immobilien und -Systemen verarbeitet. DynamoDB bedient kontinuierlich Hunderte Kunden mit Tabellen, die zu Spitzenzeiten über eine halbe Million Anfragen pro Sekunde erhalten. Außerdem bedient es Hunderte Kunden, deren Tabellengrößen 200 TB überschreiten, und verarbeitet über eine Milliarde Anfragen pro Stunde.

Themen

- Merkmale von DynamoDB
- Anwendungsfälle für DynamoDB
- Funktionen von DynamoDB
- Service-Integrationen
- Sicherheit
- Ausfallsicherheit
- Zugreifen auf DynamoDB

- DynamoDB – Preise
- Erste Schritte mit DynamoDB

Merkmale von DynamoDB

Serverless

Mit DynamoDB müssen Sie keine Server bereitstellen oder Software patchen, verwalten, installieren, pflegen oder betreiben. DynamoDB ermöglicht eine Wartung ohne Ausfallzeiten. Es gibt weder Versionen (Hauptversion, Nebenversionen oder Patches) noch Wartungszeitfenster.

Der On-Demand-Kapazitätsmodus von DynamoDB bietet pay-as-you-go Preise für Lese- und Schreibanfragen, sodass Sie nur für das bezahlen, was Sie tatsächlich nutzen. Bei Bedarf skaliert DynamoDB Ihre Tabellen sofort nach oben oder unten, um für die nötige Kapazität zu sorgen. Dabei bleibt das Leistungsniveau ohne Verwaltungsaufwand erhalten. Es wird auch auf Null herunterskaliert, sodass Sie nicht für Durchsatz zahlen, wenn Ihre Tabelle keinen Datenverkehr verzeichnet. Zudem gibt es keine Kaltstarts.

NoSQL

Als NoSQL-Datenbank wurde DynamoDB speziell dafür entwickelt, im Vergleich zu herkömmlichen relationalen Datenbanken eine verbesserte Leistung, Skalierbarkeit, Verwaltbarkeit und Flexibilität zu bieten. Um eine Vielzahl von Anwendungsfällen zu unterstützen, kann DynamoDB sowohl mit Schlüsselwert- als auch Dokumentdatenmodellen verwendet werden.

Im Gegensatz zu relationalen Datenbanken unterstützt DynamoDB keinen JOIN-Operator. Deshalb wird empfohlen, Ihr Datenmodell zu denormalisieren, um Austauschläufe der Datenbank und die für die Beantwortung von Abfragen benötigte Rechenleistung zu reduzieren. Als NoSQL-Datenbank bietet DynamoDB hohe Lesekonsistenz und ACID-Transaktionen zur Erstellung von Unternehmensanwendungen.

Vollständig verwaltet

Als vollständig verwalteter Datenbankservice übernimmt DynamoDB die aufwendigsten Aufgaben der Datenbankverwaltung. So können Sie sich darauf konzentrieren, Mehrwert für Ihre Kunden zu schaffen. Es kümmert sich um Einrichtung, Konfiguration, Wartung, Hochverfügbarkeit, Hardwarebereitstellung, Sicherheit, Backups, Überwachung und mehr. Dadurch ist eine DynamoDB-Tabelle sofort nach dem Erstellen für Produktionsworkloads bereit. DynamoDB verbessert ständig

seine Verfügbarkeit, Zuverlässigkeit, Leistung, Sicherheit und Funktionalität, ohne dass Upgrades oder Ausfallzeiten erforderlich sind.

Leistung im einstelligen Millisekundenbereich

DynamoDB wurde speziell dafür entwickelt, die Performance und Skalierbarkeit relationaler Datenbanken zu verbessern und in jeder Größenordnung eine Leistung im einstelligen Millisekundenbereich zu erzielen. Um diese Skalierbarkeit und Leistung zu erreichen, ist DynamoDB für Hochleistungs-Workloads optimiert und bietet Lösungen APIs , die eine effiziente Datenbanknutzung fördern. Funktionen, die ineffizient sind und sich nicht für Einsätze im großen Umfang eignen, wie z. B. JOIN-Operationen, werden weggelassen. DynamoDB bietet eine konsistente Leistung im einstelligen Millisekundenbereich für Ihre Anwendung, unabhängig davon, ob Sie 100 oder 100 Millionen Benutzer haben.

Anwendungsfälle für DynamoDB

Kunden aller Größen, Branchen und Regionen verwenden DynamoDB, um moderne Serverless-Anwendungen zu entwickeln. Diese können von einer kleinen Ausgangsgröße global skaliert werden. DynamoDB ist ideal für Anwendungsfälle, die eine gleichbleibende Leistung in jeder Größenordnung mit geringem bis gar keinem Betriebsaufwand erfordern. In der folgenden Liste finden Sie einige Anwendungsfälle von DynamoDB:

- Anwendungen für Finanzdienstleistungen – Nehmen wir an, Sie sind ein Finanzdienstleistungsunternehmen, das Anwendungen wie Live-Handel und -Routing, Kreditmanagement, Token-Generierung und Transaktionsbücher entwickelt. Mit globalen DynamoDB-Tabellen können Ihre Anwendungen auf Ereignisse reagieren und Traffic Ihrer Wahl AWS-Regionen mit schneller, lokaler Lese- und Schreibleistung bereitstellen.

  DynamoDB eignet sich für Anwendungen mit den strengsten Verfügbarkeitsanforderungen. Es beseitigt den betrieblichen Aufwand, der durch die manuelle Skalierung von Instances entsteht, um Speicherplatz oder Durchsatz zu erhöhen sowie Versionsverwaltung und Lizenzierung zu regeln.

  Sie können DynamoDB-Transaktionen verwenden, um mit einer einzigen Anfrage Atomizität, Konsistenz, Isolierung und Zuverlässigkeit (ACID) für eine oder mehrere Tabellen zu erreichen. (ACID)-Transaktionen eignen sich für Workloads wie die Verarbeitung von Finanztransaktionen oder die Ausführung von Bestellungen. DynamoDB passt sich umgehend den steigenden oder sinkenden Anforderungen Ihrer Workloads an. So können Sie Ihre Datenbank effizient auf die Marktbedingungen, wie etwa Handelszeiten, abstimmen.

- Spieleanwendungen – Als Spielehersteller können Sie DynamoDB für alle Bereiche von Gaming-Plattformen verwenden, z. B. Spielstatus, Spielerdaten, Sitzungsverlauf und Bestenlisten. Entscheiden Sie sich für DynamoDB wegen seiner Skalierbarkeit, konsistenten Leistung und der Benutzerfreundlichkeit seiner Serverless-Architektur. DynamoDB eignet sich gut für Scale-Out-Architekturen, die zur Unterstützung erfolgreicher Spiele benötigt werden. Es skaliert den Durchsatz Ihres Spiels schnell auf und ab (ohne Kaltstart auf Null). Durch diese Skalierbarkeit wird die Effizienz Ihrer Architektur optimiert, ganz gleich, ob Sie bei hohem Datenverkehr aufskalieren oder bei geringer Gameplay-Auslastung wieder zurückfahren.
- Streaming-Anwendungen – Medien- und Unterhaltungsunternehmen verwenden DynamoDB als Metadatenindex für Inhalte, Content-Management-Dienste oder zur Bereitstellung von Sportstatistiken nahezu in Echtzeit. Außerdem nutzen sie DynamoDB, um Benutzer-Watchlist- und Bookmarking-Dienste auszuführen und Milliarden von täglichen Kundenereignissen zu verarbeiten, um Empfehlungen zu generieren. Diese Kunden profitieren von der Skalierbarkeit, Leistung und Stabilität von DynamoDB. DynamoDB passt sich flexibel an Workload-Änderungen an und ermöglicht so Streaming-Media-Anwendungsfälle, die alle Nachfrageanforderungen erfüllen können.

Weitere Informationen darüber, wie Kunden aus verschiedenen Branchen DynamoDB verwenden, finden Sie unter Amazon DynamoDB Customers und This is My Architecture.

Funktionen von DynamoDB

Multiaktive Replikation mit globalen Tabellen

Globale Tabellen ermöglichen eine multiaktive Replikation Ihrer Daten für alle von Ihnen ausgewählten Daten AWS-Regionen mit einer Verfügbarkeit von 99,999%. Globale Tabellen bieten eine vollständig verwaltete Lösung für die Bereitstellung einer multiregionalen, multiaktiven Datenbank, ohne dass eine eigene Replikationslösung erstellt und gepflegt werden muss. Bei globalen Tabellen können Sie angeben, AWS-Regionen wo die Tabellen verfügbar sein sollen. DynamoDB repliziert laufende Datenänderungen in all diesen Tabellen.

Ihre global verteilten Anwendungen können lokal auf Daten in den ausgewählten Regionen zugreifen, um eine Lese- und Schreibleistung im einstelligen Millisekundenbereich zu erreichen. Da globale Tabellen multiaktiv sind, benötigen Sie keine Primärtabelle. Das bedeutet, dass es beim Failover einer Anwendung zwischen Regionen keine komplizierten oder verzögerten Failovers oder Datenbankausfälle gibt.

ACID-Transaktionen

DynamoDB wurde für geschäftskritische Workloads entwickelt. Es umfasst (ACID)-Transaktionsunterstützung für Anwendungen, die eine komplexe Geschäftslogik erfordern. DynamoDB bietet native serverseitige Unterstützung für Transaktionen und vereinfacht so die Entwicklererfahrung, koordinierte all-or-nothing Änderungen an mehreren Elementen innerhalb und zwischen Tabellen vorzunehmen.

Erfassung von Datenänderungen für ereignisgesteuerte Architekturen

DynamoDB unterstützt das Streaming von Datensätzen für Änderungsdaten auf Elementebene in nahezu Echtzeit. Dafür bietet DynamoDB zwei Streaming-Modelle: DynamoDB Streams und Kinesis Data Streams für DynamoDB. Wenn eine Anwendung Elemente in einer Tabelle erstellt, aktualisiert oder löscht, zeichnet Streams eine zeitgeordnete Reihenfolge aller Änderungen auf Elementebene nahezu in Echtzeit auf. Dies macht DynamoDB Streams ideal für Anwendungen mit ereignisgesteuerter Architektur, um die Änderungen aufzunehmen und darauf zu reagieren.

Sekundäre Indizes

DynamoDB bietet die Möglichkeit, sowohl globale als auch lokale sekundäre Indizes zu erstellen, mit denen Sie die Tabellendaten über einen alternativen Schlüssel abfragen können. Mit diesen sekundären Indizes können Sie auf Daten mit anderen Attributen als dem Primärschlüssel zugreifen. Dies bietet Ihnen maximale Flexibilität beim Datenzugriff.

Service-Integrationen

DynamoDB lässt sich umfassend in mehrere Systeme integrieren AWS-Services , um Ihnen zu helfen, mehr aus Ihren Daten herauszuholen, undifferenzierte Schwerarbeit zu vermeiden und Ihre Workloads skalierbar zu betreiben. Einige Beispiele sind: Amazon AWS CloudFormation CloudWatch, Amazon S3, AWS Identity and Access Management (IAM) und AWS Auto Scaling. In den folgenden Abschnitten werden einige der Service-Integrationen beschrieben, die Sie mit DynamoDB durchführen können:

Serverless-Integrationen

Um end-to-end serverlose Anwendungen zu erstellen, lässt sich DynamoDB nativ in eine Reihe von serverlosen Anwendungen integrieren. AWS-Services So können Sie DynamoDB beispielsweise in

AWS Lambda integrieren, um Auslöser zu erstellen – Codeteile, die automatisch auf Ereignisse in DynamoDB Streams reagieren. Mit Auslösern können Sie ereignisgesteuerte Anwendungen erstellen, die auf Datenänderungen in DynamoDB-Tabellen reagieren. Zur Kostenoptimierung können Sie Ereignisse filtern, die Lambda aus einem DynamoDB-Datenstrom verarbeitet.

Die folgende Liste enthält einige Beispiele für Serverless-Integrationen mit DynamoDB:

- AWS AppSyncfür die Erstellung von GraphQL APIs
- Amazon API Gateway für die Erstellung von REST APIs
- Lambda für Serverless-Computing
- Amazon Kinesis Data Streams für die Erfassung von Datenänderungen

Importieren und Exportieren von Daten in Amazon S3

Durch die Integration von DynamoDB in Amazon S3 können Sie Daten für Analysen und Machine Learning einfach in einen Amazon-S3-Bucket exportieren. DynamoDB unterstützt vollständige Tabellenexporte und inkrementelle Exporte von geänderten, aktualisierten oder gelöschten Daten aus einem bestimmten Zeitraum. Sie können auch Daten aus Amazon S3 in eine neue DynamoDB-Tabelle importieren.

Null-ETL-Integration

DynamoDB unterstützt die Zero-ETL-Integration mit Amazon Redshift und die Verwendung einer OpenSearch Ingestion-Pipeline mit Amazon DynamoDB. Durch diese Integrationen lassen sich komplexe Analysen ausführen und erweiterte Suchfunktionen für DynamoDB-Tabellendaten verwenden. Sie können beispielsweise eine Volltext- und Vektorsuche sowie eine semantische Suche in Ihren DynamoDB-Daten durchführen. Null-ETL-Integrationen wirken sich nicht auf Produktionsworkloads aus, die in DynamoDB ausgeführt werden.

Caching

DynamoDB Accelerator (DAX) ist ein vollständig verwalteter, hoch verfügbarer Caching-Service für DynamoDB. DAX bietet eine bis zu 10-fache Leistungssteigerung – von Millisekunde zu Mikrosekunde – sogar bei Millionen von Anfragen pro Sekunde. DAX fügt Ihren DynamoDB-Tabellen eigenständig speicherinterne Beschleunigung hinzuzufügen, ohne dass Sie sich um Cache-Invalidierung, das Ausfüllen von Daten oder die Clusterverwaltung kümmern müssen.

Sicherheit

DynamoDB nutzt IAM, um einen sicheren Zugriff auf Ressourcen zu ermöglichen. Mit IAM können Sie Berechtigungen, die festlegen, welche DynamoDB-Benutzer auf Ressourcen zugreifen dürfen, zentral verwalten. Sie verwenden IAM, um zu steuern, wer authentifiziert (angemeldet) und autorisiert (Berechtigungen besitzt) ist, Ressourcen zu nutzen. Durch die Verwendung von IAM gibt es keine Benutzernamen oder Passwörter für den Zugriff auf DynamoDB. Dies vereinfacht Ihren Sicherheitsstatus, da Sie keine komplizierten Richtlinien für die Passwortrotation verwalten müssen. Mit IAM können Sie auch eine differenzierte Zugriffskontrolle aktivieren, um Autorisierungen auf Attributebene zu gewähren. Außerdem lassen sich ressourcenbasierte Richtlinien mit Unterstützung für IAM Access Analyzer und Block Public Access (BPA) definieren, um die Richtlinienverwaltung zu vereinfachen.

Standardmäßig verschlüsselt DynamoDB alle Kundendaten im Ruhezustand. Durch Verschlüsselung im Ruhezustand wird die Sicherheit Ihrer Daten erhöht, indem Verschlüsselungsschlüssel verwendet werden, die in AWS Key Management Service (AWS KMS) gespeichert sind. Mit der Verschlüsselung ruhender Daten können Sie sicherheitsrelevante Anwendungen erstellen, die eine strenge Einhaltung der Verschlüsselungsvorschriften und der gesetzlichen Bestimmungen erfordern. Wenn Sie auf eine verschlüsselte Tabelle zugreifen, entschlüsselt DynamoDB die Tabellendaten transparent. Sie müssen keinen Code oder Anwendungen ändern, um verschlüsselte Tabellen zu verwenden oder zu verwalten. DynamoDB bietet weiterhin dieselbe Latenz im einstelligen Millisekundenbereich, die Sie erwarten, und alle DynamoDB-Abfragen funktionieren problemlos mit Ihren verschlüsselten Daten.

Sie können angeben, ob DynamoDB einen AWS-eigener Schlüssel (Standardverschlüsselungstyp) oder einen vom Kunden verwalteten Schlüssel zum Verschlüsseln von Benutzerdaten verwenden soll. Von AWS verwalteter Schlüssel Die Standardverschlüsselung mit AWS-eigenen KMS-Schlüsseln ist ohne Aufpreis verfügbar. Für eine clientseitige Verschlüsselung können Sie das AWS Database Encryption SDK verwenden.

DynamoDB erfüllt außerdem verschiedene Compliance-Standards, darunter HIPAA, PCI DSS und DSGVO, sodass Sie gesetzliche Anforderungen einhalten können.

Ausfallsicherheit

Standardmäßig repliziert DynamoDB Ihre Daten automatisch über drei Availability Zones hinweg, um für eine lange Haltbarkeit und ein Verfügbarkeits-SLA von 99,99 % zu sorgen. Außerdem bietet DynamoDB zusätzliche Funktionen, mit deren Hilfe Sie Ihre Ziele für Geschäftskontinuität und Notfallwiederherstellung erreichen können.

Mit folgenden Funktionen unterstützt DynamoDB Ihre Anforderungen in Bezug auf Ausfallsicherheit und Datensicherung:

Features

- Globale Tabellen
- Kontinuierliche Backups und Wiederherstellung point-in-time
- On-Demand-Backup und Wiederherstellung

Globale Tabellen

Globale DynamoDB-Tabellen ermöglichen ein SLA mit einer Verfügbarkeit von 99,999 % und Ausfallsicherheit in mehreren Regionen. Dies hilft Ihnen, robuste Anwendungen zu erstellen und sie für das niedrigste Recovery Time Objective (RTO) und Recovery Point Objective (RPO) zu optimieren. Globale Tabellen lassen sich auch in AWS Fault Injection Service (AWS FIS) integrieren, um Fehlerinjektionstests für Ihre globalen Tabellenworkloads durchzuführen. Dazu gehört zum Beispiel das Anhalten der globalen Tabellenreplikation für eine beliebige Replikattabelle.

Kontinuierliche Backups und Wiederherstellung point-in-time

Kontinuierliche Backups bieten Ihnen eine Genauigkeit pro Sekunde und die Möglichkeit, eine point-in-time Wiederherstellung einzuleiten. Mit point-in-time Recovery können Sie eine Tabelle zu einem beliebigen Zeitpunkt bis zur Sekunde der letzten 35 Tage wiederherstellen. Der Wiederherstellungszeitraum kann auf einen Wert zwischen 1 und 35 Tagen festgelegt werden.

Kontinuierliche Backups und das Initiieren einer point-in-time Wiederherstellung verbrauchen keine bereitgestellte Kapazität. Sie wirken sich auch nicht auf die Leistung oder Verfügbarkeit Ihrer Anwendungen aus.

On-Demand-Backup und Wiederherstellung

Sie können die Funktion On-Demand-Sicherung und Wiederherstellung nutzen, um vollständige Sicherungen einer Tabelle für die langfristige Aufbewahrung und Archivierung für behördliche Compliance-Anforderungen zu erstellen. Sicherungen wirken sich nicht auf die Leistung der Tabelle aus. Zudem können Sie Tabellen jeder Größe sichern. Mit der AWS Backup Integration können Sie AWS Backup den Lebenszyklus Ihrer DynamoDB-Backups auf Abruf automatisch planen, kopieren, taggen und verwalten. Mithilfe AWS Backup können Sie On-Demand-Backups zwischen Konten und Regionen kopieren und ältere Backups zur Kostenoptimierung in Cold Storage migrieren.

Zugreifen auf DynamoDB

Sie können mit DynamoDB arbeiten, indem Sie AWS-Managementkonsole, AWS Command Line InterfaceNoSQL-Workbench für DynamoDB, oder DynamoDB verwenden. APIs

Weitere Informationen finden Sie unter Zugreifen auf DynamoDB.

DynamoDB – Preise

DynamoDB berechnet Gebühren für das Lesen, Schreiben und Speichern von Daten in Ihren Tabellen sowie für alle optionalen Funktionen, die Sie aktivieren. DynamoDB bietet zwei Kapazitätsmodi mit spezifischen Abrechnungsoptionen für die Verarbeitung von Lese- und Schreibvorgängen für Ihre Tabellen: On-Demand und Bereitgestellt.

DynamoDB ist auch im Kontingent „Immer kostenlos“ mit 25 GB Speicherplatz enthalten. Darüber hinaus beinhaltet das Kontingent „Immer kostenlos“ 25 bereitgestellte Schreib- und 25 bereitgestellte Lesekapazitätseinheiten (WCU, RCU), was für die Verarbeitung von 200 Millionen Anfragen pro Monat ausreicht.

Weitere Informationen finden Sie unter Amazon DynamoDB – Preise.

Erste Schritte mit DynamoDB

Wenn Sie DynamoDB zum ersten Mal verwenden, empfehlen wir, die folgenden Themen zu lesen:

- Erste Schritte mit DynamoDB – erläutert die Einrichtung von DynamoDB, die Erstellung von Beispieltabellen und das Hochladen von Daten. Dieses Thema enthält auch Informationen zur Ausführung einiger grundlegender Datenbankoperationen mit der AWS-Managementkonsole, AWS CLI, NoSQL Workbench und DynamoDB. APIs
- DynamoDB-Kernkomponenten – beschreibt die grundlegenden DynamoDB-Konzepte.
- Bewährte Methoden für Design und Architektur mit DynamoDB – Empfehlungen zum NoSQL-Design, zu DynamoDB Well-Architected Lens, zum Tabellendesign und zu verschiedenen anderen DynamoDB-Funktionen. Mithilfe dieser bewährten Methoden können Sie bei der Arbeit mit DynamoDB die Leistung maximieren und die Durchsatzkosten minimieren.

Wir empfehlen Ihnen außerdem, die folgenden Tutorials zu lesen, in denen vollständige end-to-end Verfahren vorgestellt werden, um sich mit DynamoDB vertraut zu machen. Sie können diese Tutorials im Rahmen des Kontingents „Immer kostenlos“ absolvieren.

- Create and Query a NoSQL Table with Amazon DynamoDB
- Build an Application Using a NoSQL Key-Value Data Store

Informationen zu Ressourcen, Tools und Strategien für die Migration zu DynamoDB finden Sie unter Migrating to DynamoDB. Aktuelle Blogs und Whitepaper finden Sie unter Amazon DynamoDB resources.

Erste Schritte mit DynamoDB

In den folgenden Abschnitten erfahren Sie, wie Sie eine Verbindung zu DynamoDB-Tabellen herstellen, diese erstellen und verwalten.

Bevor Sie beginnen, sollten Sie sich mit den Basic-Konzepten in Amazon DynamoDB vertraut machen. Einen schnellen Überblick über Was ist Amazon DynamoDB? und einen tieferen Einblick erhalten Sie unter Kernkomponenten von Amazon DynamoDB. Fahren Sie dann mit den Voraussetzungen fort.

> **Note**
>
> Wenn Sie sich für anmelden AWS, können Sie mithilfe des AWS kostenlosen Kontingents mit DynamoDB beginnen. Sofern Sie das kostenlose Kontingent für Amazon DynamoDB noch nicht überschritten haben, können Sie die Beispiele in diesem Abschnitt kostenlos bearbeiten. Andernfalls fallen die üblichen DynamoDB-Nutzungsgebühren an, und zwar zwischen dem Zeitpunkt, an dem Sie sie erstellen, und zu dem Zeitpunkt, an dem Sie sie löschen.
> Wenn Sie sich nicht für ein Konto mit kostenlosem Kontingent registrieren möchten, können Sie die lokale Version von DynamoDB (herunterladbare Version) auf dem Computer einrichten. Mit der herunterladbaren Version können Sie Anwendungen lokal entwickeln und testen, ohne sich für ein AWS Konto registrieren oder auf den DynamoDB-Webservice zugreifen zu müssen.

Themen

- Amazon DynamoDB-Ressourcen für Erstbenutzer
- Zugreifen auf DynamoDB
- Voraussetzungen
- Einrichten von DynamoDB
- Schritt 1: Erstellen einer Tabelle in DynamoDB
- Schritt 2: Schreiben von Daten in eine DynamoDB-Tabelle
- Schritt 3: Lesen von Daten aus einer DynamoDB-Tabelle
- Schritt 4: Aktualisieren von Daten in einer DynamoDB-Tabelle
- Schritt 5: Abfragen der Daten in der DynamoDB-Tabelle

- Schritt 6: (Optional) Löschen der DynamoDB-Tabelle, um Ressourcen zu bereinigen
- Erfahren Sie mehr über DynamoDB
- Generieren Sie Infrastrukturcode für Amazon DynamoDB mit Console-to-Code

Amazon DynamoDB-Ressourcen für Erstbenutzer

Wir empfehlen Erstbenutzern, mit der Lektüre der folgenden Abschnitte zu beginnen und bei Bedarf darauf zurückzugreifen.

Example Leistungsschwerpunkte und Preise Bietet einen allgemeinen Produktüberblick über DynamoDB, allgemeine Anwendungsfälle, Servicehighlights und Preise.	Example DynamoDB-Ressourcen Videos, Tutorials und Anleitungen, die Sie mit dem Service, den Konzepten der Datenmodellierung und den wichtigsten Funktionen und Fähigkeiten vertraut machen.
Example Erste Schritte Informationen zum Einrichten von DynamoDB, zum Erstellen von Beispieltabellen und zum Hochladen von Daten.	Example DynamoDB-Grundlagenkurs Kostenloser digitaler Kurs, der die Grundlagen von DynamoDB vermittelt, einschließlich Tabellendesign, Datentypen und Grundoperationen.
Example DynamoDB-Nuggets Kurze, gezielte Video-Tutorials, in denen die wichtigsten Konzepte und Funktionen von DynamoDB erklärt werden.	Example DynamoDB-Codebeispiel-Repository Praktische DynamoDB-Codebeispiele in verschiedenen Programmiersprachen.
Example Kostenlose DynamoDB-Schulung	Example NoSQL Workbench für DynamoDB

AWS bietet kostenlose digitale Schulungen, die sich mit Konzepten, Funktionen und bewährten Methoden von DynamoDB befassen.	Einheitliches visuelles Tool, das Funktionen für Datenmodellierung, Datenvisualisierung und Abfrageentwicklung bietet.
Example DynamoDB-Entwurfsmuster Bewährte Methoden und Beispiele für Datenmodellierung für verschiedene Anwendungsfälle mit praktischen Codebeispielen.	Example Praktische Tutorials Step-by-step Tutorials im AWS-Managementkonsole , die Sie durch allgemeine DynamoDB-Aufgaben führen.
Example Migration zu DynamoDB Überblick über den Prozess, die Tools und Strategien für die Migration einer Datenbank zu DynamoDB.	Example AWS Well-Architected Objektiv für DynamoDB Bewährte Architekturpraktiken für die Entwicklung und den Betrieb zuverlässiger, sicherer, effizienter und kostengünstiger Anwendungen mit DynamoDB.

Zusätzliche bewährte Methoden für Amazon DynamoDB für Erstbenutzer

Wenn Sie die vorhergehenden Abschnitte abgeschlossen haben, lesen die folgenden:

- DynamoDB-Durchsatzkapazität

  Bietet Überblick über die beiden Durchsatzmodi, die für Ihre DynamoDB-Tabelle verfügbar sind, sowie Überlegungen zur Auswahl des geeigneten Kapazitätsmodus für Ihre Anwendung. Der On-Demand-Modus ist die standardmäßige und empfohlene Durchsatzoption für die meisten DynamoDB-Workloads.

- Best Practices

  Identifizieren und beheben Sie Probleme, um die Leistung zu maximieren und die Kosten bei der Arbeit mit DynamoDB zu minimieren.

AWS CLI Ressourcen

Wenn Sie die AWS Befehlszeilenschnittstelle (AWS CLI) verwenden möchten, können Sie diese Dokumente verwenden, um Ihnen den Einstieg zu erleichtern:

- AWS CLI -Dokumentation

  Dieser Abschnitt enthält Informationen zum Herunterladen von AWS CLI, zum AWS CLI Betrieb auf Ihrem System und zum Eingeben Ihrer AWS Anmeldeinformationen.
- AWS CLI Dokumentation für DynamoDB

  Dieses separate Dokument behandelt alle Befehle AWS CLI für DynamoDB, einschließlich Syntax und Beispielen.

Programmierressourcen

Sie können Anwendungsprogramme unter Verwendung der DynamoDB-API mit einer Reihe gängiger Programmiersprachen erstellen. Nachfolgend finden Sie einige Ressourcen:

- Tools für Amazon Web Services

  AWS bietet eine Reihe von Software Development Kits (SDKs) mit Unterstützung für DynamoDB. Sie können Java, .NET, PHP, Ruby und andere Sprachen für die DynamoDB-Programmierung verwenden. Diese SDKs können Ihre Anwendungsentwicklung erheblich vereinfachen, indem sie Ihre Anfragen in DynamoDB formatieren, Antworten analysieren und Wiederholungslogik und Fehlerbehandlung bereitstellen.
- Referenz zur DynamoDB-API

  Wenn Sie das nicht verwenden möchten AWS SDKs, können Sie über die DynamoDB-API direkt mit DynamoDB interagieren. In diesem separaten Dokument finden Sie alle DynamoDB-API-Operationen zusammen mit Syntax und Beispielen. In diesem Abschnitt finden Sie auch Tipps zur Fehlerbehebung sowie Informationen zur Erstellung und Authentifizierung von Anfragen und der Behandlung von Antworten.

Zugreifen auf DynamoDB

Sie können mit der AWS-Managementkonsole, der AWS Command Line Interface (AWS CLI) oder der DynamoDB-API auf Amazon DynamoDB zugreifen.

Themen

- Verwenden der Konsole
- Mit dem AWS CLI
- Verwenden der API
- Verwenden von NoSQL-Workbench für DynamoDB
- IP-Adressbereiche
- Dual-Stack-Endpunkte für Internet Protocol Version 6 () IPv6

Verwenden der Konsole

Sie können zu Hause auf die AWS-Managementkonsole für Amazon DynamoDB zugreifen. https://console.aws.amazon.com/dynamodb/

Hier sind einige der Aktionen, die Sie in der DynamoDB-Konsole ausführen können:

- Tabellen verwalten: Erstellen, aktualisieren und löschen Sie Tabellen. Der Kapazitätsrechner kann Ihnen helfen, den Kapazitätsbedarf abzuschätzen.
- Mit Daten interagieren: Sie können Elemente in Ihren Tabellen anzeigen, hinzufügen, aktualisieren und löschen. Außerdem können Sie die Einstellungen für Time to Live (TTL) verwalten.
- Überwachen und analysieren: Zeigen Sie Dashboards an, überwachen und richten Sie Alarme ein und analysieren Sie Metriken und Warnungen für Ihre DynamoDB-Tabellen.
- Optimieren und erweitern: Verwalten Sie sekundäre Indizes, Streams, Trigger, reservierte Kapazität und andere erweiterte Funktionen, um Ihre DynamoDB-Nutzung zu verbessern.

Die DynamoDB-Konsole bietet eine umfassende Oberfläche für die Verwaltung Ihrer DynamoDB-Ressourcen. Wir empfehlen Ihnen, auf die Konsole zuzugreifen und mit ihr zu interagieren, um mehr zu erfahren.

Mit dem AWS CLI

Sie können die AWS Command Line Interface (AWS CLI) verwenden, um mehrere AWS Dienste von der Befehlszeile aus zu steuern und sie mithilfe von Skripten zu automatisieren. Sie können das AWS CLI für Ad-hoc-Operationen verwenden, z. B. das Erstellen einer Tabelle. Sie können damit auch Amazon-DynamoDB-Operationen in Hilfsprogrammskripts einbetten.

Bevor Sie das AWS CLI mit DynamoDB verwenden können, benötigen Sie eine Zugriffsschlüssel-ID und einen geheimen Zugriffsschlüssel. Weitere Informationen finden Sie unter Erteilen programmgesteuerten Zugriffs.

Eine vollständige Liste aller Befehle, die für DynamoDB in verfügbar sind AWS CLI, finden Sie in der AWS CLI Befehlsreferenz.

Herunterladen und Konfigurieren der AWS CLI

Das AWS CLI ist unter http://aws.amazon.com/cli verfügbar. Sie kann auf Windows, macOS oder Linux ausgeführt werden. Gehen Sie nach dem Herunterladen wie folgt vor AWS CLI, um es zu installieren und zu konfigurieren:

1. Rufen Sie auf das AWS Command Line Interface -Benutzerhandbuch auf.
2. Befolgen Sie die Anweisungen unter Installieren der AWS CLI und Konfigurieren der AWS CLI.

Verwenden von AWS CLI mit DynamoDB

Das Befehlszeilenformat besteht aus einem DynamoDB-Operationsnamen gefolgt von den Parametern für diese Operation. Das AWS CLI unterstützt eine Kurzsyntax für die Parameterwerte sowie JSON.

Mit dem folgenden Befehl wird beispielsweise eine Tabelle namens Musik erstellt. Der Partitionsschlüssel ist Artist und der Sortierschlüssel ist. SongTitle (Für eine bessere Lesbarkeit werden lange Befehle in diesem Abschnitt über mehrere Zeilen verteilt.)

```
aws dynamodb create-table \
    --table-name Music \
    --attribute-definitions \
        AttributeName=Artist,AttributeType=S \
        AttributeName=SongTitle,AttributeType=S \
    --key-schema AttributeName=Artist,KeyType=HASH
 AttributeName=SongTitle,KeyType=RANGE \
    --billing-mode PAY_PER_REQUEST \
    --table-class STANDARD
```

Mit den folgenden Befehlen werden der Tabelle neue Elemente hinzugefügt. Diese Beispiele verwenden eine Kombination von Syntax-Kurznotation und JSON.

```
aws dynamodb put-item \
```

```
    --table-name Music \
    --item \
        '{"Artist": {"S": "No One You Know"}, "SongTitle": {"S": "Call Me Today"},
 "AlbumTitle": {"S": "Somewhat Famous"}}' \
    --return-consumed-capacity TOTAL

aws dynamodb put-item \
    --table-name Music \
    --item '{
        "Artist": {"S": "Acme Band"},
        "SongTitle": {"S": "Happy Day"},
        "AlbumTitle": {"S": "Songs About Life"} }' \
    --return-consumed-capacity TOTAL
```

Es ist nicht einfach, in der Befehlszeile gültigen JSON-Code zu erstellen. Die AWS CLI kann aber JSON-Dateien lesen. Betrachten Sie dazu das folgende Beispiel für einen JSON-Codeausschnitt, der in einer Datei mit dem Namen key-conditions.json gespeichert ist:

```
{
    "Artist": {
        "AttributeValueList": [
            {
                "S": "No One You Know"
            }
        ],
        "ComparisonOperator": "EQ"
    },
    "SongTitle": {
        "AttributeValueList": [
            {
                "S": "Call Me Today"
            }
        ],
        "ComparisonOperator": "EQ"
    }
}
```

Sie können jetzt eine `Query`-Anforderung über die AWS CLI ausgeben. In diesem Beispiel werden die Inhalte der Datei key-conditions.json für den Parameter `--key-conditions` verwendet:

```
aws dynamodb query --table-name Music --key-conditions file://key-conditions.json
```

Verwenden von AWS CLI mit DynamoDB local

AWS CLI Sie können auch mit DynamoDB Local (herunterladbare Version) interagieren, die auf Ihrem Computer läuft. Um diese Funktion zu aktivieren, fügen Sie jedem Befehl den folgenden Parameter hinzu:

```
--endpoint-url http://localhost:8000
```

Das folgende Beispiel verwendet die AWS CLI , um die Tabellen in einer lokalen Datenbank aufzulisten.

```
aws dynamodb list-tables --endpoint-url http://localhost:8000
```

Wenn DynamoDB eine andere Portnummer verwendet als den Standardport (8000), ändern Sie den `--endpoint-url` Wert entsprechend.

> **Note**
>
> Die lokale DynamoDB-Version (herunterladbare Version) AWS CLI kann nicht als Standardendpunkt verwendet werden. Daher müssen Sie mit jedem Befehl `--endpoint-url` angeben.

Verwenden der API

Sie können die AWS-Managementkonsole und die verwenden AWS Command Line Interface , um interaktiv mit Amazon DynamoDB zu arbeiten. Um DynamoDB jedoch optimal zu nutzen, können Sie Anwendungscode mit dem schreiben. AWS SDKs

AWS SDKs Sie bieten umfassende Unterstützung für DynamoDB in Java, JavaScript im Browser, .NET, Node.js, PHP, Python, Ruby, C++, Go, Android und iOS.

Bevor Sie das AWS SDKs mit DynamoDB verwenden können, benötigen Sie eine AWS Zugriffsschlüssel-ID und einen geheimen Zugriffsschlüssel. Weitere Informationen finden Sie unter Einrichten von DynamoDB (Webservice).

Einen allgemeinen Überblick über die DynamoDB-Anwendungsprogrammierung mit dem finden Sie AWS SDKs unter. Programmieren mit DynamoDB und AWS SDKs

Verwenden von NoSQL-Workbench für DynamoDB

Sie können auch die NoSQL-Workbench für DynamoDB herunterladen und verwenden, um auf DynamoDB zuzugreifen.

NoSQL Workbench for Amazon DynamoDB ist eine plattformübergreifende clientseitige GUI-Anwendung für moderne Datenbankentwicklung und -operationen. Sie ist für Windows, macOS und Linux verfügbar. NoSQL Workbench ist ein visuelles Entwicklungstool, das Funktionen zur Datenmodellierung, Datenvisualisierung und Abfrageentwicklung bereitstellt, mit denen Sie DynamoDB-Tabellen entwerfen, erstellen, abfragen und verwalten können. NoSQL Workbench enthält jetzt die lokale Version von DynamoDB als optionalen Bestandteil des Installationsprozesses. Dies macht es einfacher, Ihre Daten in die lokale Version von DynamoDB zu modellieren. Weitere Informationen über die lokale Version von DynamoDB und ihre Anforderungen finden Sie unter Lokale Einrichtung von DynamoDB (herunterladbare Version).

> **Note**
>
> Die NoSQL Workbench für DynamoDB unterstützt derzeit keine AWS Anmeldungen, die mit Zwei-Faktor-Authentifizierung (2FA) konfiguriert sind.

Datenmodellierung

Mit NoSQL-Workbench für DynamoDB können Sie neue Datenmodelle aus vorhandenen Datenmodellen erstellen oder Modelle basierend auf vorhandenen Datenmodellen entwerfen, die den Datenzugriffsmustern Ihrer Anwendung entsprechen. Sie können das gestaltete Datenmodell am Ende des Prozesses auch importieren und exportieren. Weitere Informationen finden Sie unter Erstellen von Datenmodellen mit NoSQL Workbench.

Erstellen von Operationen

NoSQL Workbench stellt eine umfassende Grafikbenutzeroberfläche für die Entwicklung und den Test von Abfragen bereit. Sie können den Operation Builder verwenden, um Live-Datensätze anzuzeigen, zu erkunden und abzufragen. Sie können auch den strukturieren Operation Builder verwenden, um Datenebenen-Operationen zu erstellen und durchzuführen. Er unterstützt Projektions- und Konditionsausdrücke und ermöglicht das Generieren von Beispielcode in mehreren Sprachen. Weitere Informationen finden Sie unter Erkunden von Datasets und Erstellen von Vorgängen mit NoSQL Workbench.

IP-Adressbereiche

Amazon Web Services (AWS) veröffentlicht seine aktuellen IP-Adressbereiche im JSON-Format. Laden Sie die Datei ip-ranges.json herunter, um die aktuellen Bereiche anzuzeigen. Weitere Informationen finden Sie unter AWS IP-Adressbereiche im Allgemeine AWS-Referenz.

Um die IP-Adressbereiche zu suchen, die Sie verwenden können, um auf die DynamoDB-Tabellen und Indizes zuzugreifen, suchen Sie in der Datei "ip-ranges.json" nach folgender Zeichenfolge: `"service": "DYNAMODB"`.

> **Note**
>
> Die IP-Adressbereiche gelten nicht für DynamoDB Streams oder DynamoDB Accelerator (DAX).

Dual-Stack-Endpunkte für Internet Protocol Version 6 () IPv6

DynamoDB bietet Dual-Stack-Endpunkte, die sowohl mit als auch kompatibel sind. IPv4 IPv6 Die Benennungskonventionen für Endpunkte lauten wie folgt:

- `dynamodb.<region>.api.aws`
- `<account-id>.ddb.<region>.api.aws`
- `streams-dynamodb.<region>.api.aws`
- `dax.<region>.api.aws`
- `dynamodb-fips.<region>.api.aws`

Eine vollständige Liste der DynamoDB-Endpunkte und der regionalen Verfügbarkeit finden Sie im Thema Amazon DynamoDB DynamoDB-Endpunkte und Kontingente im Allgemeinen Referenzhandbuch.AWS

Weitere Informationen zur Einrichtung der AWS CLI für die Verwendung von Dual-Stack-Endpunkten finden Sie im Abschnitt Zur Verwendung von Dual-Stack-Endpunkten für alle AWS Dienste einrichten im Leitfaden zur AWS Befehlszeilenschnittstelle.

Weitere Informationen zur Einrichtung Ihrer SDK-Clients für die Verwendung von Dual-Stack-Endpunkten finden Sie unter dem Thema Dual-Stack- und FIPS-Endpunkte im Handbuch und Tools.AWS SDKs

Bevor Sie DynamoDB mit verwenden IPv6, müssen Sie Ihre IAM-Benutzerrolle oder die ressourcenbasierten Richtlinien, die Sie für die IP-Adressfilterung verwenden, aktualisieren, um Adressbereiche einzubeziehen. IPv6 IP-Adressfilterrichtlinien, die Adressen nicht berücksichtigen, können zu IPv6 Zugriffsproblemen führen. Weitere Informationen finden Sie im Abschnitt Operatoren für IP-Adressbedingungen im Handbuch AWS Identity and Access Management.

Voraussetzungen

Machen Sie sich vor dem Start des Amazon-DynamoDB-Tutorials damit vertraut, wie Sie in Zugreifen auf DynamoDB auf DynamoDB zugreifen können. Richten Sie DynamoDB anschließend entweder über den Webservice oder die lokal heruntergeladene Version in Einrichten von DynamoDB ein. Fahren Sie danach fort mit Schritt 1: Erstellen einer Tabelle in DynamoDB.

> **Note**
>
> - Wenn Sie nur über den mit DynamoDB interagieren möchten AWS-Managementkonsole, benötigen Sie keinen AWS Zugriffsschlüssel. Führen Sie die Schritte unter Registrierung für AWS aus und fahren Sie dann fort mit. Schritt 1: Erstellen einer Tabelle in DynamoDB
> - Wenn Sie sich nicht für ein Konto mit kostenlosem Kontingent registrieren möchten, können Sie die lokale Version von DynamoDB (herunterladbare Version) einrichten. Fahren Sie dann mit Schritt 1: Erstellen einer Tabelle in DynamoDB fort.
> - Bei der Arbeit mit CLI-Befehlen in Terminals unter Linux und unter Windows gibt es Unterschiede. Die folgende Anleitung enthält Befehle, die für Linux-Terminals (einschließlich macOS) formatiert sind, sowie Befehle, die für Windows CMD formatiert sind. Wählen Sie den Befehl, der am besten zu der von Ihnen verwendeten Terminalanwendung passt.

Einrichten von DynamoDB

AWS Bietet zusätzlich zum Amazon DynamoDB DynamoDB-Webservice eine herunterladbare Version von DynamoDB, die Sie auf Ihrem Computer ausführen können. Die herunterladbare Version ist nützlich für die Entwicklung und das Testen Ihres Codes. Damit können Sie Anwendungen ohne Zugriff auf den DynamoDB-Webservice lokal entwickeln und testen.

Die Themen in diesem Abschnitt beschreiben, wie Sie DynamoDB (herunterladbare Version) und den DynamoDB-Webservice einrichten.

Themen

- Einrichten von DynamoDB (Webservice)
- Lokale Einrichtung von DynamoDB (herunterladbare Version)

Einrichten von DynamoDB (Webservice)

So verwenden Sie den Amazon-DynamoDB-Webservice:

1. Registrieren für AWS.
2. Rufen Sie einen AWS Zugriffsschlüssel ab (wird verwendet, um programmgesteuert auf DynamoDB zuzugreifen).

   > **Note**
   >
   > Wenn Sie planen, nur über den mit DynamoDB zu interagieren AWS-Managementkonsole, benötigen Sie keinen AWS Zugriffsschlüssel, und Sie können direkt zu. Verwenden der Konsole

3. Konfigurieren der Anmeldeinformationen (für den programmgesteuerten Zugriff auf DynamoDB).

Melden Sie sich an für AWS

Um den DynamoDB-Dienst nutzen zu können, benötigen Sie ein AWS Konto. Wenn Sie noch kein Konto besitzen, werden Sie bei der Anmeldung aufgefordert, eines zu erstellen. Für AWS Dienste, für die Sie sich registrieren, werden Ihnen keine Gebühren berechnet, es sei denn, Sie nutzen sie.

Um sich anzumelden für AWS

1. Öffnen Sie https://portal.aws.amazon.com/billing/die Anmeldung.
2. Folgen Sie den Online-Anweisungen.

   Während der Anmeldung erhalten Sie einen Telefonanruf oder eine Textnachricht und müssen einen Verifizierungscode über die Telefontasten eingeben.

   Wenn Sie sich für eine anmelden AWS-Konto, Root-Benutzer des AWS-Kontoswird eine erstellt. Der Root-Benutzer hat Zugriff auf alle AWS-Services und Ressourcen des Kontos. Als bewährte Sicherheitsmethode weisen Sie einem Benutzer Administratorzugriff zu und verwenden Sie nur den Root-Benutzer, um Aufgaben auszuführen, die Root-Benutzerzugriff erfordern.

Erteilen programmgesteuerten Zugriffs

Bevor Sie programmgesteuert oder über AWS Command Line Interface (AWS CLI) auf DynamoDB zugreifen können, benötigen Sie programmatischen Zugriff. Sie benötigen keinen programmgesteuerten Zugriff, wenn Sie nur die DynamoDB-Konsole verwenden möchten.

Benutzer benötigen programmatischen Zugriff, wenn sie mit außerhalb von interagieren möchten. AWS AWS-Managementkonsole Die Art und Weise, wie programmatischer Zugriff gewährt wird, hängt von der Art des Benutzers ab, der zugreift. AWS

Um Benutzern programmgesteuerten Zugriff zu gewähren, wählen Sie eine der folgenden Optionen.

Welcher Benutzer benötigt programmgesteuerten Zugriff?	Bis	Von
IAM	(Empfohlen) Verwenden Sie Konsolenanmeldeinformationen als temporäre Anmeldeinformationen, um programmatische Anfragen an AWS CLI AWS SDKs, oder zu signieren. AWS APIs	Befolgen Sie die Anweisungen für die Schnittstelle, die Sie verwenden möchten. • Informationen zum AWS CLI finden Sie unter Anmeldung für AWS lokale Entwicklung im AWS Command Line Interface Benutzerhandbuch. • Weitere Informationen finden Sie unter Anmeldung für AWS lokale Entwicklung im Referenzhandbuch AWS SDKs und im Tools-Referenzhandbuch. AWS SDKs
Mitarbeiteridentität (Benutzer, die in IAM Identity Center verwaltet werden)	Verwenden Sie temporäre Anmeldeinformationen, um programmatische Anfragen an das AWS CLI AWS SDKs, oder AWS APIs zu signieren.	Befolgen Sie die Anweisungen für die Schnittstelle, die Sie verwenden möchten.

Welcher Benutzer benötigt programmgesteuerten Zugriff?	Bis	Von
		• Informationen zu den AWS CLI finden Sie unter Konfiguration der AWS CLI zur Verwendung AWS IAM Identity Center im AWS Command Line Interface Benutzerhandbuch. • Informationen zu AWS SDKs Tools und AWS APIs finden Sie unter IAM Identity Center-Authentifizierung im Referenzhandbuch AWS SDKs und im Tools-Ref erenzhandbuch.
IAM	Verwenden Sie temporäre Anmeldeinformationen, um programmatische Anfragen an das AWS CLI AWS SDKs, oder zu signieren. AWS APIs	Folgen Sie den Anweisungen unter Verwenden temporäre r Anmeldeinformationen mit AWS Ressourcen im IAM-Benutzerhandbuch.

Welcher Benutzer benötigt programmgesteuerten Zugriff?	Bis	Von
IAM	(Nicht empfohlen) Verwenden Sie langfristige Anmeldeinformationen, um programmatische Anfragen an das AWS CLI AWS SDKs, oder zu signieren. AWS APIs	Befolgen Sie die Anweisungen für die Schnittstelle, die Sie verwenden möchten. • Informationen dazu AWS CLI finden Sie unter Authentifizierung mithilfe von IAM-Benutzeranmeldeinformationen im AWS Command Line Interface Benutzerhandbuch. • Informationen zu AWS SDKs und Tools finden Sie unter Authentifizieren mit langfristigen Anmeldeinformationen im Referenzhandbuch AWS SDKs und im Tools-Referenzhandbuch. • Weitere Informationen finden Sie unter Verwaltung von Zugriffsschlüsseln für IAM-Benutzer im IAM-Benutzerhandbuch. AWS APIs

Konfigurieren der Anmeldeinformationen

Bevor Sie programmgesteuert oder über die auf DynamoDB zugreifen können, müssen Sie Ihre Anmeldeinformationen so konfigurieren AWS CLI, dass die Autorisierung für Ihre Anwendungen aktiviert wird.

Dazu stehen verschiedene Möglichkeiten zur Verfügung. Beispielsweise können Sie die Datei mit den Anmeldeinformationen zum Speichern Ihrer Zugriffsschlüssel-ID und des geheimen Zugriffsschlüssels manuell erstellen. Sie können den AWS CLI Befehl auch verwenden, um die

Datei automatisch `aws configure` zu erstellen. Alternativ können Sie Umgebungsvariablen verwenden. Weitere Informationen zur Konfiguration Ihrer Anmeldeinformationen finden Sie im entwicklerspezifischen AWS SDK-Entwicklerhandbuch.

Informationen zur Installation und Konfiguration von finden Sie AWS CLI unter. Mit dem AWS CLI

Integration mit anderen DynamoDB-Services

Sie können DynamoDB in viele andere AWS Dienste integrieren. Weitere Informationen finden Sie hier:

- DynamoDB mit anderen Diensten verwenden AWS
- CloudFormation für DynamoDB
- Verwendung von AWS Backup mit DynamoDB
- AWS Identity and Access Management (IAM) und DynamoDB
- Verwendung AWS Lambda mit Amazon DynamoDB

Lokale Einrichtung von DynamoDB (herunterladbare Version)

Mit der herunterladbaren Version von Amazon DynamoDB können Sie Anwendungen entwickeln und testen, ohne auf den DynamoDB-Webservice zuzugreifen. Die Datenbank wird stattdessen als unabhängige Komponente auf Ihrem Computer ausgeführt. Wenn Sie bereit sind, Ihre Anwendung in der Produktion bereitzustellen, entfernen Sie den lokalen Endpunkt im Code. Er verweist dann auf den DynamoDB-Webservice.

Mit dieser lokalen Version sind Einsparungen im Hinblick auf den Durchsatz, die Datenspeicherung und Datenübertragungsgebühren möglich. Darüber hinaus benötigen Sie für die Entwicklung der Anwendung keine Internetverbindung.

Die lokale Version von DynamoDB ist als Download (erfordert JRE), als Apache-Maven-Abhängigkeit oder als Docker-Image verfügbar.

Wenn Sie stattdessen den Amazon-DynamoDB-Webservice verwenden möchten, finden Sie unter Einrichten von DynamoDB (Webservice) weitere Informationen.

Themen

- Bereitstellen der lokalen Version von DynamoDB auf Ihrem Computer

- DynamoDB-Local-Nutzungshinweise
- Versionsverlauf für die lokale Version von DynamoDB
- Telemetrie in der lokalen Version von DynamoDB

Bereitstellen der lokalen Version von DynamoDB auf Ihrem Computer

> **Note**
>
> - Die lokale Version von DynamoDB ist in drei Versionen verfügbar: v3.x (aktuell), v2.x (Legacy) und v1.x (veraltet).
> - DynamoDB v3.x wird für Ihre lokalen Test- und Entwicklungszwecke empfohlen.
> - Die Migration der lokalen Version von DynamoDB V2.x auf V3.x erfordert die Aktualisierung der Importanweisungen von `com.amazonaws.services.dynamodbv2` auf `software.amazon.dynamodb` und die Aktualisierung der Maven-Abhängigkeiten für Maven-Benutzer.
> - Wenn Sie eine Anwendung zum SDK für Java 2.x migrieren, die das SDK für Java v1.x verwendet, folgen Sie den Schritten unter AWS -SDK für Java 2.x.

Herunterladen der lokalen Version von DynamoDB

Gehen Sie wie folgt vor, um DynamoDB einzurichten und auf Ihrem Computer auszuführen.

So richten Sie DynamoDB auf dem Computer ein

1. Laden Sie die lokale Version von DynamoDB kostenlos von einem der folgenden Speicherorte herunter.

Download-Links	Prüfsummen
.tar.gz | .zip	.tar.gz.sha256 | .zip.sha256

> **Important**
>
> Um DynamoDB v2.6.0 oder höher auf Ihrem Computer ausführen zu können, benötigen Sie die Java Runtime Environment (JRE) Version 17.x oder neuer. Die Anwendung kann nicht unter früheren JRE-Versionen ausgeführt werden.

2. Nachdem Sie das Archiv heruntergeladen haben, extrahieren Sie die Inhalte und kopieren Sie das entpackte Verzeichnis an einen Speicherort Ihrer Wahl.
3. Zum Starten von DynamoDB auf Ihrem Computer, öffnen Sie ein Befehlszeilenfenster, gehen Sie zu dem Verzeichnis, in das Sie `DynamoDBLocal.jar` extrahiert haben, und geben Sie den folgenden Befehl ein.

```
java -Djava.library.path=./DynamoDBLocal_lib -jar DynamoDBLocal.jar -sharedDb
```

> **Note**
>
> Wenn Sie Windows verwenden, achten Sie darauf PowerShell, den Parameternamen oder den gesamten Namen und Wert wie folgt einzuschließen:
> `java -D"java.library.path=./DynamoDBLocal_lib" -jar DynamoDBLocal.jar`
> DynamoDB verarbeitet eingehende Anforderungen, bis Sie den Service beenden. Zum Beenden von DynamoDB geben Sie Strg+C in die Befehlszeile ein.
> DynamoDB verwendet standardmäßig Port 8000. Wenn Port 8000 nicht verfügbar ist, wird eine Ausnahme ausgelöst. Um eine vollständige Liste der DynamoDB-Laufzeitoptionen, einschließlich `-port`, zu erhalten, geben Sie folgenden Befehl ein.
> `java -Djava.library.path=./DynamoDBLocal_lib -jar DynamoDBLocal.jar -help`

4. Bevor Sie programmgesteuert oder über die AWS Command Line Interface (AWS CLI) auf DynamoDB zugreifen können, müssen Sie die Anmeldeinformationen so konfigurieren, dass die Autorisierung für die Anwendungen aktiviert wird. Die herunterladbare Version von DynamoDB benötigt Anmeldeinformationen, um arbeiten zu können, wie in dem folgenden Beispiel gezeigt.

```
AWS Access Key ID: "fakeMyKeyId"
AWS Secret Access Key: "fakeSecretAccessKey"
```

```
Default Region Name: "fakeRegion"
```

Sie können die Anmeldeinformationen mit dem AWS CLI -Befehl `aws configure` einrichten. Weitere Informationen finden Sie unter Mit dem AWS CLI.

5. Beginnen Sie mit dem Schreiben von Anwendungen. Verwenden Sie den Parameter, um auf DynamoDB zuzugreifen AWS CLI, das lokal mit dem ausgeführt wird`--endpoint-url` . Sie können z. B. den folgenden Befehl verwenden, um DynamoDB-Tabellen aufzulisten.

```
aws dynamodb list-tables --endpoint-url http://localhost:8000
```

Ausführen der lokalen Version von DynamoDB als Docker-Image

Die herunterladbare Version von Amazon DynamoDB ist als Docker-Image verfügbar. Weitere Informationen finden Sie unter dynamodb-local. Geben Sie folgenden Befehl ein, um Ihre aktuelle lokale Version von DynamoDB zu sehen:

```
java -Djava.library.path=./DynamoDBLocal_lib -jar DynamoDBLocal.jar -version
```

Ein Beispiel für die Verwendung von DynamoDB local als Teil einer REST-Anwendung, die auf der AWS Serverless Application Model (AWS SAM) aufbaut, finden Sie unter SAM DynamoDB-Anwendung zur Verwaltung von Bestellungen. Diese Beispielanwendung demonstriert die Verwendung der lokalen Version von DynamoDB für Testzwecke.

Wenn Sie eine Anwendung mit mehreren Containern ausführen möchten, die auch den lokalen DynamoDB-Container nutzt, verwenden Sie Docker Compose, um alle Services in Ihrer Anwendung zu definieren und auszuführen, einschließlich DynamoDB lokal.

So gehen Sie vor, um DynamoDB zu installieren und lokal auszuführen:

1. Führen Sie Download und Installation von Docker Desktop durch.
2. Kopieren Sie den folgenden Code in eine Datei, und speichern Sie ihn unter `docker-compose.yml`.

```
services:
 dynamodb-local:
   command: "-jar DynamoDBLocal.jar -sharedDb -dbPath ./data"
   image: "amazon/dynamodb-local:latest"
```

```
    container_name: dynamodb-local
    ports:
      - "8000:8000"
    volumes:
      - "./docker/dynamodb:/home/dynamodblocal/data"
    working_dir: /home/dynamodblocal
```

Wenn sich Ihre Anwendung und DynamoDB lokal in separaten Containern befinden sollen, verwenden Sie die folgende YAML-Datei:

```
version: '3.8'
services:
 dynamodb-local:
   command: "-jar DynamoDBLocal.jar -sharedDb -dbPath ./data"
   image: "amazon/dynamodb-local:latest"
   container_name: dynamodb-local
   ports:
     - "8000:8000"
   volumes:
     - "./docker/dynamodb:/home/dynamodblocal/data"
   working_dir: /home/dynamodblocal
 app-node:
   depends_on:
     - dynamodb-local
   image: amazon/aws-cli
   container_name: app-node
   ports:
    - "8080:8080"
   environment:
     AWS_ACCESS_KEY_ID: 'DUMMYIDEXAMPLE'
     AWS_SECRET_ACCESS_KEY: 'DUMMYEXAMPLEKEY'
   command:
     dynamodb describe-limits --endpoint-url http://dynamodb-local:8000 --region
 us-west-2
```

Dieses docker-compose.yml-Skript erstellt einen `app-node`-Container und einen `dynamodb-local`-Container. Das Skript führt einen Befehl im Container `app-node` aus, der mithilfe der AWS CLI eine Verbindung mit dem `dynamodb-local`-Container herstellt und die Konten- und Tabellenlimits beschreibt.

Wenn Sie diesen Vorgang bei Ihrem eigenen Anwendungsimage verwenden möchten, ersetzen Sie den Wert `image` im folgenden Beispiel durch den Wert Ihrer Anwendung.

```
version: '3.8'
services:
 dynamodb-local:
   command: "-jar DynamoDBLocal.jar -sharedDb -dbPath ./data"
   image: "amazon/dynamodb-local:latest"
   container_name: dynamodb-local
   ports:
     - "8000:8000"
   volumes:
     - "./docker/dynamodb:/home/dynamodblocal/data"
   working_dir: /home/dynamodblocal
 app-node:
   image: location-of-your-dynamodb-demo-app:latest
   container_name: app-node
   ports:
     - "8080:8080"
   depends_on:
     - "dynamodb-local"
   links:
     - "dynamodb-local"
   environment:
     AWS_ACCESS_KEY_ID: 'DUMMYIDEXAMPLE'
     AWS_SECRET_ACCESS_KEY: 'DUMMYEXAMPLEKEY'
     REGION: 'eu-west-1'
```

> **Note**
>
> Die YAML-Skripts erfordern, dass Sie einen AWS Zugriffsschlüssel und einen AWS geheimen Schlüssel angeben, aber es müssen keine gültigen AWS Schlüssel sein, damit Sie lokal auf DynamoDB zugreifen können.

3. Führen Sie die folgende Befehlzeilen-Befehle aus:

```
docker-compose up
```

Ausführen der lokalen Version von DynamoDB als Apache Maven-Abhängigkeit

> **Note**
>
> Wenn Sie eine Anwendung zum SDK für Java 2.x migrieren, die das SDK für Java v1.x verwendet, folgen Sie den Schritten unter AWS -SDK für Java 2.x.

Führen Sie die folgenden Schritte aus, um Amazon DynamoDB in Ihrer Anwendung als Abhängigkeit zu verwenden.

So stellen Sie DynamoDB als Apache-Maven-Repository bereit

1. Laden Sie Apache Maven herunter und installieren Sie es. Weitere Informationen finden Sie unter Downloading Apache Maven und Installing Apache Maven.
2. Fügen Sie das DynamoDB-Maven-Repository zur POM-Datei (Project Object Model) Ihrer Anwendung hinzu.

```
<!--Dependency:-->
<dependencies>
   <dependency>
      <groupId>com.amazonaws</groupId>
      <artifactId>DynamoDBLocal</artifactId>
      <version>3.3.0</version>
   </dependency>
</dependencies>
```

Beispielvorlage zur Verwendung mit Spring Boot 3 and/or Spring Framework 6:

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://
maven.apache.org/xsd/maven-4.0.0.xsd">
<modelVersion>4.0.0</modelVersion>

<groupId>org.example</groupId>
<artifactId>SpringMavenDynamoDB</artifactId>
<version>1.0-SNAPSHOT</version>

<properties>
```

```
    <spring-boot.version>3.0.1</spring-boot.version>
    <maven.compiler.source>17</maven.compiler.source>
    <maven.compiler.target>17</maven.compiler.target>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
</properties>

    <parent>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-parent</artifactId>
        <version>3.1.0</version>
    </parent>

<dependencies>
    <dependency>
        <groupId>com.amazonaws</groupId>
        <artifactId>DynamoDBLocal</artifactId>
        <version>3.3.0</version>
    </dependency>
    <!-- Spring Boot -->
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter</artifactId>
        <version>${spring-boot.version}</version>
    </dependency>
    <!-- Spring Web -->
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-web</artifactId>
        <version>${spring-boot.version}</version>
    </dependency>
    <!-- Spring Data JPA -->
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-data-jpa</artifactId>
        <version>${spring-boot.version}</version>
    </dependency>
    <!-- Other Spring dependencies -->
    <!-- Replace the version numbers with the desired version -->
    <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-context</artifactId>
        <version>6.0.0</version>
    </dependency>
    <dependency>
```

```
        <groupId>org.springframework</groupId>
        <artifactId>spring-core</artifactId>
        <version>6.0.0</version>
    </dependency>
    <!-- Add other Spring dependencies as needed -->
    <!-- Add any other dependencies your project requires -->
</dependencies>
</project>
```

> **Note**
>
> Sie können auch die URL des Maven Central Repository verwenden.

Führen Sie DynamoDB lokal aus in AWS CloudShell

AWS CloudShell ist eine browserbasierte, vorauthentifizierte Shell, die Sie direkt von der aus starten können. AWS-Managementkonsole Sie können auf verschiedene Arten AWS CloudShell zu ihr navigieren. AWS-Managementkonsole Weitere Informationen finden Sie unter Erste Schritte mit AWS CloudShell.

Gehen Sie wie folgt vor, um DynamoDB lokal an Ihrem AWS CloudShell beliebigen Ort in der auszuführen. AWS-Managementkonsole

Um DynamoDB lokal in Ihrem AWS CloudShell auszuführen AWS-Managementkonsole

1. Starten Sie AWS CloudShell von der Konsolenoberfläche aus, wählen Sie eine verfügbare AWS-Region aus und wechseln Sie zu Ihrer bevorzugten Shell, z. B. Bash oder PowerShell Z-Shell.
2. Um eine auszuwählen AWS-Region, gehen Sie zum Menü „Region auswählen“ und wählen Sie eine unterstützte AWS-Region aus. (Verfügbare Regionen sind hervorgehoben.)
3. Starten Sie von der aus AWS-Managementkonsole, AWS CloudShell indem Sie eine der folgenden Optionen wählen:

    a. Klicken Sie in der Navigationsleiste auf das AWS CloudShell-Symbol.

    b. Geben Sie das Wort in das Suchfeld ein CloudShell, und wählen Sie dann CloudShell.

    c. Wählen Sie im Widget „Zuletzt besucht“ CloudShell.

    d. Wählen Sie in der Werkzeugleiste der Konsole CloudShell.

4. Um DynamoDB lokal in auszuführen, können AWS CloudShell Sie den `dynamodb-local` Alias verwenden. Sie können zusätzliche Befehlszeilenoptionen zum Ändern der Einstellungen Ihrer lokalen Version von DynamoDB angeben. Die verfügbaren Optionen finden Sie unter the section called “Nutzungshinweise”.

   > **Note**
   >
   > Um DynamoDB local im Hintergrund auszuführen, führen Sie DynamoDB local mit: aus. AWS CloudShell `dynamodb-local &`

5. Verwenden Sie den Parameter, um auf DynamoDB zuzugreifen AWS CLI, das lokal AWS CloudShell mit dem ausgeführt wird`--endpoint-url`. Sie können z. B. den folgenden Befehl verwenden, um DynamoDB-Tabellen aufzulisten:

   ```
   aws dynamodb list-tables --endpoint-url http://localhost:8000
   ```

Ein Beispielprojekt, das verschiedene Ansätze zur Einrichtung und Nutzung der lokalen Version von DynamoDB zeigt – darunter das Herunterladen von JAR-Dateien, die Ausführung als Docker-Image und die Einbindung als Maven-Abhängigkeit – finden Sie unter Java-Beispielprojekt für die lokale Version von DynamoDB.

DynamoDB-Local-Nutzungshinweise

Mit Ausnahme des Endpunkts sollten Anwendungen, die mit der herunterladbaren Version von Amazon DynamoDB ausgeführt werden, auch mit dem DynamoDB-Webservice funktionieren. Bei der lokalen Verwendung von DynamoDB sollten Sie allerdings Folgendes beachten:

- Wenn Sie die *`-sharedDb`* Option verwenden, erstellt DynamoDB eine einzelne Datenbankdatei mit dem Namen shared-local-instance .db. Jedes Programm, das eine Verbindung mit DynamoDB herstellt, greift auf diese Datei zu. Wenn Sie die Datei löschen, gehen alle darin gespeicherten Daten verloren.
- Wenn Sie diese Option weglassen`-sharedDb`, erhält die Datenbankdatei den Namen myaccesskeyid_region.db mit der AWS Zugriffsschlüssel-ID und AWS Region, wie sie in Ihrer Anwendungskonfiguration erscheinen. Wenn Sie die Datei löschen, gehen alle darin gespeicherten Daten verloren.
- Bei Verwendung der `-inMemory`-Option schreibt DynamoDB keine Datenbankdateien. Stattdessen werden alle Daten im Arbeitsspeicher abgelegt und beim Beenden von DynamoDB nicht gespeichert.

- Wenn Sie die `-inMemory`-Option verwenden, ist die `-sharedDb`-Option ebenfalls erforderlich.
- Bei Verwendung der Option `-optimizeDbBeforeStartup` müssen Sie auch den `-dbPath`-Parameter angeben, damit DynamoDB die entsprechende Datenbankdatei finden kann.
- Die AWS SDKs für DynamoDB erfordern, dass Ihre Anwendungskonfiguration einen Zugriffsschlüsselwert und einen AWS Regionswert angibt. DynamoDB verwendet diese Werte, um die lokale Datenbankdatei zu benennen, sofern Sie nicht die Option `-sharedDb` oder `-inMemory` verwenden. Diese Werte müssen keine gültigen AWS Werte sein, um lokal ausgeführt zu werden. Möglicherweise ist es jedoch komfortabler, gültige Werte zu verwenden, damit Sie Ihren Code später in der Cloud ausführen können, indem Sie den verwendeten Endpunkt ändern.
- Die lokale Version von DynamoDB gibt für `billingModeSummary.` immer null zurück
- Die lokale Version von DynamoDB `AWS_ACCESS_KEY_ID` kann nur Buchstaben (A–Z, a–z) und Zahlen (0–9) enthalten.
- DynamoDB local unterstützt keine Point-in-time Wiederherstellung (PITR).

Themen

- Befehlszeilenoptionen
- Festlegen des lokalen Endpunkts
- Unterschiede zwischen der herunterladbaren Version von DynamoDB und dem DynamoDB-Webservice

Befehlszeilenoptionen

Mit der herunterladbaren Version von DynamoDB können Sie die folgenden Befehlszeilenoptionen verwenden:

- `-corsvalue`— Aktiviert die Unterstützung für Cross-Origin Resource Sharing (CORS) für. JavaScript Sie müssen eine durch Komma getrennte Liste zum Zulassen spezifischer Domänen bereitstellen. Die Standardeinstellung für `-cors` ist ein Stern (*), der öffentlichen Zugriff zulässt.
- `-dbPath value` – Das Verzeichnis, in das DynamoDB seine Datenbankdatei schreibt. Wenn Sie diese Option nicht angeben, wird die Datei in das aktuelle Verzeichnis geschrieben. Sie können nicht `-dbPath` und `-inMemory` gleichzeitig angeben.
- `-delayTransientStatuses` – Veranlasst DynamoDB, Verzögerungen für bestimmte Operationen einzuführen. DynamoDB (herunterladbare Version) kann einige Aufgaben fast sofort ausführen, z. B. create/update/delete Operationen an Tabellen und Indizes. Der Service

DynamoDB benötigt für diese Aufgaben jedoch mehr Zeit. Durch Festlegen dieses Parameters kann DynamoDB bei Ausführung auf einem Computer das Verhalten des DynamoDB-Webservices besser simulieren. (Gegenwärtig führt dieser Parameter Verzögerungen nur für globale sekundäre Indizes mit dem Status CREATING oder DELETING ein.)

- `-help` – Druckt eine Nutzungszusammenfassung sowie Optionen.
- `-inMemory` – DynamoDB wird im Speicher anstatt mit einer Datenbankdatei ausgeführt. Wenn Sie DynamoDB beenden, werden keine Daten gespeichert. Sie können nicht `-dbPath` und `-inMemory` gleichzeitig angeben.
- `-optimizeDbBeforeStartup` – Optimiert die zugrunde liegenden Datenbanktabellen vor dem Start von DynamoDB auf Ihrem Computer. Wenn Sie diesen Parameter verwenden, müssen Sie außerdem `-dbPath` angeben.
- `-port value` – Die Portnummer, die DynamoDB für die Kommunikation mit Ihrer Anwendung verwendet. Wenn Sie diese Option nicht angeben, lautet der Standardport `8000`.

> **Note**
>
> DynamoDB verwendet standardmäßig Port 8000. Wenn Port 8000 nicht verfügbar ist, wird eine Ausnahme ausgelöst. Mit der Option `-port` können Sie eine andere Portnummer angeben. Um eine vollständige Liste der DynamoDB-Laufzeitoptionen, einschließlich `-port`, zu erhalten, geben Sie folgenden Befehl ein:
> `java -Djava.library.path=./DynamoDBLocal_lib -jar DynamoDBLocal.jar -help`

- `-sharedDb` – `-sharedDb` DynamoDB verwendet eine einzige Datenbankdatei anstatt separater Dateien für jede Anmeldeinformation und Region.
- `-disableTelemetry`: Wenn dies angegeben ist, sendet die lokale Version von DynamoDB keine Telemetrie.
- `-version` – Zeigt die Version der lokalen Version von DynamoDB.

Festlegen des lokalen Endpunkts

Standardmäßig verwenden die Tools AWS SDKs und Endpunkte für den Amazon DynamoDB DynamoDB-Webservice. Um die Tools SDKs und mit der herunterladbaren Version von DynamoDB zu verwenden, müssen Sie den lokalen Endpunkt angeben:

```
http://localhost:8000
```

AWS Command Line Interface

Sie können die AWS Command Line Interface (AWS CLI) verwenden, um mit herunterladbarem DynamoDB zu interagieren.

Verwenden Sie den `--endpoint-url`-Parameter , um auf die lokal ausgeführte DynamoDB-Version zuzugreifen. Im Folgenden finden Sie ein Beispiel für die Verwendung von AWS CLI , um die Tabellen in DynamoDB auf Ihrem Computer aufzulisten.

```
aws dynamodb list-tables --endpoint-url http://localhost:8000
```

> **Note**
>
> Die herunterladbare Version von DynamoDB AWS CLI kann nicht als Standardendpunkt verwendet werden. Daher müssen Sie `--endpoint-url` bei jedem AWS CLI Befehl angeben.

AWS SDKs

Die Art und Weise, wie Sie einen Endpunkt angeben, hängt davon ab, welche Programmiersprache und welches AWS -SDK Sie verwenden. In den folgenden Abschnitten wird die entsprechende Vorgehensweise beschrieben:

- Java: AWS Region und Endpunkt festlegen(DynamoDB local unterstützt das AWS SDK for Java V1 und V2)
- CodeSamples.Java. RegionAndEndpoint .NET: AWS Region und Endpunkt festlegen

Unterschiede zwischen der herunterladbaren Version von DynamoDB und dem DynamoDB-Webservice

Die herunterladbare Version von DynamoDB ist nur für Zwecke der Anwendungsentwicklung und Tests vorgesehen. Der DynamoDB-Webservice hingegen ist ein verwalteter Service mit Skalierbarkeits-, Verfügbarkeits- und Haltbarkeitsfunktionen, die ideal für den Einsatz in der Produktion geeignet sind.

Die herunterladbare Version von DynamoDB unterscheidet sich folgendermaßen vom Webservice:

- AWS-Regionen und distinct AWS-Konten werden auf Client-Ebene nicht unterstützt.

- Einstellungen für den bereitgestellten Durchsatz werden in der herunterladbaren Version von DynamoDB ignoriert, auch wenn die `CreateTable`-Operation diese benötigt. Für `CreateTable` können Sie beliebige Zahlen für den bereitgestellten Lese- und Schreibdurchsatz angeben, auch wenn diese Zahlen nicht verwendet werden. Sie können `UpdateTable` beliebig oft pro Tag aufrufen. Alle Änderungen an Werten des bereitgestellten Durchsatzes werden jedoch ignoriert.
- `Scan`-Operationen werden sequenziell durchgeführt. Parallele Scans werden nicht unterstützt. Die Parameter `Segment` und `TotalSegments` der `Scan`-Operation werden ignoriert.
- Die Geschwindigkeit der Lese- und Schreibvorgänge in den Tabellendaten ist nur durch die Geschwindigkeit des Computers begrenzt. Die Vorgänge `CreateTable`, `UpdateTable` und `DeleteTable` werden umgehend ausgeführt und der Tabellenstatus ist jederzeit ACTIVE. `UpdateTable`-Vorgänge, die nur die Einstellungen zum bereitgestellten Durchsatz für Tabellen oder globale sekundäre Indizes ändern, werden umgehend ausgeführt. Wenn eine `UpdateTable`-Operation globale sekundäre Indizes erstellt oder löscht, durchlaufen diese Indizes die normalen Status (wie CREATING bzw. DELETING), bevor sie in den Status ACTIVE übergehen. Die Tabelle bleibt während dieser Zeit im Status ACTIVE.
- Leseoperationen sind Eventually Consistent. Aufgrund der Geschwindigkeit, mit der die lokale Version von DynamoDB auf Ihrem Computer ausgeführt wird, sind die meisten Lesevorgänge allerdings tatsächlich vom Typ „Strongly Consistent”.
- Elementauflistungsmetriken und Elementauflistungsgrößen werden nicht nachverfolgt. In Operationsantworten werden anstelle der Elementauflistungsmetriken Nullen zurückgegeben.
- In DynamoDB gibt es für Daten, die pro Ergebnissatz zurückgegeben werden, ein Limit von 1 MB. Sowohl der DynamoDB-Webservice als auch die herunterladbare Version setzen dieses Limit durch. Beim Abfragen eines Indexes berechnet der DynamoDB-Service jedoch nur die Größe des projizierten Schlüssels und der Attribute. Im Gegensatz dazu berechnet die herunterladbare Version von DynamoDB die Größe des gesamten Elements.
- Bei der Verwendung von DynamoDB Streams kann die Rate, mit der Shards erstellt werden, abweichen. Im DynamoDB-Webservice wird die Shard-Erstellung teilweise von Aktivitäten der Tabellenpartition beeinflusst. Wenn Sie DynamoDB lokal ausführen, gibt es keine Tabellenpartitionierung. In beiden Fällen sind Shards flüchtig, sodass Ihre Anwendung nicht vom Shard-Verhalten abhängt.
- `TransactionConflictExceptions`werden nicht von herunterladbarem DynamoDB für Transaktionen ausgelöst. APIs Wir empfehlen die Verwendung eines Java-Mocking-Frameworks für die Simulation von `TransactionConflictExceptions` im DynamoDB-Handler, um zu prüfen, wie Ihre Anwendung auf miteinander in Konflikt stehende Transaktionen reagiert.

- Im DynamoDB-Webdienst wird zwischen Groß- und Kleinschreibung unterschieden, unabhängig davon, ob auf Tabellennamen über die AWS CLI Konsole oder über zugegriffen wird. Eine Tabelle mit dem Namen `Authors` und eine mit dem Namen `authors` können als separate Tabellen vorhanden sein. In der Downloadversion muss bei Tabellennamen die Groß-/Kleinschreibung beachtet werden. Ein Versuch, diese beiden Tabellen zu erstellen, würde einen Fehler verursachen.
- Tagging wird in der herunterladbaren Version von DynamoDB nicht unterstützt.
- Die herunterladbare Version von DynamoDB ignoriert den Limit-Parameter in. ExecuteStatement

Versionsverlauf für die lokale Version von DynamoDB

In der folgenden Tabelle sind wichtige Änderungen in jeder Version von der lokalen Version von DynamoDB beschrieben.

Version	Änderungen	Beschreibung	Date
3.3.0	Unterstützung für Schlüssel mit mehreren Attributen für globale Sekundärindizes hinzufügen	• Unterstützung für Schlüssel mit mehreren Attributen für globale Sekundärindizes hinzugefügt • Aktualisierung der SDK-Java-Version von 2.41.0 auf 2.41.7 auf die neueste öffentliche Version	19. Januar 2026
3.2.0	Kompatibilitätsprobleme mit mehreren Kotlin-Versionen wurden behoben	• Support für ShardFilter Param DescribeStream in der API hinzufügen • Aktualisierung der SDK-Java-Version auf die neueste	09. Januar 2026

Version	Änderungen	Beschreibung	Date
		öffentliche Version von 2.33.0 auf 2.41.0 • Fehler behoben bei DeletionProtection UpdateTable	
3.1.0	Verbesserung der Leistung von PartiQL-Abfragen, Einbeziehung der Joda-Zeitabhängigkeit	• Aktualisierung der SDK-Java-Version von 2.25.50 auf 2.33.0 auf die neueste öffentliche Version • Einbeziehung der Joda-Zeitabhängigkeit in die Datei „Pom.xml“ • Verbesserung der Leistung für PartiQL-Abfragen • Aktualisierung von Abhängigkeiten zur Behebung mehrerer CVE-Schwachstellen	14. September 2025

Version	Änderungen	Beschreibung	Date
3.0.0	Migration von AWS SDK Java V1 zu V2	• Migration von AWS SDK Java V1 zu V2 • Aktualisierung der Paketstruktur von `com.amazonaws.services.dynamodbv2` auf `software.amazon.dynamodb.services` aktualisiert • Die Abhängigkeiten von AWS SDK Java V1 wurden entfernt	17. Juli 2025

Version	Änderungen	Beschreibung	Date
2.6.0	Support Tabellen-ARN als Tabellennamen in DynamoDB APIs Leistungsverbesserungen und Sicherheitsupdates	• Unterstützung für die Verwendung von Tabellen-ARN als Tabellennamen in mehreren DynamoDB hinzugefügt APIs • Behebung des Fehlers `CreateStreamTable` auf Hochleistungsrechnern wie Mac M3 • Aktualisierung von Abhängigkeiten zur Behebung von Schwachstellen (CVE-2022-49043, CVE-2024-56732, CVE-2020-29582, CVE-2025-21502, CVE-2024-50602, CVE-2025-24970, CVE-2025-25193)	13. März 2025

Version	Änderungen	Beschreibung	Date
2.5.4	Aktualisierung von Jetty-Abhängigkeiten	• Aktualisierung von Jetty 12.0.8 auf Jetty 12.0.14 (Behebung von CVE-2024-6763, CVE-2024-8184 und CVE-2024-47535) Beseitigung von (CVE-2024-21634)	12. Dezember 2024
2.5.3	Aktualisierung von Jackson-Abhängigkeiten auf 2.17.x in Log4j Core (Behebung der Schwachstelle CVE-2022-1471)	• Aktualisierung der Jackson-Abhängigkeiten auf 2.17.x in Log4j Core (Behebung der Schwachstelle CVE-2022-1471), um eine kritische Schwachstelle in der SnakeYAML-Bibliothek zu schließen, bei der es sich um eine transitive Abhängigkeit handelt	6. November 2024

Version	Änderungen	Beschreibung	Date
2.5.2	Bugfix für den Workflow „Tabelle aktualisieren“	• Behebung des folgenden Fehlers: Der Workflow „Tabelle aktualisieren“ versucht, den Fakturierungsmodus einer Tabelle im „On-Demand“ mit GSI in den Modus „Bereitgestellte Kapazität“ zu ändern	20. Juni 2024
2.5.1	Patch für mit der `OndemandThroughPut` - Feature eingeführte Fehler	• Es wurden einige Fehler im Zusammenhang mit `OndemandThroughPut` behoben	5. Juni 2024
2.5.0	Unterstützung des konfigurierbaren maximalen Durchsatzes für On-Demand-Tabellen, `ReturnValuesOnConditionCheckFailure` , `BatchExecuteStatement` und `ExecuteTransaction Request`	• Erweiterung des eingebetteten Modus um Telemetrie • Korrektur der SDKv2 Übersetzung für `ConditionalCheckException`	28. Mai 2024

Version	Änderungen	Beschreibung	Date
2.4.0	Unterstützung von `ReturnValuesOnConditionCheckFailure` – Embedded-Modus	• Korrektur eines Fehlers mit dem eingebetteten Modus für `TrimmedDataAccessException` beim Betrieb mit mehreren Streams • Korrektur der Ausnahmeübersetzung für SDKv2 im eingebetteten Modus	17. April 2024
2.3.0	Aktualisierung von Jetty und JDK	• Aktualisierung Jetty 12.0.2 • Aktualisierung auf JDK 17 • Aktualisierung ANTLR4 auf 4.10.1	14. März 2024
2.2.0	Löschschutz für Tabellen und Parameter `ReturnValuesOnConditionCheckFailure` hinzugefügt	• Löschschutz für Tabellen hinzugefügt • Unterstützung hinzugefügt für ReturnValuesOnConditionCheckFailure • Unterstützung für das „-version “-Kennzeichen hinzugefügt	14. Dezember 2023

Version	Änderungen	Beschreibung	Date
2.1.0	Support für SQLLite native Bibliotheken für Maven-Projekte und Hinzufügen von Telemetrie	• Hinzufügen von Telemetrie zur lokalen Version von DynamoDB • Kopieren Sie dynamisch SQLLite native Bibliotheken für Maven-Projekte • Bibliothek io.github .ganadist.sqlite4j ava wurde aus der Maven-Abh ängigkeit entfernt • Aktualisierung auf GoogleGuava 32.1.1-jre	23. Oktober 2023
2.0.0	Migration von Javax zu Jakarta Namespace und Support JDK11	• Migration von Javax zum Jakarta-N amespace und Support JDK11 • Fehlerbehebung für den Umgang mit ungültige n Zugriffs- und Geheimschlüsseln beim Serverstart • Behebung der von Maven identifiz ierten Schwachst ellen durch Aktualisierung von Abhängigkeiten	5. Juli 2023

Version	Änderungen	Beschreibung	Date
1.25.1	Aktualisierung von Jackson-A bhängigkeiten auf 2.17.x in Log4j Core (Behebung der Schwachstelle CVE-2022-1471)	Aktualisierung der Jackson-A bhängigkeiten auf 2.17.x in Log4j Core (Behebung der Schwachstelle CVE-2022-1471), um eine kritische Schwachstelle in der SnakeYAML-Biblioth ek zu schließen, bei der es sich um eine transitive Abhängigk eit handelt	6. November 2024
1.25.0	Löschschutz für Tabellen und Parameter `ReturnVal uesOnCond itionChec kFailure` hinzugefügt	• Löschschutz für Tabellen hinzugefü gt • Unterstützung hinzugefügt für ReturnVal uesOnCond itionCheckFailure • Unterstützung für das „-version “-Kennzeichen hinzugefügt	18. Dezember 2023

Version	Änderungen	Beschreibung	Date
1.24.0	Support für SQLLite native Bibliotheken für Maven-Projekte und Hinzufügen von Telemetrie	• Hinzufügen von Telemetrie zur lokalen Version von DynamoDB • Kopieren Sie dynamisch SQLLite native Bibliotheken für Maven-Projekte • Bibliothek io.github .ganadist.sqlite4j ava wurde aus der Maven-Abh ängigkeit entfernt • Aktualisierung auf GoogleGuava 32.1.1-jre	23. Oktober 2023
1.23.0	Umgang mit ungültige n Zugriffs- und Geheimschlüsseln beim Serverstart	• Fehlerbehebung für den Umgang mit ungültige n Zugriffs- und Geheimschlüsseln beim Serverstart • Behebung der von Maven identifiz ierten Schwachst ellen durch Aktualisierung von Abhängigkeiten	28. Juni 2023

Version	Änderungen	Beschreibung	Date
1.22.0	Unterstützung von Limit-Operation für PartiQL	• Optimierung der IN-Klausel für PartiQL • Unterstützung für Limit-Operation • M1-Unterstützung für Maven-Projekte	08. Juni 2023
1.21.0	Unterstützung für 100 Aktionen pro Transaktion	• Erhöhung der Aktionen pro Transaktion von 25 auf 100 • Aktualisierung des Docker-Image-Open-JDK auf 11 • Korrektur der Parität für Ausnahmen, die ausgelöst werden, wenn doppelte Elemente in BatchExec uteStatement	26. Januar 2023
1.20.0	Unterstützung für M1 Mac hinzugefügt	• Unterstützung für M1 Mac hinzugefü gt • Aktualisierung der Jetty-Abhängigkeit auf 9.4.48.v2 0220622	12. September 2022

Version	Änderungen	Beschreibung	Date
1.19.0	PartiQL-Parser aktualisiert	PartiQL-Parser und andere verwandte Bibliotheken aktualisiert	27. Juli 2022
1.18.0	log4j-Core und Jackson-Core aktualisiert	log4j-Core auf 2.17.1 und Jackson-Core 2.10.x auf 2.12.0 aktualisiert	10. Januar 2022
1.17.2	log4j-Core aktualisiert	log4j-Core-Abhängigkeit auf Version 2.16 aktualisiert	16. Januar 2021
1.17.1	log4j-Core aktualisiert	log4j-core-Abhängigkeit aktualisiert, um den Zero-Day-Exploit zu patchen und die Ausführung von Code aus der Ferne zu verhindern – Log4Shel	10. Januar 2021
1.17.0	Veraltete Javascript-Web-Shell	• Die AWS SDK-Abhängigkeit wurde auf AWS SDK for Java 1.12.x aktualisiert • Veraltete Javascript-Web-Shell	08. Januar 2021

Telemetrie in der lokalen Version von DynamoDB

Bei AWS entwickeln und lancieren wir Dienstleistungen auf der Grundlage dessen, was wir aus Interaktionen mit Kunden lernen, und wir nutzen Kundenfeedback, um unsere Produkte zu

verbessern. Bei Telemetrie handelt es sich um zusätzliche Informationen, die uns dabei helfen, die Anforderungen unserer Kunden besser zu verstehen, Probleme zu diagnostizieren und Features bereitzustellen, mit denen das Erlebnis unserer Kunden verbessert wird.

Die lokale Version von DynamoDB erfasst Telemetriedaten, z. B. allgemeine Nutzungsmetriken, System- und Umgebungsinformationen sowie Fehler. Einzelheiten zu der Art der erfassten Telemetrie finden Sie unter Arten von erfassten Informationen.

Die lokale Version von DynamoDB erfasst keine personenbezogenen Daten wie Benutzernamen oder E-Mail-Adressen. Außerdem werden keine sensiblen Informationen auf Projektebene extrahiert.

Als Kunde haben Sie die Kontrolle darüber, ob die Telemetrie aktiviert ist, und Sie können Ihre Einstellungen jederzeit ändern. Wenn die Telemetrie aktiviert bleibt, sendet die lokale Version von DynamoDB Telemetriedaten im Hintergrund, ohne dass eine zusätzliche Kundeninteraktion erforderlich ist.

Telemetrie mithilfe von Befehlszeilenoptionen ausschalten

Sie können Telemetrie beim Start der lokalen Version von DynamoDB mit der Option `-disableTelemetry` mithilfe von Befehlszeilenoptionen deaktivieren. Weitere Informationen finden Sie unter Befehlszeilenoptionen.

Telemetrie für eine einzelne Sitzung ausschalten

In macOS- und Linux-Betriebssystemen können Sie die Telemetrie für eine einzelne Sitzung ausschalten. Um die Telemetrie für Ihre aktuelle Sitzung zu deaktivieren, führen Sie den folgenden Befehl aus, um die Umgebungsvariable `DDB_LOCAL_TELEMETRY` auf `false` festzulegen. Wiederholen Sie den Befehl für jedes neue Terminal oder jede neue Sitzung.

```
export DDB_LOCAL_TELEMETRY=0
```

Telemetrie für Ihr Profil in allen Sitzungen ausschalten

Führen Sie die folgenden Befehle aus, um die Telemetrie für alle Sitzungen zu deaktivieren, wenn Sie die lokale Version von DynamoDB auf Ihrem Betriebssystem ausführen.

Aktivieren von Telemetrie unter Linux

1. Führen Sie Folgendes aus:

   ```
   echo "export DDB_LOCAL_TELEMETRY=0" >>~/.profile
   ```

2. Führen Sie Folgendes aus:

```
source ~/.profile
```

Aktivieren von Telemetrie unter macOS

1. Führen Sie Folgendes aus:

```
echo "export DDB_LOCAL_TELEMETRY=0" >>~/.profile
```

2. Führen Sie Folgendes aus:

```
source ~/.profile
```

Aktivieren von Telemetrie unter Windows

1. Führen Sie Folgendes aus:

```
setx DDB_LOCAL_TELEMETRY 0
```

2. Führen Sie Folgendes aus:

```
refreshenv
```

Deaktivieren der Telemetrie mithilfe der lokalen eingebetteten Version von DynamoDB in Maven-Projekten

Sie können die Telemetrie mithilfe der lokalen eingebetteten Version von DynamoDB in Maven-Projekten deaktivieren.

```
boolean disableTelemetry = true;
// AWS SDK v1
 AmazonDynamoDB amazonDynamoDB =
 DynamoDBEmbedded.create(disableTelemetry).amazonDynamoDB();

// AWS SDK v2
DynamoDbClient ddbClientSDKv2Local =
 DynamoDBEmbedded.create(disableTelemetry).dynamoDbClient();
```

Arten von erfassten Informationen

- Nutzungsinformationen — Die generische Telemetrie wie Server start/stop und die API oder Operation werden aufgerufen.
- System- und Umgebungsinformationen: Die Java-Version, das Betriebssystem (Windows, Linux oder macOS), die Umgebung, in der die lokale Version von DynamoDB ausgeführt wird (z. B. eigenständiges JAR, Docker-Container oder als Maven-Abhängigkeit), und Hashwerte von Nutzungsattributen.

Weitere Informationen

Die Telemetriedaten, die DynamoDB local sammelt, entsprechen den AWS Datenschutzrichtlinien. Weitere Informationen finden Sie hier:

- AWS Nutzungsbedingungen
- Häufig gestellte Fragen zum Datenschutz

Schritt 1: Erstellen einer Tabelle in DynamoDB

In diesem Schritt erstellen Sie eine `Music`-Tabelle im Amazon DynamoDB. In dieser Tabelle befinden sich die folgenden Details:

- Partitionsschlüssel: `Artist`
- Sortierschlüssel: `SongTitle`

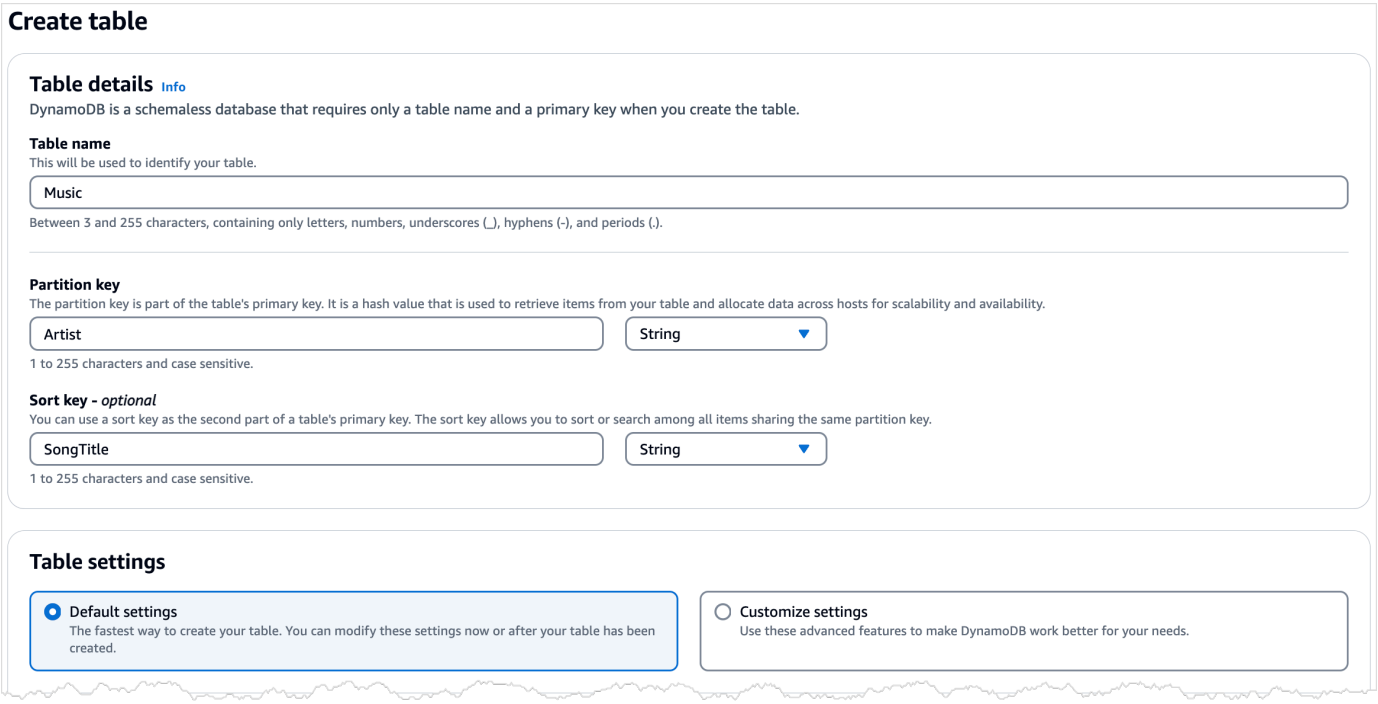
Weitere Informationen über Tabellen-Operationen finden Sie unter Arbeiten mit Tabellen und Daten in DynamoDB.

> **Note**
>
> Bevor Sie beginnen, sollten Sie sicherstellen, dass Sie die in Voraussetzungen beschriebenen Schritte befolgt haben.

AWS-Managementkonsole

So erstellen Sie eine neue `Music`-Tabelle mithilfe der DynamoDB-Konsole:

1. Melden Sie sich bei der an AWS-Managementkonsole und öffnen Sie die DynamoDB-Konsole unter. https://console.aws.amazon.com/dynamodb/
2. Wählen Sie im linken Navigationsbereich Tables (Tabellen) aus.
3. Wählen Sie Create table (Tabelle erstellen) aus.
4. Geben Sie die Tabellendetails wie folgt ein:
   a. Geben Sie unter Tabellenname **`Music`** ein.
   b. Geben Sie unter Partitionsschlüssel den Wert **`Artist`** ein.
   c. Geben Sie unter Sortierschlüssel den Wert **`SongTitle`** ein.
5. Behalten Sie unter Tabelleneinstellungen die Standardauswahl Standardeinstellungen bei.
6. Klicken Sie auf Tabelle erstellen, um die Tabelle zu erstellen.



7. Sobald die Tabelle den Status `ACTIVE` aufweist, empfehlen wir, für die Tabelle die Option Zeitpunktbezogene Backups für DynamoDB zu aktivieren, indem Sie die folgenden Schritte ausführen:
   a. Klicken Sie auf den Tabellennamen, um die Tabelle zu öffnen.
   b. Wählen Sie Backups aus.
   c. Wählen Sie im Bereich Point-in-time Wiederherstellung (PITR) die Option Bearbeiten aus.
   d. Wählen Sie auf der Seite „ point-in-timeWiederherstellungseinstellungen bearbeiten“ die Option point-in-timeWiederherstellung aktivieren aus.

e. Wählen Sie Änderungen speichern aus.

AWS CLI

Im folgenden AWS CLI Beispiel wird eine neue `Music` Tabelle mit erstellt`create-table`.

Linux

```
aws dynamodb create-table \
    --table-name Music \
    --attribute-definitions \
        AttributeName=Artist,AttributeType=S \
        AttributeName=SongTitle,AttributeType=S \
    --key-schema AttributeName=Artist,KeyType=HASH
 AttributeName=SongTitle,KeyType=RANGE \
    --billing-mode PAY_PER_REQUEST \
    --table-class STANDARD
```

Windows CMD

```
aws dynamodb create-table ^
    --table-name Music ^
    --attribute-definitions ^
        AttributeName=Artist,AttributeType=S ^
        AttributeName=SongTitle,AttributeType=S ^
    --key-schema AttributeName=Artist,KeyType=HASH
 AttributeName=SongTitle,KeyType=RANGE ^
    --billing-mode PAY_PER_REQUEST ^
    --table-class STANDARD
```

Bei Verwenden von `create-table` wird das folgende Beispielergebnis zurückgegeben.

```
{
    "TableDescription": {
        "AttributeDefinitions": [
            {
                "AttributeName": "Artist",
                "AttributeType": "S"
            },
            {
                "AttributeName": "SongTitle",
```

```
                "AttributeType": "S"
            }
        ],
        "TableName": "Music",
        "KeySchema": [
            {
                "AttributeName": "Artist",
                "KeyType": "HASH"
            },
            {
                "AttributeName": "SongTitle",
                "KeyType": "RANGE"
            }
        ],
        "TableStatus": "CREATING",
        "CreationDateTime": "2023-03-29T12:11:43.379000-04:00",
        "ProvisionedThroughput": {
            "NumberOfDecreasesToday": 0,
            "ReadCapacityUnits": 5,
            "WriteCapacityUnits": 5
        },
        "TableSizeBytes": 0,
        "ItemCount": 0,
        "TableArn": "arn:aws:dynamodb:us-east-1:111122223333:table/Music",
        "TableId": "60abf404-1839-4917-a89b-a8b0ab2a1b87",
        "TableClassSummary": {
            "TableClass": "STANDARD"
        }
    }
}
}
```

Beachten Sie, dass für das Feld `TableStatus` der Wert `CREATING` festgelegt ist.

Um zu überprüfen, dass DynamoDB die Erstellung der `Music`-Tabelle abgeschlossen hat, verwenden Sie den Befehl `describe-table`.

Linux

```
aws dynamodb describe-table --table-name Music | grep TableStatus
```

Windows CMD

```
aws dynamodb describe-table --table-name Music | findstr TableStatus
```

Dieser Befehl gibt das folgende Ergebnis zurück. Wenn DynamoDB die Erstellung der Tabelle abgeschlossen hat, wird der Wert `TableStatus` des Felds auf `ACTIVE` festgelegt.

```
"TableStatus": "ACTIVE",
```

Sobald die Tabelle den Status `ACTIVE` aufweist, sollten Sie Zeitpunktbezogene Backups für DynamoDB für die Tabelle aktivieren, indem Sie den folgenden Befehl ausführen:

Linux

```
aws dynamodb update-continuous-backups \
    --table-name Music \
    --point-in-time-recovery-specification \
        PointInTimeRecoveryEnabled=true
```

Windows CMD

```
aws dynamodb update-continuous-backups --table-name Music --point-in-time-recovery-
specification PointInTimeRecoveryEnabled=true
```

Dieser Befehl gibt das folgende Ergebnis zurück.

```
{
    "ContinuousBackupsDescription": {
        "ContinuousBackupsStatus": "ENABLED",
        "PointInTimeRecoveryDescription": {
            "PointInTimeRecoveryStatus": "ENABLED",
            "EarliestRestorableDateTime": "2023-03-29T12:18:19-04:00",
            "LatestRestorableDateTime": "2023-03-29T12:18:19-04:00"
        }
    }
}
```

> **Note**
>
> Die Aktivierung kontinuierlicher Backups mit point-in-time Wiederherstellung hat Auswirkungen auf die Kosten. Weitere Informationen zu Preisen finden Sie unter Amazon DynamoDB – Preise.

AWS SDK

In den folgenden Codebeispielen wird gezeigt, wie eine DynamoDB-Tabelle mit einem AWS -SDK erstellt wird.

.NET

SDK für .NET (v4)

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
    /// <summary>
    /// Creates a new Amazon DynamoDB table and then waits for the new
    /// table to become active.
    /// </summary>
    /// <param name="tableName">The name of the table to create.</param>
    /// <returns>A Boolean value indicating the success of the operation.</
returns>
    public async Task<bool> CreateMovieTableAsync(string tableName)
    {
        try
        {
            var response = await _amazonDynamoDB.CreateTableAsync(new
 CreateTableRequest
            {
                TableName = tableName,
                AttributeDefinitions = new List<AttributeDefinition>()
                {
                    new AttributeDefinition
```

```
                    {
                        AttributeName = "title",
                        AttributeType = ScalarAttributeType.S,
                    },
                    new AttributeDefinition
                    {
                        AttributeName = "year",
                        AttributeType = ScalarAttributeType.N,
                    },
                },
                KeySchema = new List<KeySchemaElement>()
                {
                    new KeySchemaElement
                    {
                        AttributeName = "year",
                        KeyType = KeyType.HASH,
                    },
                    new KeySchemaElement
                    {
                        AttributeName = "title",
                        KeyType = KeyType.RANGE,
                    },
                },
                BillingMode = BillingMode.PAY_PER_REQUEST,
            });

            // Wait until the table is ACTIVE and then report success.
            Console.Write("Waiting for table to become active...");

            var request = new DescribeTableRequest
            {
                TableName = response.TableDescription.TableName,
            };

            TableStatus status;

            int sleepDuration = 2000;

            do
            {
                Thread.Sleep(sleepDuration);

                var describeTableResponse = await
_amazonDynamoDB.DescribeTableAsync(request);
```

```
                status = describeTableResponse.Table.TableStatus;

                Console.Write(".");
            }
            while (status != "ACTIVE");

            return status == TableStatus.ACTIVE;
        }
        catch (ResourceInUseException ex)
        {
            Console.WriteLine($"Table {tableName} already exists. {ex.Message}");
            throw;
        }
        catch (AmazonDynamoDBException ex)
        {
            Console.WriteLine($"An Amazon DynamoDB error occurred while creating
table {tableName}. {ex.Message}");
            throw;
        }
        catch (Exception ex)
        {
            Console.WriteLine($"An error occurred while creating table
{tableName}. {ex.Message}");
            throw;
        }
    }
```

- Einzelheiten zur API finden Sie CreateTablein der AWS SDK für .NET API-Referenz.

Bash

AWS CLI mit Bash-Skript

> **Note**
>
> Es gibt noch mehr dazu. GitHub Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
##############################################################################
```

```
# function dynamodb_create_table
#
# This function creates an Amazon DynamoDB table.
#
# Parameters:
#       -n table_name  -- The name of the table to create.
#       -a attribute_definitions -- JSON file path of a list of attributes and
 their types.
#       -k key_schema -- JSON file path of a list of attributes and their key
 types.
#
#  Returns:
#       0 - If successful.
#       1 - If it fails.
###############################################################################
function dynamodb_create_table() {
  local table_name attribute_definitions key_schema response
  local option OPTARG # Required to use getopts command in a function.

  ######################################
  # Function usage explanation
  ######################################
  function usage() {
    echo "function dynamodb_create_table"
    echo "Creates an Amazon DynamoDB table with on-demand billing."
    echo " -n table_name  -- The name of the table to create."
    echo " -a attribute_definitions -- JSON file path of a list of attributes and
 their types."
    echo " -k key_schema -- JSON file path of a list of attributes and their key
 types."
    echo ""
  }

  # Retrieve the calling parameters.
  while getopts "n:a:k:h" option; do
    case "${option}" in
      n) table_name="${OPTARG}" ;;
      a) attribute_definitions="${OPTARG}" ;;
      k) key_schema="${OPTARG}" ;;
      h)
        usage
        return 0
        ;;
      \?)
```

```
        echo "Invalid parameter"
        usage
        return 1
        ;;
    esac
  done
  export OPTIND=1

  if [[ -z "$table_name" ]]; then
    errecho "ERROR: You must provide a table name with the -n parameter."
    usage
    return 1
  fi

  if [[ -z "$attribute_definitions" ]]; then
    errecho "ERROR: You must provide an attribute definitions json file path the
-a parameter."
    usage
    return 1
  fi

  if [[ -z "$key_schema" ]]; then
    errecho "ERROR: You must provide a key schema json file path the -k
parameter."
    usage
    return 1
  fi

  iecho "Parameters:\n"
  iecho "    table_name:   $table_name"
  iecho "    attribute_definitions:   $attribute_definitions"
  iecho "    key_schema:   $key_schema"
  iecho ""

  response=$(aws dynamodb create-table \
    --table-name "$table_name" \
    --attribute-definitions file://"$attribute_definitions" \
    --billing-mode PAY_PER_REQUEST \
    --key-schema file://"$key_schema" )

  local error_code=${?}

  if [[ $error_code -ne 0 ]]; then
    aws_cli_error_log $error_code
```

```
    errecho "ERROR: AWS reports create-table operation failed.$response"
    return 1
  fi

  return 0
}
```

Die in diesem Beispiel verwendeten Hilfsfunktionen.

```
###############################################################################
# function iecho
#
# This function enables the script to display the specified text only if
# the global variable $VERBOSE is set to true.
###############################################################################
function iecho() {
  if [[ $VERBOSE == true ]]; then
    echo "$@"
  fi
}

###############################################################################
# function errecho
#
# This function outputs everything sent to it to STDERR (standard error output).
###############################################################################
function errecho() {
  printf "%s\n" "$*" 1>&2
}

###############################################################################
# function aws_cli_error_log()
#
# This function is used to log the error messages from the AWS CLI.
#
# See https://docs.aws.amazon.com/cli/latest/topic/return-codes.html#cli-aws-
help-return-codes.
#
# The function expects the following argument:
#         $1 - The error code returned by the AWS CLI.
#
#  Returns:
```

```
#          0: - Success.
#
###############################################################################
function aws_cli_error_log() {
  local err_code=$1
  errecho "Error code : $err_code"
  if [ "$err_code" == 1 ]; then
    errecho "  One or more S3 transfers failed."
  elif [ "$err_code" == 2 ]; then
    errecho "  Command line failed to parse."
  elif [ "$err_code" == 130 ]; then
    errecho "  Process received SIGINT."
  elif [ "$err_code" == 252 ]; then
    errecho "  Command syntax invalid."
  elif [ "$err_code" == 253 ]; then
    errecho "  The system environment or configuration was invalid."
  elif [ "$err_code" == 254 ]; then
    errecho "  The service returned an error."
  elif [ "$err_code" == 255 ]; then
    errecho "  255 is a catch-all error."
  fi

  return 0
}
```

- Einzelheiten zur API finden Sie CreateTablein der AWS CLI Befehlsreferenz.

C++

SDK für C++

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
//! Create an Amazon DynamoDB table.
/*!
  \sa createTable()
  \param tableName: Name for the DynamoDB table.
```

```
  \param primaryKey: Primary key for the DynamoDB table.
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
 */
bool AwsDoc::DynamoDB::createTable(const Aws::String &tableName,
                                   const Aws::String &primaryKey,
                                   const Aws::Client::ClientConfiguration
 &clientConfiguration) {
    Aws::DynamoDB::DynamoDBClient dynamoClient(clientConfiguration);

    std::cout << "Creating table " << tableName <<
              " with a simple primary key: \"" << primaryKey << "\"." <<
 std::endl;

    Aws::DynamoDB::Model::CreateTableRequest request;

    Aws::DynamoDB::Model::AttributeDefinition hashKey;
    hashKey.SetAttributeName(primaryKey);
    hashKey.SetAttributeType(Aws::DynamoDB::Model::ScalarAttributeType::S);
    request.AddAttributeDefinitions(hashKey);

    Aws::DynamoDB::Model::KeySchemaElement keySchemaElement;
    keySchemaElement.WithAttributeName(primaryKey).WithKeyType(
            Aws::DynamoDB::Model::KeyType::HASH);
    request.AddKeySchema(keySchemaElement);

    Aws::DynamoDB::Model::ProvisionedThroughput throughput;
    throughput.WithReadCapacityUnits(5).WithWriteCapacityUnits(5);
    request.SetProvisionedThroughput(throughput);
    request.SetTableName(tableName);

    const Aws::DynamoDB::Model::CreateTableOutcome &outcome =
 dynamoClient.CreateTable(
              request);
    if (outcome.IsSuccess()) {
        std::cout << "Table \""
                  << outcome.GetResult().GetTableDescription().GetTableName() <<
                  " created!" << std::endl;
    }
    else {
        std::cerr << "Failed to create table: " <<
 outcome.GetError().GetMessage()
                  << std::endl;
        return false;
```

```
    }

    return waitTableActive(tableName, dynamoClient);
}
```

Code, der darauf wartet, dass die Tabelle aktiv wird.

```
//! Query a newly created DynamoDB table until it is active.
/*!
  \sa waitTableActive()
  \param waitTableActive: The DynamoDB table's name.
  \param dynamoClient: A DynamoDB client.
  \return bool: Function succeeded.
*/
bool AwsDoc::DynamoDB::waitTableActive(const Aws::String &tableName,
                                       const Aws::DynamoDB::DynamoDBClient
 &dynamoClient) {

    // Repeatedly call DescribeTable until table is ACTIVE.
    const int MAX_QUERIES = 20;
    Aws::DynamoDB::Model::DescribeTableRequest request;
    request.SetTableName(tableName);

    int count = 0;
    while (count < MAX_QUERIES) {
        const Aws::DynamoDB::Model::DescribeTableOutcome &result =
 dynamoClient.DescribeTable(
                request);
        if (result.IsSuccess()) {
            Aws::DynamoDB::Model::TableStatus status =
 result.GetResult().GetTable().GetTableStatus();

            if (Aws::DynamoDB::Model::TableStatus::ACTIVE != status) {
                std::this_thread::sleep_for(std::chrono::seconds(1));
            }
            else {
                return true;
            }
        }
        else {
            std::cerr << "Error DynamoDB::waitTableActive "
                      << result.GetError().GetMessage() << std::endl;
```

```
            return false;
        }
        count++;
    }
    return false;
}
```

- Einzelheiten zur API finden Sie CreateTablein der AWS SDK für C++ API-Referenz.

CLI

AWS CLI

Beispiel 1: So erstellen Sie eine Tabelle mit Tags

Im folgenden Beispiel für `create-table` werden die angegebenen Attribute und das angegebene Schlüsselschema verwendet, um eine Tabelle mit dem Namen `MusicCollection` zu erstellen. Diese Tabelle verwendet den bereitgestellten Durchsatz und wird im Ruhezustand mit dem standardmäßigen AWS eigenen CMK verschlüsselt. Der Befehl weist der Tabelle außerdem ein Tag mit dem Schlüssel `Owner` und dem Wert `blueTeam` zu.

```
aws dynamodb create-table \
    --table-name MusicCollection \
    --attribute-
definitions AttributeName=Artist,AttributeType=S AttributeName=SongTitle,AttributeType=S
 \
    --key-
schema AttributeName=Artist,KeyType=HASH AttributeName=SongTitle,KeyType=RANGE \
    --provisioned-throughput ReadCapacityUnits=5,WriteCapacityUnits=5 \
    --tags Key=Owner,Value=blueTeam
```

Ausgabe:

```
{
    "TableDescription": {
        "AttributeDefinitions": [
            {
                "AttributeName": "Artist",
                "AttributeType": "S"
            },
```

```
            {
                "AttributeName": "SongTitle",
                "AttributeType": "S"
            }
        ],
        "ProvisionedThroughput": {
            "NumberOfDecreasesToday": 0,
            "WriteCapacityUnits": 5,
            "ReadCapacityUnits": 5
        },
        "TableSizeBytes": 0,
        "TableName": "MusicCollection",
        "TableStatus": "CREATING",
        "KeySchema": [
            {
                "KeyType": "HASH",
                "AttributeName": "Artist"
            },
            {
                "KeyType": "RANGE",
                "AttributeName": "SongTitle"
            }
        ],
        "ItemCount": 0,
        "CreationDateTime": "2020-05-26T16:04:41.627000-07:00",
        "TableArn": "arn:aws:dynamodb:us-west-2:123456789012:table/
MusicCollection",
        "TableId": "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111"
    }
}
```

Weitere Informationen finden Sie unter Grundlegende Operationen für Tabellen im Entwicklerhandbuch zu Amazon DynamoDB.

Beispiel 2: So erstellen Sie eine Tabelle im On-Demand-Modus

Im folgenden Beispiel wird mit dem On-Demand-Modus anstelle des bereitgestellten Durchsatzmodus eine Tabelle mit dem Namen `MusicCollection` erstellt. Dies ist nützlich für Tabellen mit unvorhersehbaren Workloads.

```
aws dynamodb create-table \
    --table-name MusicCollection \
```

```
    --attribute-
definitions AttributeName=Artist,AttributeType=S AttributeName=SongTitle,AttributeType=S
 \
    --key-
schema AttributeName=Artist,KeyType=HASH AttributeName=SongTitle,KeyType=RANGE \
    --billing-mode PAY_PER_REQUEST
```

Ausgabe:

```
{
    "TableDescription": {
        "AttributeDefinitions": [
            {
                "AttributeName": "Artist",
                "AttributeType": "S"
            },
            {
                "AttributeName": "SongTitle",
                "AttributeType": "S"
            }
        ],
        "TableName": "MusicCollection",
        "KeySchema": [
            {
                "AttributeName": "Artist",
                "KeyType": "HASH"
            },
            {
                "AttributeName": "SongTitle",
                "KeyType": "RANGE"
            }
        ],
        "TableStatus": "CREATING",
        "CreationDateTime": "2020-05-27T11:44:10.807000-07:00",
        "ProvisionedThroughput": {
            "NumberOfDecreasesToday": 0,
            "ReadCapacityUnits": 0,
            "WriteCapacityUnits": 0
        },
        "TableSizeBytes": 0,
        "ItemCount": 0,
        "TableArn": "arn:aws:dynamodb:us-west-2:123456789012:table/
MusicCollection",
```

```
        "TableId": "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
        "BillingModeSummary": {
            "BillingMode": "PAY_PER_REQUEST"
        }
    }
}
```

Weitere Informationen finden Sie unter Grundlegende Operationen für Tabellen im Entwicklerhandbuch zu Amazon DynamoDB.

Beispiel 3: So erstellen Sie eine Tabelle und verschlüsseln sie mit einem kundenseitig verwalteten Schlüssel (Customer Managed Key; CMK).

Im folgenden Beispiel wird eine Tabelle mit dem Namen `MusicCollection` erstellt und mithilfe eines kundenseitig verwalteten Schlüssels (Customer Managed Key; CMK) verschlüsselt.

```
aws dynamodb create-table \
    --table-name MusicCollection \
    --attribute-
definitions AttributeName=Artist,AttributeType=S AttributeName=SongTitle,AttributeType=S
 \
    --key-
schema AttributeName=Artist,KeyType=HASH AttributeName=SongTitle,KeyType=RANGE \
    --provisioned-throughput ReadCapacityUnits=5,WriteCapacityUnits=5 \
    --sse-specification Enabled=true,SSEType=KMS,KMSMasterKeyId=abcd1234-
abcd-1234-a123-ab1234a1b234
```

Ausgabe:

```
{
    "TableDescription": {
        "AttributeDefinitions": [
            {
                "AttributeName": "Artist",
                "AttributeType": "S"
            },
            {
                "AttributeName": "SongTitle",
                "AttributeType": "S"
            }
        ],
```

```
        "TableName": "MusicCollection",
        "KeySchema": [
            {
                "AttributeName": "Artist",
                "KeyType": "HASH"
            },
            {
                "AttributeName": "SongTitle",
                "KeyType": "RANGE"
            }
        ],
        "TableStatus": "CREATING",
        "CreationDateTime": "2020-05-27T11:12:16.431000-07:00",
        "ProvisionedThroughput": {
            "NumberOfDecreasesToday": 0,
            "ReadCapacityUnits": 5,
            "WriteCapacityUnits": 5
        },
        "TableSizeBytes": 0,
        "ItemCount": 0,
        "TableArn": "arn:aws:dynamodb:us-west-2:123456789012:table/
MusicCollection",
        "TableId": "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
        "SSEDescription": {
            "Status": "ENABLED",
            "SSEType": "KMS",
            "KMSMasterKeyArn": "arn:aws:kms:us-west-2:123456789012:key/abcd1234-
abcd-1234-a123-ab1234a1b234"
        }
    }
}
```

Weitere Informationen finden Sie unter Grundlegende Operationen für Tabellen im Entwicklerhandbuch zu Amazon DynamoDB.

Bespiel 4: So erstellen Sie eine Tabelle mit einem lokalen sekundären Index

Im folgenden Beispiel werden die angegebenen Attribute und das angegebene Schlüsselschema verwendet, um eine Tabelle namens `MusicCollection` mit dem lokalen sekundären Index `AlbumTitleIndex` zu erstellen.

```
aws dynamodb create-table \
    --table-name MusicCollection \
```

```
    --attribute-
definitions AttributeName=Artist,AttributeType=S AttributeName=SongTitle,AttributeType=S
 \
    --key-
schema AttributeName=Artist,KeyType=HASH AttributeName=SongTitle,KeyType=RANGE \
    --provisioned-throughput ReadCapacityUnits=10,WriteCapacityUnits=5 \
    --local-secondary-indexes \
        "[
            {
                \"IndexName\": \"AlbumTitleIndex\",
                \"KeySchema\": [
                    {\"AttributeName\": \"Artist\",\"KeyType\":\"HASH\"},
                    {\"AttributeName\": \"AlbumTitle\",\"KeyType\":\"RANGE\"}
                ],
                \"Projection\": {
                    \"ProjectionType\": \"INCLUDE\",
                    \"NonKeyAttributes\": [\"Genre\", \"Year\"]
                }
            }
        ]"
```

Ausgabe:

```
{
    "TableDescription": {
        "AttributeDefinitions": [
            {
                "AttributeName": "AlbumTitle",
                "AttributeType": "S"
            },
            {
                "AttributeName": "Artist",
                "AttributeType": "S"
            },
            {
                "AttributeName": "SongTitle",
                "AttributeType": "S"
            }
        ],
        "TableName": "MusicCollection",
        "KeySchema": [
            {
                "AttributeName": "Artist",
```

```
                "KeyType": "HASH"
            },
            {
                "AttributeName": "SongTitle",
                "KeyType": "RANGE"
            }
        ],
        "TableStatus": "CREATING",
        "CreationDateTime": "2020-05-26T15:59:49.473000-07:00",
        "ProvisionedThroughput": {
            "NumberOfDecreasesToday": 0,
            "ReadCapacityUnits": 10,
            "WriteCapacityUnits": 5
        },
        "TableSizeBytes": 0,
        "ItemCount": 0,
        "TableArn": "arn:aws:dynamodb:us-west-2:123456789012:table/
MusicCollection",
        "TableId": "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
        "LocalSecondaryIndexes": [
            {
                "IndexName": "AlbumTitleIndex",
                "KeySchema": [
                    {
                        "AttributeName": "Artist",
                        "KeyType": "HASH"
                    },
                    {
                        "AttributeName": "AlbumTitle",
                        "KeyType": "RANGE"
                    }
                ],
                "Projection": {
                    "ProjectionType": "INCLUDE",
                    "NonKeyAttributes": [
                        "Genre",
                        "Year"
                    ]
                },
                "IndexSizeBytes": 0,
                "ItemCount": 0,
                "IndexArn": "arn:aws:dynamodb:us-west-2:123456789012:table/
MusicCollection/index/AlbumTitleIndex"
            }
```

```
        ]
    }
}
```

Weitere Informationen finden Sie unter Grundlegende Operationen für Tabellen im Entwicklerhandbuch zu Amazon DynamoDB.

Beispiel 5: So erstellen Sie eine Tabelle mit einem globalen sekundären Index

Im folgenden Beispiel wird eine Tabelle mit dem Namen `GameScores` mit einem globalen sekundären Index namens `GameTitleIndex` erstellt. Die Basistabelle hat einen Partitionsschlüssel von `UserId` und einen Sortierschlüssel von `GameTitle`, mit dem Sie effizient die beste Punktzahl eines einzelnen Benutzers für ein bestimmtes Spiel finden können, während die GSI einen Partitionsschlüssel von `GameTitle` und einen Sortierschlüssel von `TopScore` hat, mit dem Sie finden Sie schnell die höchste Gesamtpunktzahl für ein bestimmtes Spiel.

```
aws dynamodb create-table \
    --table-name GameScores \
    --attribute-
definitions AttributeName=UserId,AttributeType=S AttributeName=GameTitle,AttributeType=S
 \
    --key-schema AttributeName=UserId,KeyType=HASH \
                 AttributeName=GameTitle,KeyType=RANGE \
    --provisioned-throughput ReadCapacityUnits=10,WriteCapacityUnits=5 \
    --global-secondary-indexes \
        "[
            {
                \"IndexName\": \"GameTitleIndex\",
                \"KeySchema\": [
                    {\"AttributeName\":\"GameTitle\",\"KeyType\":\"HASH\"},
                    {\"AttributeName\":\"TopScore\",\"KeyType\":\"RANGE\"}
                ],
                \"Projection\": {
                    \"ProjectionType\":\"INCLUDE\",
                    \"NonKeyAttributes\":[\"UserId\"]
                },
                \"ProvisionedThroughput\": {
                    \"ReadCapacityUnits\": 10,
                    \"WriteCapacityUnits\": 5
                }
            }
```

```
        ]"
```

Ausgabe:

```
{
    "TableDescription": {
        "AttributeDefinitions": [
            {
                "AttributeName": "GameTitle",
                "AttributeType": "S"
            },
            {
                "AttributeName": "TopScore",
                "AttributeType": "N"
            },
            {
                "AttributeName": "UserId",
                "AttributeType": "S"
            }
        ],
        "TableName": "GameScores",
        "KeySchema": [
            {
                "AttributeName": "UserId",
                "KeyType": "HASH"
            },
            {
                "AttributeName": "GameTitle",
                "KeyType": "RANGE"
            }
        ],
        "TableStatus": "CREATING",
        "CreationDateTime": "2020-05-26T17:28:15.602000-07:00",
        "ProvisionedThroughput": {
            "NumberOfDecreasesToday": 0,
            "ReadCapacityUnits": 10,
            "WriteCapacityUnits": 5
        },
        "TableSizeBytes": 0,
        "ItemCount": 0,
        "TableArn": "arn:aws:dynamodb:us-west-2:123456789012:table/GameScores",
        "TableId": "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
        "GlobalSecondaryIndexes": [
```

```
            {
                "IndexName": "GameTitleIndex",
                "KeySchema": [
                    {
                        "AttributeName": "GameTitle",
                        "KeyType": "HASH"
                    },
                    {
                        "AttributeName": "TopScore",
                        "KeyType": "RANGE"
                    }
                ],
                "Projection": {
                    "ProjectionType": "INCLUDE",
                    "NonKeyAttributes": [
                        "UserId"
                    ]
                },
                "IndexStatus": "CREATING",
                "ProvisionedThroughput": {
                    "NumberOfDecreasesToday": 0,
                    "ReadCapacityUnits": 10,
                    "WriteCapacityUnits": 5
                },
                "IndexSizeBytes": 0,
                "ItemCount": 0,
                "IndexArn": "arn:aws:dynamodb:us-west-2:123456789012:table/
GameScores/index/GameTitleIndex"
            }
        ]
    }
}
```

Weitere Informationen finden Sie unter Grundlegende Operationen für Tabellen im Entwicklerhandbuch zu Amazon DynamoDB.

Beispiel 6: So erstellen Sie eine Tabelle mit mehreren globalen sekundären Indizes gleichzeitig

Im folgenden Beispiel wird eine Tabelle namens `GameScores` mit zwei globalen sekundären Indizes erstellt. Die GSI-Schemas werden über eine Datei und nicht über die Befehlszeile übergeben.

```
aws dynamodb create-table \
```

```
    --table-name GameScores \
    --attribute-
definitions AttributeName=UserId,AttributeType=S AttributeName=GameTitle,AttributeType=S
 \
    --key-
schema AttributeName=UserId,KeyType=HASH AttributeName=GameTitle,KeyType=RANGE \
    --provisioned-throughput ReadCapacityUnits=10,WriteCapacityUnits=5 \
    --global-secondary-indexes file://gsi.json
```

Inhalt von `gsi.json`:

```
[
    {
        "IndexName": "GameTitleIndex",
        "KeySchema": [
            {
                "AttributeName": "GameTitle",
                "KeyType": "HASH"
            },
            {
                "AttributeName": "TopScore",
                "KeyType": "RANGE"
            }
        ],
        "Projection": {
            "ProjectionType": "ALL"
        },
        "ProvisionedThroughput": {
            "ReadCapacityUnits": 10,
            "WriteCapacityUnits": 5
        }
    },
    {
        "IndexName": "GameDateIndex",
        "KeySchema": [
            {
                "AttributeName": "GameTitle",
                "KeyType": "HASH"
            },
            {
                "AttributeName": "Date",
                "KeyType": "RANGE"
            }
```

```
        ],
        "Projection": {
            "ProjectionType": "ALL"
        },
        "ProvisionedThroughput": {
            "ReadCapacityUnits": 5,
            "WriteCapacityUnits": 5
        }
    }
]
```

Ausgabe:

```
{
    "TableDescription": {
        "AttributeDefinitions": [
            {
                "AttributeName": "Date",
                "AttributeType": "S"
            },
            {
                "AttributeName": "GameTitle",
                "AttributeType": "S"
            },
            {
                "AttributeName": "TopScore",
                "AttributeType": "N"
            },
            {
                "AttributeName": "UserId",
                "AttributeType": "S"
            }
        ],
        "TableName": "GameScores",
        "KeySchema": [
            {
                "AttributeName": "UserId",
                "KeyType": "HASH"
            },
            {
                "AttributeName": "GameTitle",
                "KeyType": "RANGE"
            }
```

```
        ],
        "TableStatus": "CREATING",
        "CreationDateTime": "2020-08-04T16:40:55.524000-07:00",
        "ProvisionedThroughput": {
            "NumberOfDecreasesToday": 0,
            "ReadCapacityUnits": 10,
            "WriteCapacityUnits": 5
        },
        "TableSizeBytes": 0,
        "ItemCount": 0,
        "TableArn": "arn:aws:dynamodb:us-west-2:123456789012:table/GameScores",
        "TableId": "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
        "GlobalSecondaryIndexes": [
            {
                "IndexName": "GameTitleIndex",
                "KeySchema": [
                    {
                        "AttributeName": "GameTitle",
                        "KeyType": "HASH"
                    },
                    {
                        "AttributeName": "TopScore",
                        "KeyType": "RANGE"
                    }
                ],
                "Projection": {
                    "ProjectionType": "ALL"
                },
                "IndexStatus": "CREATING",
                "ProvisionedThroughput": {
                    "NumberOfDecreasesToday": 0,
                    "ReadCapacityUnits": 10,
                    "WriteCapacityUnits": 5
                },
                "IndexSizeBytes": 0,
                "ItemCount": 0,
                "IndexArn": "arn:aws:dynamodb:us-west-2:123456789012:table/
GameScores/index/GameTitleIndex"
            },
            {
                "IndexName": "GameDateIndex",
                "KeySchema": [
                    {
                        "AttributeName": "GameTitle",
```

```
                        "KeyType": "HASH"
                    },
                    {
                        "AttributeName": "Date",
                        "KeyType": "RANGE"
                    }
                ],
                "Projection": {
                    "ProjectionType": "ALL"
                },
                "IndexStatus": "CREATING",
                "ProvisionedThroughput": {
                    "NumberOfDecreasesToday": 0,
                    "ReadCapacityUnits": 5,
                    "WriteCapacityUnits": 5
                },
                "IndexSizeBytes": 0,
                "ItemCount": 0,
                "IndexArn": "arn:aws:dynamodb:us-west-2:123456789012:table/
GameScores/index/GameDateIndex"
            }
        ]
    }
}
```

Weitere Informationen finden Sie unter Grundlegende Operationen für Tabellen im Entwicklerhandbuch zu Amazon DynamoDB.

Beispiel 7: So erstellen Sie eine Tabelle mit aktivierten Streams

Im folgenden Beispiel wird die Tabelle `GameScores` mit aktiviertem DynamoDB Streams aufgerufen. Sowohl neue als auch alte Images von jedem Element werden in den Stream geschrieben.

```
aws dynamodb create-table \
    --table-name GameScores \
    --attribute-
definitions AttributeName=UserId,AttributeType=S AttributeName=GameTitle,AttributeType=S
 \
    --key-
schema AttributeName=UserId,KeyType=HASH AttributeName=GameTitle,KeyType=RANGE \
    --provisioned-throughput ReadCapacityUnits=10,WriteCapacityUnits=5 \
    --stream-specification StreamEnabled=TRUE,StreamViewType=NEW_AND_OLD_IMAGES
```

Ausgabe:

```
{
    "TableDescription": {
        "AttributeDefinitions": [
            {
                "AttributeName": "GameTitle",
                "AttributeType": "S"
            },
            {
                "AttributeName": "UserId",
                "AttributeType": "S"
            }
        ],
        "TableName": "GameScores",
        "KeySchema": [
            {
                "AttributeName": "UserId",
                "KeyType": "HASH"
            },
            {
                "AttributeName": "GameTitle",
                "KeyType": "RANGE"
            }
        ],
        "TableStatus": "CREATING",
        "CreationDateTime": "2020-05-27T10:49:34.056000-07:00",
        "ProvisionedThroughput": {
            "NumberOfDecreasesToday": 0,
            "ReadCapacityUnits": 10,
            "WriteCapacityUnits": 5
        },
        "TableSizeBytes": 0,
        "ItemCount": 0,
        "TableArn": "arn:aws:dynamodb:us-west-2:123456789012:table/GameScores",
        "TableId": "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
        "StreamSpecification": {
            "StreamEnabled": true,
            "StreamViewType": "NEW_AND_OLD_IMAGES"
        },
        "LatestStreamLabel": "2020-05-27T17:49:34.056",
        "LatestStreamArn": "arn:aws:dynamodb:us-west-2:123456789012:table/
GameScores/stream/2020-05-27T17:49:34.056"
    }
```

```
}
```

Weitere Informationen finden Sie unter Grundlegende Operationen für Tabellen im Entwicklerhandbuch zu Amazon DynamoDB.

Beispiel 8: So erstellen Sie eine Tabelle mit aktiviertem Keys-Only-Stream

Im folgenden Beispiel wird die Tabelle `GameScores` mit aktiviertem DynamoDB Streams aufgerufen. Nur die Schlüsselattribute von geänderten Elementen werden in den Stream geschrieben.

```
aws dynamodb create-table \
    --table-name GameScores \
    --attribute-
definitions AttributeName=UserId,AttributeType=S AttributeName=GameTitle,AttributeType=S
 \
    --key-
schema AttributeName=UserId,KeyType=HASH AttributeName=GameTitle,KeyType=RANGE \
    --provisioned-throughput ReadCapacityUnits=10,WriteCapacityUnits=5 \
    --stream-specification StreamEnabled=TRUE,StreamViewType=KEYS_ONLY
```

Ausgabe:

```
{
    "TableDescription": {
        "AttributeDefinitions": [
            {
                "AttributeName": "GameTitle",
                "AttributeType": "S"
            },
            {
                "AttributeName": "UserId",
                "AttributeType": "S"
            }
        ],
        "TableName": "GameScores",
        "KeySchema": [
            {
                "AttributeName": "UserId",
                "KeyType": "HASH"
            },
            {
```

```
                "AttributeName": "GameTitle",
                "KeyType": "RANGE"
            }
        ],
        "TableStatus": "CREATING",
        "CreationDateTime": "2023-05-25T18:45:34.140000+00:00",
        "ProvisionedThroughput": {
            "NumberOfDecreasesToday": 0,
            "ReadCapacityUnits": 10,
            "WriteCapacityUnits": 5
        },
        "TableSizeBytes": 0,
        "ItemCount": 0,
        "TableArn": "arn:aws:dynamodb:us-west-2:123456789012:table/GameScores",
        "TableId": "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
        "StreamSpecification": {
            "StreamEnabled": true,
            "StreamViewType": "KEYS_ONLY"
        },
        "LatestStreamLabel": "2023-05-25T18:45:34.140",
        "LatestStreamArn": "arn:aws:dynamodb:us-west-2:123456789012:table/
GameScores/stream/2023-05-25T18:45:34.140",
        "DeletionProtectionEnabled": false
    }
}
```

Weitere Informationen finden Sie unter Ändern der Datenerfassung für DynamoDB Streams im Entwicklerhandbuch zu Amazon DynamoDB.

Beispiel 9: So erstellen Sie eine Tabelle mit der Klasse Standard Infrequent Access

Im folgenden Beispiel wird eine Tabelle mit dem Namen `GameScores` erstellt und die Tabellenklasse Standard-Infrequent Access (DynamoDB Standard-IA) zugewiesen. Diese Tabellenklasse ist dafür optimiert, dass Speicher der Hauptkostenfaktor ist.

```
aws dynamodb create-table \
    --table-name GameScores \
    --attribute-
definitions AttributeName=UserId,AttributeType=S AttributeName=GameTitle,AttributeType=S
 \
    --key-
schema AttributeName=UserId,KeyType=HASH AttributeName=GameTitle,KeyType=RANGE \
    --provisioned-throughput ReadCapacityUnits=10,WriteCapacityUnits=5 \
```

```
    --table-class STANDARD_INFREQUENT_ACCESS
```

Ausgabe:

```
{
    "TableDescription": {
        "AttributeDefinitions": [
            {
                "AttributeName": "GameTitle",
                "AttributeType": "S"
            },
            {
                "AttributeName": "UserId",
                "AttributeType": "S"
            }
        ],
        "TableName": "GameScores",
        "KeySchema": [
            {
                "AttributeName": "UserId",
                "KeyType": "HASH"
            },
            {
                "AttributeName": "GameTitle",
                "KeyType": "RANGE"
            }
        ],
        "TableStatus": "CREATING",
        "CreationDateTime": "2023-05-25T18:33:07.581000+00:00",
        "ProvisionedThroughput": {
            "NumberOfDecreasesToday": 0,
            "ReadCapacityUnits": 10,
            "WriteCapacityUnits": 5
        },
        "TableSizeBytes": 0,
        "ItemCount": 0,
        "TableArn": "arn:aws:dynamodb:us-west-2:123456789012:table/GameScores",
        "TableId": "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
        "TableClassSummary": {
            "TableClass": "STANDARD_INFREQUENT_ACCESS"
        },
        "DeletionProtectionEnabled": false
    }
```

```
}
```

Weitere Informationen finden Sie unter Tabellenklassen im Entwicklerhandbuch zu Amazon DynamoDB.

Beispiel 10: So erstellen Sie eine Tabelle mit aktiviertem Löschschutz

Im folgenden Beispiel wird eine Tabelle mit dem Namen `GameScores` erstellt und der Löschschutz aktiviert.

```
aws dynamodb create-table \
    --table-name GameScores \
    --attribute-
definitions AttributeName=UserId,AttributeType=S AttributeName=GameTitle,AttributeType=S
 \
    --key-
schema AttributeName=UserId,KeyType=HASH AttributeName=GameTitle,KeyType=RANGE \
    --provisioned-throughput ReadCapacityUnits=10,WriteCapacityUnits=5 \
    --deletion-protection-enabled
```

Ausgabe:

```
{
    "TableDescription": {
        "AttributeDefinitions": [
            {
                "AttributeName": "GameTitle",
                "AttributeType": "S"
            },
            {
                "AttributeName": "UserId",
                "AttributeType": "S"
            }
        ],
        "TableName": "GameScores",
        "KeySchema": [
            {
                "AttributeName": "UserId",
                "KeyType": "HASH"
            },
            {
                "AttributeName": "GameTitle",
                "KeyType": "RANGE"
```

```
            }
        ],
        "TableStatus": "CREATING",
        "CreationDateTime": "2023-05-25T23:02:17.093000+00:00",
        "ProvisionedThroughput": {
            "NumberOfDecreasesToday": 0,
            "ReadCapacityUnits": 10,
            "WriteCapacityUnits": 5
        },
        "TableSizeBytes": 0,
        "ItemCount": 0,
        "TableArn": "arn:aws:dynamodb:us-west-2:123456789012:table/GameScores",
        "TableId": "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
        "DeletionProtectionEnabled": true
    }
}
```

Weitere Informationen finden Sie unter Verwenden des Löschschutzes im Entwicklerhandbuch zu Amazon DynamoDB.

- Einzelheiten zur API finden Sie unter CreateTable AWS CLIBefehlsreferenz.

Go

SDK für Go V2

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
import (
 "context"
 "errors"
 "log"
 "time"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/attributevalue"
 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/expression"
```

```
 "github.com/aws/aws-sdk-go-v2/service/dynamodb"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb/types"
)

// TableBasics encapsulates the Amazon DynamoDB service actions used in the
 examples.
// It contains a DynamoDB service client that is used to act on the specified
 table.
type TableBasics struct {
 DynamoDbClient *dynamodb.Client
 TableName      string
}



// CreateMovieTable creates a DynamoDB table with a composite primary key defined
 as
// a string sort key named `title`, and a numeric partition key named `year`.
// This function uses NewTableExistsWaiter to wait for the table to be created by
// DynamoDB before it returns.
func (basics TableBasics) CreateMovieTable(ctx context.Context)
 (*types.TableDescription, error) {
 var tableDesc *types.TableDescription
 table, err := basics.DynamoDbClient.CreateTable(ctx, &dynamodb.CreateTableInput{
  AttributeDefinitions: []types.AttributeDefinition{{
   AttributeName: aws.String("year"),
   AttributeType: types.ScalarAttributeTypeN,
  }, {
   AttributeName: aws.String("title"),
   AttributeType: types.ScalarAttributeTypeS,
  }},
  KeySchema: []types.KeySchemaElement{{
   AttributeName: aws.String("year"),
   KeyType:       types.KeyTypeHash,
  }, {
   AttributeName: aws.String("title"),
   KeyType:       types.KeyTypeRange,
  }},
  TableName:   aws.String(basics.TableName),
  BillingMode: types.BillingModePayPerRequest,
 })
 if err != nil {
  log.Printf("Couldn't create table %v. Here's why: %v\n", basics.TableName, err)
 } else {
```

```
  waiter := dynamodb.NewTableExistsWaiter(basics.DynamoDbClient)
  err = waiter.Wait(ctx, &dynamodb.DescribeTableInput{
   TableName: aws.String(basics.TableName)}, 5*time.Minute)
  if err != nil {
   log.Printf("Wait for table exists failed. Here's why: %v\n", err)
  }
  tableDesc = table.TableDescription
  log.Printf("Ccreating table test")
 }
 return tableDesc, err
}
```

- Einzelheiten zur API finden Sie CreateTablein der AWS SDK für Go API-Referenz.

Java

SDK für Java 2.x

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
import software.amazon.awssdk.core.waiters.WaiterResponse;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.AttributeDefinition;
import software.amazon.awssdk.services.dynamodb.model.BillingMode;
import software.amazon.awssdk.services.dynamodb.model.CreateTableRequest;
import software.amazon.awssdk.services.dynamodb.model.CreateTableResponse;
import software.amazon.awssdk.services.dynamodb.model.DescribeTableRequest;
import software.amazon.awssdk.services.dynamodb.model.DescribeTableResponse;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.model.KeySchemaElement;
import software.amazon.awssdk.services.dynamodb.model.KeyType;
import software.amazon.awssdk.services.dynamodb.model.OnDemandThroughput;
import software.amazon.awssdk.services.dynamodb.model.ProvisionedThroughput;
import software.amazon.awssdk.services.dynamodb.model.ScalarAttributeType;
```

```
import software.amazon.awssdk.services.dynamodb.waiters.DynamoDbWaiter;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 * <p>
 * For more information, see the following documentation topic:
 * <p>
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 */
public class CreateTable {
    public static void main(String[] args) {
        final String usage = """

            Usage:
                <tableName> <key>

            Where:
                tableName - The Amazon DynamoDB table to create (for example,
 Music3).
                key - The key for the Amazon DynamoDB table (for example,
 Artist).
            """;

        if (args.length != 2) {
            System.out.println(usage);
            System.exit(1);
        }

        String tableName = args[0];
        String key = args[1];
        System.out.println("Creating an Amazon DynamoDB table " + tableName + "
 with a simple primary key: " + key);
        Region region = Region.US_EAST_1;
        DynamoDbClient ddb = DynamoDbClient.builder()
            .region(region)
            .build();

        String result = createTable(ddb, tableName, key);
        System.out.println("New table is " + result);
        ddb.close();
    }
```

```
    public static String createTable(DynamoDbClient ddb, String tableName, String
 key) {
        DynamoDbWaiter dbWaiter = ddb.waiter();
        CreateTableRequest request = CreateTableRequest.builder()
            .attributeDefinitions(AttributeDefinition.builder()
                .attributeName(key)
                .attributeType(ScalarAttributeType.S)
                .build())
            .keySchema(KeySchemaElement.builder()
                .attributeName(key)
                .keyType(KeyType.HASH)
                .build())
            .billingMode(BillingMode.PAY_PER_REQUEST) //  DynamoDB automatically
 scales based on traffic.
            .tableName(tableName)
            .build();

        String newTable;
        try {
            CreateTableResponse response = ddb.createTable(request);
            DescribeTableRequest tableRequest = DescribeTableRequest.builder()
                .tableName(tableName)
                .build();

            // Wait until the Amazon DynamoDB table is created.
            WaiterResponse<DescribeTableResponse> waiterResponse =
 dbWaiter.waitUntilTableExists(tableRequest);
            waiterResponse.matched().response().ifPresent(System.out::println);
            newTable = response.tableDescription().tableName();
            return newTable;

        } catch (DynamoDbException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
        return "";
    }
}
```

- Einzelheiten zur API finden Sie CreateTablein der AWS SDK for Java 2.x API-Referenz.

JavaScript

SDK für JavaScript (v3)

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
import { CreateTableCommand, DynamoDBClient } from "@aws-sdk/client-dynamodb";

const client = new DynamoDBClient({});

export const main = async () => {
  const command = new CreateTableCommand({
    TableName: "EspressoDrinks",
    // For more information about data types,
    // see https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/
HowItWorks.NamingRulesDataTypes.html#HowItWorks.DataTypes and
    // https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/
Programming.LowLevelAPI.html#Programming.LowLevelAPI.DataTypeDescriptors
    AttributeDefinitions: [
      {
        AttributeName: "DrinkName",
        AttributeType: "S",
      },
    ],
    KeySchema: [
      {
        AttributeName: "DrinkName",
        KeyType: "HASH",
      },
    ],
    BillingMode: "PAY_PER_REQUEST",
  });

  const response = await client.send(command);
  console.log(response);
  return response;
};
```

- Weitere Informationen finden Sie im AWS SDK für JavaScript -Entwicklerhandbuch.
- Einzelheiten zur API finden Sie CreateTablein der AWS SDK für JavaScript API-Referenz.

SDK für JavaScript (v2)

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create the DynamoDB service object
var ddb = new AWS.DynamoDB({ apiVersion: "2012-08-10" });

var params = {
  AttributeDefinitions: [
    {
      AttributeName: "CUSTOMER_ID",
      AttributeType: "N",
    },
    {
      AttributeName: "CUSTOMER_NAME",
      AttributeType: "S",
    },
  ],
  KeySchema: [
    {
      AttributeName: "CUSTOMER_ID",
      KeyType: "HASH",
    },
    {
      AttributeName: "CUSTOMER_NAME",
      KeyType: "RANGE",
    },
  ],
  ProvisionedThroughput: {
    ReadCapacityUnits: 1,
    WriteCapacityUnits: 1,
```

```
  },
  TableName: "CUSTOMER_LIST",
  StreamSpecification: {
    StreamEnabled: false,
  },
};

// Call DynamoDB to create the table
ddb.createTable(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Table Created", data);
  }
});
```

- Weitere Informationen finden Sie im AWS SDK für JavaScript -Entwicklerhandbuch.
- Einzelheiten zur API finden Sie CreateTablein der AWS SDK für JavaScript API-Referenz.

Kotlin

SDK für Kotlin

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
suspend fun createNewTable(
    tableNameVal: String,
    key: String,
): String? {
    val attDef =
        AttributeDefinition {
            attributeName = key
            attributeType = ScalarAttributeType.S
        }

    val keySchemaVal =
```

```
        KeySchemaElement {
            attributeName = key
            keyType = KeyType.Hash
        }

    val request =
        CreateTableRequest {
            attributeDefinitions = listOf(attDef)
            keySchema = listOf(keySchemaVal)
            billingMode = BillingMode.PayPerRequest
            tableName = tableNameVal
        }

    DynamoDbClient.fromEnvironment { region = "us-east-1" }.use { ddb ->
        var tableArn: String
        val response = ddb.createTable(request)
        ddb.waitUntilTableExists {
            // suspend call
            tableName = tableNameVal
        }
        tableArn = response.tableDescription!!.tableArn.toString()
        println("Table $tableArn is ready")
        return tableArn
    }
}
```

- API-Details finden Sie CreateTablein der API-Referenz zum AWS SDK für Kotlin.

PHP

SDK für PHP

> **Note**
>
> Es gibt noch mehr dazu. GitHub Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

Erstellen Sie eine -Tabelle.

```
        $tableName = "ddb_demo_table_$uuid";
```

```
        $service->createTable(
            $tableName,
            [
                new DynamoDBAttribute('year', 'N', 'HASH'),
                new DynamoDBAttribute('title', 'S', 'RANGE')
            ]
        );

    public function createTable(string $tableName, array $attributes)
    {
        $keySchema = [];
        $attributeDefinitions = [];
        foreach ($attributes as $attribute) {
            if (is_a($attribute, DynamoDBAttribute::class)) {
                $keySchema[] = ['AttributeName' => $attribute->AttributeName,
'KeyType' => $attribute->KeyType];
                $attributeDefinitions[] =
                    ['AttributeName' => $attribute->AttributeName,
'AttributeType' => $attribute->AttributeType];
            }
        }

        $this->dynamoDbClient->createTable([
            'TableName' => $tableName,
            'KeySchema' => $keySchema,
            'AttributeDefinitions' => $attributeDefinitions,
            'ProvisionedThroughput' => ['ReadCapacityUnits' => 10,
'WriteCapacityUnits' => 10],
        ]);
    }
```

- Einzelheiten zur API finden Sie CreateTablein der AWS SDK für PHP API-Referenz.

PowerShell

Tools für PowerShell V4

Beispiel 1: In diesem Beispiel wird eine Tabelle mit dem Namen Thread erstellt, deren Primärschlüssel aus 'ForumName' (Schlüsseltyp-Hash) und 'Subject' (Schlüsseltypbereich) besteht. Das zur Erstellung der Tabelle verwendete Schema kann wie gezeigt an jedes Cmdlet übergeben oder mit dem Parameter „-Schema“ angegeben werden.

```
$schema = New-DDBTableSchema
$schema | Add-DDBKeySchema -KeyName "ForumName" -KeyDataType "S"
$schema | Add-DDBKeySchema -KeyName "Subject" -KeyType RANGE -KeyDataType "S"
$schema | New-DDBTable -TableName "Thread" -ReadCapacity 10 -WriteCapacity 5
```

Ausgabe:

```
AttributeDefinitions   : {ForumName, Subject}
TableName              : Thread
KeySchema              : {ForumName, Subject}
TableStatus            : CREATING
CreationDateTime       : 10/28/2013 4:39:49 PM
ProvisionedThroughput  : Amazon.DynamoDBv2.Model.ProvisionedThroughputDescription
TableSizeBytes         : 0
ItemCount              : 0
LocalSecondaryIndexes  : {}
```

Beispiel 2: In diesem Beispiel wird eine Tabelle mit dem Namen Thread erstellt, deren Primärschlüssel aus 'ForumName' (Schlüsseltyp-Hash) und 'Subject' (Schlüsseltypbereich) besteht. Ein lokaler sekundärer Index wird ebenfalls definiert. Der Schlüssel des lokalen sekundären Indexes wird automatisch anhand des primären Hashschlüssels in der Tabelle (ForumName) festgelegt. Das zur Erstellung der Tabelle verwendete Schema kann wie gezeigt an jedes Cmdlet übergeben oder mit dem Parameter „-Schema“ angegeben werden.

```
$schema = New-DDBTableSchema
$schema | Add-DDBKeySchema -KeyName "ForumName" -KeyDataType "S"
$schema | Add-DDBKeySchema -KeyName "Subject" -KeyDataType "S"
$schema | Add-DDBIndexSchema -IndexName "LastPostIndex" -RangeKeyName
 "LastPostDateTime" -RangeKeyDataType "S" -ProjectionType "keys_only"
$schema | New-DDBTable -TableName "Thread" -ReadCapacity 10 -WriteCapacity 5
```

Ausgabe:

```
AttributeDefinitions   : {ForumName, LastPostDateTime, Subject}
TableName              : Thread
KeySchema              : {ForumName, Subject}
TableStatus            : CREATING
CreationDateTime       : 10/28/2013 4:39:49 PM
ProvisionedThroughput  : Amazon.DynamoDBv2.Model.ProvisionedThroughputDescription
TableSizeBytes         : 0
ItemCount              : 0
```

```
LocalSecondaryIndexes  : {LastPostIndex}
```

Beispiel 3: Dieses Beispiel zeigt, wie eine einzelne Pipeline verwendet wird, um eine Tabelle mit dem Namen Thread zu erstellen, deren Primärschlüssel aus 'ForumName' (Schlüsseltyp Hash) und 'Subject' (Schlüsseltypbereich) und einem lokalen sekundären Index besteht. Mit den Optionen „Add- DDBKey Schema“ und DDBIndex „Add- Schema“ wird ein neues TableSchema Objekt für Sie erstellt, falls keines über die Pipeline oder den Parameter -Schema bereitgestellt wird.

```
New-DDBTableSchema |
  Add-DDBKeySchema -KeyName "ForumName" -KeyDataType "S" |
  Add-DDBKeySchema -KeyName "Subject" -KeyDataType "S" |
  Add-DDBIndexSchema -IndexName "LastPostIndex" `
                     -RangeKeyName "LastPostDateTime" `
                     -RangeKeyDataType "S" `
                     -ProjectionType "keys_only" |
  New-DDBTable -TableName "Thread" -ReadCapacity 10 -WriteCapacity 5
```

Ausgabe:

```
AttributeDefinitions   : {ForumName, LastPostDateTime, Subject}
TableName              : Thread
KeySchema              : {ForumName, Subject}
TableStatus            : CREATING
CreationDateTime       : 10/28/2013 4:39:49 PM
ProvisionedThroughput  : Amazon.DynamoDBv2.Model.ProvisionedThroughputDescription
TableSizeBytes         : 0
ItemCount              : 0
LocalSecondaryIndexes  : {LastPostIndex}
```

- Einzelheiten zur API finden Sie unter CreateTable AWS -Tools für PowerShellCmdlet-Referenz (V4).

Tools für V5 PowerShell

Beispiel 1: In diesem Beispiel wird eine Tabelle mit dem Namen Thread erstellt, deren Primärschlüssel aus 'ForumName' (Schlüsseltyp-Hash) und 'Subject' (Schlüsseltypbereich) besteht. Das zur Erstellung der Tabelle verwendete Schema kann wie gezeigt an jedes Cmdlet übergeben oder mit dem Parameter „-Schema“ angegeben werden.

```
$schema = New-DDBTableSchema
```

```
$schema | Add-DDBKeySchema -KeyName "ForumName" -KeyDataType "S"
$schema | Add-DDBKeySchema -KeyName "Subject" -KeyType RANGE -KeyDataType "S"
$schema | New-DDBTable -TableName "Thread" -ReadCapacity 10 -WriteCapacity 5
```

Ausgabe:

```
AttributeDefinitions   : {ForumName, Subject}
TableName              : Thread
KeySchema              : {ForumName, Subject}
TableStatus            : CREATING
CreationDateTime       : 10/28/2013 4:39:49 PM
ProvisionedThroughput  : Amazon.DynamoDBv2.Model.ProvisionedThroughputDescription
TableSizeBytes         : 0
ItemCount              : 0
LocalSecondaryIndexes  : {}
```

Beispiel 2: In diesem Beispiel wird eine Tabelle mit dem Namen Thread erstellt, deren Primärschlüssel aus 'ForumName' (Schlüsseltyp-Hash) und 'Subject' (Schlüsseltypbereich) besteht. Ein lokaler sekundärer Index wird ebenfalls definiert. Der Schlüssel des lokalen sekundären Indexes wird automatisch anhand des primären Hashschlüssels in der Tabelle (ForumName) festgelegt. Das zur Erstellung der Tabelle verwendete Schema kann wie gezeigt an jedes Cmdlet übergeben oder mit dem Parameter „-Schema“ angegeben werden.

```
$schema = New-DDBTableSchema
$schema | Add-DDBKeySchema -KeyName "ForumName" -KeyDataType "S"
$schema | Add-DDBKeySchema -KeyName "Subject" -KeyDataType "S"
$schema | Add-DDBIndexSchema -IndexName "LastPostIndex" -RangeKeyName
 "LastPostDateTime" -RangeKeyDataType "S" -ProjectionType "keys_only"
$schema | New-DDBTable -TableName "Thread" -ReadCapacity 10 -WriteCapacity 5
```

Ausgabe:

```
AttributeDefinitions   : {ForumName, LastPostDateTime, Subject}
TableName              : Thread
KeySchema              : {ForumName, Subject}
TableStatus            : CREATING
CreationDateTime       : 10/28/2013 4:39:49 PM
ProvisionedThroughput  : Amazon.DynamoDBv2.Model.ProvisionedThroughputDescription
TableSizeBytes         : 0
ItemCount              : 0
LocalSecondaryIndexes  : {LastPostIndex}
```

Beispiel 3: Dieses Beispiel zeigt, wie eine einzelne Pipeline verwendet wird, um eine Tabelle mit dem Namen Thread zu erstellen, deren Primärschlüssel aus 'ForumName' (Schlüsseltyp Hash) und 'Subject' (Schlüsseltypbereich) und einem lokalen sekundären Index besteht. Mit den Optionen „Add- DDBKey Schema“ und DDBIndex „Add- Schema“ wird ein neues TableSchema Objekt für Sie erstellt, falls keines über die Pipeline oder den Parameter -Schema bereitgestellt wird.

```
New-DDBTableSchema |
  Add-DDBKeySchema -KeyName "ForumName" -KeyDataType "S" |
  Add-DDBKeySchema -KeyName "Subject" -KeyDataType "S" |
  Add-DDBIndexSchema -IndexName "LastPostIndex" `
                     -RangeKeyName "LastPostDateTime" `
                     -RangeKeyDataType "S" `
                     -ProjectionType "keys_only" |
  New-DDBTable -TableName "Thread" -ReadCapacity 10 -WriteCapacity 5
```

Ausgabe:

```
AttributeDefinitions   : {ForumName, LastPostDateTime, Subject}
TableName              : Thread
KeySchema              : {ForumName, Subject}
TableStatus            : CREATING
CreationDateTime       : 10/28/2013 4:39:49 PM
ProvisionedThroughput  : Amazon.DynamoDBv2.Model.ProvisionedThroughputDescription
TableSizeBytes         : 0
ItemCount              : 0
LocalSecondaryIndexes  : {LastPostIndex}
```

- Einzelheiten zur API finden Sie unter CreateTable AWS -Tools für PowerShellCmdlet-Referenz (V5).

Python

SDK für Python (Boto3)

> **Note**
>
> Es gibt noch mehr dazu. GitHub Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

Erstellen Sie eine Tabelle zum Speichern von Filmdaten.

```
class Movies:
    """Encapsulates an Amazon DynamoDB table of movie data.

    Example data structure for a movie record in this table:
        {
            "year": 1999,
            "title": "For Love of the Game",
            "info": {
                "directors": ["Sam Raimi"],
                "release_date": "1999-09-15T00:00:00Z",
                "rating": 6.3,
                "plot": "A washed up pitcher flashes through his career.",
                "rank": 4987,
                "running_time_secs": 8220,
                "actors": [
                    "Kevin Costner",
                    "Kelly Preston",
                    "John C. Reilly"
                ]
            }
        }
    """

    def __init__(self, dyn_resource):
        """
        :param dyn_resource: A Boto3 DynamoDB resource.
        """
        self.dyn_resource = dyn_resource
        # The table variable is set during the scenario in the call to
        # 'exists' if the table exists. Otherwise, it is set by 'create_table'.
        self.table = None


    def create_table(self, table_name):
        """
        Creates an Amazon DynamoDB table that can be used to store movie data.
        The table uses the release year of the movie as the partition key and the
        title as the sort key.

        :param table_name: The name of the table to create.
        :return: The newly created table.
        """
```

```
        try:
            self.table = self.dyn_resource.create_table(
                TableName=table_name,
                KeySchema=[
                    {"AttributeName": "year", "KeyType": "HASH"},  # Partition
key
                    {"AttributeName": "title", "KeyType": "RANGE"},  # Sort key
                ],
                AttributeDefinitions=[
                    {"AttributeName": "year", "AttributeType": "N"},
                    {"AttributeName": "title", "AttributeType": "S"},
                ],
                BillingMode='PAY_PER_REQUEST',
            )
            self.table.wait_until_exists()
        except ClientError as err:
            logger.error(
                "Couldn't create table %s. Here's why: %s: %s",
                table_name,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return self.table
```

- Einzelheiten zur API finden Sie CreateTablein AWS SDK for Python (Boto3) API Reference.

Ruby

SDK für Ruby

> **Note**
>
> Es gibt noch mehr dazu. GitHub Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
# Encapsulates an Amazon DynamoDB table of movie data.
class Scaffold
```

```
  attr_reader :dynamo_resource, :table_name, :table

  def initialize(table_name)
    client = Aws::DynamoDB::Client.new(region: 'us-east-1')
    @dynamo_resource = Aws::DynamoDB::Resource.new(client: client)
    @table_name = table_name
    @table = nil
    @logger = Logger.new($stdout)
    @logger.level = Logger::DEBUG
  end

  # Creates an Amazon DynamoDB table that can be used to store movie data.
  # The table uses the release year of the movie as the partition key and the
  # title as the sort key.
  #
  # @param table_name [String] The name of the table to create.
  # @return [Aws::DynamoDB::Table] The newly created table.
  def create_table(table_name)
    @table = @dynamo_resource.create_table(
      table_name: table_name,
      key_schema: [
        { attribute_name: 'year', key_type: 'HASH' }, # Partition key
        { attribute_name: 'title', key_type: 'RANGE' } # Sort key
      ],
      attribute_definitions: [
        { attribute_name: 'year', attribute_type: 'N' },
        { attribute_name: 'title', attribute_type: 'S' }
      ],
      billing_mode: 'PAY_PER_REQUEST'
    )
    @dynamo_resource.client.wait_until(:table_exists, table_name: table_name)
    @table
  rescue Aws::DynamoDB::Errors::ServiceError => e
    @logger.error("Failed create table #{table_name}:\n#{e.code}: #{e.message}")
    raise
  end
```

- Einzelheiten zur API finden Sie CreateTablein der AWS SDK für Ruby API-Referenz.

Rust

SDK für Rust

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
pub async fn create_table(
    client: &Client,
    table: &str,
    key: &str,
) -> Result<CreateTableOutput, Error> {
    let a_name: String = key.into();
    let table_name: String = table.into();

    let ad = AttributeDefinition::builder()
        .attribute_name(&a_name)
        .attribute_type(ScalarAttributeType::S)
        .build()
        .map_err(Error::BuildError)?;

    let ks = KeySchemaElement::builder()
        .attribute_name(&a_name)
        .key_type(KeyType::Hash)
        .build()
        .map_err(Error::BuildError)?;

    let create_table_response = client
        .create_table()
        .table_name(table_name)
        .key_schema(ks)
        .attribute_definitions(ad)
        .billing_mode(BillingMode::PayPerRequest)
        .send()
        .await;

    match create_table_response {
        Ok(out) => {
            println!("Added table {} with key {}", table, key);
```

```
            Ok(out)
        }
        Err(e) => {
            eprintln!("Got an error creating table:");
            eprintln!("{}", e);
            Err(Error::unhandled(e))
        }
    }
}
```

- Einzelheiten zur API finden Sie CreateTablein der API-Referenz zum AWS SDK für Rust.

SAP ABAP

SDK für SAP ABAP

> **Note**
>
> Es gibt noch mehr dazu. GitHub Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
    TRY.
        DATA(lt_keyschema) = VALUE /aws1/cl_dynkeyschemaelement=>tt_keyschema(
          ( NEW /aws1/cl_dynkeyschemaelement( iv_attributename = 'year'
                                              iv_keytype = 'HASH' ) )
          ( NEW /aws1/cl_dynkeyschemaelement( iv_attributename = 'title'
                                              iv_keytype = 'RANGE' ) ) ).
        DATA(lt_attributedefinitions) = VALUE /aws1/
cl_dynattributedefn=>tt_attributedefinitions(
          ( NEW /aws1/cl_dynattributedefn( iv_attributename = 'year'
                                           iv_attributetype = 'N' ) )
          ( NEW /aws1/cl_dynattributedefn( iv_attributename = 'title'
                                           iv_attributetype = 'S' ) ) ).

        " Adjust read/write capacities as desired.
        DATA(lo_dynprovthroughput)  = NEW /aws1/cl_dynprovthroughput(
          iv_readcapacityunits = 5
          iv_writecapacityunits = 5 ).
        oo_result = lo_dyn->createtable(
          it_keyschema = lt_keyschema
```

```
          iv_tablename = iv_table_name
          it_attributedefinitions = lt_attributedefinitions
          io_provisionedthroughput = lo_dynprovthroughput ).
        " Table creation can take some time. Wait till table exists before
returning.
        lo_dyn->get_waiter( )->tableexists(
          iv_max_wait_time = 200
          iv_tablename     = iv_table_name ).
        MESSAGE 'DynamoDB Table' && iv_table_name && 'created.' TYPE 'I'.
        " This exception can happen if the table already exists.
      CATCH /aws1/cx_dynresourceinuseex INTO DATA(lo_resourceinuseex).
        DATA(lv_error) = |"{ lo_resourceinuseex->av_err_code }" -
{ lo_resourceinuseex->av_err_msg }|.
        MESSAGE lv_error TYPE 'E'.
    ENDTRY.
```

- Einzelheiten zur API finden Sie CreateTablein der API-Referenz zum AWS SDK für SAP ABAP.

Swift

SDK für Swift

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
import AWSDynamoDB


    ///
    /// Create a movie table in the Amazon DynamoDB data store.
    ///
    private func createTable() async throws {
        do {
            guard let client = self.ddbClient else {
                throw MoviesError.UninitializedClient
            }
```

```
            let input = CreateTableInput(
                attributeDefinitions: [
                    DynamoDBClientTypes.AttributeDefinition(attributeName:
"year", attributeType: .n),
                    DynamoDBClientTypes.AttributeDefinition(attributeName:
"title", attributeType: .s)
                ],
                billingMode: DynamoDBClientTypes.BillingMode.payPerRequest,
                keySchema: [
                    DynamoDBClientTypes.KeySchemaElement(attributeName: "year",
keyType: .hash),
                    DynamoDBClientTypes.KeySchemaElement(attributeName: "title",
keyType: .range)
                ],
                tableName: self.tableName
            )
            let output = try await client.createTable(input: input)
            if output.tableDescription == nil {
                throw MoviesError.TableNotFound
            }
        } catch {
            print("ERROR: createTable:", dump(error))
            throw error
        }
    }
```

- Einzelheiten zur API finden Sie CreateTablein der API-Referenz zum AWS SDK für Swift.

Weitere DynamoDB-Beispiele finden Sie unter Codebeispiele für DynamoDB mit AWS SDKs.

Fahren Sie nach dem Erstellen der neuen Tabelle mit Schritt 2: Schreiben von Daten in eine DynamoDB-Tabelle fort.

Schritt 2: Schreiben von Daten in eine DynamoDB-Tabelle

In diesem Schritt fügen Sie mehrere Elemente in die Tabelle `Music` ein, die Sie in Schritt 1: Erstellen einer Tabelle in DynamoDB erstellt haben.

Weitere Informationen über Schreibvorgänge finden Sie unter Schreiben eines Elements.

AWS-Managementkonsole

Gehen Sie wie folgt vor, um mithilfe der DynamoDB-Konsole Daten in die `Music`-Tabelle zu schreiben.

1. Öffnen Sie die DynamoDB-Konsole unter. https://console.aws.amazon.com/dynamodb/
2. Wählen Sie im linken Navigationsbereich Tables (Tabellen) aus.
3. Wählen Sie auf der Seite Tabellen die Tabelle Musik aus.
4. Wählen Sie Tabellenelemente erkunden aus.
5. Wählen Sie im Abschnitt Zurückgegebene Elemente die Option Element erstellen aus.
6. Gehen Sie auf der Seite Element erstellen wie folgt vor, um Elemente zur Tabelle hinzuzufügen:
    a. Klicken Sie auf Neues Attribut hinzufügen und wählen Sie dann Zahl aus.
    b. Geben Sie als Attributnamen den Wert **`Awards`** ein.
    c. Wiederholen Sie diesen Vorgang, um ein **`AlbumTitle`** vom Typ String zu erstellen.
    d. Geben Sie die folgenden Werte für Ihr Element ein:
        i. Machen Sie für Artist die Eingabe **`No One You Know`**.
        ii. Geben Sie unter SongTitle den Wert **`Call Me Today`** ein.
        iii. Geben Sie unter AlbumTitle den Wert **`Somewhat Famous`** ein.
        iv. Machen Sie für Awards die Eingabe **`1`**.
7. Wählen Sie Create item (Element erstellen) aus.
8. Wiederholen Sie diesen Vorgang und erstellen Sie ein anderes Element mit den folgenden Werten:
    a. Machen Sie für Artist die Eingabe **`Acme Band`**.
    b. Zum SongTitleEingeben. **`Happy Day`**
    c. Geben Sie unter AlbumTitle den Wert **`Songs About Life`** ein.
    d. Machen Sie für Awards die Eingabe **`10`**.
9. Tun Sie dies noch einmal, um ein anderes Element mit demselben Artist wie im vorherigen Schritt, aber andere Werte für die anderen Attribute zu erstellen:
    a. Machen Sie für Artist die Eingabe **`Acme Band`**.
    b. Zum SongTitleEingeben**`PartiQL Rocks`**.

c. Geben Sie unter AlbumTitle den Wert **`Another Album Title`** ein.

d. Machen Sie für Awards die Eingabe **8**.

AWS CLI

Im folgenden AWS CLI Beispiel werden mehrere neue Elemente in der `Music` Tabelle erstellt. Sie können dies entweder über die DynamoDB-API tun oder PartiQL, eine SQL-kompatible Abfragesprache für DynamoDB.

DynamoDB API

Linux

```
aws dynamodb put-item \
    --table-name Music  \
    --item \
        '{"Artist": {"S": "No One You Know"}, "SongTitle": {"S": "Call Me Today"},
 "AlbumTitle": {"S": "Somewhat Famous"}, "Awards": {"N": "1"}}'

aws dynamodb put-item \
    --table-name Music  \
    --item \
        '{"Artist": {"S": "No One You Know"}, "SongTitle": {"S": "Howdy"},
 "AlbumTitle": {"S": "Somewhat Famous"}, "Awards": {"N": "2"}}'

aws dynamodb put-item \
    --table-name Music \
    --item \
        '{"Artist": {"S": "Acme Band"}, "SongTitle": {"S": "Happy Day"},
 "AlbumTitle": {"S": "Songs About Life"}, "Awards": {"N": "10"}}'

aws dynamodb put-item \
    --table-name Music \
    --item \
        '{"Artist": {"S": "Acme Band"}, "SongTitle": {"S": "PartiQL Rocks"},
 "AlbumTitle": {"S": "Another Album Title"}, "Awards": {"N": "8"}}'
```

Windows CMD

```
aws dynamodb put-item ^
    --table-name Music  ^
```

```
    --item ^
        "{\"Artist\": {\"S\": \"No One You Know\"}, \"SongTitle\": {\"S\": \"Call
 Me Today\"}, \"AlbumTitle\": {\"S\": \"Somewhat Famous\"}, \"Awards\": {\"N\":
 \"1\"}}"

aws dynamodb put-item ^
    --table-name Music  ^
    --item ^
        "{\"Artist\": {\"S\": \"No One You Know\"}, \"SongTitle\": {\"S\": \"Howdy
\"}, \"AlbumTitle\": {\"S\": \"Somewhat Famous\"}, \"Awards\": {\"N\": \"2\"}}"

aws dynamodb put-item ^
    --table-name Music ^
    --item ^
        "{\"Artist\": {\"S\": \"Acme Band\"}, \"SongTitle\": {\"S\": \"Happy Day\"},
 \"AlbumTitle\": {\"S\": \"Songs About Life\"}, \"Awards\": {\"N\": \"10\"}}"

aws dynamodb put-item ^
    --table-name Music ^
    --item ^
        "{\"Artist\": {\"S\": \"Acme Band\"}, \"SongTitle\": {\"S\": \"PartiQL Rocks
\"}, \"AlbumTitle\": {\"S\": \"Another Album Title\"}, \"Awards\": {\"N\": \"8\"}}"
```

PartiQL for DynamoDB

Linux

```
aws dynamodb execute-statement --statement "INSERT INTO Music  \
               VALUE  \
               {'Artist':'No One You Know','SongTitle':'Call Me Today',
 'AlbumTitle':'Somewhat Famous', 'Awards':'1'}"

aws dynamodb execute-statement --statement "INSERT INTO Music  \
               VALUE  \
               {'Artist':'No One You Know','SongTitle':'Howdy',
 'AlbumTitle':'Somewhat Famous', 'Awards':'2'}"

aws dynamodb execute-statement --statement "INSERT INTO Music  \
               VALUE  \
               {'Artist':'Acme Band','SongTitle':'Happy Day', 'AlbumTitle':'Songs
 About Life', 'Awards':'10'}"

aws dynamodb execute-statement --statement "INSERT INTO Music  \
               VALUE  \
```

```
                {'Artist':'Acme Band','SongTitle':'PartiQL Rocks',
 'AlbumTitle':'Another Album Title', 'Awards':'8'}"
```

Windows CMD

```
aws dynamodb execute-statement --statement "INSERT INTO Music VALUE {'Artist':'No
 One You Know','SongTitle':'Call Me Today', 'AlbumTitle':'Somewhat Famous',
 'Awards':'1'}"

aws dynamodb execute-statement --statement "INSERT INTO Music VALUE {'Artist':'No
 One You Know','SongTitle':'Howdy', 'AlbumTitle':'Somewhat Famous', 'Awards':'2'}"

aws dynamodb execute-statement --statement "INSERT INTO Music VALUE {'Artist':'Acme
 Band','SongTitle':'Happy Day', 'AlbumTitle':'Songs About Life', 'Awards':'10'}"

aws dynamodb execute-statement --statement "INSERT INTO Music VALUE {'Artist':'Acme
 Band','SongTitle':'PartiQL Rocks', 'AlbumTitle':'Another Album Title',
 'Awards':'8'}"
```

Für weitere Informationen zum Schreiben von Daten mit PartiQL siehe PartiQL Insert Statements (PartiQL-Insert-Anweisungen).

Weitere Informationen zu unterstützten Datentypen in DynamoDB finden Sie unter Datentypen.

Weitere Informationen zur Darstellung von DynamoDB-Datentypen in JSON finden Sie unter Attributwerte.

AWS SDK

Die folgenden Codebeispiele zeigen, wie ein Element mithilfe eines SDK in eine DynamoDB-Tabelle geschrieben wird. AWS

.NET

SDK für .NET (v4)

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
    /// <summary>
    /// Adds a new item to the table.
    /// </summary>
    /// <param name="newMovie">A Movie object containing informtation for
    /// the movie to add to the table.</param>
    /// <param name="tableName">The name of the table where the item will be
 added.</param>
    /// <returns>A Boolean value that indicates the results of adding the item.</
returns>
    public async Task<bool> PutItemAsync(Movie newMovie, string tableName)
    {
        try
        {
            var item = new Dictionary<string, AttributeValue>
            {
                ["title"] = new AttributeValue { S = newMovie.Title },
                ["year"] = new AttributeValue { N = newMovie.Year.ToString() },
            };

            var request = new PutItemRequest
            {
                TableName = tableName,
                Item = item,
            };

            await _amazonDynamoDB.PutItemAsync(request);
            return true;
        }
        catch (ResourceNotFoundException ex)
        {
            Console.WriteLine($"Table {tableName} was not found. {ex.Message}");
            return false;
        }
        catch (AmazonDynamoDBException ex)
        {
            Console.WriteLine($"An Amazon DynamoDB error occurred while putting
 item. {ex.Message}");
            throw;
        }
        catch (Exception ex)
        {
```

```
            Console.WriteLine($"An error occurred while putting item.
 {ex.Message}");
            throw;
        }
    }
```

- Einzelheiten zur API finden Sie PutItemin der AWS SDK für .NET API-Referenz.

Bash

AWS CLI mit Bash-Skript

> **Note**
>
> Es gibt noch mehr dazu. GitHub Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
##############################################################################
# function dynamodb_put_item
#
# This function puts an item into a DynamoDB table.
#
# Parameters:
#       -n table_name  -- The name of the table.
#       -i item  -- Path to json file containing the item values.
#
#  Returns:
#       0 - If successful.
#       1 - If it fails.
##############################################################################
function dynamodb_put_item() {
  local table_name item response
  local option OPTARG # Required to use getopts command in a function.

  ######################################
  # Function usage explanation
  ######################################
  function usage() {
    echo "function dynamodb_put_item"
```

```
  echo "Put an item into a DynamoDB table."
  echo " -n table_name  -- The name of the table."
  echo " -i item  -- Path to json file containing the item values."
  echo ""
}

while getopts "n:i:h" option; do
  case "${option}" in
    n) table_name="${OPTARG}" ;;
    i) item="${OPTARG}" ;;
    h)
      usage
      return 0
      ;;
    \?)
      echo "Invalid parameter"
      usage
      return 1
      ;;
  esac
done
export OPTIND=1

if [[ -z "$table_name" ]]; then
  errecho "ERROR: You must provide a table name with the -n parameter."
  usage
  return 1
fi

if [[ -z "$item" ]]; then
  errecho "ERROR: You must provide an item with the -i parameter."
  usage
  return 1
fi

iecho "Parameters:\n"
iecho "    table_name:   $table_name"
iecho "    item:   $item"
iecho ""
iecho ""

response=$(aws dynamodb put-item \
  --table-name "$table_name" \
  --item file://"$item")
```

```
  local error_code=${?}

  if [[ $error_code -ne 0 ]]; then
    aws_cli_error_log $error_code
    errecho "ERROR: AWS reports put-item operation failed.$response"
    return 1
  fi

  return 0

}
```

Die in diesem Beispiel verwendeten Hilfsfunktionen.

```
################################################################################
# function iecho
#
# This function enables the script to display the specified text only if
# the global variable $VERBOSE is set to true.
################################################################################
function iecho() {
  if [[ $VERBOSE == true ]]; then
    echo "$@"
  fi
}

################################################################################
# function errecho
#
# This function outputs everything sent to it to STDERR (standard error output).
################################################################################
function errecho() {
  printf "%s\n" "$*" 1>&2
}

################################################################################
# function aws_cli_error_log()
#
# This function is used to log the error messages from the AWS CLI.
#
```

```
# See https://docs.aws.amazon.com/cli/latest/topic/return-codes.html#cli-aws-
help-return-codes.
#
# The function expects the following argument:
#         $1 - The error code returned by the AWS CLI.
#
#  Returns:
#          0: - Success.
#
################################################################################
function aws_cli_error_log() {
  local err_code=$1
  errecho "Error code : $err_code"
  if [ "$err_code" == 1 ]; then
    errecho "  One or more S3 transfers failed."
  elif [ "$err_code" == 2 ]; then
    errecho "  Command line failed to parse."
  elif [ "$err_code" == 130 ]; then
    errecho "  Process received SIGINT."
  elif [ "$err_code" == 252 ]; then
    errecho "  Command syntax invalid."
  elif [ "$err_code" == 253 ]; then
    errecho "  The system environment or configuration was invalid."
  elif [ "$err_code" == 254 ]; then
    errecho "  The service returned an error."
  elif [ "$err_code" == 255 ]; then
    errecho "  255 is a catch-all error."
  fi

  return 0
}
```

- Einzelheiten zur API finden Sie PutItemin der AWS CLI Befehlsreferenz.

C++

SDK für C++

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
//! Put an item in an Amazon DynamoDB table.
/*!
  \sa putItem()
  \param tableName: The table name.
  \param artistKey: The artist key. This is the partition key for the table.
  \param artistValue: The artist value.
  \param albumTitleKey: The album title key.
  \param albumTitleValue: The album title value.
  \param awardsKey: The awards key.
  \param awardsValue: The awards value.
  \param songTitleKey: The song title key.
  \param songTitleValue: The song title value.
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
 */
bool AwsDoc::DynamoDB::putItem(const Aws::String &tableName,
                               const Aws::String &artistKey,
                               const Aws::String &artistValue,
                               const Aws::String &albumTitleKey,
                               const Aws::String &albumTitleValue,
                               const Aws::String &awardsKey,
                               const Aws::String &awardsValue,
                               const Aws::String &songTitleKey,
                               const Aws::String &songTitleValue,
                               const Aws::Client::ClientConfiguration
 &clientConfiguration) {
    Aws::DynamoDB::DynamoDBClient dynamoClient(clientConfiguration);

    Aws::DynamoDB::Model::PutItemRequest putItemRequest;
    putItemRequest.SetTableName(tableName);
```

```
    putItemRequest.AddItem(artistKey,
 Aws::DynamoDB::Model::AttributeValue().SetS(
            artistValue)); // This is the hash key.
    putItemRequest.AddItem(albumTitleKey,
 Aws::DynamoDB::Model::AttributeValue().SetS(
            albumTitleValue));
    putItemRequest.AddItem(awardsKey,

 Aws::DynamoDB::Model::AttributeValue().SetS(awardsValue));
    putItemRequest.AddItem(songTitleKey,

 Aws::DynamoDB::Model::AttributeValue().SetS(songTitleValue));

    const Aws::DynamoDB::Model::PutItemOutcome outcome = dynamoClient.PutItem(
            putItemRequest);
    if (outcome.IsSuccess()) {
        std::cout << "Successfully added Item!" << std::endl;
    }
    else {
        std::cerr << outcome.GetError().GetMessage() << std::endl;
        return false;
    }

    return waitTableActive(tableName, dynamoClient);
}
```

Code, der darauf wartet, dass die Tabelle aktiv wird.

```
//! Query a newly created DynamoDB table until it is active.
/*!
  \sa waitTableActive()
  \param waitTableActive: The DynamoDB table's name.
  \param dynamoClient: A DynamoDB client.
  \return bool: Function succeeded.
*/
bool AwsDoc::DynamoDB::waitTableActive(const Aws::String &tableName,
                                       const Aws::DynamoDB::DynamoDBClient
 &dynamoClient) {

    // Repeatedly call DescribeTable until table is ACTIVE.
    const int MAX_QUERIES = 20;
```

```
    Aws::DynamoDB::Model::DescribeTableRequest request;
    request.SetTableName(tableName);

    int count = 0;
    while (count < MAX_QUERIES) {
        const Aws::DynamoDB::Model::DescribeTableOutcome &result =
 dynamoClient.DescribeTable(
                request);
        if (result.IsSuccess()) {
            Aws::DynamoDB::Model::TableStatus status =
 result.GetResult().GetTable().GetTableStatus();

            if (Aws::DynamoDB::Model::TableStatus::ACTIVE != status) {
                std::this_thread::sleep_for(std::chrono::seconds(1));
            }
            else {
                return true;
            }
        }
        else {
            std::cerr << "Error DynamoDB::waitTableActive "
                      << result.GetError().GetMessage() << std::endl;
            return false;
        }
        count++;
    }
    return false;
}
```

- Einzelheiten zur API finden Sie PutItemin der AWS SDK für C++ API-Referenz.

CLI

AWS CLI

Beispiel 1: So fügen Sie ein Element zu einer Tabelle hinzu

Das folgende `put-item` Beispiel fügt der MusicCollectionTabelle ein neues Element hinzu.

```
aws dynamodb put-item \
    --table-name MusicCollection \
    --item file://item.json \
```

```
    --return-consumed-capacity TOTAL \
    --return-item-collection-metrics SIZE
```

Inhalt von `item.json`:

```
{
    "Artist": {"S": "No One You Know"},
    "SongTitle": {"S": "Call Me Today"},
    "AlbumTitle": {"S": "Greatest Hits"}
}
```

Ausgabe:

```
{
    "ConsumedCapacity": {
        "TableName": "MusicCollection",
        "CapacityUnits": 1.0
    },
    "ItemCollectionMetrics": {
        "ItemCollectionKey": {
            "Artist": {
                "S": "No One You Know"
            }
        },
        "SizeEstimateRangeGB": [
            0.0,
            1.0
        ]
    }
}
```

Weitere Informationen finden Sie unter Schreiben eines Elements im Entwicklerhandbuch zu Amazon DynamoDB.

Beispiel 2: So überschreiben Sie ein Element in einer Tabelle bedingungsabhängig

Im folgenden Beispiel für `put-item` wird ein vorhandenes Element in der `MusicCollection`-Tabelle nur dann überschrieben, wenn dieses vorhandene Element ein `AlbumTitle`-Attribut mit dem Wert `Greatest Hits` hat. Der Befehl gibt den vorherigen Wert des Elements zurück.

```
aws dynamodb put-item \
```

```
    --table-name MusicCollection \
    --item file://item.json \
    --condition-expression "#A = :A" \
    --expression-attribute-names file://names.json \
    --expression-attribute-values file://values.json \
    --return-values ALL_OLD
```

Inhalt von `item.json`:

```
{
    "Artist": {"S": "No One You Know"},
    "SongTitle": {"S": "Call Me Today"},
    "AlbumTitle": {"S": "Somewhat Famous"}
}
```

Inhalt von `names.json`:

```
{
    "#A": "AlbumTitle"
}
```

Inhalt von `values.json`:

```
{
    ":A": {"S": "Greatest Hits"}
}
```

Ausgabe:

```
{
    "Attributes": {
        "AlbumTitle": {
            "S": "Greatest Hits"
        },
        "Artist": {
            "S": "No One You Know"
        },
        "SongTitle": {
            "S": "Call Me Today"
        }
    }
```

```
}
```

Wenn der Schlüssel bereits existiert, sollten Sie die folgende Ausgabe sehen:

```
A client error (ConditionalCheckFailedException) occurred when calling the
 PutItem operation: The conditional request failed.
```

Weitere Informationen finden Sie unter Schreiben eines Elements im Entwicklerhandbuch zu Amazon DynamoDB.

- Einzelheiten zur API finden Sie PutItemunter AWS CLI Befehlsreferenz.

Go

SDK für Go V2

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
import (
 "context"
 "errors"
 "log"
 "time"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/attributevalue"
 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/expression"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb/types"
)

// TableBasics encapsulates the Amazon DynamoDB service actions used in the
 examples.
// It contains a DynamoDB service client that is used to act on the specified
 table.
type TableBasics struct {
```

```
 DynamoDbClient *dynamodb.Client
 TableName      string
}



// AddMovie adds a movie the DynamoDB table.
func (basics TableBasics) AddMovie(ctx context.Context, movie Movie) error {
 item, err := attributevalue.MarshalMap(movie)
 if err != nil {
  panic(err)
 }
 _, err = basics.DynamoDbClient.PutItem(ctx, &dynamodb.PutItemInput{
  TableName: aws.String(basics.TableName), Item: item,
 })
 if err != nil {
  log.Printf("Couldn't add item to table. Here's why: %v\n", err)
 }
 return err
}
```

Definieren Sie eine Movie-Struktur, die in diesem Beispiel verwendet wird.

```
import (
 "archive/zip"
 "bytes"
 "encoding/json"
 "fmt"
 "io"
 "log"
 "net/http"

 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/attributevalue"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb/types"
)

// Movie encapsulates data about a movie. Title and Year are the composite
 primary key
// of the movie in Amazon DynamoDB. Title is the sort key, Year is the partition
 key,
```

```
// and Info is additional data.
type Movie struct {
 Title string                `dynamodbav:"title"`
 Year  int                   `dynamodbav:"year"`
 Info  map[string]interface{} `dynamodbav:"info"`
}

// GetKey returns the composite primary key of the movie in a format that can be
// sent to DynamoDB.
func (movie Movie) GetKey() map[string]types.AttributeValue {
 title, err := attributevalue.Marshal(movie.Title)
 if err != nil {
  panic(err)
 }
 year, err := attributevalue.Marshal(movie.Year)
 if err != nil {
  panic(err)
 }
 return map[string]types.AttributeValue{"title": title, "year": year}
}

// String returns the title, year, rating, and plot of a movie, formatted for the
 example.
func (movie Movie) String() string {
 return fmt.Sprintf("%v\n\tReleased: %v\n\tRating: %v\n\tPlot: %v\n",
  movie.Title, movie.Year, movie.Info["rating"], movie.Info["plot"])
}
```

- Einzelheiten zur API finden Sie PutItemin der AWS SDK für Go API-Referenz.

Java

SDK für Java 2.x

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

Fügt ein Element in eine Tabelle ein mit DynamoDbClient.

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.AttributeValue;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.model.PutItemRequest;
import software.amazon.awssdk.services.dynamodb.model.PutItemResponse;
import software.amazon.awssdk.services.dynamodb.model.ResourceNotFoundException;
import java.util.HashMap;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 *
 * To place items into an Amazon DynamoDB table using the AWS SDK for Java V2,
 * its better practice to use the
 * Enhanced Client. See the EnhancedPutItem example.
 */
public class PutItem {
    public static void main(String[] args) {
        final String usage = """

                Usage:
                    <tableName> <key> <keyVal> <albumtitle> <albumtitleval>
 <awards> <awardsval> <Songtitle> <songtitleval>

                Where:
                    tableName - The Amazon DynamoDB table in which an item is
 placed (for example, Music3).
                    key - The key used in the Amazon DynamoDB table (for example,
 Artist).
                    keyval - The key value that represents the item to get (for
 example, Famous Band).
                    albumTitle - The Album title (for example, AlbumTitle).
                    AlbumTitleValue - The name of the album (for example, Songs
 About Life ).
                    Awards - The awards column (for example, Awards).
                    AwardVal - The value of the awards (for example, 10).
```

```
                    SongTitle - The song title (for example, SongTitle).
                    SongTitleVal - The value of the song title (for example,
Happy Day).
                **Warning** This program will  place an item that you specify
into a table!
                """;

        if (args.length != 9) {
            System.out.println(usage);
            System.exit(1);
        }

        String tableName = args[0];
        String key = args[1];
        String keyVal = args[2];
        String albumTitle = args[3];
        String albumTitleValue = args[4];
        String awards = args[5];
        String awardVal = args[6];
        String songTitle = args[7];
        String songTitleVal = args[8];

        Region region = Region.US_EAST_1;
        DynamoDbClient ddb = DynamoDbClient.builder()
                .region(region)
                .build();

        putItemInTable(ddb, tableName, key, keyVal, albumTitle, albumTitleValue,
awards, awardVal, songTitle,
                songTitleVal);
        System.out.println("Done!");
        ddb.close();
    }

    public static void putItemInTable(DynamoDbClient ddb,
            String tableName,
            String key,
            String keyVal,
            String albumTitle,
            String albumTitleValue,
            String awards,
            String awardVal,
            String songTitle,
            String songTitleVal) {
```

```
        HashMap<String, AttributeValue> itemValues = new HashMap<>();
        itemValues.put(key, AttributeValue.builder().s(keyVal).build());
        itemValues.put(songTitle,
 AttributeValue.builder().s(songTitleVal).build());
        itemValues.put(albumTitle,
 AttributeValue.builder().s(albumTitleValue).build());
        itemValues.put(awards, AttributeValue.builder().s(awardVal).build());

        PutItemRequest request = PutItemRequest.builder()
                .tableName(tableName)
                .item(itemValues)
                .build();

        try {
            PutItemResponse response = ddb.putItem(request);
            System.out.println(tableName + " was successfully updated. The
 request id is "
                    + response.responseMetadata().requestId());

        } catch (ResourceNotFoundException e) {
            System.err.format("Error: The Amazon DynamoDB table \"%s\" can't be
 found.\n", tableName);
            System.err.println("Be sure that it exists and that you've typed its
 name correctly!");
            System.exit(1);
        } catch (DynamoDbException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
    }
}
```

- Einzelheiten zur API finden Sie PutItemunter AWS SDK for Java 2.x API-Referenz.

JavaScript

SDK für JavaScript (v3)

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

In diesem Beispiel wird der Dokument-Client verwendet, um die Arbeit mit Elementen in DynamoDB zu vereinfachen. Einzelheiten zur API finden Sie unter PutCommand.

```
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";
import { PutCommand, DynamoDBDocumentClient } from "@aws-sdk/lib-dynamodb";

const client = new DynamoDBClient({});
const docClient = DynamoDBDocumentClient.from(client);

export const main = async () => {
  const command = new PutCommand({
    TableName: "HappyAnimals",
    Item: {
      CommonName: "Shiba Inu",
    },
  });

  const response = await docClient.send(command);
  console.log(response);
  return response;
};
```

- Einzelheiten zur API finden Sie PutItemunter AWS SDK für JavaScript API-Referenz.

SDK für JavaScript (v2)

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

Fügen Sie ein Element in eine Tabelle ein.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create the DynamoDB service object
var ddb = new AWS.DynamoDB({ apiVersion: "2012-08-10" });

var params = {
  TableName: "CUSTOMER_LIST",
  Item: {
    CUSTOMER_ID: { N: "001" },
    CUSTOMER_NAME: { S: "Richard Roe" },
  },
};

// Call DynamoDB to add the item to the table
ddb.putItem(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data);
  }
});
```

Fügen Sie ein Element mithilfe des DynamoDB-Dokument-Clients in eine Tabelle ein.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create DynamoDB document client
var docClient = new AWS.DynamoDB.DocumentClient({ apiVersion: "2012-08-10" });

var params = {
  TableName: "TABLE",
  Item: {
    HASHKEY: VALUE,
    ATTRIBUTE_1: "STRING_VALUE",
```

```
    ATTRIBUTE_2: VALUE_2,
  },
};

docClient.put(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data);
  }
});
```

- Weitere Informationen finden Sie im AWS SDK für JavaScript -Entwicklerhandbuch.
- Einzelheiten zur API finden Sie PutItemin der AWS SDK für JavaScript API-Referenz.

Kotlin

SDK für Kotlin

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
suspend fun putItemInTable(
    tableNameVal: String,
    key: String,
    keyVal: String,
    albumTitle: String,
    albumTitleValue: String,
    awards: String,
    awardVal: String,
    songTitle: String,
    songTitleVal: String,
) {
    val itemValues = mutableMapOf<String, AttributeValue>()

    // Add all content to the table.
    itemValues[key] = AttributeValue.S(keyVal)
```

```
    itemValues[songTitle] = AttributeValue.S(songTitleVal)
    itemValues[albumTitle] = AttributeValue.S(albumTitleValue)
    itemValues[awards] = AttributeValue.S(awardVal)

    val request =
        PutItemRequest {
            tableName = tableNameVal
            item = itemValues
        }

    DynamoDbClient.fromEnvironment { region = "us-east-1" }.use { ddb ->
        ddb.putItem(request)
        println(" A new item was placed into $tableNameVal.")
    }
}
```

- API-Details finden Sie PutItemin der API-Referenz zum AWS SDK für Kotlin.

PHP

SDK für PHP

> **Note**
>
> Es gibt noch mehr dazu. GitHub Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
        echo "What's the name of the last movie you watched?\n";
        while (empty($movieName)) {
            $movieName = testable_readline("Movie name: ");
        }
        echo "And what year was it released?\n";
        $movieYear = "year";
        while (!is_numeric($movieYear) || intval($movieYear) != $movieYear) {
            $movieYear = testable_readline("Year released: ");
        }

        $service->putItem([
            'Item' => [
                'year' => [
```

```
                    'N' => "$movieYear",
                ],
                'title' => [
                    'S' => $movieName,
                ],
            ],
            'TableName' => $tableName,
        ]);

    public function putItem(array $array)
    {
        $this->dynamoDbClient->putItem($array);
    }
```

- Einzelheiten zur API finden Sie PutItemin der AWS SDK für PHP API-Referenz.

PowerShell

Tools für PowerShell V4

Beispiel 1: Erstellt ein neues Element oder ersetzt ein vorhandenes Element durch ein neues.

```
$item = @{
  SongTitle = 'Somewhere Down The Road'
  Artist = 'No One You Know'
        AlbumTitle = 'Somewhat Famous'
        Price = 1.94
        Genre = 'Country'
        CriticRating = 9.0
} | ConvertTo-DDBItem
Set-DDBItem -TableName 'Music' -Item $item
```

- Einzelheiten zur API finden Sie unter PutItem AWS -Tools für PowerShellCmdlet-Referenz (V4).

Tools für V5 PowerShell

Beispiel 1: Erstellt ein neues Element oder ersetzt ein vorhandenes Element durch ein neues.

```
$item = @{
  SongTitle = 'Somewhere Down The Road'
  Artist = 'No One You Know'
```

```
        AlbumTitle = 'Somewhat Famous'
        Price = 1.94
        Genre = 'Country'
        CriticRating = 9.0
} | ConvertTo-DDBItem
Set-DDBItem -TableName 'Music' -Item $item
```

- Einzelheiten zur API finden Sie unter PutItem AWS -Tools für PowerShellCmdlet-Referenz (V5).

Python

SDK für Python (Boto3)

> **Note**
>
> Es gibt noch mehr dazu. GitHub Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
class Movies:
    """Encapsulates an Amazon DynamoDB table of movie data.

    Example data structure for a movie record in this table:
        {
            "year": 1999,
            "title": "For Love of the Game",
            "info": {
                "directors": ["Sam Raimi"],
                "release_date": "1999-09-15T00:00:00Z",
                "rating": 6.3,
                "plot": "A washed up pitcher flashes through his career.",
                "rank": 4987,
                "running_time_secs": 8220,
                "actors": [
                    "Kevin Costner",
                    "Kelly Preston",
                    "John C. Reilly"
                ]
            }
        }
```

```
    """

    def __init__(self, dyn_resource):
        """
        :param dyn_resource: A Boto3 DynamoDB resource.
        """
        self.dyn_resource = dyn_resource
        # The table variable is set during the scenario in the call to
        # 'exists' if the table exists. Otherwise, it is set by 'create_table'.
        self.table = None


    def add_movie(self, title, year, plot, rating):
        """
        Adds a movie to the table.

        :param title: The title of the movie.
        :param year: The release year of the movie.
        :param plot: The plot summary of the movie.
        :param rating: The quality rating of the movie.
        """
        try:
            self.table.put_item(
                Item={
                    "year": year,
                    "title": title,
                    "info": {"plot": plot, "rating": Decimal(str(rating))},
                }
            )
        except ClientError as err:
            logger.error(
                "Couldn't add movie %s to table %s. Here's why: %s: %s",
                title,
                self.table.name,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
```

- Einzelheiten zur API finden Sie PutItemin AWS SDK for Python (Boto3) API Reference.

Ruby

SDK für Ruby

> **Note**
>
> Es gibt noch mehr dazu. GitHub Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
class DynamoDBBasics
  attr_reader :dynamo_resource, :table

  def initialize(table_name)
    client = Aws::DynamoDB::Client.new(region: 'us-east-1')
    @dynamo_resource = Aws::DynamoDB::Resource.new(client: client)
    @table = @dynamo_resource.table(table_name)
  end

  # Adds a movie to the table.
  #
  # @param movie [Hash] The title, year, plot, and rating of the movie.
  def add_item(movie)
    @table.put_item(
      item: {
        'year' => movie[:year],
        'title' => movie[:title],
        'info' => { 'plot' => movie[:plot], 'rating' => movie[:rating] }
      }
    )
  rescue Aws::DynamoDB::Errors::ServiceError => e
    puts("Couldn't add movie #{title} to table #{@table.name}. Here's why:")
    puts("\t#{e.code}: #{e.message}")
    raise
  end
```

- Einzelheiten zur API finden Sie PutItemin der AWS SDK für Ruby API-Referenz.

Rust

SDK für Rust

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
pub async fn add_item(client: &Client, item: Item, table: &String) ->
 Result<ItemOut, Error> {
    let user_av = AttributeValue::S(item.username);
    let type_av = AttributeValue::S(item.p_type);
    let age_av = AttributeValue::S(item.age);
    let first_av = AttributeValue::S(item.first);
    let last_av = AttributeValue::S(item.last);

    let request = client
        .put_item()
        .table_name(table)
        .item("username", user_av)
        .item("account_type", type_av)
        .item("age", age_av)
        .item("first_name", first_av)
        .item("last_name", last_av);

    println!("Executing request [{request:?}] to add item...");

    let resp = request.send().await?;

    let attributes = resp.attributes().unwrap();

    let username = attributes.get("username").cloned();
    let first_name = attributes.get("first_name").cloned();
    let last_name = attributes.get("last_name").cloned();
    let age = attributes.get("age").cloned();
    let p_type = attributes.get("p_type").cloned();

    println!(
        "Added user {:?}, {:?} {:?}, age {:?} as {:?} user",
        username, first_name, last_name, age, p_type
```

```
    );

    Ok(ItemOut {
        p_type,
        age,
        username,
        first_name,
        last_name,
    })
}
```

- Einzelheiten zur API finden Sie PutItemin der API-Referenz zum AWS SDK für Rust.

SAP ABAP

SDK für SAP ABAP

> **Note**
>
> Es gibt noch mehr dazu. GitHub Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
    TRY.
        DATA(lo_resp) = lo_dyn->putitem(
          iv_tablename = iv_table_name
          it_item      = it_item ).
        MESSAGE '1 row inserted into DynamoDB Table' && iv_table_name TYPE 'I'.
      CATCH /aws1/cx_dyncondalcheckfaile00.
        MESSAGE 'A condition specified in the operation could not be evaluated.'
TYPE 'E'.
      CATCH /aws1/cx_dynresourcenotfoundex.
        MESSAGE 'The table or index does not exist' TYPE 'E'.
      CATCH /aws1/cx_dyntransactconflictex.
        MESSAGE 'Another transaction is using the item' TYPE 'E'.
    ENDTRY.
```

- Einzelheiten zur API finden Sie PutItemin der API-Referenz zum AWS SDK für SAP ABAP.

Swift

SDK für Swift

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
import AWSDynamoDB


    /// Add a movie specified as a `Movie` structure to the Amazon DynamoDB
    /// table.
    ///
    /// - Parameter movie: The `Movie` to add to the table.
    ///
    func add(movie: Movie) async throws {
        do {
            guard let client = self.ddbClient else {
                throw MoviesError.UninitializedClient
            }

            // Get a DynamoDB item containing the movie data.
            let item = try await movie.getAsItem()

            // Send the `PutItem` request to Amazon DynamoDB.

            let input = PutItemInput(
                item: item,
                tableName: self.tableName
            )
            _ = try await client.putItem(input: input)
        } catch {
            print("ERROR: add movie:", dump(error))
            throw error
        }
    }


    ///
```

```
    /// Return an array mapping attribute names to Amazon DynamoDB attribute
    /// values, representing the contents of the `Movie` record as a DynamoDB
    /// item.
    ///
    /// - Returns: The movie item as an array of type
    ///   `[Swift.String:DynamoDBClientTypes.AttributeValue]`.
    ///
    func getAsItem() async throws ->
[Swift.String:DynamoDBClientTypes.AttributeValue]  {
        // Build the item record, starting with the year and title, which are
        // always present.

        var item: [Swift.String:DynamoDBClientTypes.AttributeValue] = [
            "year": .n(String(self.year)),
            "title": .s(self.title)
        ]

        // Add the `info` field with the rating and/or plot if they're
        // available.

        var details: [Swift.String:DynamoDBClientTypes.AttributeValue] = [:]
        if (self.info.rating != nil || self.info.plot != nil) {
            if self.info.rating != nil {
                details["rating"] = .n(String(self.info.rating!))
            }
            if self.info.plot != nil {
                details["plot"] = .s(self.info.plot!)
            }
        }
        item["info"] = .m(details)

        return item
    }
```

- Einzelheiten zur API finden Sie PutItemin der API-Referenz zum AWS SDK für Swift.

Weitere DynamoDB-Beispiele finden Sie unter Codebeispiele für DynamoDB mit AWS SDKs.

Nachdem Daten in die Tabelle geschrieben wurden, fahren Sie mit Schritt 3: Lesen von Daten aus einer DynamoDB-Tabelle fort.

Schritt 3: Lesen von Daten aus einer DynamoDB-Tabelle

In diesem Schritt lesen Sie eines der Elemente, die Sie in Schritt 2: Schreiben von Daten in eine DynamoDB-Tabelle erstellt haben. Sie können die DynamoDB-Konsole oder die verwenden, AWS CLI um ein Element aus der `Music` Tabelle zu lesen, indem Sie und angeben`Artist`. `SongTitle`

Weitere Informationen über Leseoperationen in DynamoDB finden Sie unter Lesen eines Elements.

AWS-Managementkonsole

Gehen Sie wie folgt vor, um mithilfe der DynamoDB-Konsole Daten aus der `Music`-Tabelle zu lesen.

1. Öffnen Sie die DynamoDB-Konsole unter. https://console.aws.amazon.com/dynamodb/
2. Wählen Sie im linken Navigationsbereich Tables (Tabellen) aus.
3. Wählen Sie auf der Seite Tabellen die Tabelle Musik aus.
4. Wählen Sie Tabellenelemente erkunden aus.
5. Zeigen Sie auf der Registerkarte Elemente die Liste der in der Tabelle gespeicherten Elemente, sortiert nach `Artist` und `SongTitle`, an. Der erste Eintrag in der Liste ist der mit dem Künstler namens Acme Band und den SongTitlePartiQL Rocks.

AWS CLI

Das folgende AWS CLI Beispiel liest ein Element aus dem. `Music` Sie können dies entweder über die DynamoDB-API tun oder PartiQL, eine SQL-kompatible Abfragesprache für DynamoDB.

DynamoDB API

> **Note**
>
> Das Standardverhalten für DynamoDB sind schließlich konsistente Lesevorgänge. Nachstehend wird der Parameter `consistent-read` verwendet, um Strongly Consistent-Lesevorgänge zu veranschaulichen.

Linux

```
aws dynamodb get-item --consistent-read \
    --table-name Music \
```

```
    --key '{ "Artist": {"S": "Acme Band"}, "SongTitle": {"S": "Happy Day"}}'
```

Windows CMD

```
aws dynamodb get-item --consistent-read ^
    --table-name Music ^
    --key "{\"Artist\": {\"S\": \"Acme Band\"}, \"SongTitle\": {\"S\": \"Happy Day
\"}}"
```

Bei Verwenden von `get-item` wird das folgende Beispielergebnis zurückgegeben.

```
{
    "Item": {
        "AlbumTitle": {
            "S": "Songs About Life"
        },
        "Awards": {
            "S": "10"
        },
        "Artist": {
            "S": "Acme Band"
        },
        "SongTitle": {
            "S": "Happy Day"
        }
    }
}
```

PartiQL for DynamoDB

Linux

```
aws dynamodb execute-statement --statement "SELECT * FROM Music   \
WHERE Artist='Acme Band' AND SongTitle='Happy Day'"
```

Windows CMD

```
aws dynamodb execute-statement --statement "SELECT * FROM Music WHERE Artist='Acme
 Band' AND SongTitle='Happy Day'"
```

Verwendung der PartiQL `Select`-Anweisung gibt das folgende Beispielergebnis zurück.

```
{
    "Items": [
        {
            "AlbumTitle": {
                "S": "Songs About Life"
            },
            "Awards": {
                "S": "10"
            },
            "Artist": {
                "S": "Acme Band"
            },
            "SongTitle": {
                "S": "Happy Day"
            }
        }
    ]
}
```

Für weitere Informationen zum Lesen von Daten mit PartiQL siehe PartiQL Select Statements (PartiQL-Select-Anweisungen).

AWS SDK

Die folgenden Codebeispiele zeigen, wie Sie mithilfe eines SDK ein Element aus einer DynamoDB-Tabelle lesen. AWS

.NET

SDK für .NET (v4)

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
    /// <summary>
    /// Gets information about an existing movie from the table.
```

```
    /// </summary>
    /// <param name="newMovie">A Movie object containing information about
    /// the movie to retrieve.</param>
    /// <param name="tableName">The name of the table containing the movie.</
param>
    /// <returns>A Dictionary object containing information about the item
    /// retrieved.</returns>
    public async Task<Dictionary<string, AttributeValue>> GetItemAsync(Movie
 newMovie, string tableName)
    {
        try
        {
            var key = new Dictionary<string, AttributeValue>
            {
                ["title"] = new AttributeValue { S = newMovie.Title },
                ["year"] = new AttributeValue { N = newMovie.Year.ToString() },
            };

            var request = new GetItemRequest
            {
                Key = key,
                TableName = tableName,
            };

            var response = await _amazonDynamoDB.GetItemAsync(request);
            return response.Item;
        }
        catch (ResourceNotFoundException ex)
        {
            Console.WriteLine($"Table {tableName} was not found. {ex.Message}");
            return new Dictionary<string, AttributeValue>();
        }
        catch (AmazonDynamoDBException ex)
        {
            Console.WriteLine($"An Amazon DynamoDB error occurred while getting
 item. {ex.Message}");
            throw;
        }
        catch (Exception ex)
        {
            Console.WriteLine($"An error occurred while getting item.
 {ex.Message}");
            throw;
        }
```

```
    }
```

- Einzelheiten zur API finden Sie GetItemin der AWS SDK für .NET API-Referenz.

Bash

AWS CLI mit Bash-Skript

> **Note**
>
> Es gibt noch mehr dazu. GitHub Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
##############################################################################
# function dynamodb_get_item
#
# This function gets an item from a DynamoDB table.
#
# Parameters:
#       -n table_name  -- The name of the table.
#       -k keys  -- Path to json file containing the keys that identify the item
 to get.
#       [-q query]  -- Optional JMESPath query expression.
#
#  Returns:
#       The item as text output.
#  And:
#       0 - If successful.
#       1 - If it fails.
##############################################################################
function dynamodb_get_item() {
  local table_name keys query response
  local option OPTARG # Required to use getopts command in a function.

  # ######################################
  # Function usage explanation
  ######################################
  function usage() {
    echo "function dynamodb_get_item"
```

```
    echo "Get an item from a DynamoDB table."
    echo " -n table_name  -- The name of the table."
    echo " -k keys  -- Path to json file containing the keys that identify the
item to get."
    echo " [-q query]  -- Optional JMESPath query expression."
    echo ""
 }
 query=""
 while getopts "n:k:q:h" option; do
    case "${option}" in
      n) table_name="${OPTARG}" ;;
      k) keys="${OPTARG}" ;;
      q) query="${OPTARG}" ;;
      h)
        usage
        return 0
        ;;
      \?)
        echo "Invalid parameter"
        usage
        return 1
        ;;
    esac
 done
 export OPTIND=1

 if [[ -z "$table_name" ]]; then
    errecho "ERROR: You must provide a table name with the -n parameter."
    usage
    return 1
 fi

 if [[ -z "$keys" ]]; then
    errecho "ERROR: You must provide a keys json file path the -k parameter."
    usage
    return 1
 fi

 if [[ -n "$query" ]]; then
    response=$(aws dynamodb get-item \
      --table-name "$table_name" \
      --key file://"$keys" \
      --output text \
      --query "$query")
```

```
  else
    response=$(
      aws dynamodb get-item \
        --table-name "$table_name" \
        --key file://"$keys" \
        --output text
    )
  fi

  local error_code=${?}

  if [[ $error_code -ne 0 ]]; then
    aws_cli_error_log $error_code
    errecho "ERROR: AWS reports get-item operation failed.$response"
    return 1
  fi

  if [[ -n "$query" ]]; then
    echo "$response" | sed "/^\t/s/\t//1" # Remove initial tab that the JMSEPath
 query inserts on some strings.
  else
    echo "$response"
  fi

  return 0
}
```

Die in diesem Beispiel verwendeten Hilfsfunktionen.

```
###############################################################################
# function errecho
#
# This function outputs everything sent to it to STDERR (standard error output).
###############################################################################
function errecho() {
  printf "%s\n" "$*" 1>&2
}

###############################################################################
# function aws_cli_error_log()
#
# This function is used to log the error messages from the AWS CLI.
```

```
#
# See https://docs.aws.amazon.com/cli/latest/topic/return-codes.html#cli-aws-
help-return-codes.
#
# The function expects the following argument:
#         $1 - The error code returned by the AWS CLI.
#
#  Returns:
#          0: - Success.
#
###############################################################################
function aws_cli_error_log() {
  local err_code=$1
  errecho "Error code : $err_code"
  if [ "$err_code" == 1 ]; then
    errecho "  One or more S3 transfers failed."
  elif [ "$err_code" == 2 ]; then
    errecho "  Command line failed to parse."
  elif [ "$err_code" == 130 ]; then
    errecho "  Process received SIGINT."
  elif [ "$err_code" == 252 ]; then
    errecho "  Command syntax invalid."
  elif [ "$err_code" == 253 ]; then
    errecho "  The system environment or configuration was invalid."
  elif [ "$err_code" == 254 ]; then
    errecho "  The service returned an error."
  elif [ "$err_code" == 255 ]; then
    errecho "  255 is a catch-all error."
  fi

  return 0
}
```

- Einzelheiten zur API finden Sie GetItemin der AWS CLI Befehlsreferenz.

C++

SDK für C++

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
//! Get an item from an Amazon DynamoDB table.
/*!
  \sa getItem()
  \param tableName: The table name.
  \param partitionKey: The partition key.
  \param partitionValue: The value for the partition key.
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
 */

bool AwsDoc::DynamoDB::getItem(const Aws::String &tableName,
                               const Aws::String &partitionKey,
                               const Aws::String &partitionValue,
                               const Aws::Client::ClientConfiguration
 &clientConfiguration) {
    Aws::DynamoDB::DynamoDBClient dynamoClient(clientConfiguration);
    Aws::DynamoDB::Model::GetItemRequest request;

    // Set up the request.
    request.SetTableName(tableName);
    request.AddKey(partitionKey,
                   Aws::DynamoDB::Model::AttributeValue().SetS(partitionValue));

    // Retrieve the item's fields and values.
    const Aws::DynamoDB::Model::GetItemOutcome &outcome =
 dynamoClient.GetItem(request);
    if (outcome.IsSuccess()) {
        // Reference the retrieved fields/values.
        const Aws::Map<Aws::String, Aws::DynamoDB::Model::AttributeValue> &item =
 outcome.GetResult().GetItem();
        if (!item.empty()) {
            // Output each retrieved field and its value.
```

```
            for (const auto &i: item)
                std::cout << "Values: " << i.first << ": " << i.second.GetS()
                          << std::endl;
        }
        else {
            std::cout << "No item found with the key " << partitionKey <<
 std::endl;
        }
    }
    else {
        std::cout << "Failed to get item: " << outcome.GetError().GetMessage();
    }

    return outcome.IsSuccess();
}
```

- Einzelheiten zur API finden Sie GetItemin der AWS SDK für C++ API-Referenz.

CLI

AWS CLI

Beispiel 1: So lesen Sie ein Element in einer Tabelle

Das folgende Beispiel für `get-item` ruft ein Element aus der `MusicCollection`-Tabelle ab. Die Tabelle hat einen hash-and-range Primärschlüssel (`ArtistundSongTitle`), daher müssen Sie diese beiden Attribute angeben. Der Befehl fragt auch Informationen über die Lesekapazität ab, die von der Operation verbraucht wird.

```
aws dynamodb get-item \
    --table-name MusicCollection \
    --key file://key.json \
    --return-consumed-capacity TOTAL
```

Inhalt von `key.json`:

```
{
    "Artist": {"S": "Acme Band"},
    "SongTitle": {"S": "Happy Day"}
}
```

Ausgabe:

```
{
    "Item": {
        "AlbumTitle": {
            "S": "Songs About Life"
        },
        "SongTitle": {
            "S": "Happy Day"
        },
        "Artist": {
            "S": "Acme Band"
        }
    },
    "ConsumedCapacity": {
        "TableName": "MusicCollection",
        "CapacityUnits": 0.5
    }
}
```

Weitere Informationen finden Sie unter Lesen eines Elements im Entwicklerhandbuch zu Amazon DynamoDB.

Beispiel 2: So lesen Sie ein Element mit einem konsistenten Lesevorgang

Das folgende Beispiel ruft mithilfe von strikt konsistenten Lesevorgängen ein Element aus der `MusicCollection`-Tabelle ab.

```
aws dynamodb get-item \
    --table-name MusicCollection \
    --key file://key.json \
    --consistent-read \
    --return-consumed-capacity TOTAL
```

Inhalt von `key.json`:

```
{
    "Artist": {"S": "Acme Band"},
    "SongTitle": {"S": "Happy Day"}
}
```

Ausgabe:

```
{
    "Item": {
        "AlbumTitle": {
            "S": "Songs About Life"
        },
        "SongTitle": {
            "S": "Happy Day"
        },
        "Artist": {
            "S": "Acme Band"
        }
    },
    "ConsumedCapacity": {
        "TableName": "MusicCollection",
        "CapacityUnits": 1.0
    }
}
```

Weitere Informationen finden Sie unter Lesen eines Elements im Entwicklerhandbuch zu Amazon DynamoDB.

Beispiel 3: So rufen Sie bestimmte Attribute eines Elements ab

Im folgenden Beispiel wird ein Projektionsausdruck verwendet, um nur drei Attribute des gewünschten Elements abzurufen.

```
aws dynamodb get-item \
    --table-name ProductCatalog \
    --key '{"Id": {"N": "102"}}' \
    --projection-expression "#T, #C, #P" \
    --expression-attribute-names file://names.json
```

Inhalt von `names.json`:

```
{
    "#T": "Title",
    "#C": "ProductCategory",
    "#P": "Price"
}
```

Ausgabe:

```
{
    "Item": {
        "Price": {
            "N": "20"
        },
        "Title": {
            "S": "Book 102 Title"
        },
        "ProductCategory": {
            "S": "Book"
        }
    }
}
```

Weitere Informationen finden Sie unter Lesen eines Elements im Entwicklerhandbuch zu Amazon DynamoDB.

- Einzelheiten zur API finden Sie GetItemunter AWS CLI Befehlsreferenz.

Go

SDK für Go V2

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
import (
 "context"
 "errors"
 "log"
 "time"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/attributevalue"
 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/expression"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb/types"
```

```
)

// TableBasics encapsulates the Amazon DynamoDB service actions used in the
 examples.
// It contains a DynamoDB service client that is used to act on the specified
 table.
type TableBasics struct {
 DynamoDbClient *dynamodb.Client
 TableName      string
}



// GetMovie gets movie data from the DynamoDB table by using the primary
 composite key
// made of title and year.
func (basics TableBasics) GetMovie(ctx context.Context, title string, year int)
 (Movie, error) {
 movie := Movie{Title: title, Year: year}
 response, err := basics.DynamoDbClient.GetItem(ctx, &dynamodb.GetItemInput{
  Key: movie.GetKey(), TableName: aws.String(basics.TableName),
 })
 if err != nil {
  log.Printf("Couldn't get info about %v. Here's why: %v\n", title, err)
 } else {
  err = attributevalue.UnmarshalMap(response.Item, &movie)
  if err != nil {
   log.Printf("Couldn't unmarshal response. Here's why: %v\n", err)
  }
 }
 return movie, err
}
```

Definieren Sie eine Movie-Struktur, die in diesem Beispiel verwendet wird.

```
import (
 "archive/zip"
 "bytes"
 "encoding/json"
 "fmt"
```

```
 "io"
 "log"
 "net/http"

 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/attributevalue"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb/types"
)

// Movie encapsulates data about a movie. Title and Year are the composite
 primary key
// of the movie in Amazon DynamoDB. Title is the sort key, Year is the partition
 key,
// and Info is additional data.
type Movie struct {
 Title string                `dynamodbav:"title"`
 Year  int                   `dynamodbav:"year"`
 Info  map[string]interface{} `dynamodbav:"info"`
}

// GetKey returns the composite primary key of the movie in a format that can be
// sent to DynamoDB.
func (movie Movie) GetKey() map[string]types.AttributeValue {
 title, err := attributevalue.Marshal(movie.Title)
 if err != nil {
  panic(err)
 }
 year, err := attributevalue.Marshal(movie.Year)
 if err != nil {
  panic(err)
 }
 return map[string]types.AttributeValue{"title": title, "year": year}
}

// String returns the title, year, rating, and plot of a movie, formatted for the
 example.
func (movie Movie) String() string {
 return fmt.Sprintf("%v\n\tReleased: %v\n\tRating: %v\n\tPlot: %v\n",
  movie.Title, movie.Year, movie.Info["rating"], movie.Info["plot"])
}
```

- Einzelheiten zur API finden Sie GetItemin der AWS SDK für Go API-Referenz.

Java

SDK für Java 2.x

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

Ruft mithilfe von ein Element aus einer Tabelle ab DynamoDbClient.

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.AttributeValue;
import software.amazon.awssdk.services.dynamodb.model.GetItemRequest;
import java.util.HashMap;
import java.util.Map;
import java.util.Set;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 *
 * To get an item from an Amazon DynamoDB table using the AWS SDK for Java V2,
 * its better practice to use the
 * Enhanced Client, see the EnhancedGetItem example.
 */
public class GetItem {
    public static void main(String[] args) {
        final String usage = """

                Usage:
                    <tableName> <key> <keyVal>

                Where:
```

```
                    tableName - The Amazon DynamoDB table from which an item is
retrieved (for example, Music3).\s
                    key - The key used in the Amazon DynamoDB table (for example,
Artist).\s
                    keyval - The key value that represents the item to get (for
example, Famous Band).
                """";

        if (args.length != 3) {
            System.out.println(usage);
            System.exit(1);
        }

        String tableName = args[0];
        String key = args[1];
        String keyVal = args[2];
        System.out.format("Retrieving item \"%s\" from \"%s\"\n", keyVal,
tableName);
        Region region = Region.US_EAST_1;
        DynamoDbClient ddb = DynamoDbClient.builder()
                .region(region)
                .build();

        getDynamoDBItem(ddb, tableName, key, keyVal);
        ddb.close();
    }

    public static void getDynamoDBItem(DynamoDbClient ddb, String tableName,
String key, String keyVal) {
        HashMap<String, AttributeValue> keyToGet = new HashMap<>();
        keyToGet.put(key, AttributeValue.builder()
                .s(keyVal)
                .build());

        GetItemRequest request = GetItemRequest.builder()
                .key(keyToGet)
                .tableName(tableName)
                .build();

        try {
            // If there is no matching item, GetItem does not return any data.
            Map<String, AttributeValue> returnedItem =
ddb.getItem(request).item();
            if (returnedItem.isEmpty())
```

```
                System.out.format("No item found with the key %s!\n", key);
            else {
                Set<String> keys = returnedItem.keySet();
                System.out.println("Amazon DynamoDB table attributes: \n");
                for (String key1 : keys) {
                    System.out.format("%s: %s\n", key1,
 returnedItem.get(key1).toString());
                }
            }

        } catch (DynamoDbException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
    }
}
```

- Einzelheiten zur API finden Sie GetItemunter AWS SDK for Java 2.x API-Referenz.

JavaScript

SDK für JavaScript (v3)

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

In diesem Beispiel wird der Dokument-Client verwendet, um die Arbeit mit Elementen in DynamoDB zu vereinfachen. Einzelheiten zur API finden Sie unter GetCommand.

```
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";
import { DynamoDBDocumentClient, GetCommand } from "@aws-sdk/lib-dynamodb";

const client = new DynamoDBClient({});
const docClient = DynamoDBDocumentClient.from(client);

export const main = async () => {
  const command = new GetCommand({
    TableName: "AngryAnimals",
```

```
    Key: {
      CommonName: "Shoebill",
    },
  });

  const response = await docClient.send(command);
  console.log(response);
  return response;
};
```

- Einzelheiten zur API finden Sie GetItemunter AWS SDK für JavaScript API-Referenz.

SDK für JavaScript (v2)

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

Rufen Sie ein Element aus einer Tabelle ab.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create the DynamoDB service object
var ddb = new AWS.DynamoDB({ apiVersion: "2012-08-10" });

var params = {
  TableName: "TABLE",
  Key: {
    KEY_NAME: { N: "001" },
  },
  ProjectionExpression: "ATTRIBUTE_NAME",
};

// Call DynamoDB to read the item from the table
ddb.getItem(params, function (err, data) {
  if (err) {
    console.log("Error", err);
```

```
  } else {
    console.log("Success", data.Item);
  }
});
```

Rufen Sie ein Element mithilfe des DynamoDB-Dokument-Clients aus einer Tabelle ab.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create DynamoDB document client
var docClient = new AWS.DynamoDB.DocumentClient({ apiVersion: "2012-08-10" });

var params = {
  TableName: "EPISODES_TABLE",
  Key: { KEY_NAME: VALUE },
};

docClient.get(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data.Item);
  }
});
```

- Weitere Informationen finden Sie im AWS SDK für JavaScript -Entwicklerhandbuch.
- Einzelheiten zur API finden Sie GetItemin der AWS SDK für JavaScript API-Referenz.

Kotlin

SDK für Kotlin

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
suspend fun getSpecificItem(
    tableNameVal: String,
    keyName: String,
    keyVal: String,
) {
    val keyToGet = mutableMapOf<String, AttributeValue>()
    keyToGet[keyName] = AttributeValue.S(keyVal)

    val request =
        GetItemRequest {
            key = keyToGet
            tableName = tableNameVal
        }

    DynamoDbClient.fromEnvironment { region = "us-east-1" }.use { ddb ->
        val returnedItem = ddb.getItem(request)
        val numbersMap = returnedItem.item
        numbersMap?.forEach { key1 ->
            println(key1.key)
            println(key1.value)
        }
    }
}
```

- API-Details finden Sie GetItemin der API-Referenz zum AWS SDK für Kotlin.

PHP

SDK für PHP

> **Note**
>
> Es gibt noch mehr dazu. GitHub Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
        $movie = $service->getItemByKey($tableName, $key);
        echo "\nThe movie {$movie['Item']['title']['S']} was released in
{$movie['Item']['year']['N']}.\n";
```

```
    public function getItemByKey(string $tableName, array $key)
    {
        return $this->dynamoDbClient->getItem([
            'Key' => $key['Item'],
            'TableName' => $tableName,
        ]);
    }
```

- Einzelheiten zur API finden Sie GetItemin der AWS SDK für PHP API-Referenz.

PowerShell

Tools für PowerShell V4

Beispiel 1: Gibt das DynamoDB-Element mit dem Partitionsschlüssel SongTitle und dem Sortierschlüssel Artist zurück.

```
$key = @{
  SongTitle = 'Somewhere Down The Road'
  Artist = 'No One You Know'
} | ConvertTo-DDBItem

Get-DDBItem -TableName 'Music' -Key $key | ConvertFrom-DDBItem
```

Ausgabe:

```
Name                           Value
----                           -----
Genre                          Country
SongTitle                      Somewhere Down The Road
Price                          1.94
Artist                         No One You Know
CriticRating                   9
AlbumTitle                     Somewhat Famous
```

- Einzelheiten zur API finden Sie unter GetItem AWS -Tools für PowerShellCmdlet-Referenz (V4).

Tools für V5 PowerShell

Beispiel 1: Gibt das DynamoDB-Element mit dem Partitionsschlüssel SongTitle und dem Sortierschlüssel Artist zurück.

```
$key = @{
  SongTitle = 'Somewhere Down The Road'
  Artist = 'No One You Know'
} | ConvertTo-DDBItem

Get-DDBItem -TableName 'Music' -Key $key | ConvertFrom-DDBItem
```

Ausgabe:

```
Name                           Value
----                           -----
Genre                          Country
SongTitle                      Somewhere Down The Road
Price                          1.94
Artist                         No One You Know
CriticRating                   9
AlbumTitle                     Somewhat Famous
```

- Einzelheiten zur API finden Sie unter GetItem AWS -Tools für PowerShellCmdlet-Referenz (V5).

Python

SDK für Python (Boto3)

> **Note**
>
> Es gibt noch mehr dazu. GitHub Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
class Movies:
    """Encapsulates an Amazon DynamoDB table of movie data.

    Example data structure for a movie record in this table:
        {
```

```
            "year": 1999,
            "title": "For Love of the Game",
            "info": {
                "directors": ["Sam Raimi"],
                "release_date": "1999-09-15T00:00:00Z",
                "rating": 6.3,
                "plot": "A washed up pitcher flashes through his career.",
                "rank": 4987,
                "running_time_secs": 8220,
                "actors": [
                    "Kevin Costner",
                    "Kelly Preston",
                    "John C. Reilly"
                ]
            }
        }
    """

    def __init__(self, dyn_resource):
        """
        :param dyn_resource: A Boto3 DynamoDB resource.
        """
        self.dyn_resource = dyn_resource
        # The table variable is set during the scenario in the call to
        # 'exists' if the table exists. Otherwise, it is set by 'create_table'.
        self.table = None


    def get_movie(self, title, year):
        """
        Gets movie data from the table for a specific movie.

        :param title: The title of the movie.
        :param year: The release year of the movie.
        :return: The data about the requested movie.
        """
        try:
            response = self.table.get_item(Key={"year": year, "title": title})
        except ClientError as err:
            logger.error(
                "Couldn't get movie %s from table %s. Here's why: %s: %s",
                title,
                self.table.name,
                err.response["Error"]["Code"],
```

```
                err.response["Error"]["Message"],
            )
            raise
        else:
            return response["Item"]
```

- Einzelheiten zur API finden Sie GetItemin AWS SDK for Python (Boto3) API Reference.

Ruby

SDK für Ruby

> **Note**
>
> Es gibt noch mehr dazu. GitHub Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
class DynamoDBBasics
  attr_reader :dynamo_resource, :table

  def initialize(table_name)
    client = Aws::DynamoDB::Client.new(region: 'us-east-1')
    @dynamo_resource = Aws::DynamoDB::Resource.new(client: client)
    @table = @dynamo_resource.table(table_name)
  end

  # Gets movie data from the table for a specific movie.
  #
  # @param title [String] The title of the movie.
  # @param year [Integer] The release year of the movie.
  # @return [Hash] The data about the requested movie.
  def get_item(title, year)
    @table.get_item(key: { 'year' => year, 'title' => title })
  rescue Aws::DynamoDB::Errors::ServiceError => e
    puts("Couldn't get movie #{title} (#{year}) from table #{@table.name}:\n")
    puts("\t#{e.code}: #{e.message}")
    raise
  end
```

- Einzelheiten zur API finden Sie GetItemin der AWS SDK für Ruby API-Referenz.

SAP ABAP

SDK für SAP ABAP

> **Note**
>
> Es gibt noch mehr dazu. GitHub Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
TRY.
    oo_item = lo_dyn->getitem(
      iv_tablename               = iv_table_name
      it_key                     = it_key ).
    DATA(lt_attr) = oo_item->get_item( ).
    DATA(lo_title) = lt_attr[ key = 'title' ]-value.
    DATA(lo_year) = lt_attr[ key = 'year' ]-value.
    DATA(lo_rating) = lt_attr[ key = 'rating' ]-value.
    MESSAGE 'Movie name is: ' && lo_title->get_s( )
      && 'Movie year is: ' && lo_year->get_n( )
      && 'Moving rating is: ' && lo_rating->get_n( ) TYPE 'I'.
  CATCH /aws1/cx_dynresourcenotfoundex.
    MESSAGE 'The table or index does not exist' TYPE 'E'.
ENDTRY.
```

- Einzelheiten zur API finden Sie GetItemin der API-Referenz zum AWS SDK für SAP ABAP.

Swift

SDK für Swift

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
import AWSDynamoDB


    /// Return a `Movie` record describing the specified movie from the Amazon
    /// DynamoDB table.
    ///
    /// - Parameters:
    ///   - title: The movie's title (`String`).
    ///   - year: The movie's release year (`Int`).
    ///
    /// - Throws: `MoviesError.ItemNotFound` if the movie isn't in the table.
    ///
    /// - Returns: A `Movie` record with the movie's details.
    func get(title: String, year: Int) async throws -> Movie {
        do {
            guard let client = self.ddbClient else {
                throw MoviesError.UninitializedClient
            }

            let input = GetItemInput(
                key: [
                    "year": .n(String(year)),
                    "title": .s(title)
                ],
                tableName: self.tableName
            )
            let output = try await client.getItem(input: input)
            guard let item = output.item else {
                throw MoviesError.ItemNotFound
            }

            let movie = try Movie(withItem: item)
```

```
            return movie
        } catch {
            print("ERROR: get:", dump(error))
            throw error
        }
    }
```

- Einzelheiten zur API finden Sie GetItemin der API-Referenz zum AWS SDK für Swift.

Weitere DynamoDB-Beispiele finden Sie unter Codebeispiele für DynamoDB mit AWS SDKs.

Um die Daten in Ihrer Tabelle zu aktualisieren, fahren Sie mit Schritt 4: Aktualisieren von Daten in einer DynamoDB-Tabelle fort.

Schritt 4: Aktualisieren von Daten in einer DynamoDB-Tabelle

In diesem Schritt aktualisieren Sie ein von Ihnen in Schritt 2: Schreiben von Daten in eine DynamoDB-Tabelle erstelltes Element. Sie können die DynamoDB-Konsole oder die verwenden AWS CLI , um ein Element in `AlbumTitle` der `Music` Tabelle zu aktualisieren, indem Sie `ArtistSongTitle`, und das Aktualisierte angeben. `AlbumTitle`

Weitere Informationen über Schreibvorgänge finden Sie unter Schreiben eines Elements.

AWS-Managementkonsole

Sie können mithilfe der DynamoDB-Konsole Daten in der Tabelle `Music` aktualisieren.

1. Öffnen Sie die DynamoDB-Konsole unter. https://console.aws.amazon.com/dynamodb/
2. Wählen Sie im linken Navigationsbereich Tables (Tabellen) aus.
3. Wählen Sie in der Tabellenliste die Tabelle Music (Musik) aus.
4. Wählen Sie Tabellenelemente erkunden aus.
5. Gehen Sie unter Zurückgesendete Artikel für die Artikelzeile mit Acme Band Artist und Happy Day SongTitlewie folgt vor:

    a. Platzieren Sie den Mauszeiger auf den AlbumTitleTiteln Songs About Life.
    b. Wählen Sie das Symbol „Bearbeiten“ aus.
    c. Geben Sie im Popup-Fenster Zeichenfolge bearbeiten **`Songs of Twilight`** ein.

d. Wählen Sie Speichern.

> **Tip**
>
> Um ein Element zu aktualisieren, gehen Sie alternativ im Abschnitt Zurückgegebene Element wie folgt vor:
>
> 1. Wählen Sie die Artikelzeile mit dem Künstler namens Acme Band und SongTitledem Namen Happy Day.
> 2. Wählen Sie in der Dropdown-Liste Aktionen die Option Element bearbeiten aus.
> 3. Geben Sie für Enter AlbumTitleein**`Songs of Twilight`**.
> 4. Klicken Sie auf Save and close.

AWS CLI

Im folgenden AWS CLI Beispiel wird ein Element in der `Music` Tabelle aktualisiert. Sie können dies entweder über die DynamoDB-API tun oder PartiQL, eine SQL-kompatible Abfragesprache für DynamoDB.

DynamoDB API

Linux

```
aws dynamodb update-item \
    --table-name Music \
    --key '{ "Artist": {"S": "Acme Band"}, "SongTitle": {"S": "Happy Day"}}' \
    --update-expression "SET AlbumTitle = :newval" \
    --expression-attribute-values '{":newval":{"S":"Updated Album Title"}}' \
    --return-values ALL_NEW
```

Windows CMD

```
aws dynamodb update-item ^
    --table-name Music ^
    --key "{\"Artist\": {\"S\": \"Acme Band\"}, \"SongTitle\": {\"S\": \"Happy Day
\"}}" ^
    --update-expression "SET AlbumTitle = :newval" ^
    --expression-attribute-values "{\":newval\":{\"S\":\"Updated Album Title\"}}" ^
```

```
    --return-values ALL_NEW
```

Die Verwendung von `update-item` gibt das folgende Beispielergebnis zurück, da `return-values ALL_NEW` angegeben wurde.

```
{
    "Attributes": {
        "AlbumTitle": {
            "S": "Updated Album Title"
        },
        "Awards": {
            "S": "10"
        },
        "Artist": {
            "S": "Acme Band"
        },
        "SongTitle": {
            "S": "Happy Day"
        }
    }
}
```

PartiQL for DynamoDB

Linux

```
aws dynamodb execute-statement --statement "UPDATE Music  \
    SET AlbumTitle='Updated Album Title'  \
    WHERE Artist='Acme Band' AND SongTitle='Happy Day' \
    RETURNING ALL NEW *"
```

Windows CMD

```
aws dynamodb execute-statement --statement "UPDATE Music SET AlbumTitle='Updated
 Album Title' WHERE Artist='Acme Band' AND SongTitle='Happy Day' RETURNING ALL NEW
 *"
```

Die Verwendung der `Update`-Anweisung gibt das folgende Beispielergebnis zurück, da `RETURNING ALL NEW *` angegeben wurde.

```
{
    "Items": [
```

```
        {
            "AlbumTitle": {
                "S": "Updated Album Title"
            },
            "Awards": {
                "S": "10"
            },
            "Artist": {
                "S": "Acme Band"
            },
            "SongTitle": {
                "S": "Happy Day"
            }
        }
    ]
}
```

Für weitere Informationen zum Aktualisieren von Daten mit PartiQL siehe PartiQL Update Statements (PartiQL-Update-Anweisungen).

AWS SDK

Die folgenden Codebeispiele zeigen, wie ein Element in einer DynamoDB-Tabelle mithilfe eines AWS SDK aktualisiert wird.

.NET

SDK für .NET (v4)

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
    /// <summary>
    /// Updates an existing item in the movies table.
    /// </summary>
    /// <param name="newMovie">A Movie object containing information for
    /// the movie to update.</param>
```

```
    /// <param name="newInfo">A MovieInfo object that contains the
    /// information that will be changed.</param>
    /// <param name="tableName">The name of the table that contains the movie.</
param>
    /// <returns>A Boolean value that indicates the success of the operation.</
returns>
    public async Task<bool> UpdateItemAsync(
        Movie newMovie,
        MovieInfo newInfo,
        string tableName)
    {
        try
        {
            var key = new Dictionary<string, AttributeValue>
            {
                ["title"] = new AttributeValue { S = newMovie.Title },
                ["year"] = new AttributeValue { N = newMovie.Year.ToString() },
            };
            var updates = new Dictionary<string, AttributeValueUpdate>
            {
                ["info.plot"] = new AttributeValueUpdate
                {
                    Action = AttributeAction.PUT,
                    Value = new AttributeValue { S = newInfo.Plot },
                },

                ["info.rating"] = new AttributeValueUpdate
                {
                    Action = AttributeAction.PUT,
                    Value = new AttributeValue { N = newInfo.Rank.ToString() },
                },
            };

            var request = new UpdateItemRequest
            {
                AttributeUpdates = updates,
                Key = key,
                TableName = tableName,
            };

            await _amazonDynamoDB.UpdateItemAsync(request);
            return true;
        }
        catch (ResourceNotFoundException ex)
```

```
        {
            Console.WriteLine($"Table {tableName} or item was not found.
{ex.Message}");
            return false;
        }
        catch (AmazonDynamoDBException ex)
        {
            Console.WriteLine($"An Amazon DynamoDB error occurred while updating
item. {ex.Message}");
            throw;
        }
        catch (Exception ex)
        {
            Console.WriteLine($"An error occurred while updating item.
{ex.Message}");
            throw;
        }
    }
```

- Einzelheiten zur API finden Sie UpdateItemin der AWS SDK für .NET API-Referenz.

Bash

AWS CLI mit Bash-Skript

> **Note**
>
> Es gibt noch mehr dazu. GitHub Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
###############################################################################
# function dynamodb_update_item
#
# This function updates an item in a DynamoDB table.
#
#
# Parameters:
#       -n table_name  -- The name of the table.
```

```
#       -k keys  -- Path to json file containing the keys that identify the item
 to update.
#       -e update expression  -- An expression that defines one or more
 attributes to be updated.
#       -v values  -- Path to json file containing the update values.
#
#  Returns:
#       0 - If successful.
#       1 - If it fails.
###############################################################################
function dynamodb_update_item() {
  local table_name keys update_expression values response
  local option OPTARG # Required to use getopts command in a function.

  ######################################
  # Function usage explanation
  ######################################
  function usage() {
    echo "function dynamodb_update_item"
    echo "Update an item in a DynamoDB table."
    echo " -n table_name  -- The name of the table."
    echo " -k keys  -- Path to json file containing the keys that identify the
 item to update."
    echo " -e update expression  -- An expression that defines one or more
 attributes to be updated."
    echo " -v values  -- Path to json file containing the update values."
    echo ""
  }

  while getopts "n:k:e:v:h" option; do
    case "${option}" in
      n) table_name="${OPTARG}" ;;
      k) keys="${OPTARG}" ;;
      e) update_expression="${OPTARG}" ;;
      v) values="${OPTARG}" ;;
      h)
        usage
        return 0
        ;;
      \?)
        echo "Invalid parameter"
        usage
        return 1
        ;;
```

```
  esac
done
export OPTIND=1

if [[ -z "$table_name" ]]; then
  errecho "ERROR: You must provide a table name with the -n parameter."
  usage
  return 1
fi

if [[ -z "$keys" ]]; then
  errecho "ERROR: You must provide a keys json file path the -k parameter."
  usage
  return 1
fi
if [[ -z "$update_expression" ]]; then
  errecho "ERROR: You must provide an update expression with the -e parameter."
  usage
  return 1
fi

if [[ -z "$values" ]]; then
  errecho "ERROR: You must provide a values json file path the -v parameter."
  usage
  return 1
fi

iecho "Parameters:\n"
iecho "    table_name:   $table_name"
iecho "    keys:   $keys"
iecho "    update_expression:   $update_expression"
iecho "    values:   $values"

response=$(aws dynamodb update-item \
  --table-name "$table_name" \
  --key file://"$keys" \
  --update-expression "$update_expression" \
  --expression-attribute-values file://"$values")

local error_code=${?}

if [[ $error_code -ne 0 ]]; then
  aws_cli_error_log $error_code
  errecho "ERROR: AWS reports update-item operation failed.$response"
```

```
    return 1
  fi

  return 0

}
```

Die in diesem Beispiel verwendeten Hilfsfunktionen.

```
################################################################################
# function iecho
#
# This function enables the script to display the specified text only if
# the global variable $VERBOSE is set to true.
################################################################################
function iecho() {
  if [[ $VERBOSE == true ]]; then
    echo "$@"
  fi
}

################################################################################
# function errecho
#
# This function outputs everything sent to it to STDERR (standard error output).
################################################################################
function errecho() {
  printf "%s\n" "$*" 1>&2
}

################################################################################
# function aws_cli_error_log()
#
# This function is used to log the error messages from the AWS CLI.
#
# See https://docs.aws.amazon.com/cli/latest/topic/return-codes.html#cli-aws-
help-return-codes.
#
# The function expects the following argument:
#         $1 - The error code returned by the AWS CLI.
#
#  Returns:
```

```
#          0: - Success.
#
###############################################################################
function aws_cli_error_log() {
  local err_code=$1
  errecho "Error code : $err_code"
  if [ "$err_code" == 1 ]; then
    errecho "  One or more S3 transfers failed."
  elif [ "$err_code" == 2 ]; then
    errecho "  Command line failed to parse."
  elif [ "$err_code" == 130 ]; then
    errecho "  Process received SIGINT."
  elif [ "$err_code" == 252 ]; then
    errecho "  Command syntax invalid."
  elif [ "$err_code" == 253 ]; then
    errecho "  The system environment or configuration was invalid."
  elif [ "$err_code" == 254 ]; then
    errecho "  The service returned an error."
  elif [ "$err_code" == 255 ]; then
    errecho "  255 is a catch-all error."
  fi

  return 0
}
```

- Einzelheiten zur API finden Sie UpdateItemin der AWS CLI Befehlsreferenz.

C++

SDK für C++

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
//! Update an Amazon DynamoDB table item.
/*!
  \sa updateItem()
  \param tableName: The table name.
```

```
  \param partitionKey: The partition key.
  \param partitionValue: The value for the partition key.
  \param attributeKey: The key for the attribute to be updated.
  \param attributeValue: The value for the attribute to be updated.
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
  */

/*
 *  The example code only sets/updates an attribute value. It processes
 *  the attribute value as a string, even if the value could be interpreted
 *  as a number. Also, the example code does not remove an existing attribute
 *  from the key value.
 */

bool AwsDoc::DynamoDB::updateItem(const Aws::String &tableName,
                                  const Aws::String &partitionKey,
                                  const Aws::String &partitionValue,
                                  const Aws::String &attributeKey,
                                  const Aws::String &attributeValue,
                                  const Aws::Client::ClientConfiguration
 &clientConfiguration) {
    Aws::DynamoDB::DynamoDBClient dynamoClient(clientConfiguration);

    // *** Define UpdateItem request arguments.
    // Define TableName argument.
    Aws::DynamoDB::Model::UpdateItemRequest request;
    request.SetTableName(tableName);

    // Define KeyName argument.
    Aws::DynamoDB::Model::AttributeValue attribValue;
    attribValue.SetS(partitionValue);
    request.AddKey(partitionKey, attribValue);

    // Construct the SET update expression argument.
    Aws::String update_expression("SET #a = :valueA");
    request.SetUpdateExpression(update_expression);

    // Construct attribute name argument.
    Aws::Map<Aws::String, Aws::String> expressionAttributeNames;
    expressionAttributeNames["#a"] = attributeKey;
    request.SetExpressionAttributeNames(expressionAttributeNames);

    // Construct attribute value argument.
```

```
    Aws::DynamoDB::Model::AttributeValue attributeUpdatedValue;
    attributeUpdatedValue.SetS(attributeValue);
    Aws::Map<Aws::String, Aws::DynamoDB::Model::AttributeValue>
 expressionAttributeValues;
    expressionAttributeValues[":valueA"] = attributeUpdatedValue;
    request.SetExpressionAttributeValues(expressionAttributeValues);

    // Update the item.
    const Aws::DynamoDB::Model::UpdateItemOutcome &outcome =
 dynamoClient.UpdateItem(
            request);
    if (outcome.IsSuccess()) {
        std::cout << "Item was updated" << std::endl;
    } else {
        std::cerr << outcome.GetError().GetMessage() << std::endl;
        return false;
    }

    return waitTableActive(tableName, dynamoClient);
}
```

Code, der darauf wartet, dass die Tabelle aktiv wird.

```
//! Query a newly created DynamoDB table until it is active.
/*!
  \sa waitTableActive()
  \param waitTableActive: The DynamoDB table's name.
  \param dynamoClient: A DynamoDB client.
  \return bool: Function succeeded.
*/
bool AwsDoc::DynamoDB::waitTableActive(const Aws::String &tableName,
                                       const Aws::DynamoDB::DynamoDBClient
 &dynamoClient) {

    // Repeatedly call DescribeTable until table is ACTIVE.
    const int MAX_QUERIES = 20;
    Aws::DynamoDB::Model::DescribeTableRequest request;
    request.SetTableName(tableName);

    int count = 0;
    while (count < MAX_QUERIES) {
```

```
        const Aws::DynamoDB::Model::DescribeTableOutcome &result =
 dynamoClient.DescribeTable(
                request);
        if (result.IsSuccess()) {
            Aws::DynamoDB::Model::TableStatus status =
 result.GetResult().GetTable().GetTableStatus();

            if (Aws::DynamoDB::Model::TableStatus::ACTIVE != status) {
                std::this_thread::sleep_for(std::chrono::seconds(1));
            }
            else {
                return true;
            }
        }
        else {
            std::cerr << "Error DynamoDB::waitTableActive "
                      << result.GetError().GetMessage() << std::endl;
            return false;
        }
        count++;
    }
    return false;
}
```

- Einzelheiten zur API finden Sie UpdateItemin der AWS SDK für C++ API-Referenz.

CLI

AWS CLI

Beispiel 1: So aktualisieren Sie ein Element in einer Tabelle

Im folgenden Beispiel für `update-item` wird ein Element in der Tabelle `MusicCollection` aktualisiert. Es wird ein neues Attribut (`Year`) hinzugefügt und das `AlbumTitle`-Attribut geändert. Alle Attribute im Element, wie sie nach dem Update erscheinen, werden in der Antwort zurückgegeben.

```
aws dynamodb update-item \
    --table-name MusicCollection \
    --key file://key.json \
    --update-expression "SET #Y = :y, #AT = :t" \
```

```
    --expression-attribute-names file://expression-attribute-names.json \
    --expression-attribute-values file://expression-attribute-values.json  \
    --return-values ALL_NEW \
    --return-consumed-capacity TOTAL \
    --return-item-collection-metrics SIZE
```

Inhalt von `key.json`:

```
{
    "Artist": {"S": "Acme Band"},
    "SongTitle": {"S": "Happy Day"}
}
```

Inhalt von `expression-attribute-names.json`:

```
{
    "#Y":"Year", "#AT":"AlbumTitle"
}
```

Inhalt von `expression-attribute-values.json`:

```
{
    ":y":{"N": "2015"},
    ":t":{"S": "Louder Than Ever"}
}
```

Ausgabe:

```
{
    "Attributes": {
        "AlbumTitle": {
            "S": "Louder Than Ever"
        },
        "Awards": {
            "N": "10"
        },
        "Artist": {
            "S": "Acme Band"
        },
        "Year": {
            "N": "2015"
        },
```

```
        "SongTitle": {
            "S": "Happy Day"
        }
    },
    "ConsumedCapacity": {
        "TableName": "MusicCollection",
        "CapacityUnits": 3.0
    },
    "ItemCollectionMetrics": {
        "ItemCollectionKey": {
            "Artist": {
                "S": "Acme Band"
            }
        },
        "SizeEstimateRangeGB": [
            0.0,
            1.0
        ]
    }
}
```

Weitere Informationen finden Sie unter Schreiben eines Elements im Entwicklerhandbuch zu Amazon DynamoDB.

Beispiel 2: So aktualisieren Sie ein Element bedingungsabhängig

Im folgenden Beispiel wird ein Element in der `MusicCollection`-Tabelle aktualisiert, jedoch nur, wenn das vorhandene Element noch kein `Year`-Attribut besitzt.

```
aws dynamodb update-item \
    --table-name MusicCollection \
    --key file://key.json \
    --update-expression "SET #Y = :y, #AT = :t" \
    --expression-attribute-names file://expression-attribute-names.json \
    --expression-attribute-values file://expression-attribute-values.json  \
    --condition-expression "attribute_not_exists(#Y)"
```

Inhalt von `key.json`:

```
{
    "Artist": {"S": "Acme Band"},
    "SongTitle": {"S": "Happy Day"}
```

```
}
```

Inhalt von `expression-attribute-names.json`:

```
{
    "#Y":"Year",
    "#AT":"AlbumTitle"
}
```

Inhalt von `expression-attribute-values.json`:

```
{
    ":y":{"N": "2015"},
    ":t":{"S": "Louder Than Ever"}
}
```

Wenn das Element bereits über ein `Year`-Attribut verfügt, gibt DynamoDB die folgende Ausgabe zurück.

```
An error occurred (ConditionalCheckFailedException) when calling the UpdateItem
 operation: The conditional request failed
```

Weitere Informationen finden Sie unter Schreiben eines Elements im Entwicklerhandbuch zu Amazon DynamoDB.

- Einzelheiten zur API finden Sie UpdateItemin der AWS CLI Befehlsreferenz.

Go

SDK für Go V2

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
import (
 "context"
```

```
 "errors"
 "log"
 "time"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/attributevalue"
 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/expression"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb/types"
)

// TableBasics encapsulates the Amazon DynamoDB service actions used in the
 examples.
// It contains a DynamoDB service client that is used to act on the specified
 table.
type TableBasics struct {
 DynamoDbClient *dynamodb.Client
 TableName      string
}



// UpdateMovie updates the rating and plot of a movie that already exists in the
// DynamoDB table. This function uses the `expression` package to build the
 update
// expression.
func (basics TableBasics) UpdateMovie(ctx context.Context, movie Movie)
 (map[string]map[string]interface{}, error) {
 var err error
 var response *dynamodb.UpdateItemOutput
 var attributeMap map[string]map[string]interface{}
 update := expression.Set(expression.Name("info.rating"),
 expression.Value(movie.Info["rating"]))
 update.Set(expression.Name("info.plot"), expression.Value(movie.Info["plot"]))
 expr, err := expression.NewBuilder().WithUpdate(update).Build()
 if err != nil {
  log.Printf("Couldn't build expression for update. Here's why: %v\n", err)
 } else {
  response, err = basics.DynamoDbClient.UpdateItem(ctx,
 &dynamodb.UpdateItemInput{
   TableName:                 aws.String(basics.TableName),
   Key:                       movie.GetKey(),
   ExpressionAttributeNames:  expr.Names(),
   ExpressionAttributeValues: expr.Values(),
```

```
   UpdateExpression:          expr.Update(),
   ReturnValues:              types.ReturnValueUpdatedNew,
  })
  if err != nil {
   log.Printf("Couldn't update movie %v. Here's why: %v\n", movie.Title, err)
  } else {
   err = attributevalue.UnmarshalMap(response.Attributes, &attributeMap)
   if err != nil {
    log.Printf("Couldn't unmarshall update response. Here's why: %v\n", err)
   }
  }
 }
 return attributeMap, err
}
```

Definieren Sie eine Movie-Struktur, die in diesem Beispiel verwendet wird.

```
import (
 "archive/zip"
 "bytes"
 "encoding/json"
 "fmt"
 "io"
 "log"
 "net/http"

 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/attributevalue"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb/types"
)

// Movie encapsulates data about a movie. Title and Year are the composite
 primary key
// of the movie in Amazon DynamoDB. Title is the sort key, Year is the partition
 key,
// and Info is additional data.
type Movie struct {
 Title string                 `dynamodbav:"title"`
 Year  int                    `dynamodbav:"year"`
 Info  map[string]interface{} `dynamodbav:"info"`
}
```

```
// GetKey returns the composite primary key of the movie in a format that can be
// sent to DynamoDB.
func (movie Movie) GetKey() map[string]types.AttributeValue {
 title, err := attributevalue.Marshal(movie.Title)
 if err != nil {
  panic(err)
 }
 year, err := attributevalue.Marshal(movie.Year)
 if err != nil {
  panic(err)
 }
 return map[string]types.AttributeValue{"title": title, "year": year}
}

// String returns the title, year, rating, and plot of a movie, formatted for the
 example.
func (movie Movie) String() string {
 return fmt.Sprintf("%v\n\tReleased: %v\n\tRating: %v\n\tPlot: %v\n",
  movie.Title, movie.Year, movie.Info["rating"], movie.Info["plot"])
}
```

- Einzelheiten zur API finden Sie UpdateItemin der AWS SDK für Go API-Referenz.

Java

SDK für Java 2.x

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

Aktualisiert ein Element in einer Tabelle mit DynamoDbClient.

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.model.AttributeAction;
import software.amazon.awssdk.services.dynamodb.model.AttributeValue;
```

```
import software.amazon.awssdk.services.dynamodb.model.AttributeValueUpdate;
import software.amazon.awssdk.services.dynamodb.model.UpdateItemRequest;
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import java.util.HashMap;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 *
 * To update an Amazon DynamoDB table using the AWS SDK for Java V2, its better
 * practice to use the
 * Enhanced Client, See the EnhancedModifyItem example.
 */
public class UpdateItem {
    public static void main(String[] args) {
        final String usage = """

                Usage:
                    <tableName> <key> <keyVal> <name> <updateVal>

                Where:
                    tableName - The Amazon DynamoDB table (for example, Music3).
                    key - The name of the key in the table (for example, Artist).
                    keyVal - The value of the key (for example, Famous Band).
                    name - The name of the column where the value is updated (for
 example, Awards).
                    updateVal - The value used to update an item (for example,
 14).
                 Example:
                    UpdateItem Music3 Artist Famous Band Awards 14
                """;

        if (args.length != 5) {
            System.out.println(usage);
            System.exit(1);
        }

        String tableName = args[0];
        String key = args[1];
```

```
        String keyVal = args[2];
        String name = args[3];
        String updateVal = args[4];

        Region region = Region.US_EAST_1;
        DynamoDbClient ddb = DynamoDbClient.builder()
                .region(region)
                .build();
        updateTableItem(ddb, tableName, key, keyVal, name, updateVal);
        ddb.close();
    }

    public static void updateTableItem(DynamoDbClient ddb,
            String tableName,
            String key,
            String keyVal,
            String name,
            String updateVal) {

        HashMap<String, AttributeValue> itemKey = new HashMap<>();
        itemKey.put(key, AttributeValue.builder()
                .s(keyVal)
                .build());

        HashMap<String, AttributeValueUpdate> updatedValues = new HashMap<>();
        updatedValues.put(name, AttributeValueUpdate.builder()
                .value(AttributeValue.builder().s(updateVal).build())
                .action(AttributeAction.PUT)
                .build());

        UpdateItemRequest request = UpdateItemRequest.builder()
                .tableName(tableName)
                .key(itemKey)
                .attributeUpdates(updatedValues)
                .build();

        try {
            ddb.updateItem(request);
        } catch (DynamoDbException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
        System.out.println("The Amazon DynamoDB table was updated!");
    }
```

```
}
```

- Einzelheiten zur API finden Sie UpdateItemunter AWS SDK for Java 2.x API-Referenz.

JavaScript

SDK für JavaScript (v3)

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

In diesem Beispiel wird der Dokument-Client verwendet, um die Arbeit mit Elementen in DynamoDB zu vereinfachen. Einzelheiten zur API finden Sie unter UpdateCommand.

```
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";
import { DynamoDBDocumentClient, UpdateCommand } from "@aws-sdk/lib-dynamodb";

const client = new DynamoDBClient({});
const docClient = DynamoDBDocumentClient.from(client);

export const main = async () => {
  const command = new UpdateCommand({
    TableName: "Dogs",
    Key: {
      Breed: "Labrador",
    },
    UpdateExpression: "set Color = :color",
    ExpressionAttributeValues: {
      ":color": "black",
    },
    ReturnValues: "ALL_NEW",
  });

  const response = await docClient.send(command);
  console.log(response);
  return response;
};
```

- Einzelheiten zur API finden Sie UpdateItemunter AWS SDK für JavaScript API-Referenz.

Kotlin

SDK für Kotlin

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
suspend fun updateTableItem(
    tableNameVal: String,
    keyName: String,
    keyVal: String,
    name: String,
    updateVal: String,
) {
    val itemKey = mutableMapOf<String, AttributeValue>()
    itemKey[keyName] = AttributeValue.S(keyVal)

    val updatedValues = mutableMapOf<String, AttributeValueUpdate>()
    updatedValues[name] =
        AttributeValueUpdate {
            value = AttributeValue.S(updateVal)
            action = AttributeAction.Put
        }

    val request =
        UpdateItemRequest {
            tableName = tableNameVal
            key = itemKey
            attributeUpdates = updatedValues
        }

    DynamoDbClient.fromEnvironment { region = "us-east-1" }.use { ddb ->
        ddb.updateItem(request)
        println("Item in $tableNameVal was updated")
    }
```

```
}
```

- API-Details finden Sie UpdateItemin der API-Referenz zum AWS SDK für Kotlin.

PHP

SDK für PHP

> **Note**
>
> Es gibt noch mehr dazu. GitHub Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
        echo "What rating would you like to give {$movie['Item']['title']['S']}?
\n";
        $rating = 0;
        while (!is_numeric($rating) || intval($rating) != $rating || $rating < 1
 || $rating > 10) {
            $rating = testable_readline("Rating (1-10): ");
        }
        $service->updateItemAttributeByKey($tableName, $key, 'rating', 'N',
 $rating);

    public function updateItemAttributeByKey(
        string $tableName,
        array $key,
        string $attributeName,
        string $attributeType,
        string $newValue
    ) {
        $this->dynamoDbClient->updateItem([
            'Key' => $key['Item'],
            'TableName' => $tableName,
            'UpdateExpression' => "set #NV=:NV",
            'ExpressionAttributeNames' => [
                '#NV' => $attributeName,
            ],
            'ExpressionAttributeValues' => [
                ':NV' => [
                    $attributeType => $newValue
```

```
                ]
            ],
        ]);
    }
```

- Einzelheiten zur API finden Sie UpdateItemin der AWS SDK für PHP API-Referenz.

PowerShell

Tools für PowerShell V4

Beispiel 1: Setzt das Genre-Attribut auf 'Rap' für das DynamoDB-Element mit dem Partitionsschlüssel SongTitle und dem Sortierschlüssel Artist.

```
$key = @{
    SongTitle = 'Somewhere Down The Road'
    Artist = 'No One You Know'
} | ConvertTo-DDBItem

$updateDdbItem = @{
    TableName = 'Music'
    Key = $key
    UpdateExpression = 'set Genre = :val1'
    ExpressionAttributeValue = (@{
        ':val1' = ([Amazon.DynamoDBv2.Model.AttributeValue]'Rap')
    })
}
Update-DDBItem @updateDdbItem
```

Ausgabe:

```
Name                           Value
----                           -----
Genre                          Rap
```

- Einzelheiten zur API finden Sie unter UpdateItem AWS -Tools für PowerShellCmdlet-Referenz (V4).

Tools für V5 PowerShell

Beispiel 1: Setzt das Genre-Attribut auf 'Rap' für das DynamoDB-Element mit dem Partitionsschlüssel SongTitle und dem Sortierschlüssel Artist.

```
$key = @{
    SongTitle = 'Somewhere Down The Road'
    Artist = 'No One You Know'
} | ConvertTo-DDBItem

$updateDdbItem = @{
    TableName = 'Music'
    Key = $key
    UpdateExpression = 'set Genre = :val1'
    ExpressionAttributeValue = (@{
        ':val1' = ([Amazon.DynamoDBv2.Model.AttributeValue]'Rap')
    })
}
Update-DDBItem @updateDdbItem
```

Ausgabe:

```
Name                           Value
----                           -----
Genre                          Rap
```

- Einzelheiten zur API finden Sie unter UpdateItem AWS -Tools für PowerShellCmdlet-Referenz (V5).

Python

SDK für Python (Boto3)

> **Note**
>
> Es gibt noch mehr dazu. GitHub Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

Aktualisieren Sie ein Element mithilfe eines Aktualisierungsausdrucks.

```
class Movies:
    """Encapsulates an Amazon DynamoDB table of movie data.

    Example data structure for a movie record in this table:
        {
            "year": 1999,
            "title": "For Love of the Game",
            "info": {
                "directors": ["Sam Raimi"],
                "release_date": "1999-09-15T00:00:00Z",
                "rating": 6.3,
                "plot": "A washed up pitcher flashes through his career.",
                "rank": 4987,
                "running_time_secs": 8220,
                "actors": [
                    "Kevin Costner",
                    "Kelly Preston",
                    "John C. Reilly"
                ]
            }
        }
    """

    def __init__(self, dyn_resource):
        """
        :param dyn_resource: A Boto3 DynamoDB resource.
        """
        self.dyn_resource = dyn_resource
        # The table variable is set during the scenario in the call to
        # 'exists' if the table exists. Otherwise, it is set by 'create_table'.
        self.table = None


    def update_movie(self, title, year, rating, plot):
        """
        Updates rating and plot data for a movie in the table.

        :param title: The title of the movie to update.
        :param year: The release year of the movie to update.
        :param rating: The updated rating to the give the movie.
        :param plot: The updated plot summary to give the movie.
        :return: The fields that were updated, with their new values.
        """
```

```
        try:
            response = self.table.update_item(
                Key={"year": year, "title": title},
                UpdateExpression="set info.rating=:r, info.plot=:p",
                ExpressionAttributeValues={":r": Decimal(str(rating)), ":p":
 plot},
                ReturnValues="UPDATED_NEW",
            )
        except ClientError as err:
            logger.error(
                "Couldn't update movie %s in table %s. Here's why: %s: %s",
                title,
                self.table.name,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return response["Attributes"]
```

Aktualisieren Sie ein Element mithilfe eines Aktualisierungsausdrucks, der eine arithmetische Operation enthält.

```
class UpdateQueryWrapper:
    def __init__(self, table):
        self.table = table


    def update_rating(self, title, year, rating_change):
        """
        Updates the quality rating of a movie in the table by using an arithmetic
        operation in the update expression. By specifying an arithmetic
 operation,
        you can adjust a value in a single request, rather than first getting its
        value and then setting its new value.

        :param title: The title of the movie to update.
        :param year: The release year of the movie to update.
        :param rating_change: The amount to add to the current rating for the
 movie.
        :return: The updated rating.
```

```
        """
        try:
            response = self.table.update_item(
                Key={"year": year, "title": title},
                UpdateExpression="set info.rating = info.rating + :val",
                ExpressionAttributeValues={":val": Decimal(str(rating_change))},
                ReturnValues="UPDATED_NEW",
            )
        except ClientError as err:
            logger.error(
                "Couldn't update movie %s in table %s. Here's why: %s: %s",
                title,
                self.table.name,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return response["Attributes"]
```

Aktualisieren Sie ein Element nur, wenn es bestimmte Bedingungen erfüllt.

```
class UpdateQueryWrapper:
    def __init__(self, table):
        self.table = table


    def remove_actors(self, title, year, actor_threshold):
        """
        Removes an actor from a movie, but only when the number of actors is
 greater
        than a specified threshold. If the movie does not list more than the
 threshold,
        no actors are removed.

        :param title: The title of the movie to update.
        :param year: The release year of the movie to update.
        :param actor_threshold: The threshold of actors to check.
        :return: The movie data after the update.
        """
        try:
```

```
            response = self.table.update_item(
                Key={"year": year, "title": title},
                UpdateExpression="remove info.actors[0]",
                ConditionExpression="size(info.actors) > :num",
                ExpressionAttributeValues={":num": actor_threshold},
                ReturnValues="ALL_NEW",
            )
        except ClientError as err:
            if err.response["Error"]["Code"] ==
 "ConditionalCheckFailedException":
                logger.warning(
                    "Didn't update %s because it has fewer than %s actors.",
                    title,
                    actor_threshold + 1,
                )
            else:
                logger.error(
                    "Couldn't update movie %s. Here's why: %s: %s",
                    title,
                    err.response["Error"]["Code"],
                    err.response["Error"]["Message"],
                )
            raise
        else:
            return response["Attributes"]
```

- Einzelheiten zur API finden Sie UpdateItemin AWS SDK for Python (Boto3) API Reference.

Ruby

SDK für Ruby

> **Note**
>
> Es gibt noch mehr dazu. GitHub Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
class DynamoDBBasics
  attr_reader :dynamo_resource, :table
```

```
  def initialize(table_name)
    client = Aws::DynamoDB::Client.new(region: 'us-east-1')
    @dynamo_resource = Aws::DynamoDB::Resource.new(client: client)
    @table = @dynamo_resource.table(table_name)
  end

  # Updates rating and plot data for a movie in the table.
  #
  # @param movie [Hash] The title, year, plot, rating of the movie.
  def update_item(movie)
    response = @table.update_item(
      key: { 'year' => movie[:year], 'title' => movie[:title] },
      update_expression: 'set info.rating=:r',
      expression_attribute_values: { ':r' => movie[:rating] },
      return_values: 'UPDATED_NEW'
    )
  rescue Aws::DynamoDB::Errors::ServiceError => e
    puts("Couldn't update movie #{movie[:title]} (#{movie[:year]}) in table
#{@table.name}\n")
    puts("\t#{e.code}: #{e.message}")
    raise
  else
    response.attributes
  end
```

- Einzelheiten zur API finden Sie UpdateItemin der AWS SDK für Ruby API-Referenz.

SAP ABAP

SDK für SAP ABAP

> **Note**
>
> Es gibt noch mehr dazu. GitHub Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
    TRY.
        oo_output = lo_dyn->updateitem(
          iv_tablename        = iv_table_name
```

```
          it_key              = it_item_key
          it_attributeupdates = it_attribute_updates ).
        MESSAGE '1 item updated in DynamoDB Table' && iv_table_name TYPE 'I'.
      CATCH /aws1/cx_dyncondalcheckfaile00.
        MESSAGE 'A condition specified in the operation could not be evaluated.'
TYPE 'E'.
      CATCH /aws1/cx_dynresourcenotfoundex.
        MESSAGE 'The table or index does not exist' TYPE 'E'.
      CATCH /aws1/cx_dyntransactconflictex.
        MESSAGE 'Another transaction is using the item' TYPE 'E'.
    ENDTRY.
```

- Einzelheiten zur API finden Sie UpdateItemin der API-Referenz zum AWS SDK für SAP ABAP.

Swift

SDK für Swift

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
import AWSDynamoDB


    /// Update the specified movie with new `rating` and `plot` information.
    ///
    /// - Parameters:
    ///   - title: The title of the movie to update.
    ///   - year: The release year of the movie to update.
    ///   - rating: The new rating for the movie.
    ///   - plot: The new plot summary string for the movie.
    ///
    /// - Returns: An array of mappings of attribute names to their new
    ///   listing each item actually changed. Items that didn't need to change
    ///   aren't included in this list. `nil` if no changes were made.
    ///
```

```
   func update(title: String, year: Int, rating: Double? = nil, plot: String? =
nil) async throws
       -> [Swift.String: DynamoDBClientTypes.AttributeValue]?
   {
       do {
           guard let client = self.ddbClient else {
               throw MoviesError.UninitializedClient
           }

           // Build the update expression and the list of expression attribute
           // values. Include only the information that's changed.

           var expressionParts: [String] = []
           var attrValues: [Swift.String: DynamoDBClientTypes.AttributeValue] =
[:]

           if rating != nil {
               expressionParts.append("info.rating=:r")
               attrValues[":r"] = .n(String(rating!))
           }
           if plot != nil {
               expressionParts.append("info.plot=:p")
               attrValues[":p"] = .s(plot!)
           }
           let expression = "set \(expressionParts.joined(separator: ", "))"

           let input = UpdateItemInput(
               // Create substitution tokens for the attribute values, to ensure
               // no conflicts in expression syntax.
               expressionAttributeValues: attrValues,
               // The key identifying the movie to update consists of the
release
               // year and title.
               key: [
                   "year": .n(String(year)),
                   "title": .s(title)
               ],
               returnValues: .updatedNew,
               tableName: self.tableName,
               updateExpression: expression
           )
           let output = try await client.updateItem(input: input)
```

```
            guard let attributes: [Swift.String:
DynamoDBClientTypes.AttributeValue] = output.attributes else {
                throw MoviesError.InvalidAttributes
            }
            return attributes
        } catch {
            print("ERROR: update:", dump(error))
            throw error
        }
    }
```

- Einzelheiten zur API finden Sie UpdateItemin der API-Referenz zum AWS SDK für Swift.

Weitere DynamoDB-Beispiele finden Sie unter Codebeispiele für DynamoDB mit AWS SDKs.

Um die Daten in der Tabelle `Music` abzufragen, fahren Sie mit Schritt 5: Abfragen der Daten in der DynamoDB-Tabelle fort.

Schritt 5: Abfragen der Daten in der DynamoDB-Tabelle

In diesem Schritt fragen Sie durch Angabe von `Artist` die Daten ab, die Sie in the section called “Schritt 2: Schreiben von Daten” in die Tabelle `Music` geschrieben haben. Dadurch werden alle Lieder angezeigt, die mit dem Partitionsschlüssel verknüpft sind: `Artist`.

Weitere Informationen über Abfrageoperationen finden Sie unter Abfragen von Tabellen in DynamoDB.

AWS-Managementkonsole

Gehen Sie wie folgt vor, um mithilfe der DynamoDB-Konsole Daten in der Tabelle `Music` abzufragen.

1. Öffnen Sie die DynamoDB-Konsole unter. https://console.aws.amazon.com/dynamodb/
2. Wählen Sie im linken Navigationsbereich Tables (Tabellen) aus.
3. Wählen Sie in der Tabellenliste die Tabelle Music (Musik) aus.
4. Wählen Sie Tabellenelemente erkunden aus.
5. Stellen Sie sicher, dass unter Elemente scannen oder abfragen die Option Abfrage ausgewählt ist.

6. Geben Sie für Partition key (Partitionsschlüssel) **`Acme Band`** ein und wählen Sie dann Run (Ausführen).

AWS CLI

Im folgenden AWS CLI Beispiel wird ein Element in der `Music` Tabelle abgefragt. Sie können dies entweder über die DynamoDB-API tun oder PartiQL, eine SQL-kompatible Abfragesprache für DynamoDB.

DynamoDB API

Sie fragen ein Element über die DynamoDB-API ab, indem Sie `query` verwenden und den Partitionsschlüssel bereitstellen.

Linux

```
aws dynamodb query \
    --table-name Music \
    --key-condition-expression "Artist = :name" \
    --expression-attribute-values  '{":name":{"S":"Acme Band"}}'
```

Windows CMD

```
aws dynamodb query ^
    --table-name Music ^
    --key-condition-expression "Artist = :name" ^
    --expression-attribute-values  "{\":name\":{\"S\":\"Acme Band\"}}"
```

Verwendung von `query` gibt alle Songs zurück, die mit diesem bestimmten `Artist` assoziiert sind.

```
{
    "Items": [
        {
            "AlbumTitle": {
                "S": "Updated Album Title"
            },
            "Awards": {
                "N": "10"
            },
            "Artist": {
```

```
                "S": "Acme Band"
            },
            "SongTitle": {
                "S": "Happy Day"
            }
        },
        {
            "AlbumTitle": {
                "S": "Another Album Title"
            },
            "Awards": {
                "N": "8"
            },
            "Artist": {
                "S": "Acme Band"
            },
            "SongTitle": {
                "S": "PartiQL Rocks"
            }
        }
    ],
    "Count": 2,
    "ScannedCount": 2,
    "ConsumedCapacity": null
}
```

PartiQL for DynamoDB

Sie fragen ein Element über PartiQL ab, indem Sie die `Select`-Anweisung verwenden und den Partitionsschlüssel bereitstellen.

Linux

```
aws dynamodb execute-statement --statement "SELECT * FROM Music   \
                                            WHERE Artist='Acme Band'"
```

Windows CMD

```
aws dynamodb execute-statement --statement "SELECT * FROM Music WHERE Artist='Acme
 Band'"
```

Verwendung der `Select`-Anweisung gibt auf diese Weise alle Songs zurück, die mit diesem bestimmten `Artist` assoziiert sind.

```
{
    "Items": [
        {
            "AlbumTitle": {
                "S": "Updated Album Title"
            },
            "Awards": {
                "S": "10"
            },
            "Artist": {
                "S": "Acme Band"
            },
            "SongTitle": {
                "S": "Happy Day"
            }
        },
        {
            "AlbumTitle": {
                "S": "Another Album Title"
            },
            "Awards": {
                "S": "8"
            },
            "Artist": {
                "S": "Acme Band"
            },
            "SongTitle": {
                "S": "PartiQL Rocks"
            }
        }
    ]
}
```

Für weitere Informationen zum Abfragen von Daten mit PartiQL siehe PartiQL Select Statements (PartiQL-Select-Anweisungen).

AWS SDK

In den folgenden Codebeispielen wird gezeigt, wie eine DynamoDB-Tabelle mit einem AWS -SDK abgefragt wird.

.NET

SDK für .NET (v4)

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
/// <summary>
/// Queries the table for movies released in a particular year and
/// then displays the information for the movies returned.
/// </summary>
/// <param name="tableName">The name of the table to query.</param>
/// <param name="year">The release year for which we want to
/// view movies.</param>
/// <returns>The number of movies that match the query.</returns>
public async Task<int> QueryMoviesAsync(string tableName, int year)
{
    try
    {
        var movieTable = new TableBuilder(_amazonDynamoDB, tableName)
            .AddHashKey("year", DynamoDBEntryType.Numeric)
            .AddRangeKey("title", DynamoDBEntryType.String)
            .Build();

        var filter = new QueryFilter("year", QueryOperator.Equal, year);

        Console.WriteLine("\nFind movies released in: {year}:");

        var config = new QueryOperationConfig()
        {
            Limit = 10, // 10 items per page.
            Select = SelectValues.SpecificAttributes,
            AttributesToGet = new List<string>
            {
                "title",
                "year",
            },
            ConsistentRead = true,
```

```
                Filter = filter,
            };

            // Value used to track how many movies match the
            // supplied criteria.
            var moviesFound = 0;

            var search = movieTable.Query(config);
            do
            {
                var movieList = await search.GetNextSetAsync();
                moviesFound += movieList.Count;

                foreach (var movie in movieList)
                {
                    DisplayDocument(movie);
                }
            }
            while (!search.IsDone);

            return moviesFound;
        }
        catch (ResourceNotFoundException ex)
        {
            Console.WriteLine($"Table {tableName} was not found. {ex.Message}");
            return 0;
        }
        catch (AmazonDynamoDBException ex)
        {
            Console.WriteLine($"An Amazon DynamoDB error occurred while querying
movies. {ex.Message}");
            throw;
        }
        catch (Exception ex)
        {
            Console.WriteLine($"An error occurred while querying movies.
{ex.Message}");
            throw;
        }
    }
```

- Weitere API-Informationen finden Sie unter Abfragen in der AWS SDK für .NET -API-Referenz.

Bash

AWS CLI mit Bash-Skript

> **Note**
>
> Es gibt noch mehr dazu. GitHub Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
###############################################################################
# function dynamodb_query
#
# This function queries a DynamoDB table.
#
# Parameters:
#       -n table_name  -- The name of the table.
#       -k key_condition_expression -- The key condition expression.
#       -a attribute_names -- Path to JSON file containing the attribute names.
#       -v attribute_values -- Path to JSON file containing the attribute values.
#       [-p projection_expression]  -- Optional projection expression.
#
#  Returns:
#       The items as json output.
#  And:
#       0 - If successful.
#       1 - If it fails.
###############################################################################
function dynamodb_query() {
  local table_name key_condition_expression attribute_names attribute_values
 projection_expression response
  local option OPTARG # Required to use getopts command in a function.

  # ######################################
  # Function usage explanation
  ######################################
  function usage() {
    echo "function dynamodb_query"
```

```
   echo "Query a DynamoDB table."
   echo " -n table_name  -- The name of the table."
   echo " -k key_condition_expression -- The key condition expression."
   echo " -a attribute_names -- Path to JSON file containing the attribute
names."
   echo " -v attribute_values -- Path to JSON file containing the attribute
values."
   echo " [-p projection_expression]  -- Optional projection expression."
   echo ""
 }

 while getopts "n:k:a:v:p:h" option; do
   case "${option}" in
     n) table_name="${OPTARG}" ;;
     k) key_condition_expression="${OPTARG}" ;;
     a) attribute_names="${OPTARG}" ;;
     v) attribute_values="${OPTARG}" ;;
     p) projection_expression="${OPTARG}" ;;
     h)
       usage
       return 0
       ;;
     \?)
       echo "Invalid parameter"
       usage
       return 1
       ;;
   esac
 done
 export OPTIND=1

 if [[ -z "$table_name" ]]; then
   errecho "ERROR: You must provide a table name with the -n parameter."
   usage
   return 1
 fi

 if [[ -z "$key_condition_expression" ]]; then
   errecho "ERROR: You must provide a key condition expression with the -k
parameter."
   usage
   return 1
 fi
```

```
  if [[ -z "$attribute_names" ]]; then
    errecho "ERROR: You must provide a attribute names with the -a parameter."
    usage
    return 1
  fi

  if [[ -z "$attribute_values" ]]; then
    errecho "ERROR: You must provide a attribute values with the -v parameter."
    usage
    return 1
  fi

  if [[ -z "$projection_expression" ]]; then
    response=$(aws dynamodb query \
      --table-name "$table_name" \
      --key-condition-expression "$key_condition_expression" \
      --expression-attribute-names file://"$attribute_names" \
      --expression-attribute-values file://"$attribute_values")
  else
    response=$(aws dynamodb query \
      --table-name "$table_name" \
      --key-condition-expression "$key_condition_expression" \
      --expression-attribute-names file://"$attribute_names" \
      --expression-attribute-values file://"$attribute_values" \
      --projection-expression "$projection_expression")
  fi

  local error_code=${?}

  if [[ $error_code -ne 0 ]]; then
    aws_cli_error_log $error_code
    errecho "ERROR: AWS reports query operation failed.$response"
    return 1
  fi

  echo "$response"

  return 0
}
```

Die in diesem Beispiel verwendeten Dienstprogrammfunktionen.

```
#############################################################################
# function errecho
#
# This function outputs everything sent to it to STDERR (standard error output).
#############################################################################
function errecho() {
  printf "%s\n" "$*" 1>&2
}

#############################################################################
# function aws_cli_error_log()
#
# This function is used to log the error messages from the AWS CLI.
#
# See https://docs.aws.amazon.com/cli/latest/topic/return-codes.html#cli-aws-
help-return-codes.
#
# The function expects the following argument:
#         $1 - The error code returned by the AWS CLI.
#
#  Returns:
#          0: - Success.
#
#############################################################################
function aws_cli_error_log() {
  local err_code=$1
  errecho "Error code : $err_code"
  if [ "$err_code" == 1 ]; then
    errecho "  One or more S3 transfers failed."
  elif [ "$err_code" == 2 ]; then
    errecho "  Command line failed to parse."
  elif [ "$err_code" == 130 ]; then
    errecho "  Process received SIGINT."
  elif [ "$err_code" == 252 ]; then
    errecho "  Command syntax invalid."
  elif [ "$err_code" == 253 ]; then
    errecho "  The system environment or configuration was invalid."
  elif [ "$err_code" == 254 ]; then
    errecho "  The service returned an error."
  elif [ "$err_code" == 255 ]; then
    errecho "  255 is a catch-all error."
  fi
```

```
  return 0
}
```

- Weitere API-Informationen finden Sie unter Abfragen in der AWS CLI -Befehlsreferenz.

C++

SDK für C++

> **Note**
>
> Es gibt noch mehr GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
//! Perform a query on an Amazon DynamoDB Table and retrieve items.
/*!
  \sa queryItem()
  \param tableName: The table name.
  \param partitionKey: The partition key.
  \param partitionValue: The value for the partition key.
  \param projectionExpression: The projections expression, which is ignored if
 empty.
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
  */

/*
 * The partition key attribute is searched with the specified value. By default,
 all fields and values
 * contained in the item are returned. If an optional projection expression is
 * specified on the command line, only the specified fields and values are
 * returned.
 */

bool AwsDoc::DynamoDB::queryItems(const Aws::String &tableName,
                                  const Aws::String &partitionKey,
                                  const Aws::String &partitionValue,
                                  const Aws::String &projectionExpression,
                                  const Aws::Client::ClientConfiguration
 &clientConfiguration) {
```

```
   Aws::DynamoDB::DynamoDBClient dynamoClient(clientConfiguration);
   Aws::DynamoDB::Model::QueryRequest request;

   request.SetTableName(tableName);

   if (!projectionExpression.empty()) {
       request.SetProjectionExpression(projectionExpression);
   }

   // Set query key condition expression.
   request.SetKeyConditionExpression(partitionKey + "= :valueToMatch");

   // Set Expression AttributeValues.
   Aws::Map<Aws::String, Aws::DynamoDB::Model::AttributeValue> attributeValues;
   attributeValues.emplace(":valueToMatch", partitionValue);

   request.SetExpressionAttributeValues(attributeValues);

   bool result = true;

   // "exclusiveStartKey" is used for pagination.
   Aws::Map<Aws::String, Aws::DynamoDB::Model::AttributeValue>
exclusiveStartKey;
   do {
       if (!exclusiveStartKey.empty()) {
           request.SetExclusiveStartKey(exclusiveStartKey);
           exclusiveStartKey.clear();
       }
       // Perform Query operation.
       const Aws::DynamoDB::Model::QueryOutcome &outcome =
dynamoClient.Query(request);
       if (outcome.IsSuccess()) {
           // Reference the retrieved items.
           const Aws::Vector<Aws::Map<Aws::String,
Aws::DynamoDB::Model::AttributeValue>> &items = outcome.GetResult().GetItems();
           if (!items.empty()) {
               std::cout << "Number of items retrieved from Query: " <<
items.size()
                         << std::endl;
               // Iterate each item and print.
               for (const auto &item: items) {
                   std::cout
                           <<
"*************************************************"
```

```
                            << std::endl;
                    // Output each retrieved field and its value.
                    for (const auto &i: item)
                        std::cout << i.first << ": " << i.second.GetS() <<
 std::endl;
                }
            }
            else {
                std::cout << "No item found in table: " << tableName <<
 std::endl;
            }

            exclusiveStartKey = outcome.GetResult().GetLastEvaluatedKey();
        }
        else {
            std::cerr << "Failed to Query items: " <<
 outcome.GetError().GetMessage();
            result = false;
            break;
        }
    } while (!exclusiveStartKey.empty());

    return result;
}
```

- Weitere API-Informationen finden Sie unter Abfragen in der AWS SDK für C++ -API-Referenz.

CLI

AWS CLI

Beispiel 1: So fragen Sie eine Tabelle ab

Im folgenden Beispiel für `query` werden Elemente in der Tabelle `MusicCollection` abgefragt. Die Tabelle hat einen hash-and-range Primärschlüssel (`ArtistundSongTitle`), aber diese Abfrage spezifiziert nur den Hashschlüsselwert. Es werden Songtitel des Künstlers mit dem Namen „No One You Know“ zurückgegeben.

```
aws dynamodb query \
    --table-name MusicCollection \
```

```
    --projection-expression "SongTitle" \
    --key-condition-expression "Artist = :v1" \
    --expression-attribute-values file://expression-attributes.json \
    --return-consumed-capacity TOTAL
```

Inhalt von `expression-attributes.json`:

```
{
    ":v1": {"S": "No One You Know"}
}
```

Ausgabe:

```
{
    "Items": [
        {
            "SongTitle": {
                "S": "Call Me Today"
            },
            "SongTitle": {
                "S": "Scared of My Shadow"
            }
        }
    ],
    "Count": 2,
    "ScannedCount": 2,
    "ConsumedCapacity": {
        "TableName": "MusicCollection",
        "CapacityUnits": 0.5
    }
}
```

Weitere Informationen finden Sie unter Arbeiten mit Abfragen in DynamoDB im Entwicklerhandbuch zu Amazon DynamoDB.

Beispiel 2: So fragen Sie eine Tabelle mit strikt konsistenten Lesevorgängen ab und durchlaufen den Index in absteigender Reihenfolge

Im folgenden Beispiel wird dieselbe Abfrage wie im ersten Beispiel ausgeführt, die Ergebnisse werden jedoch in umgekehrter Reihenfolge zurückgegeben und es werden strikt konsistente Lesevorgänge verwendet.

```
aws dynamodb query \
    --table-name MusicCollection \
    --projection-expression "SongTitle" \
    --key-condition-expression "Artist = :v1" \
    --expression-attribute-values file://expression-attributes.json \
    --consistent-read \
    --no-scan-index-forward \
    --return-consumed-capacity TOTAL
```

Inhalt von `expression-attributes.json`:

```
{
    ":v1": {"S": "No One You Know"}
}
```

Ausgabe:

```
{
    "Items": [
        {
            "SongTitle": {
                "S": "Scared of My Shadow"
            }
        },
        {
            "SongTitle": {
                "S": "Call Me Today"
            }
        }
    ],
    "Count": 2,
    "ScannedCount": 2,
    "ConsumedCapacity": {
        "TableName": "MusicCollection",
        "CapacityUnits": 1.0
    }
}
```

Weitere Informationen finden Sie unter Arbeiten mit Abfragen in DynamoDB im Entwicklerhandbuch zu Amazon DynamoDB.

Beispiel 3: So filtern Sie bestimmte Ergebnisse heraus

Im folgende Beispiel wird `MusicCollection` abgefragt, Ergebnisse mit bestimmten Werten im `AlbumTitle`-Attribut werden jedoch ausgeschlossen. Beachten Sie, dass sich dies nicht auf `ScannedCount` oder `ConsumedCapacity` auswirkt, da der Filter angewendet wird, nachdem die Elemente gelesen wurden.

```
aws dynamodb query \
    --table-name MusicCollection \
    --key-condition-expression "#n1 = :v1" \
    --filter-expression "NOT (#n2 IN (:v2, :v3))" \
    --expression-attribute-names file://names.json \
    --expression-attribute-values file://values.json \
    --return-consumed-capacity TOTAL
```

Inhalt von `values.json`:

```
{
    ":v1": {"S": "No One You Know"},
    ":v2": {"S": "Blue Sky Blues"},
    ":v3": {"S": "Greatest Hits"}
}
```

Inhalt von `names.json`:

```
{
    "#n1": "Artist",
    "#n2": "AlbumTitle"
}
```

Ausgabe:

```
{
    "Items": [
        {
            "AlbumTitle": {
                "S": "Somewhat Famous"
            },
            "Artist": {
                "S": "No One You Know"
            },
            "SongTitle": {
                "S": "Call Me Today"
            }
```

```
        }
    ],
    "Count": 1,
    "ScannedCount": 2,
    "ConsumedCapacity": {
        "TableName": "MusicCollection",
        "CapacityUnits": 0.5
    }
}
```

Weitere Informationen finden Sie unter Arbeiten mit Abfragen in DynamoDB im Entwicklerhandbuch zu Amazon DynamoDB.

Beispiel 4: So rufen Sie nur die Elementanzahl ab

Das folgende Beispiel ruft eine Anzahl von Elementen ab, die der Abfrage entsprechen, ruft jedoch keines der Elemente selbst ab.

```
aws dynamodb query \
    --table-name MusicCollection \
    --select COUNT \
    --key-condition-expression "Artist = :v1" \
    --expression-attribute-values file://expression-attributes.json
```

Inhalt von `expression-attributes.json`:

```
{
    ":v1": {"S": "No One You Know"}
}
```

Ausgabe:

```
{
    "Count": 2,
    "ScannedCount": 2,
    "ConsumedCapacity": null
}
```

Weitere Informationen finden Sie unter Arbeiten mit Abfragen in DynamoDB im Entwicklerhandbuch zu Amazon DynamoDB.

Beispiel 5: So rufen Sie einen Index ab

Im folgenden Beispiel wird der lokale sekundäre Index `AlbumTitleIndex` abgefragt. Die Abfrage gibt alle Attribute aus der Basistabelle zurück, die in den lokalen sekundären Index projiziert wurden. Beachten Sie, dass Sie bei der Abfrage eines lokalen sekundären Index oder eines globalen sekundären Index auch den Namen der Basistabelle mit dem `table-name`-Parameter angeben müssen.

```
aws dynamodb query \
    --table-name MusicCollection \
    --index-name AlbumTitleIndex \
    --key-condition-expression "Artist = :v1" \
    --expression-attribute-values file://expression-attributes.json \
    --select ALL_PROJECTED_ATTRIBUTES \
    --return-consumed-capacity INDEXES
```

Inhalt von `expression-attributes.json`:

```
{
    ":v1": {"S": "No One You Know"}
}
```

Ausgabe:

```
{
    "Items": [
        {
            "AlbumTitle": {
                "S": "Blue Sky Blues"
            },
            "Artist": {
                "S": "No One You Know"
            },
            "SongTitle": {
                "S": "Scared of My Shadow"
            }
        },
        {
            "AlbumTitle": {
                "S": "Somewhat Famous"
            },
            "Artist": {
                "S": "No One You Know"
            },
```

```
            "SongTitle": {
                "S": "Call Me Today"
            }
        }
    ],
    "Count": 2,
    "ScannedCount": 2,
    "ConsumedCapacity": {
        "TableName": "MusicCollection",
        "CapacityUnits": 0.5,
        "Table": {
            "CapacityUnits": 0.0
        },
        "LocalSecondaryIndexes": {
            "AlbumTitleIndex": {
                "CapacityUnits": 0.5
            }
        }
    }
}
```

Weitere Informationen finden Sie unter Arbeiten mit Abfragen in DynamoDB im Entwicklerhandbuch zu Amazon DynamoDB.

- Weitere API-Informationen finden Sie unter Abfragen in der AWS CLI -Befehlsreferenz.

Go

SDK für Go V2

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
import (
 "context"
 "errors"
 "log"
 "time"
```

```
 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/attributevalue"
 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/expression"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb/types"
)

// TableBasics encapsulates the Amazon DynamoDB service actions used in the
 examples.
// It contains a DynamoDB service client that is used to act on the specified
 table.
type TableBasics struct {
 DynamoDbClient *dynamodb.Client
 TableName      string
}



// Query gets all movies in the DynamoDB table that were released in the
 specified year.
// The function uses the `expression` package to build the key condition
 expression
// that is used in the query.
func (basics TableBasics) Query(ctx context.Context, releaseYear int) ([]Movie,
 error) {
 var err error
 var response *dynamodb.QueryOutput
 var movies []Movie
 keyEx := expression.Key("year").Equal(expression.Value(releaseYear))
 expr, err := expression.NewBuilder().WithKeyCondition(keyEx).Build()
 if err != nil {
  log.Printf("Couldn't build expression for query. Here's why: %v\n", err)
 } else {
  queryPaginator := dynamodb.NewQueryPaginator(basics.DynamoDbClient,
 &dynamodb.QueryInput{
   TableName:                 aws.String(basics.TableName),
   ExpressionAttributeNames:  expr.Names(),
   ExpressionAttributeValues: expr.Values(),
   KeyConditionExpression:    expr.KeyCondition(),
  })
  for queryPaginator.HasMorePages() {
   response, err = queryPaginator.NextPage(ctx)
   if err != nil {
```

```
    log.Printf("Couldn't query for movies released in %v. Here's why: %v\n",
 releaseYear, err)
    break
   } else {
    var moviePage []Movie
    err = attributevalue.UnmarshalListOfMaps(response.Items, &moviePage)
    if err != nil {
     log.Printf("Couldn't unmarshal query response. Here's why: %v\n", err)
     break
    } else {
     movies = append(movies, moviePage...)
    }
   }
  }
 }
 return movies, err
}
```

Definieren Sie eine Movie-Struktur, die in diesem Beispiel verwendet wird.

```
import (
 "archive/zip"
 "bytes"
 "encoding/json"
 "fmt"
 "io"
 "log"
 "net/http"

 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/attributevalue"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb/types"
)

// Movie encapsulates data about a movie. Title and Year are the composite
 primary key
// of the movie in Amazon DynamoDB. Title is the sort key, Year is the partition
 key,
// and Info is additional data.
type Movie struct {
 Title string                `dynamodbav:"title"`
```

```
 Year  int                    `dynamodbav:"year"`
 Info  map[string]interface{} `dynamodbav:"info"`
}

// GetKey returns the composite primary key of the movie in a format that can be
// sent to DynamoDB.
func (movie Movie) GetKey() map[string]types.AttributeValue {
 title, err := attributevalue.Marshal(movie.Title)
 if err != nil {
  panic(err)
 }
 year, err := attributevalue.Marshal(movie.Year)
 if err != nil {
  panic(err)
 }
 return map[string]types.AttributeValue{"title": title, "year": year}
}

// String returns the title, year, rating, and plot of a movie, formatted for the
 example.
func (movie Movie) String() string {
 return fmt.Sprintf("%v\n\tReleased: %v\n\tRating: %v\n\tPlot: %v\n",
  movie.Title, movie.Year, movie.Info["rating"], movie.Info["plot"])
}
```

- Weitere API-Informationen finden Sie unter Query in der AWS SDK für Go -API-Referenz.

Java

SDK für Java 2.x

> **Note**
>
> Es gibt noch mehr GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

Fragt eine Tabelle ab mithilfe von DynamoDbClient.

```
import software.amazon.awssdk.regions.Region;
```

```
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.AttributeValue;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.model.QueryRequest;
import software.amazon.awssdk.services.dynamodb.model.QueryResponse;
import java.util.HashMap;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 *
 * To query items from an Amazon DynamoDB table using the AWS SDK for Java V2,
 * its better practice to use the
 * Enhanced Client. See the EnhancedQueryRecords example.
 */
public class Query {
    public static void main(String[] args) {
        final String usage = """

                Usage:
                    <tableName> <partitionKeyName> <partitionKeyVal>

                Where:
                    tableName - The Amazon DynamoDB table to put the item in (for
 example, Music3).
                    partitionKeyName - The partition key name of the Amazon
 DynamoDB table (for example, Artist).
                    partitionKeyVal - The value of the partition key that should
 match (for example, Famous Band).
                """;

        if (args.length != 3) {
            System.out.println(usage);
            System.exit(1);
        }

        String tableName = args[0];
        String partitionKeyName = args[1];
        String partitionKeyVal = args[2];
```

```
        // For more information about an alias, see:
        // https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/
Expressions.ExpressionAttributeNames.html
        String partitionAlias = "#a";

        System.out.format("Querying %s", tableName);
        System.out.println("");
        Region region = Region.US_EAST_1;
        DynamoDbClient ddb = DynamoDbClient.builder()
                .region(region)
                .build();

        int count = queryTable(ddb, tableName, partitionKeyName, partitionKeyVal,
 partitionAlias);
        System.out.println("There were " + count + "  record(s) returned");
        ddb.close();
    }

    public static int queryTable(DynamoDbClient ddb, String tableName, String
 partitionKeyName, String partitionKeyVal,
            String partitionAlias) {
        // Set up an alias for the partition key name in case it's a reserved
 word.
        HashMap<String, String> attrNameAlias = new HashMap<String, String>();
        attrNameAlias.put(partitionAlias, partitionKeyName);

        // Set up mapping of the partition name with the value.
        HashMap<String, AttributeValue> attrValues = new HashMap<>();
        attrValues.put(":" + partitionKeyName, AttributeValue.builder()
                .s(partitionKeyVal)
                .build());

        QueryRequest queryReq = QueryRequest.builder()
                .tableName(tableName)
                .keyConditionExpression(partitionAlias + " = :" +
 partitionKeyName)
                .expressionAttributeNames(attrNameAlias)
                .expressionAttributeValues(attrValues)
                .build();

        try {
            QueryResponse response = ddb.query(queryReq);
            return response.count();
```

```
        } catch (DynamoDbException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
        return -1;
    }
}
```

Fragt eine Tabelle mithilfe von `DynamoDbClient` und eines sekundären Index ab.

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.AttributeValue;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.model.QueryRequest;
import software.amazon.awssdk.services.dynamodb.model.QueryResponse;
import java.util.HashMap;
import java.util.Map;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 *
 * Create the Movies table by running the Scenario example and loading the Movie
 * data from the JSON file. Next create a secondary
 * index for the Movies table that uses only the year column. Name the index
 * **year-index**. For more information, see:
 *
 * https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/GSI.html
 */
public class QueryItemsUsingIndex {
    public static void main(String[] args) {
        String tableName = "Movies";
        Region region = Region.US_EAST_1;
        DynamoDbClient ddb = DynamoDbClient.builder()
                .region(region)
```

```
                .build();

        queryIndex(ddb, tableName);
        ddb.close();
    }

    public static void queryIndex(DynamoDbClient ddb, String tableName) {
        try {
            Map<String, String> expressionAttributesNames = new HashMap<>();
            expressionAttributesNames.put("#year", "year");
            Map<String, AttributeValue> expressionAttributeValues = new
 HashMap<>();
            expressionAttributeValues.put(":yearValue",
 AttributeValue.builder().n("2013").build());

            QueryRequest request = QueryRequest.builder()
                    .tableName(tableName)
                    .indexName("year-index")
                    .keyConditionExpression("#year = :yearValue")
                    .expressionAttributeNames(expressionAttributesNames)
                    .expressionAttributeValues(expressionAttributeValues)
                    .build();

            System.out.println("=== Movie Titles ===");
            QueryResponse response = ddb.query(request);
            response.items()
                    .forEach(movie ->
 System.out.println(movie.get("title").s()));

        } catch (DynamoDbException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
    }
}
```

- Weitere API-Informationen finden Sie unter Abfragen in der AWS SDK for Java 2.x -API-Referenz.

JavaScript

SDK für JavaScript (v3)

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

In diesem Beispiel wird der Dokument-Client verwendet, um die Arbeit mit Elementen in DynamoDB zu vereinfachen. Einzelheiten zur API finden Sie unter QueryCommand.

```
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";
import { QueryCommand, DynamoDBDocumentClient } from "@aws-sdk/lib-dynamodb";

const client = new DynamoDBClient({});
const docClient = DynamoDBDocumentClient.from(client);

export const main = async () => {
  const command = new QueryCommand({
    TableName: "CoffeeCrop",
    KeyConditionExpression:
      "OriginCountry = :originCountry AND RoastDate > :roastDate",
    ExpressionAttributeValues: {
      ":originCountry": "Ethiopia",
      ":roastDate": "2023-05-01",
    },
    ConsistentRead: true,
  });

  const response = await docClient.send(command);
  console.log(response);
  return response;
};
```

- Weitere Informationen finden Sie im AWS SDK für JavaScript -Entwicklerhandbuch.
- Weitere API-Informationen finden Sie unter Abfragen in der AWS SDK für JavaScript -API-Referenz.

SDK für JavaScript (v2)

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create DynamoDB document client
var docClient = new AWS.DynamoDB.DocumentClient({ apiVersion: "2012-08-10" });

var params = {
  ExpressionAttributeValues: {
    ":s": 2,
    ":e": 9,
    ":topic": "PHRASE",
  },
  KeyConditionExpression: "Season = :s and Episode > :e",
  FilterExpression: "contains (Subtitle, :topic)",
  TableName: "EPISODES_TABLE",
};

docClient.query(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data.Items);
  }
});
```

- Weitere Informationen finden Sie im AWS SDK für JavaScript -Entwicklerhandbuch.
- Weitere API-Informationen finden Sie unter Abfragen in der AWS SDK für JavaScript -API-Referenz.

Kotlin

SDK für Kotlin

> **Note**
>
> Es gibt noch mehr GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
suspend fun queryDynTable(
    tableNameVal: String,
    partitionKeyName: String,
    partitionKeyVal: String,
    partitionAlias: String,
): Int {
    val attrNameAlias = mutableMapOf<String, String>()
    attrNameAlias[partitionAlias] = partitionKeyName

    // Set up mapping of the partition name with the value.
    val attrValues = mutableMapOf<String, AttributeValue>()
    attrValues[":$partitionKeyName"] = AttributeValue.S(partitionKeyVal)

    val request =
        QueryRequest {
            tableName = tableNameVal
            keyConditionExpression = "$partitionAlias = :$partitionKeyName"
            expressionAttributeNames = attrNameAlias
            this.expressionAttributeValues = attrValues
        }

    DynamoDbClient.fromEnvironment { region = "us-east-1" }.use { ddb ->
        val response = ddb.query(request)
        return response.count
    }
}
```

- Weitere API-Informationen finden Sie unter Abfragen in der API–Referenz zum AWS SDK für Kotlin.

PHP

SDK für PHP

> **Note**
>
> Es gibt noch mehr GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
        $birthKey = [
            'Key' => [
                'year' => [
                    'N' => "$birthYear",
                ],
            ],
        ];
        $result = $service->query($tableName, $birthKey);

    public function query(string $tableName, $key)
    {
        $expressionAttributeValues = [];
        $expressionAttributeNames = [];
        $keyConditionExpression = "";
        $index = 1;
        foreach ($key as $name => $value) {
            $keyConditionExpression .= "#" . array_key_first($value) . " = :v
$index,";
            $expressionAttributeNames["#" . array_key_first($value)] =
 array_key_first($value);
            $hold = array_pop($value);
            $expressionAttributeValues[":v$index"] = [
                array_key_first($hold) => array_pop($hold),
            ];
        }
        $keyConditionExpression = substr($keyConditionExpression, 0, -1);
        $query = [
            'ExpressionAttributeValues' => $expressionAttributeValues,
            'ExpressionAttributeNames' => $expressionAttributeNames,
            'KeyConditionExpression' => $keyConditionExpression,
            'TableName' => $tableName,
        ];
```

```
        return $this->dynamoDbClient->query($query);
    }
```

- Weitere API-Informationen finden Sie unter Abfragen in der AWS SDK für PHP -API-Referenz.

PowerShell

Tools für PowerShell V4

Beispiel 1: Ruft eine Abfrage auf, die DynamoDB-Elemente mit dem angegebenen SongTitle Wert und Artist zurückgibt.

```
$invokeDDBQuery = @{
    TableName = 'Music'
    KeyConditionExpression = ' SongTitle = :SongTitle and Artist = :Artist'
    ExpressionAttributeValues = @{
        ':SongTitle' = 'Somewhere Down The Road'
        ':Artist' = 'No One You Know'
    } | ConvertTo-DDBItem
}
Invoke-DDBQuery @invokeDDBQuery | ConvertFrom-DDBItem
```

Ausgabe:

```
Name                           Value
----                           -----
Genre                          Country
Artist                         No One You Know
Price                          1.94
CriticRating                   9
SongTitle                      Somewhere Down The Road
AlbumTitle                     Somewhat Famous
```

- Weitere API-Informationen finden Sie unter Abfragen in der AWS -Tools für PowerShell -Cmdlet-Referenz (V4).

Tools für V5 PowerShell

Beispiel 1: Ruft eine Abfrage auf, die DynamoDB-Elemente mit dem angegebenen SongTitle Wert und Artist zurückgibt.

```
$invokeDDBQuery = @{
    TableName = 'Music'
    KeyConditionExpression = ' SongTitle = :SongTitle and Artist = :Artist'
    ExpressionAttributeValues = @{
        ':SongTitle' = 'Somewhere Down The Road'
        ':Artist' = 'No One You Know'
    } | ConvertTo-DDBItem
}
Invoke-DDBQuery @invokeDDBQuery | ConvertFrom-DDBItem
```

Ausgabe:

```
Name                           Value
----                           -----
Genre                          Country
Artist                         No One You Know
Price                          1.94
CriticRating                   9
SongTitle                      Somewhere Down The Road
AlbumTitle                     Somewhat Famous
```

- Weitere API-Informationen finden Sie unter Abfragen in der AWS -Tools für PowerShell -Cmdlet-Referenz (V5).

Python

SDK für Python (Boto3)

> **Note**
>
> Es gibt noch mehr dazu. GitHub Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

Fragen Sie Elemente mithilfe eines Schlüsselbedingungsausdrucks ab.

```
class Movies:
    """Encapsulates an Amazon DynamoDB table of movie data.

    Example data structure for a movie record in this table:
        {
```

```
            "year": 1999,
            "title": "For Love of the Game",
            "info": {
                "directors": ["Sam Raimi"],
                "release_date": "1999-09-15T00:00:00Z",
                "rating": 6.3,
                "plot": "A washed up pitcher flashes through his career.",
                "rank": 4987,
                "running_time_secs": 8220,
                "actors": [
                    "Kevin Costner",
                    "Kelly Preston",
                    "John C. Reilly"
                ]
            }
        }
    """

    def __init__(self, dyn_resource):
        """
        :param dyn_resource: A Boto3 DynamoDB resource.
        """
        self.dyn_resource = dyn_resource
        # The table variable is set during the scenario in the call to
        # 'exists' if the table exists. Otherwise, it is set by 'create_table'.
        self.table = None


    def query_movies(self, year):
        """
        Queries for movies that were released in the specified year.

        :param year: The year to query.
        :return: The list of movies that were released in the specified year.
        """
        try:
            response =
self.table.query(KeyConditionExpression=Key("year").eq(year))
        except ClientError as err:
            logger.error(
                "Couldn't query for movies released in %s. Here's why: %s: %s",
                year,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
```

```
            )
            raise
        else:
            return response["Items"]
```

Fragen Sie Elemente ab und projizieren Sie sie, um eine Teilmenge von Daten zurückzugeben.

```
class UpdateQueryWrapper:
    def __init__(self, table):
        self.table = table


    def query_and_project_movies(self, year, title_bounds):
        """
        Query for movies that were released in a specified year and that have
 titles
        that start within a range of letters. A projection expression is used
        to return a subset of data for each movie.

        :param year: The release year to query.
        :param title_bounds: The range of starting letters to query.
        :return: The list of movies.
        """
        try:
            response = self.table.query(
                ProjectionExpression="#yr, title, info.genres, info.actors[0]",
                ExpressionAttributeNames={"#yr": "year"},
                KeyConditionExpression=(
                    Key("year").eq(year)
                    & Key("title").between(
                        title_bounds["first"], title_bounds["second"]
                    )
                ),
            )
        except ClientError as err:
            if err.response["Error"]["Code"] == "ValidationException":
                logger.warning(
                    "There's a validation error. Here's the message: %s: %s",
                    err.response["Error"]["Code"],
                    err.response["Error"]["Message"],
```

```
                )
            else:
                logger.error(
                    "Couldn't query for movies. Here's why: %s: %s",
                    err.response["Error"]["Code"],
                    err.response["Error"]["Message"],
                )
                raise
        else:
            return response["Items"]
```

- Weitere API-Informationen finden Sie unter Abfragen in der API-Referenz zum AWS SDK für Python (Boto3).

Ruby

SDK für Ruby

> **Note**
>
> Es gibt noch mehr GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
class DynamoDBBasics
  attr_reader :dynamo_resource, :table

  def initialize(table_name)
    client = Aws::DynamoDB::Client.new(region: 'us-east-1')
    @dynamo_resource = Aws::DynamoDB::Resource.new(client: client)
    @table = @dynamo_resource.table(table_name)
  end

  # Queries for movies that were released in the specified year.
  #
  # @param year [Integer] The year to query.
  # @return [Array] The list of movies that were released in the specified year.
  def query_items(year)
    response = @table.query(
```

```
      key_condition_expression: '#yr = :year',
      expression_attribute_names: { '#yr' => 'year' },
      expression_attribute_values: { ':year' => year }
    )
  rescue Aws::DynamoDB::Errors::ServiceError => e
    puts("Couldn't query for movies released in #{year}. Here's why:")
    puts("\t#{e.code}: #{e.message}")
    raise
  else
    response.items
  end
```

- Weitere API-Informationen finden Sie unter Query in der AWS SDK für Ruby -API-Referenz.

Rust

SDK für Rust

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

Finden Sie die Filme, die im angegebenen Jahr gedreht wurden.

```
pub async fn movies_in_year(
    client: &Client,
    table_name: &str,
    year: u16,
) -> Result<Vec<Movie>, MovieError> {
    let results = client
        .query()
        .table_name(table_name)
        .key_condition_expression("#yr = :yyyy")
        .expression_attribute_names("#yr", "year")
        .expression_attribute_values(":yyyy",
 AttributeValue::N(year.to_string()))
        .send()
        .await?;
```

```
    if let Some(items) = results.items {
        let movies = items.iter().map(|v| v.into()).collect();
        Ok(movies)
    } else {
        Ok(vec![])
    }
}
```

- Weitere API-Informationen finden Sie unter Abfragen in der API-Referenz zum AWS SDK für Rust.

SAP ABAP

SDK für SAP ABAP

> **Note**
>
> Es gibt noch mehr dazu. GitHub Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
    TRY.
        " Query movies for a given year .
        DATA(lt_attributelist) = VALUE /aws1/
cl_dynattributevalue=>tt_attributevaluelist(
            ( NEW /aws1/cl_dynattributevalue( iv_n = |{ iv_year }| ) ) ).
        DATA(lt_key_conditions) = VALUE /aws1/cl_dyncondition=>tt_keyconditions(
          ( VALUE /aws1/cl_dyncondition=>ts_keyconditions_maprow(
          key = 'year'
          value = NEW /aws1/cl_dyncondition(
          it_attributevaluelist = lt_attributelist
          iv_comparisonoperator = |EQ|
          ) ) ) ).
        oo_result = lo_dyn->query(
          iv_tablename = iv_table_name
          it_keyconditions = lt_key_conditions ).
        DATA(lt_items) = oo_result->get_items( ).
        "You can loop over the results to get item attributes.
        LOOP AT lt_items INTO DATA(lt_item).
```

```
          DATA(lo_title) = lt_item[ key = 'title' ]-value.
          DATA(lo_year) = lt_item[ key = 'year' ]-value.
        ENDLOOP.
        DATA(lv_count) = oo_result->get_count( ).
        MESSAGE 'Item count is: ' && lv_count TYPE 'I'.
      CATCH /aws1/cx_dynresourcenotfoundex.
        MESSAGE 'The table or index does not exist' TYPE 'E'.
    ENDTRY.
```

- Weitere API-Informationen finden Sie unter Abfragen in der API-Referenz für das AWS SDK für SAP ABAP.

Swift

SDK für Swift

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
import AWSDynamoDB


    /// Get all the movies released in the specified year.
    ///
    /// - Parameter year: The release year of the movies to return.
    ///
    /// - Returns: An array of `Movie` objects describing each matching movie.
    ///
    func getMovies(fromYear year: Int) async throws -> [Movie] {
        do {
            guard let client = self.ddbClient else {
                throw MoviesError.UninitializedClient
            }

            let input = QueryInput(
                expressionAttributeNames: [
                    "#y": "year"
```

```
                ],
                expressionAttributeValues: [
                    ":y": .n(String(year))
                ],
                keyConditionExpression: "#y = :y",
                tableName: self.tableName
            )
            // Use "Paginated" to get all the movies.
            // This lets the SDK handle the 'lastEvaluatedKey' property in
"QueryOutput".

            let pages = client.queryPaginated(input: input)

            var movieList: [Movie] = []
            for try await page in pages {
                guard let items = page.items else {
                    print("Error: no items returned.")
                    continue
                }

                // Convert the found movies into `Movie` objects and return an
array
                // of them.

                for item in items {
                    let movie = try Movie(withItem: item)
                    movieList.append(movie)
                }
            }
            return movieList
        } catch {
            print("ERROR: getMovies:", dump(error))
            throw error
        }
    }
```

- Detaillierte API-Informationen finden Sie unter Query in der API-Referenz zum AWS -SDK für Swift.

Weitere DynamoDB-Beispiele finden Sie unter Codebeispiele für DynamoDB mit AWS SDKs.

Um einen globalen sekundären Index für Ihre Tabelle zu erstellen, fahren Sie mit Schritt 6: (Optional) Löschen der DynamoDB-Tabelle, um Ressourcen zu bereinigen fort.

Schritt 6: (Optional) Löschen der DynamoDB-Tabelle, um Ressourcen zu bereinigen

Wenn Sie die Amazon-DynamoDB-Tabelle, die Sie für das Tutorial verwendet haben, nicht mehr benötigen, können Sie sie löschen. Mit diesem Schritt wird sichergestellt, dass Ihnen keine Ressourcen in Rechnung gestellt werden, die Sie nicht nutzen. Sie können die DynamoDB-Konsole oder die verwenden, AWS CLI um die `Music` Tabelle zu löschen, in der Sie erstellt haben. Schritt 1: Erstellen einer Tabelle in DynamoDB

Weitere Informationen zu Tabellenoperationen in DynamoDB finden Sie unter Arbeiten mit Tabellen und Daten in DynamoDB.

AWS-Managementkonsole

So löschen Sie die Tabelle `Music` mithilfe der Konsole:

1. Öffnen Sie die DynamoDB-Konsole unter. https://console.aws.amazon.com/dynamodb/
2. Wählen Sie im linken Navigationsbereich Tables (Tabellen) aus.
3. Wählen Sie das Kontrollkästchen neben der Tabelle Musik in der Tabellenliste aus.
4. Wählen Sie Löschen aus.

AWS CLI

Das folgende AWS CLI Beispiel löscht die `Music` Tabelle mit. `delete-table`

```
aws dynamodb delete-table --table-name Music
```

AWS SDK

Mit den folgenden Codebeispielen wird demonstriert, wie Sie ein Element in einer DynamoDB-Tabelle unter Verwendung eines AWS -SDK löschen.

.NET

SDK für .NET (v4)

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
    /// <summary>
    /// Deletes a DynamoDB table.
    /// </summary>
    /// <param name="tableName">The name of the table to delete.</param>
    /// <returns>A Boolean value indicating the success of the operation.</
returns>
    public async Task<bool> DeleteTableAsync(string tableName)
    {
        try
        {
            var request = new DeleteTableRequest
            {
                TableName = tableName,
            };

            var response = await _amazonDynamoDB.DeleteTableAsync(request);

            Console.WriteLine($"Table {response.TableDescription.TableName}
 successfully deleted.");
            return true;

        }
        catch (ResourceNotFoundException ex)
        {
            Console.WriteLine($"Table {tableName} was not found and cannot be
 deleted. {ex.Message}");
            return false;
        }
        catch (AmazonDynamoDBException ex)
        {
            Console.WriteLine($"An Amazon DynamoDB error occurred while deleting
 table {tableName}. {ex.Message}");
```

```
            return false;
        }
        catch (Exception ex)
        {
            Console.WriteLine($"An error occurred while deleting table
 {tableName}. {ex.Message}");
            return false;
        }
    }
```

- Einzelheiten zur API finden Sie DeleteTablein der AWS SDK für .NET API-Referenz.

Bash

AWS CLI mit Bash-Skript

> **Note**
>
> Es gibt noch mehr dazu. GitHub Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
###############################################################################
# function dynamodb_delete_table
#
# This function deletes a DynamoDB table.
#
# Parameters:
#       -n table_name  -- The name of the table to delete.
#
#  Returns:
#       0 - If successful.
#       1 - If it fails.
###############################################################################
function dynamodb_delete_table() {
  local table_name response
  local option OPTARG # Required to use getopts command in a function.

  # bashsupport disable=BP5008
  function usage() {
```

```
  echo "function dynamodb_delete_table"
  echo "Deletes an Amazon DynamoDB table."
  echo " -n table_name  -- The name of the table to delete."
  echo ""
}

# Retrieve the calling parameters.
while getopts "n:h" option; do
  case "${option}" in
    n) table_name="${OPTARG}" ;;
    h)
      usage
      return 0
      ;;
    \?)
      echo "Invalid parameter"
      usage
      return 1
      ;;
  esac
done
export OPTIND=1

if [[ -z "$table_name" ]]; then
  errecho "ERROR: You must provide a table name with the -n parameter."
  usage
  return 1
fi

iecho "Parameters:\n"
iecho "    table_name:   $table_name"
iecho ""

response=$(aws dynamodb delete-table \
  --table-name "$table_name")

local error_code=${?}

if [[ $error_code -ne 0 ]]; then
  aws_cli_error_log $error_code
  errecho "ERROR: AWS reports delete-table operation failed.$response"
  return 1
fi
```

```
  return 0
}
```

Die in diesem Beispiel verwendeten Hilfsfunktionen.

```
###############################################################################
# function iecho
#
# This function enables the script to display the specified text only if
# the global variable $VERBOSE is set to true.
###############################################################################
function iecho() {
  if [[ $VERBOSE == true ]]; then
    echo "$@"
  fi
}

###############################################################################
# function errecho
#
# This function outputs everything sent to it to STDERR (standard error output).
###############################################################################
function errecho() {
  printf "%s\n" "$*" 1>&2
}

###############################################################################
# function aws_cli_error_log()
#
# This function is used to log the error messages from the AWS CLI.
#
# See https://docs.aws.amazon.com/cli/latest/topic/return-codes.html#cli-aws-
help-return-codes.
#
# The function expects the following argument:
#         $1 - The error code returned by the AWS CLI.
#
#  Returns:
#          0: - Success.
#
###############################################################################
function aws_cli_error_log() {
```

```
  local err_code=$1
  errecho "Error code : $err_code"
  if [ "$err_code" == 1 ]; then
    errecho "  One or more S3 transfers failed."
  elif [ "$err_code" == 2 ]; then
    errecho "  Command line failed to parse."
  elif [ "$err_code" == 130 ]; then
    errecho "  Process received SIGINT."
  elif [ "$err_code" == 252 ]; then
    errecho "  Command syntax invalid."
  elif [ "$err_code" == 253 ]; then
    errecho "  The system environment or configuration was invalid."
  elif [ "$err_code" == 254 ]; then
    errecho "  The service returned an error."
  elif [ "$err_code" == 255 ]; then
    errecho "  255 is a catch-all error."
  fi

  return 0
}
```

- Einzelheiten zur API finden Sie DeleteTablein der AWS CLI Befehlsreferenz.

C++

SDK für C++

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
//! Delete an Amazon DynamoDB table.
/*!
  \sa deleteTable()
  \param tableName: The DynamoDB table name.
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
*/
bool AwsDoc::DynamoDB::deleteTable(const Aws::String &tableName,
```

```
                                        const Aws::Client::ClientConfiguration
 &clientConfiguration) {
    Aws::DynamoDB::DynamoDBClient dynamoClient(clientConfiguration);

    Aws::DynamoDB::Model::DeleteTableRequest request;
    request.SetTableName(tableName);

    const Aws::DynamoDB::Model::DeleteTableOutcome &result =
 dynamoClient.DeleteTable(
            request);
    if (result.IsSuccess()) {
        std::cout << "Your table \""
                  << result.GetResult().GetTableDescription().GetTableName()
                  << " was deleted.\n";
    }
    else {
        std::cerr << "Failed to delete table: " << result.GetError().GetMessage()
                  << std::endl;
    }

    return result.IsSuccess();
}
```

- Einzelheiten zur API finden Sie DeleteTablein der AWS SDK für C++ API-Referenz.

CLI

AWS CLI

So löschen Sie eine Tabelle

Im folgenden Beispiel für `delete-table` wird die Tabelle `MusicCollection` gelöscht.

```
aws dynamodb delete-table \
    --table-name MusicCollection
```

Ausgabe:

```
{
    "TableDescription": {
        "TableStatus": "DELETING",
```

```
        "TableSizeBytes": 0,
        "ItemCount": 0,
        "TableName": "MusicCollection",
        "ProvisionedThroughput": {
            "NumberOfDecreasesToday": 0,
            "WriteCapacityUnits": 5,
            "ReadCapacityUnits": 5
        }
    }
}
```

Weitere Informationen finden Sie unter Löschen einer Tabelle im Entwicklerhandbuch zu Amazon DynamoDB.

- Einzelheiten zur API finden Sie DeleteTablein der AWS CLI Befehlsreferenz.

Go

SDK für Go V2

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
import (
 "context"
 "errors"
 "log"
 "time"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/attributevalue"
 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/expression"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb/types"
)

// TableBasics encapsulates the Amazon DynamoDB service actions used in the
 examples.
```

```
// It contains a DynamoDB service client that is used to act on the specified
 table.
type TableBasics struct {
 DynamoDbClient *dynamodb.Client
 TableName      string
}



// DeleteTable deletes the DynamoDB table and all of its data.
func (basics TableBasics) DeleteTable(ctx context.Context) error {
 _, err := basics.DynamoDbClient.DeleteTable(ctx, &dynamodb.DeleteTableInput{
  TableName: aws.String(basics.TableName)})
 if err != nil {
  log.Printf("Couldn't delete table %v. Here's why: %v\n", basics.TableName, err)
 }
 return err
}
```

- Einzelheiten zur API finden Sie DeleteTablein der AWS SDK für Go API-Referenz.

Java

SDK für Java 2.x

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.DeleteTableRequest;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
```

```
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 */

public class DeleteTable {
    public static void main(String[] args) {
        final String usage = """

                Usage:
                    <tableName>

                Where:
                    tableName - The Amazon DynamoDB table to delete (for example,
 Music3).

                **Warning** This program will delete the table that you specify!
                """;

        if (args.length != 1) {
            System.out.println(usage);
            System.exit(1);
        }

        String tableName = args[0];
        System.out.format("Deleting the Amazon DynamoDB table %s...\n",
 tableName);
        Region region = Region.US_EAST_1;
        DynamoDbClient ddb = DynamoDbClient.builder()
                .region(region)
                .build();

        deleteDynamoDBTable(ddb, tableName);
        ddb.close();
    }

    public static void deleteDynamoDBTable(DynamoDbClient ddb, String tableName)
 {
        DeleteTableRequest request = DeleteTableRequest.builder()
                .tableName(tableName)
                .build();

        try {
```

```
            ddb.deleteTable(request);

        } catch (DynamoDbException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
        System.out.println(tableName + " was successfully deleted!");
    }
}
```

- Einzelheiten zur API finden Sie DeleteTablein der AWS SDK for Java 2.x API-Referenz.

JavaScript

SDK für JavaScript (v3)

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
import { DeleteTableCommand, DynamoDBClient } from "@aws-sdk/client-dynamodb";

const client = new DynamoDBClient({});

export const main = async () => {
  const command = new DeleteTableCommand({
    TableName: "DecafCoffees",
  });

  const response = await client.send(command);
  console.log(response);
  return response;
};
```

- Einzelheiten zur API finden Sie DeleteTablein der AWS SDK für JavaScript API-Referenz.

SDK für JavaScript (v2)

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create the DynamoDB service object
var ddb = new AWS.DynamoDB({ apiVersion: "2012-08-10" });

var params = {
  TableName: process.argv[2],
};

// Call DynamoDB to delete the specified table
ddb.deleteTable(params, function (err, data) {
  if (err && err.code === "ResourceNotFoundException") {
    console.log("Error: Table not found");
  } else if (err && err.code === "ResourceInUseException") {
    console.log("Error: Table in use");
  } else {
    console.log("Success", data);
  }
});
```

- Weitere Informationen finden Sie im AWS SDK für JavaScript -Entwicklerhandbuch.
- Einzelheiten zur API finden Sie DeleteTablein der AWS SDK für JavaScript API-Referenz.

Kotlin

SDK für Kotlin

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
suspend fun deleteDynamoDBTable(tableNameVal: String) {
    val request =
        DeleteTableRequest {
            tableName = tableNameVal
        }

    DynamoDbClient.fromEnvironment { region = "us-east-1" }.use { ddb ->
        ddb.deleteTable(request)
        println("$tableNameVal was deleted")
    }
}
```

- API-Details finden Sie DeleteTablein der API-Referenz zum AWS SDK für Kotlin.

PHP

SDK für PHP

> **Note**
>
> Es gibt noch mehr dazu. GitHub Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
    public function deleteTable(string $TableName)
    {
        $this->customWaiter(function () use ($TableName) {
            return $this->dynamoDbClient->deleteTable([
                'TableName' => $TableName,
```

```
            ]);
        });
    }
```

- Einzelheiten zur API finden Sie DeleteTablein der AWS SDK für PHP API-Referenz.

PowerShell

Tools für PowerShell V4

Beispiel 1: Löscht die angegebene Tabelle. Sie werden um eine Bestätigung gebeten, bevor der Vorgang fortgesetzt wird.

```
Remove-DDBTable -TableName "myTable"
```

Beispiel 2: Löscht die angegebene Tabelle. Sie werden nicht zum Bestätigen aufgefordert, bevor der Vorgang fortgesetzt wird.

```
Remove-DDBTable -TableName "myTable" -Force
```

- Einzelheiten zur API finden Sie unter DeleteTable AWS -Tools für PowerShellCmdlet-Referenz (V4).

Tools für V5 PowerShell

Beispiel 1: Löscht die angegebene Tabelle. Sie werden um eine Bestätigung gebeten, bevor der Vorgang fortgesetzt wird.

```
Remove-DDBTable -TableName "myTable"
```

Beispiel 2: Löscht die angegebene Tabelle. Sie werden nicht zum Bestätigen aufgefordert, bevor der Vorgang fortgesetzt wird.

```
Remove-DDBTable -TableName "myTable" -Force
```

- Einzelheiten zur API finden Sie unter DeleteTable AWS -Tools für PowerShellCmdlet-Referenz (V5).

Python

SDK für Python (Boto3)

> **Note**
>
> Es gibt noch mehr dazu. GitHub Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
class Movies:
    """Encapsulates an Amazon DynamoDB table of movie data.

    Example data structure for a movie record in this table:
        {
            "year": 1999,
            "title": "For Love of the Game",
            "info": {
                "directors": ["Sam Raimi"],
                "release_date": "1999-09-15T00:00:00Z",
                "rating": 6.3,
                "plot": "A washed up pitcher flashes through his career.",
                "rank": 4987,
                "running_time_secs": 8220,
                "actors": [
                    "Kevin Costner",
                    "Kelly Preston",
                    "John C. Reilly"
                ]
            }
        }
    """

    def __init__(self, dyn_resource):
        """
        :param dyn_resource: A Boto3 DynamoDB resource.
        """
        self.dyn_resource = dyn_resource
        # The table variable is set during the scenario in the call to
        # 'exists' if the table exists. Otherwise, it is set by 'create_table'.
        self.table = None
```

```
    def delete_table(self):
        """
        Deletes the table.
        """
        try:
            self.table.delete()
            self.table = None
        except ClientError as err:
            logger.error(
                "Couldn't delete table. Here's why: %s: %s",
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
```

- Einzelheiten zur API finden Sie DeleteTablein AWS SDK for Python (Boto3) API Reference.

Ruby

SDK für Ruby

> **Note**
>
> Es gibt noch mehr dazu. GitHub Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
# Encapsulates an Amazon DynamoDB table of movie data.
class Scaffold
  attr_reader :dynamo_resource, :table_name, :table

  def initialize(table_name)
    client = Aws::DynamoDB::Client.new(region: 'us-east-1')
    @dynamo_resource = Aws::DynamoDB::Resource.new(client: client)
    @table_name = table_name
    @table = nil
    @logger = Logger.new($stdout)
    @logger.level = Logger::DEBUG
  end
```

```
  # Deletes the table.
  def delete_table
    @table.delete
    @table = nil
  rescue Aws::DynamoDB::Errors::ServiceError => e
    puts("Couldn't delete table. Here's why:")
    puts("\t#{e.code}: #{e.message}")
    raise
  end
```

- Einzelheiten zur API finden Sie DeleteTablein der AWS SDK für Ruby API-Referenz.

Rust

SDK für Rust

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
pub async fn delete_table(client: &Client, table: &str) ->
 Result<DeleteTableOutput, Error> {
    let resp = client.delete_table().table_name(table).send().await;

    match resp {
        Ok(out) => {
            println!("Deleted table");
            Ok(out)
        }
        Err(e) => Err(Error::Unhandled(e.into())),
    }
}
```

- Einzelheiten zur API finden Sie DeleteTablein der API-Referenz zum AWS SDK für Rust.

SAP ABAP

SDK für SAP ABAP

> **Note**
>
> Es gibt noch mehr dazu. GitHub Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
TRY.
    lo_dyn->deletetable( iv_tablename = iv_table_name ).
    " Wait till the table is actually deleted.
    lo_dyn->get_waiter( )->tablenotexists(
      iv_max_wait_time = 200
      iv_tablename     = iv_table_name ).
    MESSAGE 'Table ' && iv_table_name && ' deleted.' TYPE 'I'.
  CATCH /aws1/cx_dynresourcenotfoundex.
    MESSAGE 'The table ' && iv_table_name && ' does not exist' TYPE 'E'.
  CATCH /aws1/cx_dynresourceinuseex.
    MESSAGE 'The table cannot be deleted since it is in use' TYPE 'E'.
ENDTRY.
```

- Einzelheiten zur API finden Sie DeleteTablein der API-Referenz zum AWS SDK für SAP ABAP.

Swift

SDK für Swift

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
import AWSDynamoDB
```

```
    ///
    /// Deletes the table from Amazon DynamoDB.
    ///
    func deleteTable() async throws {
        do {
            guard let client = self.ddbClient else {
                throw MoviesError.UninitializedClient
            }

            let input = DeleteTableInput(
                tableName: self.tableName
            )
            _ = try await client.deleteTable(input: input)
        } catch {
            print("ERROR: deleteTable:", dump(error))
            throw error
        }
    }
```

- Einzelheiten zur API finden Sie DeleteTablein der API-Referenz zum AWS SDK für Swift.

Weitere DynamoDB-Beispiele finden Sie unter Codebeispiele für DynamoDB mit AWS SDKs.

Erfahren Sie mehr über DynamoDB

Weitere Informationen zu Amazon DynamoDB finden Sie in den folgenden Themen:

- Arbeiten mit Tabellen und Daten in DynamoDB
- Arbeiten mit Elementen und Attributen in DynamoDB
- Abfragen von Tabellen in DynamoDB
- Verwenden globaler sekundärer Indizes in DynamoDB
- Arbeiten mit Transaktionen
- In-Memory-Beschleunigung mit DynamoDB Accelerator (DAX)
- Programmieren mit DynamoDB und AWS SDKs

Generieren Sie Infrastrukturcode für Amazon DynamoDB mit Console-to-Code

Die Console-to-Code Funktion von Amazon Q Developer vereinfacht das Infrastrukturmanagement für Amazon DynamoDB, indem manuelle Schritte zur Tabellenerstellung in reproduzierbaren Automatisierungscode umgewandelt werden. Diese Funktion hilft Entwicklern, die Konfiguration von Datenbankressourcen in ihren Umgebungen effizient zu skalieren. Weitere Informationen finden Sie unter Automatisieren AWS-Services mit Amazon Q Developer Console-to-Code.

Console-to-Code erfasst detaillierte DynamoDB-Tabellenkonfigurationen, einschließlich Partitionsschlüssel, Sortierschlüssel, bereitgestellte Durchsatzeinstellungen und Sekundärindizes, und konvertiert diese in präzise Vorlagen. infrastructure-as-code Mithilfe generativer KI stellt das Tool sicher, dass der generierte Code die während Ihres Konsolen-Workflows festgelegte Parameterkompatibilität beibehält.

Entwickler können DynamoDB-Infrastrukturcode in verschiedenen Formaten generieren, z. B.:

- AWS Cloud Development Kit (AWS CDK) für Python und Java TypeScript
- CloudFormation in YAML oder JSON

Dieser Ansatz bietet Teams folgende Möglichkeiten:

- Standardisieren des Datenbankressourcenmanagements
- Implementieren einer versionskontrollierten Infrastruktur
- Reduzieren von manuellen Konfigurationsfehlern

Console-to-Code für Amazon DynamoDB ist in allen AWS Handelsregionen verfügbar und bietet eine leistungsstarke Lösung für die Umwandlung manueller Konfigurationen in automatisierten, reproduzierbaren Infrastrukturcode.

Funktionsweise

Bei der Verwendung Console-to-Code mit DynamoDB umfasst der Prozess in der Regel:

1. Prototyping in der Konsole: Verwenden Sie die DynamoDB-Konsole, um Ressourcen wie Tabellen zu erstellen und zu konfigurieren. Unter Connect to Amazon DynamoDB finden Sie weitere Informationen.

2. Aktionen aufzeichnen — Console-to-Code zeichnet diese Aktionen auf, während Sie sie ausführen.
3. Codegenerierung: Die Funktion nutzt die generativen KI-Funktionen von Amazon Q Developer, um Ihre Konsolenaktionen in wiederverwendbaren Code in Ihrem bevorzugten Format umzuwandeln.
4. Code-Anpassung: Sie können diesen Code dann kopieren oder herunterladen und ihn weiter an Ihre Produktions-Workloads anpassen.

Vorteile der Verwendung Console-to-Code mit DynamoDB

Vereinfachte Automatisierung

Konvertieren Sie die manuelle Erstellung und Konfiguration von DynamoDB-Tabellen mit einem einzigen Klick in wiederverwendbaren Code.

Best Practices

Der generierte Code folgt AWS bewährten Methoden für zuverlässige Bereitstellungen.

Brücke zwischen Konsole und Code

Sie müssen sich nicht zwischen der Verwendung von AWS-Managementkonsole oder Infrastructure-as-Code (IaC) entscheiden. Sie können beide Ansätze zusammen verwenden.

Beschleunigte Entwicklung

Schneller Einstieg mit Automatisierungscode, der für den Produktionseinsatz weiter angepasst werden kann.

Beispielanwendungsfälle

- Erstellen von DynamoDB-Tabellen mit bestimmten Attributen, Schlüsseln und Kapazitätseinstellungen
- Einrichtung von globalen sekundären Indizes und lokalen sekundären Indizes
- Konfiguration von Auto-Scaling-Richtlinien für DynamoDB-Tabellen
- Einrichten von Konfigurationen für Backup und Wiederherstellung
- Erstellen und Verwalten von DynamoDB Streams

Erste Schritte

So starten Sie die Verwendung Console-to-Code mit DynamoDB:

1. Melden Sie sich bei der an AWS-Managementkonsole und öffnen Sie die DynamoDB-Konsole unter. https://console.aws.amazon.com/dynamodbv2/
2. Beginnen Sie mit der Erstellung oder Änderung von DynamoDB-Ressourcen über die Konsolenschnittstelle.
3. Verwenden Sie die Console-to-Code Funktion, um Code für Ihre Aktionen in Ihrem bevorzugten Format zu generieren.
4. Kopieren Sie den generierten Code oder laden Sie ihn herunter und passen Sie ihn nach Bedarf an Ihre spezifischen Anforderungen an.

Weitere Informationen und detaillierte Anweisungen zur Verwendung Console-to-Code finden Sie unter Automatisieren AWS-Services mit Amazon Q Developer Console-to-Code im Amazon Q Developer User Guide.

Amazon DynamoDB: Funktionsweise

Die folgenden Abschnitte vermitteln eine Übersicht über Amazon-DynamoDB-Servicekomponenten und darüber, wie sie interagieren.

Themen

- Spickzettel für DynamoDB
- Kernkomponenten von Amazon DynamoDB
- DynamoDB API
- Unterstützte Datentypen und Benennungsregeln in Amazon DynamoDB
- DynamoDB-Tabellenklassen
- Partitionen und Datenverteilung in DynamoDB
- Erfahren Sie, wie Sie von SQL zu NoSQL wechseln.
- Lernressourcen und Tools für Amazon DynamoDB

Spickzettel für DynamoDB

Dieser Spickzettel bietet eine Kurzreferenz für die Arbeit mit Amazon DynamoDB und seinen verschiedenen Funktionen. AWS SDKs

Erstes Einrichten von

1. Melden Sie sich an für. AWS
2. Rufen Sie einen AWS -Zugriffsschlüssel für den programmgesteuerten Zugriff auf DynamoDB ab.
3. Konfigurieren Sie Ihre DynamoDB-Anmeldeinformationen.

Weitere Informationen finden Sie auch unter:

- Einrichten von DynamoDB (Webservice)
- Erste Schritte mit DynamoDB
- Grundlegender Überblick über die zentralen Komponenten

SDK oder CLI

Wählen Sie Ihr bevorzugtes SDK aus oder richten Sie die AWS CLI ein.

> **Note**
>
> Wenn Sie AWS CLI unter Windows den verwenden, wird ein umgekehrter Schrägstrich (\), der sich nicht in einem Anführungszeichen befindet, als Zeilenumbruch behandelt. Außerdem müssen Sie alle Anführungszeichen und geschweiften Klammern innerhalb anderer Anführungszeichen mit einem Escape-Zeichen versehen. Ein Beispiel finden Sie auf der Registerkarte Windows unter „Erstellen einer Tabelle“ im nächsten Abschnitt.

Weitere Informationen finden Sie auch unter:

- AWS CLI mit DynamoDB
- Erste Schritte mit DynamoDB – Schritt 2

Grundlegende Aktionen

In diesem Abschnitt finden Sie Code für grundlegende DynamoDB-Aufgaben. Weitere Informationen zu diesen Aufgaben finden Sie unter Erste Schritte mit DynamoDB und unter. AWS SDKs

Erstellen einer Tabelle

Default

```
aws dynamodb create-table \
    --table-name Music \
    --attribute-definitions \
        AttributeName=Artist,AttributeType=S \
        AttributeName=SongTitle,AttributeType=S \
    --key-schema AttributeName=Artist,KeyType=HASH
 AttributeName=SongTitle,KeyType=RANGE \
    --billing-mode PAY_PER_REQUEST \
    --table-class STANDARD
```

Windows

```
aws dynamodb create-table ^
```

```
    --table-name Music ^
    --attribute-definitions ^
        AttributeName=Artist,AttributeType=S ^
        AttributeName=SongTitle,AttributeType=S ^
    --key-schema AttributeName=Artist,KeyType=HASH
AttributeName=SongTitle,KeyType=RANGE ^
    --billing-mode PAY_PER_REQUEST ^
    --table-class STANDARD
```

Schreiben eines Elements in eine Tabelle

```
aws dynamodb put-item \ --table-name Music \ --item file://item.json
```

Lesen eines Elements aus einer Tabelle

```
aws dynamodb get-item \ --table-name Music \ --item file://item.json
```

Löschen eines Elements aus einer Tabelle

```
aws dynamodb delete-item --table-name Music --key file://key.json
```

Tabellen abfragen

```
aws dynamodb query --table-name Music
--key-condition-expression "ArtistName=:Artist and SongName=:Songtitle"
```

Löschen einer Tabelle

```
aws dynamodb delete-table --table-name Music
```

Auflisten von Tabellennamen

```
aws dynamodb list-tables
```

Benennungsregeln

- Alle Namen müssen mit UTF-8 kodiert werden und die Groß- und Kleinschreibung muss beachtet werden.

- Tabellen- und Indexnamen müssen zwischen 3 und 255 Zeichen lang sein und dürfen nur folgende Zeichen enthalten:
  - `a-z`
  - `A-Z`
  - `0-9`
  - `_` (Unterstrich)
  - `-` (Bindestrich)
  - `.` (Punkt)
- Attributnamen müssen mindestens ein Zeichen lang und dürfen nicht größer als 64 KB sein.

Weitere Informationen finden Sie unter Benennungsregeln.

Grundlegende Informationen zu Service Quotas

Lese- und Schreibeinheiten

- Lesekapazitätseinheit (Read capacity unit, RCU) – Ein strikt konsistenter Lesevorgang pro Sekunde oder zwei letztendlich konsistente Lesevorgänge pro Sekunde für Elemente mit einer Größe von bis zu 4 KB.
- Schreibkapazitätseinheit (Write capacity unit, WCU) – Ein Schreibvorgang pro Sekunde für Elemente mit einer Größe von bis zu 1 KB.

Tabellen-Limits

- Tabellengröße – Es gibt praktisch kein Limit für die Tabellengröße. Tabellen sind in Bezug auf die Anzahl von Elementen oder die Anzahl von Bytes unbeschränkt.
- Anzahl der Tabellen — Für jedes AWS Konto gibt es ein anfängliches Kontingent von 2.500 Tabellen pro AWS Region.
- Seitengrößenbeschränkung für Abfrage und Scan – Es gibt ein Limit von 1 MB pro Seite, pro Abfrage oder Scan. Wenn Ihre Abfrageparameter oder die Scan-Operation für eine Tabelle mehr als 1 MB an Daten ergeben, gibt DynamoDB die ersten übereinstimmenden Elemente zurück. Es wird auch eine Eigenschaft `LastEvaluatedKey` zurückgegeben, die Sie in einer neuen Anforderung verwenden können, um die nächste Seite zu lesen.

Indizes

- Lokale sekundäre Indizes (LSIs) — Sie können maximal fünf lokale sekundäre Indizes definieren. LSIs sind in erster Linie nützlich, wenn ein Index eine starke Konsistenz mit der Basistabelle aufweisen muss.
- Globale Sekundärindizes (GSIs) — Es gibt ein Standardkontingent von 20 globalen Sekundärindizes pro Tabelle.
- Projizierte sekundäre Indexattribute pro Tabelle – Sie können insgesamt bis zu 100 Attribute in alle lokalen und globalen sekundären Indizes einer Tabelle projizieren. Dies gilt nur für vom Benutzer angegebene, projizierte Attribute.

Partitionsschlüssel

- Die Mindestlänge eines Partitionsschlüsselwerts beträgt 1 Byte. Die maximale Länge beträgt 2 048 Byte.
- Es gibt praktisch keine Einschränkung in Bezug auf die Anzahl von eindeutigen Partitionsschlüsselwerten, weder für Tabellen noch für sekundäre Indizes.
- Die Mindestlänge eines Sortierschlüsselwerts beträgt 1 Byte. Die maximale Länge beträgt 1 024 Byte.
- Im Prinzip gibt es praktisch keine Einschränkung in Bezug auf die Anzahl von eindeutigen Sortierschlüsselwerten pro Partitionsschlüsselwert. Eine Ausnahme bilden Tabellen mit sekundären Indizes.

Weitere Informationen zu sekundären Indizes sowie zum Entwurf von Partitionsschlüsseln und Sortierschlüsseln finden Sie unter Bewährte Methoden.

Grenzwerte für häufig verwendete Datentypen

- Zeichenfolge – Die Länge einer Zeichenfolge ist durch die maximale Elementgröße von 400 KB beschränkt. Die Zeichenfolgen sind Unicode mit binärer UTF-8-Kodierung.
- Zahl – Eine Zahl kann bis zu 38 Nachkommastellen besitzen und positiv, negativ oder null sein.
- Binär – Die Länge eines binären Werts ist durch die maximale Elementgröße von 400 KB beschränkt. Anwendungen, die binäre Attribute verwenden, müssen die Daten vor dem Senden an DynamoDB mit der base64-Verschlüsselung kodieren.

Eine Liste der unterstützten Datentypen finden Sie unter Datentypen. Weitere Informationen finden Sie unter Service Quotas.

Elemente, Attribute und Ausdrucksparameter

Die maximale Elementgröße in DynamoDB beträgt 400 KB. Darin inbegriffen sind die binäre Länge (UTF-8-Länge) des Attributnamens und die binäre Länge (UTF-8-Länge) des Attributwerts. Der Attributname wird bei der Größenbeschränkung mit eingerechnet.

Es gibt keine Beschränkungen in Bezug auf die Anzahl der Werte in einer Liste, einer Zuordnung oder einem Satz, solange das Element, das die Werte enthält, das Limit der Elementgröße von 400 KB einhält.

Für Ausdrucksparameter beträgt die maximale Länge von Ausdruckszeichenfolgen 4 KB.

Weitere Informationen zu Elementgröße, Attributen und Ausdrucksparametern finden Sie unter Service Quotas.

Weitere Informationen

- Sicherheit
- Überwachung und Protokollierung
- Arbeiten mit Datenströmen
- Backups und Wiederherstellung Point-in-time
- Integration mit anderen AWS Diensten
- API-Referenz
- Architekturzentrum: Bewährte Methoden für Datenbanken
- Video-Tutorials
- DynamoDB-Forum

Kernkomponenten von Amazon DynamoDB

In DynamoDB sind Tabellen, Elemente und Attribute die zentralen Komponenten, mit denen Sie arbeiten. Eine Tabelle ist eine Sammlung von Elementen und jedes Element wiederum eine Sammlung von Attributen. DynamoDB nutzt Primärschlüssel, um jedes Element in einer Tabelle eindeutig zu identifizieren. Sie können DynamoDB Streams verwenden, um Datenänderungsereignisse in DynamoDB-Tabellen zu erfassen.

Es gibt Beschränkungen in DynamoDB. Weitere Informationen finden Sie unter Kontingente in Amazon DynamoDB.

Das folgende Video gibt Ihnen einen einführenden Einblick in Tabellen, Elemente und Attribute.

Tabellen, Elemente und Attribute

Tabellen, Elemente und Attribute

People

Primary key Partition key: PersonID	Attributes			
101	LastName	FirstName	Phone	
	Smith	Fred	555-4321	
102	LastName	FirstName	Address	
	Jones	Mary	{"Street":"123 Main","City":"Anytown","State":"OH","ZIPCode":12345}	
103	LastName	FirstName	FavoriteColor	Address
	Stephens	Howard	Blue	{"Street":"123 Main","City":"London","PostalCode":"ER3 5K8"}

Folgendes sind die grundlegenden DynamoDB-Komponenten:

- Tabellen – Ähnlich wie bei anderen Datenbankmanagementsystemen werden auch die Daten von DynamoDB in Tabellen gespeichert. Eine Tabelle ist eine Sammlung von Daten. Ein Beispiel ist die folgende Tabelle People, die Sie zum Speichern von persönlichen Kontaktinformationen zu Freunden, Familienmitgliedern oder anderen Personen verwenden könnten. Sie könnten auch eine Tabelle namens Cars haben, um Informationen über Fahrzeuge zu pflegen.
- Elemente - Jede Tabelle enthält keine oder mehrere Elemente. Ein Element ist eine Gruppe von Attributen, die unter allen anderen Elementen eindeutig identifizierbar ist. In einer People-Tabelle stellt jedes Element eine Person dar. In einer Cars-Tabelle stellt jedes Element ein Fahrzeug dar. Elemente in DynamoDB gleichen in vielerlei Hinsicht Zeilen, Datensätzen oder Tupel in anderen Datenbanksystemen. Bei DynamoDB gibt es keine Beschränkungen hinsichtlich Anzahl der Elemente, die in einer Tabelle gespeichert werden können.
- Attribute – Jedes Element besteht aus einem oder mehreren Attributen. Ein Attribut ist ein grundlegendes Datenelement, das nicht weiter untergliedert werden muss. Beispielsweise enthält ein Element in einer Personentabelle Attribute mit der Bezeichnung PersonID LastNameFirstName,, usw. In einer Department-Tabelle könnte ein Element Attribute wie

beispielsweise DepartmentID, Name, Manager usw. aufweisen. In DynamoDB gleichen Attribute in vielerlei Hinsicht Feldern oder Spalten in anderen Datenbanksystemen.

Das folgende Diagramm zeigt eine Tabelle mit dem Namen People mit einigen Beispielelementen und -attributen.

```
People

{
    "PersonID": 101,
    "LastName": "Smith",
    "FirstName": "Fred",
    "Phone": "555-4321"
}

{
    "PersonID": 102,
    "LastName": "Jones",
    "FirstName": "Mary",
    "Address": {
                "Street": "123 Main",
                "City": "Anytown",
                "State": "OH",
                "ZIPCode": 12345
    }
}

{
    "PersonID": 103,
    "LastName": "Stephens",
    "FirstName": "Howard",
    "Address": {
                "Street": "123 Main",
                "City": "London",
                "PostalCode": "ER3 5K8"
    },
    "FavoriteColor": "Blue"
}
```

Beachten Sie im Hinblick auf die Tabelle People Folgendes:

- Jedes Element in der Tabelle verfügt über einen eindeutigen Bezeichner oder Primärschlüssel, der das Element von allen anderen Elementen in der Tabelle unterscheidet. In der Tabelle People besteht der Primärschlüssel aus einem Attribut (PersonID).
- Mit Ausnahme des Primärschlüssels ist die Tabelle People schemalos. Dies bedeutet, dass weder die Attribute noch deren Datentypen im Vorfeld definiert werden müssen. Jedes Element kann über eigene eindeutige Attribute verfügen.
- Die meisten Attribute sind skalar, das heißt, sie können nur einen Wert annehmen. Zeichenfolgen und Zahlen sind allgemeine Beispiele für skalare Werte.
- Einige Elemente verfügen über ein verschachteltes Attribut (Address). DynamoDB unterstützt verschachtelte Attribute bis zu einer Tiefe von 32 Ebenen.

Im Folgenden finden Sie eine weitere Beispieltabelle mit dem Namen Music, die Sie verwenden können, um Ihre Musiksammlung zu erfassen.

```
Music

{
    "Artist": "No One You Know",
    "SongTitle": "My Dog Spot",
    "AlbumTitle": "Hey Now",
    "Price": 1.98,
    "Genre": "Country",
    "CriticRating": 8.4
}

{
    "Artist": "No One You Know",
    "SongTitle": "Somewhere Down The Road",
    "AlbumTitle": "Somewhat Famous",
    "Genre": "Country",
    "CriticRating": 8.4,
    "Year": 1984
}

{
    "Artist": "The Acme Band",
    "SongTitle": "Still in Love",
    "AlbumTitle": "The Buck Starts Here",
    "Price": 2.47,
```

```
    "Genre": "Rock",
    "PromotionInfo": {
        "RadioStationsPlaying": [
            "KHCR",
            "KQBX",
            "WTNR",
            "WJJH"
        ],
        "TourDates": {
            "Seattle": "20150622",
            "Cleveland": "20150630"
        },
        "Rotation": "Heavy"
    }
}

{
    "Artist": "The Acme Band",
    "SongTitle": "Look Out, World",
    "AlbumTitle": "The Buck Starts Here",
    "Price": 0.99,
    "Genre": "Rock"
}
```

Beachten Sie im Hinblick auf die Tabelle Music Folgendes:

- Der Primärschlüssel für Musik besteht aus zwei Attributen (Künstler und SongTitle). Jedes Element in der Tabelle muss über diese beiden Attribute verfügen. Die Kombination von Artist und SongTitleunterscheidet jedes Element in der Tabelle von allen anderen.
- Mit Ausnahme des Primärschlüssels ist die Tabelle Music schemalos. Dies bedeutet, dass weder die Attribute noch deren Datentypen im Vorfeld definiert werden müssen. Jedes Element kann über eigene eindeutige Attribute verfügen.
- Eines der Elemente hat ein verschachteltes Attribut (PromotionInfo), das andere verschachtelte Attribute enthält. DynamoDB unterstützt bis zu 32 Ebenen verschachtelter Attribute.

Weitere Informationen finden Sie unter Arbeiten mit Tabellen und Daten in DynamoDB.

Primärschlüssel

Wenn Sie eine Tabelle erstellen, müssen Sie außer dem Tabellennamen auch den Primärschlüssel der Tabelle angeben. Der Primärschlüssel identifiziert jedes Element in der Tabelle eindeutig, sodass keine zwei Elemente denselben Schlüssel haben können.

DynamoDB unterstützt zwei verschiedene Arten von Primärschlüsseln:

- Partitionsschlüssel – Ein einfacher Primärschlüssel bestehend aus einem Attribut, das als Partitionsschlüssel bezeichnet wird.

  DynamoDB verwendet den Wert des Partitionsschlüssels als Eingabe für eine interne Hash-Funktion. Die Ausgabe der Hash-Funktion bestimmt die Partition (physischer interner Speicher von DynamoDB), in der das Element gespeichert wird.

  In einer Tabelle mit nur einem Partitionsschlüssel können zwei Elemente in einer Tabelle nicht den gleichen Partitionsschlüsselwert haben.

  Die Tabelle People, die in Tabellen, Elemente und Attribute beschrieben wird, ist ein Beispiel für eine Tabelle mit einem einfachen Primärschlüssel (PersonID). Sie können direkt auf jedes Element in der Tabelle Personen zugreifen, indem Sie den PersonIdWert für dieses Element angeben.
- Partitionsschlüssel und Sortierschlüssel – Diese Schlüssel werden als zusammengesetzter Primärschlüssel bezeichnet, da er aus zwei Attributen besteht. Das erste Attribut ist der Partitionsschlüssel und das zweite der Sortierschlüssel.

  DynamoDB verwendet den Wert des Partitionsschlüssels als Eingabe für eine interne Hash-Funktion. Die Ausgabe der Hash-Funktion bestimmt die Partition (physischer interner Speicher von DynamoDB), in der das Element gespeichert wird. Alle Elemente mit dem gleichen Partitionsschlüsselwert werden zusammen gespeichert, und zwar sortiert nach Sortierschlüsselwert.

  In einer Tabelle, die über einen Partitionsschlüssel und einen Sortierschlüssel verfügt, ist es möglich, dass mehrere Elemente denselben Partitionsschlüsselwert aufweisen. Diese Elemente müssen aber verschiedene Sortierschlüsselwerte aufweisen.

  Die unter beschriebene Tabelle Music Tabellen, Elemente und Attribute ist ein Beispiel für eine Tabelle mit einem zusammengesetzten Primärschlüssel (Artist und SongTitle). Sie können direkt auf jedes Element in der Tabelle Musik zugreifen, wenn Sie den Interpreten und die SongTitleWerte für dieses Element angeben.

Mit einem zusammengesetzten Primärschlüssel erhalten Sie noch mehr Flexibilität bei der Abfrage von Daten. Wenn Sie beispielsweise nur den Wert für Artist angeben, ruft DynamoDB alle Songs dieses Interpreten ab. Um nur eine Teilmenge der Songs eines bestimmten Interpreten abzurufen, können Sie einen Wert für Artist zusammen mit einem Wertebereich für SongTitleangeben.

> **Note**
>
> Der Partitionsschlüssel eines Elements wird auch als Hash-Attribut bezeichnet. Der Begriff Hash-Attribut leitet sich von der Verwendung einer internen Hash-Funktion in DynamoDB ab, durch die Datenelemente basierend auf ihren Partitionsschlüsselwerten gleichmäßig auf die Partitionen verteilt werden.
> Der Sortierschlüssel eines Elements wird auch als Bereichsattribut bezeichnet. Der Begriff Bereichsattribut bezieht sich auf die Art und Weise, wie DynamoDB Elemente mit demselben Partitionsschlüssel physisch nah beieinander speichert, und zwar sortiert nach dem Sortierschlüsselwert.

Jedes Primärschlüsselattribut muss ein Skalarwert sein (das bedeutet, dass es nur einen einzigen Wert annehmen kann). Die einzigen Datentypen, die für Primärschlüsselattribute zulässig sind, sind Zeichenfolge, Zahl oder Binärwert. Es gibt keine Einschränkungen für andere Nicht-Schlüsselattribute.

Sekundäre Indizes

Sie können einen oder mehrere sekundäre Indizes für eine Tabelle erstellen. Ein Sekundärindex ermöglicht – ergänzend zu Abfragen über den Primärschlüssel – das Abfragen der Daten in der Tabelle über einen alternativen Schlüssel. Für DynamoDB ist die Verwendung von Indexen nicht erforderlich. Sie ermöglichen Ihren Anwendungen jedoch mehr Flexibilität beim Abfragen der Daten. Nachdem Sie einen sekundären Index für eine Tabelle erstellt haben, können Sie Daten aus dem Index ähnlich wie aus der Tabelle lesen.

DynamoDB unterstützt zwei Arten von Indexen:

- Globaler sekundärer Index – Ein Index mit einem Partitionsschlüssel und einem Sortierschlüssel, die sich von denen in der Tabelle unterscheiden können. Die Primärschlüsselwerte in globalen sekundären Indizes müssen nicht eindeutig sein.

- Lokaler sekundärer Index – Ein Index, der denselben Partitionsschlüssel wie die Tabelle hat, aber einen anderen Sortierschlüssel.

In DynamoDB sind globale Sekundärindizes (GSIs) Indizes, die sich über die gesamte Tabelle erstrecken, sodass Sie Abfragen über alle Partitionsschlüssel durchführen können. Lokale sekundäre Indizes (LSIs) sind Indizes, die denselben Partitionsschlüssel wie die Basistabelle, aber einen anderen Sortierschlüssel haben.

Jede Tabelle in DynamoDB verfügt über ein Kontingent von 20 globalen sekundären Indizes (Standardkontingent) und 5 lokalen sekundären Indizes.

In der zuvor gezeigten Beispieltabelle „Musik“ können Sie Datenelemente nach Künstler (Partitionsschlüssel) oder nach Künstler und SongTitle(Partitionsschlüssel und Sortierschlüssel) abfragen. Was ist, wenn Sie die Daten auch nach Genre und abfragen möchten AlbumTitle? Dazu könnten Sie einen Index für Genre und erstellen und den Index dann auf die gleiche Weise abfragen AlbumTitle, wie Sie die Musiktabelle abfragen würden.

Das folgende Diagramm zeigt die Beispieltabelle Music mit einem neuen Index namens GenreAlbumTitle. Im Index ist Genre der Partitionsschlüssel und AlbumTitleder Sortierschlüssel.

Tabelle „Musik“	GenreAlbumTitle

<pre>{
    "Artist": "No One You Know",
    "SongTitle": "My Dog Spot",
    "AlbumTitle": "Hey Now",
    "Price": 1.98,
    "Genre": "Country",
    "CriticRating": 8.4
}</pre>
	<pre>{
	    "Genre": "Country",
	    "AlbumTitle": "Hey Now",
	    "Artist": "No One You Know",
	    "SongTitle": "My Dog Spot"
	}</pre>

<pre>{
    "Artist": "No One You Know",
    "SongTitle": "Somewhere Down The
 Road",
    "AlbumTitle": "Somewhat Famous",</pre>
	<pre>{
	    "Genre": "Country",
	    "AlbumTitle": "Somewhat Famous",
	    "Artist": "No One You Know",</pre>

Tabelle „Musik“	GenreAlbumTitle

<pre>"Genre": "Country",
"CriticRating": 8.4,
"Year": 1984
}</pre>
	<pre>"SongTitle": "Somewhere Down The Road"
	}</pre>

<pre>{
"Artist": "The Acme Band",
"SongTitle": "Still in Love",
"AlbumTitle": "The Buck Starts Here",
"Price": 2.47,
"Genre": "Rock",
"PromotionInfo": {
"RadioStationsPlaying": {
"KHCR",
"KQBX",
"WTNR",
"WJJH"
},
"TourDates": {
"Seattle": "20150622",
"Cleveland": "20150630"
},
"Rotation": "Heavy"
}
}</pre>
	<pre>{
	"Genre": "Rock",
	"AlbumTitle": "The Buck Starts Here",
	"Artist": "The Acme Band",
	"SongTitle": "Still In Love"
	}</pre>

Tabelle „Musik“	GenreAlbumTitle

<pre>{
    "Artist": "The Acme Band",
    "SongTitle": "Look Out, World",
    "AlbumTitle": "The Buck Starts
 Here",
    "Price": 0.99,
    "Genre": "Rock"
}</pre>
	<pre>{
	    "Genre": "Rock",
	    "AlbumTitle": "The Buck Starts
	 Here",
	    "Artist": "The Acme Band",
	    "SongTitle": "Look Out, World"
	}</pre>

Beachten Sie im Hinblick auf den Index GenreAlbumTitle Folgendes:

- Jeder Index gehört zu einer Tabelle, die als Basistabelle für den Index bezeichnet wird. Im vorherigen Beispiel ist Music die Basistabelle für den GenreAlbumTitleIndex.
- DynamoDB verwaltet Indexe automatisch. Beim Hinzufügen, Aktualisieren und Löschen eines Elements in der Basistabelle fügt DynamoDB das entsprechende Element in allen Indexen hinzu, die zu dieser Tabelle gehören, bzw. aktualisiert oder löscht sie.
- Wenn Sie einen Index erstellen, geben Sie an, welche Attribute aus der Basistabelle in den Index kopiert oder projiziert werden. DynamoDB projiziert mindestens die Schlüsselattribute aus der Basistabelle in den Index. Dies ist der Fall bei `GenreAlbumTitle`, da hier nur die Schlüsselattribute aus der Tabelle `Music` in den Index projiziert werden.

Sie können den GenreAlbumTitleIndex abfragen, um alle Alben eines bestimmten Genres zu finden (z. B. alle Rock-Alben). Sie können den Index auch abfragen, um alle Alben in einem bestimmten Genre mit bestimmten Albumtiteln zu finden (z. B. alle Country-Alben mit Titeln, die mit dem Buchstaben H beginnen).

Weitere Informationen finden Sie unter Verbessern des Datenzugriffs mit sekundären Indizes in DynamoDB.

DynamoDB Streams

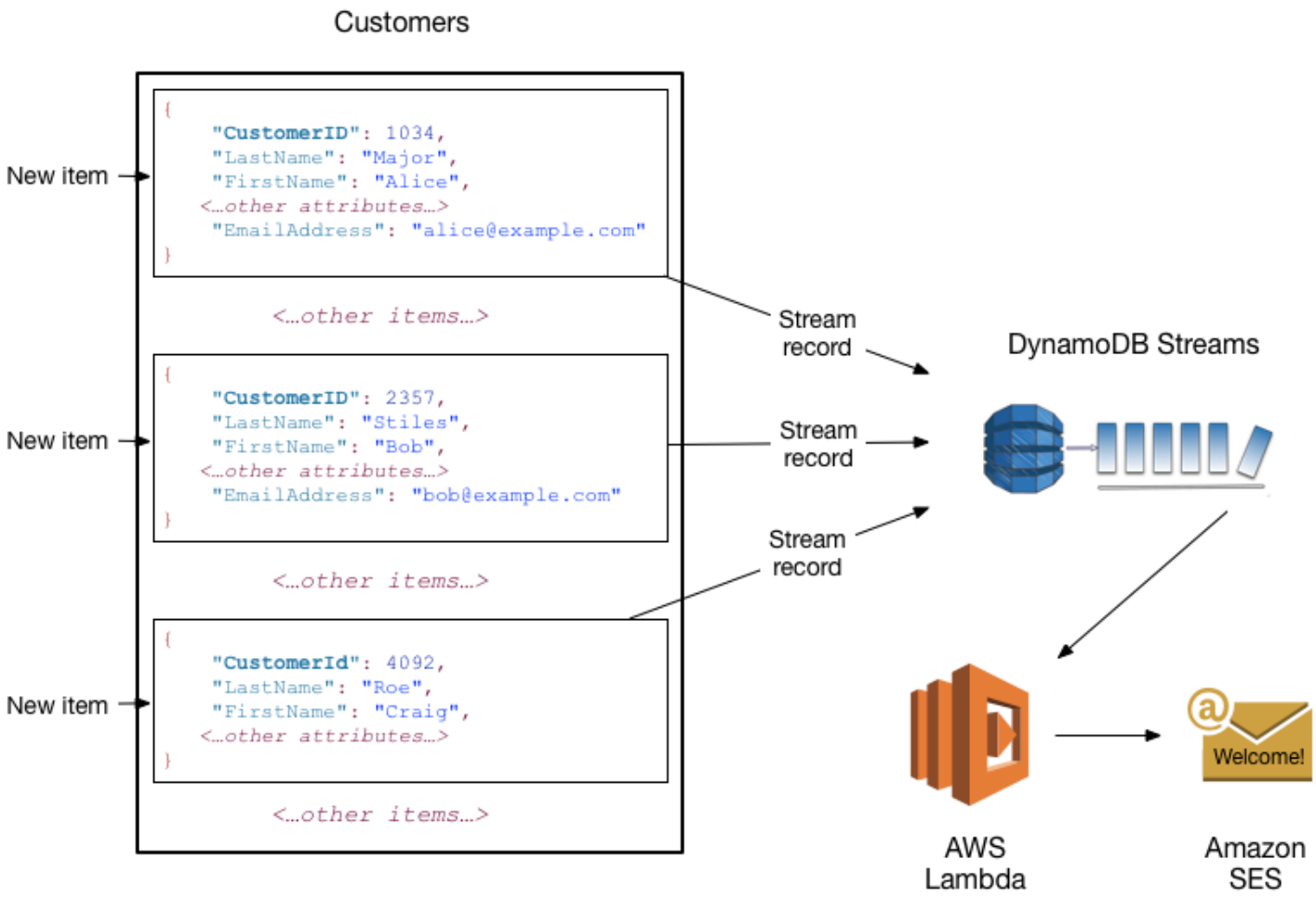
DynamoDB Streams ist eine optionale Funktion, die Datenänderungsereignisse in DynamoDB-Tabellen erfasst. Die Daten über diese Ereignisse erscheinen nahezu in Echtzeit und in der Reihenfolge, in der die Ereignisse aufgetreten sind, im Stream.

Jedes Ereignis wird durch einen Stream-Datensatz repräsentiert. Wenn Sie einen Stream für eine Tabelle aktivieren, schreibt DynamoDB Streams einen Stream-Datensatz, sobald eines der folgenden Ereignisse eintritt:

- Ein neues Element wird der Tabelle hinzugefügt: Der Stream erfasst ein Image des gesamten Elements, einschließlich aller Attribute.
- Ein Element wird aktualisiert: Der Stream erfasst das Image von Attributen, die im Element modifiziert wurden, vor und nach der Änderung.
- Ein Element wird aus der Tabelle gelöscht: Der Stream erfasst ein Image des gesamten Elements, bevor es gelöscht wurde.

Jeder Stream-Datensatz enthält auch den Namen der Tabelle, den Ereigniszeitstempel und andere Metadaten. Stream-Datensätze haben eine Lebensdauer von 24 Stunden. Danach werden sie automatisch aus dem Stream entfernt.

Sie können DynamoDB Streams zusammen mit verwenden, AWS Lambda um einen Triggercode zu erstellen, der automatisch ausgeführt wird, wenn ein interessierendes Ereignis in einem Stream auftritt. Nehmen wir als Beispiel eine Customers-Tabelle mit Kundeninformationen für ein Unternehmen. Angenommen, Sie möchten eine Willkommens-E-Mail an jeden neuen Kunden senden. Sie können einen Stream für diese Tabelle aktivieren und den Stream anschließend einer Lambda-Funktion zuordnen. Die Lambda-Funktion wird immer dann ausgeführt, wenn ein neuer Stream-Datensatz erscheint. Es werden jedoch nur neue Elemente verarbeitet, die der Tabelle Kunden hinzugefügt wurden. Für ein Element mit einem `EmailAddress`-Attribut ruft die Lambda-Funktion den Amazon Simple Email Service (Amazon SES) auf, um eine E-Mail an diese Adresse zu senden.



> **Note**
>
> In diesem Beispiel erhält der letzte Kunde, Craig Roe, keine E-Mail-Nachricht, da kein Wert für `EmailAddress` vorhanden ist.

Neben Triggern ermöglicht DynamoDB Streams leistungsstarke Lösungen wie Datenreplikation innerhalb und zwischen AWS Regionen, materialisierte Ansichten von Daten in DynamoDB-Tabellen, Datenanalyse mit materialisierten Kinesis-Ansichten und vieles mehr.

Weitere Informationen finden Sie unter Ändern Sie die Datenerfassung für DynamoDB Streams.

DynamoDB API

Um mit Amazon DynamoDB arbeiten zu können, muss Ihre Anwendung einige einfache API-Operationen verwenden. Im Folgenden finden Sie eine Zusammenfassung dieser Operationen nach Kategorie sortiert.

> **Note**
>
> Eine vollständige Liste der API-Operationen finden Sie in der Amazon-DynamoDB-API-Referenz.

Themen

- Steuerebene
- Datenebene
- DynamoDB Streams
- Transaktionen

Steuerebene

Mit Steuerungsebenenoperationen können Sie DynamoDB-Tabellen erstellen und verwalten. Außerdem ermöglichen sie die Arbeit mit Indexen, Streams und anderen Objekten, die von Tabellen abhängen.

- `CreateTable` – Erstellt eine neue Tabelle. Optional können Sie eine oder mehrere sekundäre Indxe erstellen und DynamoDB Streams für die Tabelle aktivieren
- `DescribeTable` – Gibt Informationen über eine Tabelle zurück, wie das entsprechende Primärschlüsselschema, Durchsatzeinstellungen und Indexinformationen.
- `ListTables` – Gibt die Namen aller Ihrer Tabellen in einer Liste zurück
- `UpdateTable` – Ändert die Einstellungen einer Tabelle oder deren Index, erstellt oder entfernt neue Indexe aus einer Tabelle oder ändert die DynamoDB-Streams-Einstellungen für eine Tabelle
- `DeleteTable` – Entfernt eine Tabelle und alle davon abhängigen Objekte aus DynamoDB

Datenebene

Mit Operationen auf Datenebene können Sie Aktionen zum Erstellen, Lesen, Aktualisieren und Löschen (auch als CRUD bezeichnet) für Daten in einer Tabelle ausführen. Einige Operationen auf Datenebene ermöglichen außerdem das Lesen von Daten aus einem Sekundärindex.

Sie können diese CRUD-Operationen verwendenPartiQL – Eine SQL-kompatible Abfragesprache für Amazon DynamoDB, oder Sie können das klassische APIs CRUD von DynamoDB verwenden, das jede Operation in einen eigenen API-Aufruf unterteilt.

PartiQL – Eine SQL-kompatible Abfragesprache

- `ExecuteStatement` - Liest mehrere Elemente aus einer Tabelle. Sie können auch ein einzelnes Element aus einer Tabelle schreiben oder aktualisieren. Beim Schreiben oder Aktualisieren eines einzelnen Elements müssen Sie die Primärschlüsselattribute angeben.
- `BatchExecuteStatement` - Schreibt, aktualisiert oder liest mehrere Elemente aus einer Tabelle. Dies ist effizienter, als `ExecuteStatement` weil Ihre Anwendung nur einen einzigen Netzwerk-Roundtrip zum Schreiben oder Lesen der Elemente benötigt.

Klassisch APIs

Erstellen von Daten

- `PutItem` – Schreibt ein einzelnes Element in eine Tabelle. Sie müssen nur die Primärschlüsselattribute angeben, keine anderen Attribute.
- `BatchWriteItem` – Schreibt bis zu 25 Elemente in eine Tabelle. Dies ist effizienter als `PutItem` mehrfach aufzurufen, da Ihre Anwendung nur einen einzigen Netzlauf zum Schreiben der Elemente benötigt.

Lesen von Daten

- `GetItem` – Ruft ein einzelnes Element aus einer Tabelle ab. Sie müssen den Primärschlüssel für das gewünschte Element angeben. Sie können das gesamte Element oder nur eine Teilmenge seiner Attribute abrufen.
- `BatchGetItem` – Ruft bis zu 100 Elemente aus einer oder mehreren Tabellen ab. Dies ist effizienter als `GetItem` mehrfach aufzurufen, da Ihre Anwendung nur einen einzigen Netzlauf zum Lesen der Elemente benötigt.

- `Query` – Ruft alle Elemente mit einem bestimmten Partitionsschlüssel ab. Sie müssen den Partitionsschlüsselwert angeben. Sie können ganze Elemente oder nur eine Teilmenge ihrer Attribute abrufen. Optional können Sie eine Bedingung auf die Sortierschlüsselwerte anwenden, so dass Sie nur den Teil der Daten abrufen, denen derselbe Partitionsschlüssel zugeordnet ist. Sie können diese Operation für eine Tabelle verwenden, sofern die Tabelle sowohl einen Partitions- als auch einen Sortierschlüssel hat. Außerdem können Sie diese Operation für einen Index nutzen, sofern der Index sowohl einen Partitions- als auch einen Sortierschlüssel umfasst.
- `Scan` – Ruft alle Elemente in der angegebenen Tabelle oder dem Index ab. Sie können ganze Elemente oder nur eine Teilmenge ihrer Attribute abrufen. Optional können Sie eine Filterbedingung anwenden, um nur die gewünschten Werte zurückzugeben und den Rest zu verwerfen.

Aktualisieren von Daten

- `UpdateItem` – Ändert ein oder mehrere Attribute in einem Element. Sie müssen den Primärschlüssel für das Element angeben, das Sie ändern möchten. Sie können neue Attribute hinzufügen und vorhandene Attribute ändern oder entfernen. Außerdem können Sie bedingte Aktualisierungen durchführen, so dass die Aktualisierung nur erfolgreich ist, wenn eine benutzerdefinierte Bedingung erfüllt ist. Wahlweise können Sie einen unteilbaren Zähler implementieren, der ein numerisches Attribut erhöht oder senkt, ohne dass Konflikte mit anderen Schreibanforderungen auftreten.

Löschen von Daten

- `DeleteItem` – Löscht ein einzelnes Element aus einer Tabelle. Sie müssen den Primärschlüssel für das Element angeben, das Sie löschen möchten.
- `BatchWriteItem` – Löscht bis zu 25 Elemente aus einer oder mehreren Tabellen. Dies ist effizienter, als `DeleteItem` mehrfach aufzurufen, da Ihre Anwendung nur einen einzigen Netzlauf zum Löschen der Elemente benötigt.

> **Note**
>
> Sie können `BatchWriteItem` verwenden, um sowohl Daten zu erstellen als auch Daten zu löschen.

DynamoDB Streams

Mit DynamoDB-Streams-Operationen können Sie einen Stream in einer Tabelle aktivieren oder deaktivieren und den Zugriff auf die in einem Stream enthaltenen Datenänderungsdatensätze zulassen.

- `ListStreams` – Gibt eine Liste aller Streams oder nur den Stream für eine bestimmte Tabelle zurück
- `DescribeStream` – Gibt Informationen über einen Stream, wie den Amazon-Ressourcennamen (ARN), und die Position zurück, an der Ihre Anwendung mit dem Lesen der ersten Stream-Datensätze beginnen kann
- `GetShardIterator` – Gibt einen Shard Iterator zurück. Hierbei handelt es sich um eine Datenstruktur, die Ihre Anwendung zum Abrufen der Datensätze aus dem Stream verwendet.
- `GetRecords` – Ruft einen oder mehrere Stream-Datensätze unter Verwendung eines bestimmten Shard Iterators ab

Transaktionen

Transaktionen stellen Atomizität, Konsistenz, Isolation und Haltbarkeit (Atomicity, Consistency, Isolation und Durability, ACID) bereit. Dies ermöglicht eine einfachere Aufrechterhaltung der Richtigkeit der Daten in Ihren Anwendungen.

Sie könnenPartiQL – Eine SQL-kompatible Abfragesprache für Amazon DynamoDB, verwenden, um Transaktionsoperationen durchzuführen, oder Sie können das klassische CRUD von DynamoDB verwenden APIs , das jeden Vorgang in einen eigenen API-Aufruf unterteilt.

PartiQL – Eine SQL-kompatible Abfragesprache

- `ExecuteTransaction`— Eine Batch-Operation, die CRUD-Operationen für mehrere Elemente sowohl innerhalb als auch tabellenübergreifend mit einem garantierten Ergebnis ermöglicht. all-or-nothing

Klassisch APIs

- `TransactWriteItems`— Ein Batch-Vorgang, der `Delete` Operationen mit `PutUpdate`, und für mehrere Elemente sowohl innerhalb als auch tabellenübergreifend mit einem garantierten all-or-nothing Ergebnis ermöglicht.

- `TransactGetItems` – Ein Batch-Vorgang, der `Get`-Operationen verwendet, um mehrere Elemente aus einer oder mehreren Tabellen abzurufen.

Unterstützte Datentypen und Benennungsregeln in Amazon DynamoDB

In diesem Abschnitt werden die Amazon-DynamoDB-Benennungsregeln und die verschiedenen Datentypen beschrieben, die DynamoDB unterstützt. Für Datentypen gelten bestimmte Limits. Weitere Informationen finden Sie unter Datentypen.

Themen

- Benennungsregeln
- Datentypen
- Datentypbeschreibungen

Benennungsregeln

Tabellen, Attribute und andere Objekte in DynamoDB muss einen Namen haben. Namen sollten sinnvoll und präzise sein – Namen wie beispielsweise Produkte, Bücher und Verfasser sind selbsterklärend.

Die folgenden Benennungsregeln gelten für DynamoDB:

- Alle Namen müssen mit UTF-8 kodiert werden und die Groß- und Kleinschreibung beachten.
- Tabellen- und Indexnamen müssen zwischen 3 und 255 Zeichen lang sein und dürfen nur folgende Zeichen enthalten:
  - `a-z`
  - `A-Z`
  - `0-9`
  - `_` (Unterstrich)
  - `-` (Bindestrich)
  - `.` (Punkt)
- Attributnamen müssen mindestens ein Zeichen lang und dürfen nicht größer als 64 KB sein. Eine bewährte Methode besteht darin, Attributnamen möglichst kurz zu halten. Dies trägt zur

Reduzierung der verbrauchten Leseanforderungseinheiten bei, da Attributnamen beim Erfassen der Speicher- und Durchsatznutzung berücksichtigt werden.

Dabei gibt es die folgenden Ausnahmen. Diese Attributnamen dürfen maximal 255 Zeichen lang sein:

- Partitionsschlüsselnamen des sekundären Indexes
- Sortierschlüsselnamen des sekundären Indexes
- Die Namen aller vom Benutzer angegebenen, projizierten Attribute (gilt nur für lokale sekundäre Indizes)

Reservierte Wörter und Sonderzeichen

DynamoDB verfügt über eine Liste mit reservierten Wörtern und Sonderzeichen. Eine vollständige Liste finden Sie hier: Reservierte Wörter in DynamoDB. Außerdem haben die folgenden Zeichen in DynamoDB eine besondere Bedeutung: # (Hash) und : (Doppelpunkt).

Auch wenn DynamoDB diese reservierten Wörter und Sonderzeichen für Namen zulässt, empfehlen wir, diese nicht zu verwenden, da Sie Platzhaltervariablen definieren müssen, sobald Sie diese Namen in einem Ausdruck verwenden. Weitere Informationen finden Sie unter Ausdrucksattributnamen (Aliasse) in DynamoDB.

Datentypen

DynamoDB unterstützt viele verschiedene Datentypen für Attribute in einer Tabelle. Sie können wie folgt kategorisiert werden:

- Skalare Typen – Ein skalarer Typ kann genau einen Wert repräsentieren. Die skalaren Typen sind Zahl, Zeichenfolge, Binärwert, Boolescher Wert und Null.
- Dokumenttypen – Ein Dokumenttyp kann eine komplexe Struktur mit verschachtelten Attributen repräsentieren, wie sie z. B. in einem JSON-Dokument zu finden sind. Die Dokumenttypen sind Liste und Zuordnung.
- Satztypen – Ein Satztyp kann mehrere Skalarwerte repräsentieren. Die Satztypen sind Zeichenfolgensatz, Zahlensatz und Binärwertesatz.

Wenn Sie eine Tabelle oder einen Sekundärindex erstellen, müssen Sie die Namen und Datentypen der einzelnen Primärschlüsselattribute (Partitions- und Sortierschlüssel) angeben. Darüber hinaus muss jedes Schlüsselattribut als Zeichenfolgen-, Zahlen- oder Binärtyp definiert sein.

DynamoDB ist eine NoSQL-Datenbank und schemalos. Dies bedeutet, dass Sie beim Erstellen von Tabellen abgesehen von den Primärschlüsselattributen keine Attribute oder Datentypen definieren müssen. Im Vergleich dazu erfordern relationale Datenbanken beim Erstellen einer Tabelle die Angabe der Namen und Datentypen für jede einzelne Spalte.

Es folgen Beschreibungen der einzelnen Datentypen. Außerdem sind einige Beispiele im JSON-Format angegeben.

Skalare Typen

Die skalaren Typen sind Zahl, Zeichenfolge, Binärwert, Boolescher Wert und Null.

Zahl

Zahlen können positiv, negativ oder Null sein. Zahlen können auf 38 Stellen genau sein. Wird dieser Wert überschritten, wird eine Ausnahme ausgegeben. Wenn Sie eine höhere Genauigkeit als 38 Stellen benötigen, können Sie Zeichenfolgen verwenden.

- Positiver Bereich: 1E-130 bis 9,99999999999999999999999999999999999999E+125
- Negativer Bereich: -9,99999999999999999999999999999999999999E+125 bis -1E-130

In DynamoDB werden Zahlen als variable Länge dargestellt. Nullen am Anfang und am Ende werden abgeschnitten.

Alle Zahlen werden über das Netzwerk an DynamoDB als Zeichenfolgen übertragen, um die Kompatibilität über die Sprachen und Bibliotheken hinweg zu maximieren. DynamoDB behandelt sie jedoch als Zahlentypattribute für mathematische Operationen.

Sie können den Datentyp Zahl verwenden, um ein Datum oder einen Zeitstempel zu repräsentieren. Eine Möglichkeit, dies zu tun, ist die Verwendung der Epochenzeit - die Anzahl der Sekunden seit 00:00:00 UTC am 1. Januar 1970. Die Epochenzeit `1437136300` stellt 12:31:40 UTC am 17. Juli 2015 dar.

Weitere Informationen finden Sie unter http://en.wikipedia. org/wiki/Unix_Zeit.

Zeichenfolge

Die Zeichenfolgen sind Unicode mit binärer UTF-8-Kodierung. Die Mindestlänge einer Zeichenfolge kann Null sein, wenn das Attribut nicht als Schlüssel für einen Index oder eine Tabelle verwendet wird und durch die maximale DynamoDB-Elementgrößenbeschränkung von 400 KB eingeschränkt ist.

Wenn Sie einen Primärschlüssel als Attribut des Typs Zeichenfolge definieren, gelten die folgenden zusätzlichen Einschränkungen:

- Für einen einfachen Primärschlüssel beträgt die maximale Länge des ersten Attributwerts (der Partitionsschlüssel) 2048 Byte.
- Für einen zusammengesetzten Primärschlüssel beträgt die maximale Länge des zweiten Attributwerts (Sortierschlüssel) 1024 Byte.

DynamoDB sortiert und vergleicht Zeichenfolgen mithilfe der Bytes der zugrunde liegenden UTF-8-Zeichenfolgenkodierung. Beispiel: „a” (0x61) ist größer als „A” (0x41) und „¿” (0xC2BF) ist größer als „z” (0x7A).

Sie können den Datentyp Zeichenfolge verwenden, um ein Datum oder einen Zeitstempel zu repräsentieren. Eine Möglichkeit dazu ist die Verwendung von ISO 8601-Zeichenfolgen, wie in folgenden Beispielen gezeigt:

- `2016-02-15`
- `2015-12-21T17:42:34Z`
- `20150311T122706Z`

Weitere Informationen finden Sie unter http://en.wikipedia. org/wiki/ISO_8601.

> **Note**
>
> Im Gegensatz zu herkömmlichen relationalen Datenbanken unterstützt DynamoDB nativ keinen Datums- und Uhrzeitdatentyp. Es kann hilfreich sein, Daten und Zeitdaten stattdessen als Zahlendatentyp zu speichern, wobei die Unix-Epochenzeit verwendet wird.

Binär

Attribute vom Typ Binärwert können beliebige binäre Daten speichern, wie z. B. komprimierte Textdateien, verschlüsselte Daten oder Bilder. Beim Binärtyp betrachtet DynamoDB beim Vergleichen der binären Werte jedes Byte der Binärdaten ohne Vorzeichen.

Die Länge eines binären Attributs kann Null sein, wenn das Attribut nicht als Schlüssel für einen Index oder eine Tabelle verwendet wird und durch die maximale DynamoDB-Elementgrößenbeschränkung von 400 KB eingeschränkt ist.

Wenn Sie einen Primärschlüssel als Attribut des Typs Binärwert definieren, gelten die folgenden zusätzlichen Einschränkungen:

- Für einen einfachen Primärschlüssel beträgt die maximale Länge des ersten Attributwerts (der Partitionsschlüssel) 2048 Byte.
- Für einen zusammengesetzten Primärschlüssel beträgt die maximale Länge des zweiten Attributwerts (Sortierschlüssel) 1024 Byte.

Ihre Anwendungen müssen Binärwerte im Base64-codierten Format codieren, bevor sie an DynamoDB gesendet werden. Nach dem Empfang dieser Werte dekodiert DynamoDB die Daten in ein nicht signiertes Byte-Array und verwendet dies als Länge des binären Attribut

Im folgenden Beispiel wird ein binäres Attribut mit base64-verschlüsseltem Text dargestellt.

```
dGhpcyB0ZXh0IGlzIGJhc2U2NC1lbmNvZGVk
```

Boolesch

Ein Attribut vom Typ Boolescher Wert kann entweder `true` oder `false` speichern.

Null

Null stellt ein Attribut mit einem unbekannten oder nicht definierten Status dar.

Dokumenttypen

Die Dokumenttypen sind Liste und Zuordnung. Diese Datentypen können ineinander verschachtelt werden, um komplexe Datenstrukturen bis zu 32 Ebenen tief darzustellen.

Es gibt keine Beschränkungen in Bezug auf die Anzahl der Werte in einer Liste oder Zuordnung, solange das Element, das die Werte enthält, das Limit der DynamoDB-Elementgröße (400 KB) einhält.

Ein Attributwert kann eine leere Zeichenfolge oder ein leerer Binärwert sein, wenn das Attribut nicht für einen Tabellen- oder Indexschlüssel verwendet wird. Ein Attributwert kann kein leerer Satz sein (Zeichenfolgensatz, Zahlensatz oder Binärsatz), jedoch sind leere Listen und Zuordnungen zulässig. Leere Zeichenfolgen- und Binärwerte sind in Listen und Zuordnungen zulässig. Weitere Informationen finden Sie unter Attribute.

Auflisten

Ein Attribut vom Typ Liste kann eine geordnete Sammlung von Werten speichern. Listen werden in eckige Klammern gesetzt: `[ ... ]`

Eine Liste ähnelt einem JSON-Array. Es gibt keine Einschränkungen hinsichtlich der Datentypen, die in einem Listenelement gespeichert werden können. Die Elemente in einem Listenelement müssen nicht vom selben Typ sein.

Das folgende Beispiel zeigt eine Liste mit zwei Zeichenfolgen und einer Zahl.

```
FavoriteThings: ["Cookies", "Coffee", 3.14159]
```

> **Note**
>
> Mit DynamoDB können Sie mit einzelnen Elementen innerhalb der Listen arbeiten, auch wenn diese Elemente stark verschachtelt sind. Weitere Informationen finden Sie unter Verwenden von Ausdrücken in DynamoDB.

Zuordnung

Ein Attribut vom Typ Zuordnung kann eine ungeordnete Sammlung von Name-Wert-Paaren speichern. Zuordnungen werden in geschweifte Klammern gesetzt: `{ ... }`

Eine Zuordnung ist mit einem JSON-Objekt vergleichbar. Es gibt keine Einschränkungen hinsichtlich der Datentypen, die in einem Zuordnungselement gespeichert werden können. Die Elemente in einem Zuordnungselement müssen nicht vom selben Typ sein.

Zuordnungen eignen sich optimal für das Speichern von JSON-Dokumenten in DynamoDB. Das folgende Beispiel zeigt eine Zuordnung, die eine Zeichenfolge, eine Zahl und eine verschachtelte Liste mit einer weiteren Zuordnung enthält.

```
{
    Day: "Monday",
    UnreadEmails: 42,
    ItemsOnMyDesk: [
        "Coffee Cup",
        "Telephone",
        {
            Pens: { Quantity : 3},
```

```
            Pencils: { Quantity : 2},
            Erasers: { Quantity : 1}
        }
    ]
}
```

> **Note**
>
> Mit DynamoDB können Sie mit einzelnen Elementen innerhalb von Zuordnungen arbeiten, auch wenn diese Elemente stark verschachtelt sind. Weitere Informationen finden Sie unter Verwenden von Ausdrücken in DynamoDB.

Sätze

DynamoDB unterstützt Typen, die Sätze mit Zahlen-, Zeichenfolgen- und Binärwerten darstellen. Alle Elemente in einem Satz müssen jedoch vom selben Typ sein. Ein Zahlensatz kann z. B. nur Zahlen enthalten und ein Zeichenfolgensatz kann nur Zeichenfolgen enthalten.

Es gibt keine Beschränkungen in Bezug auf die Anzahl der Werte in einem Satz, solange das Element, das die Werte enthält, das Limit der DynamoDB-Elementgröße (400 KB) einhält.

Jeder Wert innerhalb eines Satzes muss eindeutig sein. In einem Satz spielt die Reihenfolge der Werte keine Rolle. Aus diesem Grund dürfen Ihre Anwendungen nicht davon ausgehen, dass die Elemente innerhalb des Satzes in einer bestimmten Reihenfolge vorliegen. Leere Mengen werden von DynamoDB nicht unterstützt, jedoch sind leere Zeichenfolgen und Binärwerte innerhalb einer Menge zulässig.

Das folgende Beispiel zeigt einen Zeichenfolgensatz, einen Zahlensatz und einen Binärwertesatz:

```
["Black", "Green", "Red"]

[42.2, -19, 7.5, 3.14]

["U3Vubnk=", "UmFpbnk=", "U25vd3k="]
```

Datentypbeschreibungen

Das Low-Level-DynamoDB-API-Protokoll verwendet Datentypdeskriptoren als Token, die DynamoDB mitteilen, wie die einzelnen Attribute zu interpretieren sind.

Im Folgenden sehen Sie eine vollständige Liste der DynamoDB-Datentypbeschreibungen:

- **`S`** – Zeichenfolge
- **`N`** – Zahl
- **`B`** – Binary
- **`BOOL`** – Boolean
- **`NULL`** – Nullwert
- **`M`** – Zuordnung
- **`L`** – Liste
- **`SS`** – Zeichenfolgensatz
- **`NS`** – Zahlensatz
- **`BS`** – Binärzahlensatz

DynamoDB-Tabellenklassen

DynamoDB bietet zwei Tabellenklassen an, mit denen Sie die Kosten optimieren können. Die DynamoDB-Standard-Tabellenklasse ist die Standardeinstellung und wird für die große Mehrheit der Workloads empfohlen. Die Tabellenklasse DynamoDB Standard-Infrequent Access (DynamoDB Standard-IA) ist für Tabellen optimiert, in denen Speicher die dominierenden Kosten sind. Zum Beispiel sind Tabellen, die selten aufgerufene Daten speichern, wie Anwendungsprotokolle, alte Social-Media-Posts, E-Commerce-Bestellhistorie und frühere Spielerrungenschaften, gute Kandidaten für die Standard-IA Tabellenklasse. Siehe Amazon-DynamoDB-Preise für weitere Informationen zur Preisdetails.

Jede DynamoDB-Tabelle ist einer Tabellenklasse zugeordnet (Standardmäßig DynamoDB Standard). Alle der Tabelle zugeordneten sekundären Indizes verwenden dieselbe Tabellenklasse. Jede Tabellenklasse bietet unterschiedliche Preise für die Datenspeicherung sowie für Lese- und Schreibanfragen. Sie können die kostengünstigste Tabellenklasse für Ihre Tabelle basierend auf ihren Speicher- und Durchsatznutzungsmustern auswählen.

Die Wahl einer Tabellenklasse ist nicht dauerhaft — Sie können diese Einstellung mit der AWS CLI oder dem AWS-Managementkonsole SDK ändern. AWS DynamoDB unterstützt auch die Verwaltung Ihrer Tabellenklasse mithilfe von AWS CloudFormation Tabellen mit nur einer Region und globalen Tabellen. Weitere Informationen zur Auswahl Ihrer Tabellenklasse finden Sie unter Überlegungen bei der Auswahl einer Tabellenklasse in DynamoDB.

Partitionen und Datenverteilung in DynamoDB

Amazon DynamoDB speichert Daten in Partitionen. Eine Partition ist eine Speicherzuweisung für eine Tabelle, die durch Solid-State-Laufwerke (SSDs) gesichert und automatisch über mehrere Availability Zones innerhalb einer Region repliziert wird. AWS Die Partitionsverwaltung wird ausschließlich von DynamoDB ausgeführt, d. h. Sie müssen Partitionen nicht selbst verwalten.

Beim Erstellen einer Tabelle ist der Anfangsstatus `CREATING`. In dieser Phase weist DynamoDB der Tabelle ausreichende Partitionen zu, sodass diese Ihre Anforderungen für den bereitgestellten Durchsatz erfüllen kann. Sie können mit dem Schreiben und Lesen von Tabellendaten beginnen, sobald sich der Tabellenstatus zu `ACTIVE` ändert.

DynamoDB weist einer Tabelle in folgenden Situationen zusätzliche Partitionen zu:

- Wenn Sie die Einstellungen für den bereitgestellten Durchsatz der Tabelle über das Maß erhöhen, das die vorhandenen Partitionen unterstützen können.
- Wenn eine vorhandene Partition bis zur Kapazität gefüllt ist und mehr Speicherplatz erforderlich ist.

Die Partitionsverwaltung wird automatisch im Hintergrund ausgeführt und ist für Ihre Anwendungen transparent. Ihre Tabelle bleibt verfügbar und unterstützt Ihre Anforderungen des bereitgestellten Durchsatzes vollständig.

Weitere Details finden Sie unter Entwerfen von Partitionsschlüsseln.

Globale sekundäre Indizes in DynamoDB bestehen ebenfalls aus Partitionen. Die Daten in einem globalen sekundären Index werden getrennt von den Daten in der Basistabelle gespeichert. Indexpartitionen verhalten sich jedoch ähnlich wie Tabellenpartitionen.

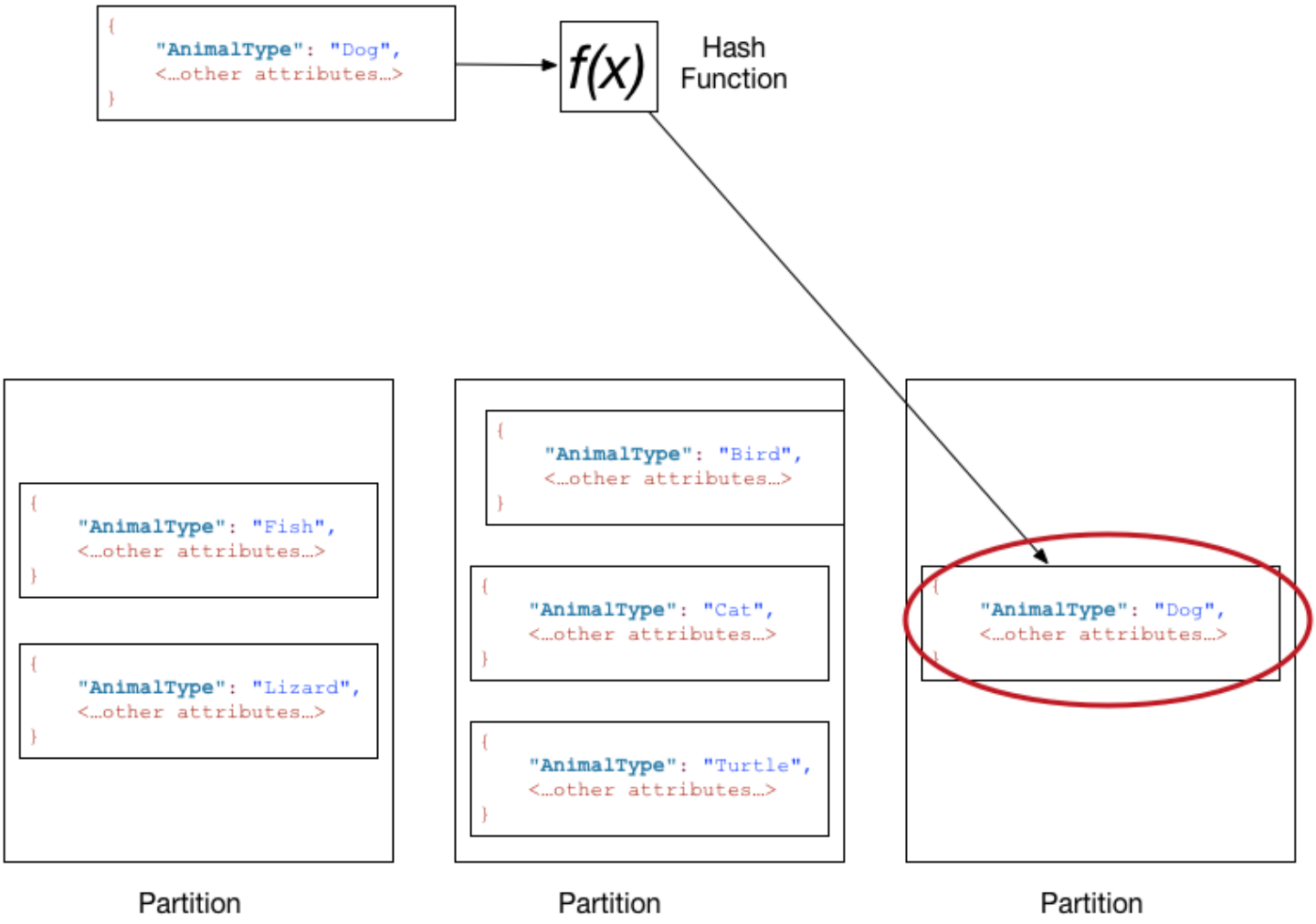
Datenverteilung: Partitionsschlüssel

Wenn Ihre Tabelle über einen einfachen Primärschlüssel (nur Partitionsschlüssel) verfügt, speichert und ruft DynamoDB jedes Element basierend auf seinem Partitionsschlüsselwert ab.

Um ein Element in die Tabelle zu schreiben, verwendet DynamoDB den Wert des Partitionsschlüssels als Eingabe für eine interne Hash-Funktion. Der Ausgabewert der Hash-Funktion bestimmt die Partition, in der das Element gespeichert wird.

Um ein Element aus der Tabelle zu lesen, müssen Sie den Partitionsschlüsselwert für das Element angeben. DynamoDB verwendet diesen Wert als Eingabe für die Hash-Funktion. Daraus ergibt sich die Partition, in der das Element gefunden werden kann.

Das folgende Diagramm zeigt eine Tabelle mit dem Namen Pets, die sich über mehrere Partitionen erstreckt. Der Primärschlüssel der Tabelle ist AnimalType(nur dieses Schlüsselattribut wird angezeigt). DynamoDB verwendet eine Hash-Funktion, um zu ermitteln, wo ein neues Element gespeichert werden soll, und zwar in diesem Fall basierend auf dem Hash-Wert der Zeichenfolge Dog. Hinweis: Die Elemente werden nicht sortiert gespeichert. Die Position jedes einzelnen Elements wird durch den Hash-Wert seines Partitionsschlüssels bestimmt.



> **Note**
>
> DynamoDB ist für eine einheitliche Verteilung von Elementen über die Partitionen einer Tabelle optimiert, unabhängig davon, wie viele Partitionen vorliegen. Wir empfehlen, dass Sie einen Partitionsschlüssel verwenden, der eine großen Anzahl von eindeutigen Werten in Bezug auf die Anzahl der Elemente in der Tabelle aufnehmen kann.

Datenverteilung: Partitions- und Sortierschlüssel

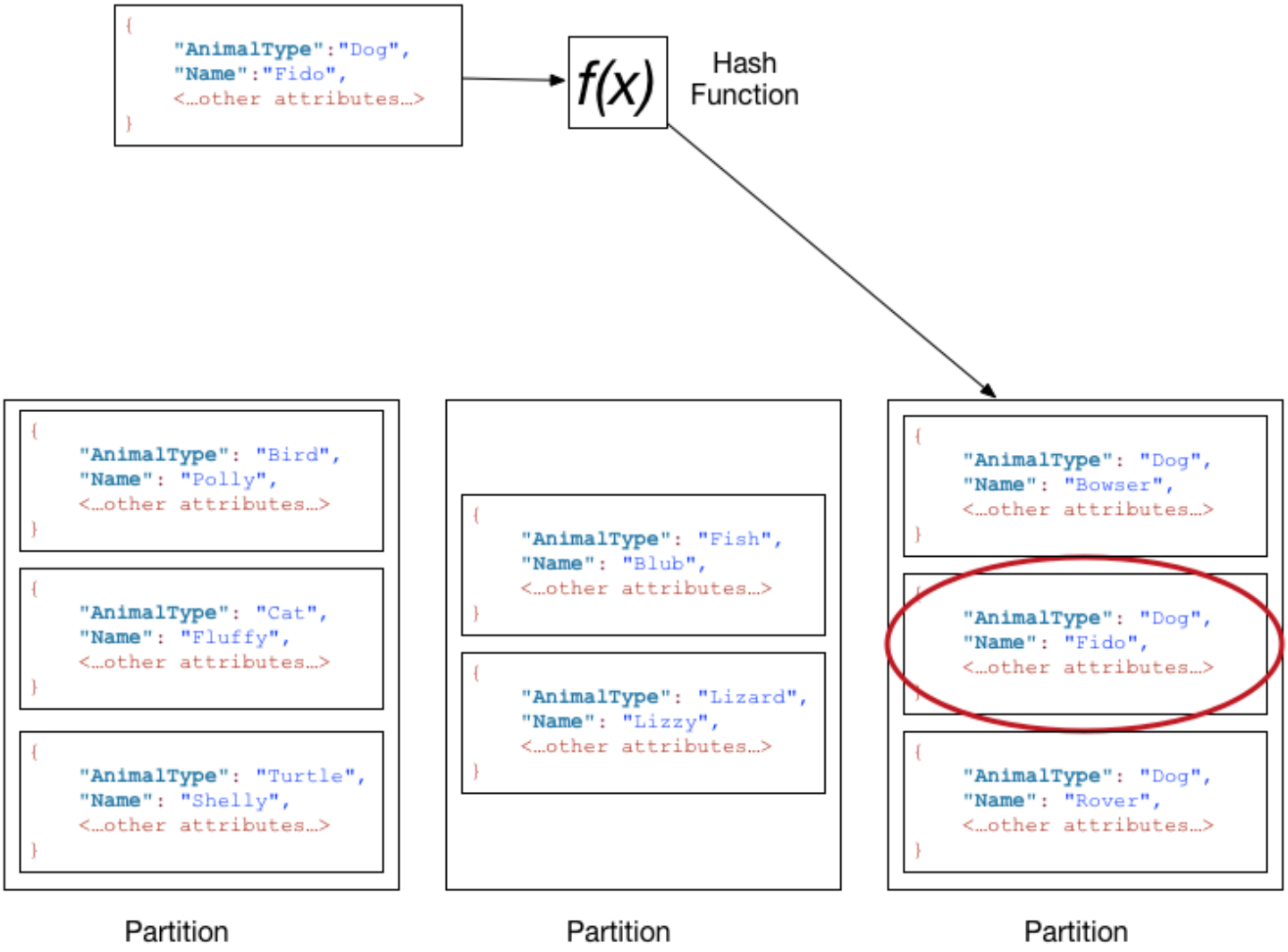
Wenn die Tabelle über einen zusammengesetzten Primärschlüssel (Partitionsschlüssel und Sortierschlüssel) verfügt, berechnet DynamoDB den Hashwert des Partitionsschlüssels auf die gleiche Weise wie unter Datenverteilung: Partitionsschlüssel beschrieben. Dabei werden jedoch Elemente, die denselben Partitionsschlüsselwert haben, nahe beieinander und in sortierter Reihenfolge nach dem Wert des Sortierschlüsselattributs beibehalten. Der Satz von Elementen, die denselben Partitionsschlüsselwert haben, wird als Elementauflistung bezeichnet. Elementauflistungen sind für das effiziente Abrufen von Elementbereichen innerhalb der Auflistung optimiert. Wenn Ihre Tabelle keine lokalen sekundären Indizes enthält, teilt DynamoDB Ihre Elementauflistung automatisch auf die Anzahl von Partitionen auf, die erforderlich ist, um die Daten zu speichern und den Lese- und Schreibdurchsatz zu gewährleisten.

Beim Schreiben eines Elements in die Tabelle berechnet DynamoDB den Hash-Wert des Partitionsschlüssels, um zu ermitteln, welche Partition das Element enthalten soll. In dieser Partition können mehrere Elemente denselben Wert für den Partitionsschlüssel haben. DynamoDB speichert dann das Element zusammen mit den anderen Elementen mit demselben Partitionsschlüssel nach Sortierschlüssel sortiert in aufsteigender Reihenfolge.

Um ein Element aus der Tabelle zu lesen, müssen Sie den Partitions- und Sortierschlüsselwert für das Element angeben. DynamoDB berechnet den Hash-Wert des Partitionsschlüssels. Daraus ergibt sich die Partition, in der sich das Element befindet.

Sie können mehrere Elemente aus der Tabelle in einer einzigen Operation (`Query`) lesen, sofern die gewünschten Elemente denselben Partitionsschlüsselwert besitzen. DynamoDB gibt alle Elemente mit diesem Partitionsschlüsselwert zurück. Optional können Sie eine Bedingung auf den Sortierschlüssel anwenden, so dass nur die Elemente in einem bestimmten Wertebereich zurückgegeben werden.

Angenommen, die Pets-Tabelle hat einen zusammengesetzten Primärschlüssel, der aus AnimalType(Partitionsschlüssel) und Name (Sortierschlüssel) besteht. Das folgende Diagramm veranschaulicht, wie DynamoDB ein Element mit dem Partitionsschlüsselwert Dog und dem Sortierschlüsselwert Fido schreibt.



Um dasselbe Element aus der Tabelle Pets zu lesen, berechnet DynamoDB den Hash-Wert von Dog. Daraus ergibt sich die Partition, in der diese Elemente gespeichert sind. DynamoDB scannt dann die Attributwerte des Sortierschlüssels, bis Fido gefunden wird.

Um alle Elemente mit dem Wert AnimalTypevon Dog zu lesen, können Sie einen `Query` Vorgang ausführen, ohne eine Sortierschlüsselbedingung anzugeben. Standardmäßig werden die Elemente in der Reihenfolge zurückgegeben, in der sie gespeichert sind (d. h. in aufsteigender Reihenfolge nach Sortierschlüssel). Wahlweise können Sie stattdessen die absteigende Reihenfolge anfordern.

Um nur einige der Dog-Elemente abzufragen, können Sie eine Bedingung auf den Sortierschlüssel anwenden (z. B. nur die Dog-Elemente, in denen Name mit einem Buchstaben im Bereich von `A` bis `K` beginnt).

> **Note**
>
> In der DynamoDB-Tabelle gibt es keine Einschränkung in Bezug auf die Anzahl von eindeutigen Sortierschlüsselwerten pro Partitionsschlüsselwert. Wenn Sie viele Milliarden von Dog-Elementen in der Tabelle Pets speichern müssen, weist DynamoDB automatisch genügend Speicherplatz zu, um diese Anforderung zu erfüllen.

Erfahren Sie, wie Sie von SQL zu NoSQL wechseln.

Als Anwendungsentwickler haben Sie möglicherweise Erfahrung mit relationalen Datenbankmanagementsystemen (RDBMS) und SQL (Structured Query Language). Wenn Sie mit Amazon DynamoDB arbeiten, werden Sie nicht nur viele Gemeinsamkeiten, sondern auch viele Unterschiede feststellen. Mit dem Begriff NoSQL werden nicht relationale Datenbanksysteme beschrieben, die hoch verfügbar, skalierbar und für hohe Leistung optimiert sind. Anstatt des relationalen Modells nutzen NoSQL-Datenbanken (wie DynamoDB) alternative Datenverwaltungsmodelle, z. B. Schlüssel-Wert-Paare oder Dokumentenspeicher. Weitere Informationen finden Sie unter Was ist NoSQL?.

Amazon DynamoDB unterstützt PartiQL, eine Open-Source-, SQL-kompatible Abfragesprache, mit der Sie Daten effizient abfragen können, unabhängig davon, wo oder in welchem Format sie gespeichert sind. Mit PartiQL können Sie problemlos strukturierte Daten aus relationalen Datenbanken, halbstrukturierte und verschachtelte Daten in offenen Datenformaten und sogar schemalose Daten in NoSQL- oder Dokumentdatenbanken verarbeiten, die verschiedene Attribute für verschiedene Zeilen zulassen. Weitere Informationen finden Sie unter PartiQL-Abfragesprache.

In den folgenden Abschnitten werden allgemeine Datenbankaufgaben beschrieben, wobei SQL-Anweisungen mit ihren entsprechenden DynamoDB-Operationen verglichen und gegenübergestellt werden.

> **Note**
>
> Die SQL-Beispiele in diesem Abschnitt sind mit dem MySQL-RDBMS kompatibel. Die in diesem Abschnitt genannten DynamoDB-Beispiele enthalten den Namen der DynamoDB-Operation zusammen mit dem Parameter für diese Operation im JSON-Format.

Themen

- Auswahl zwischen relational (SQL) und NoSQL
- Unterschiede beim Zugriff auf eine relationale (SQL) Datenbank und DynamoDB
- Unterschiede zwischen einer relationalen (SQL) Datenbank und DynamoDB beim Erstellen einer Tabelle
- Unterschiede zwischen dem Abrufen von Tabelleninformationen aus einer relationalen (SQL) Datenbank und DynamoDB
- Unterschiede zwischen einer relationalen (SQL) Datenbank und DynamoDB beim Schreiben von Daten in eine Tabelle
- Wichtige Unterschiede zwischen einer relationalen (SQL) Datenbank und DynamoDB beim Lesen von Daten aus einer Tabelle
- Unterschiede zwischen einer relationalen (SQL) Datenbank und DynamoDB beim Verwalten von Indizes
- Unterschiede zwischen einer relationalen (SQL) Datenbank und DynamoDB beim Ändern von Daten in einer Tabelle
- Unterschiede zwischen einer relationalen (SQL) Datenbank und DynamoDB beim Löschen von Daten aus einer Tabelle
- Unterschiede zwischen einer relationalen (SQL) Datenbank und DynamoDB beim Entfernen einer Tabelle

Auswahl zwischen relational (SQL) und NoSQL

Aktuelle Anwendungen stellen mehr anspruchsvolle Anforderungen als je zuvor. Beispielsweise beginnt ein Online-Spiel mit nur wenigen Benutzern und einer sehr kleinen Datenmenge. Wenn das Spiel jedoch erfolgreicher wird, kann es die Ressourcen des zugrunde liegenden Datenbankmanagementsystems sehr schnell überfordern. Häufig haben webbasierte Anwendungen Hunderte, Tausende oder Millionen von Benutzern gleichzeitig, wobei mehrere Terabyte an neuen Daten pro Tag generiert werden. Datenbanken für diese Anwendungen müssen Zehntausende (oder Hunderttausende) von Lese- und Schreibvorgängen pro Sekunde verarbeiten.

Amazon DynamoDB eignet sich besonders für solche Workloads. Als Entwickler können Sie in kleinem Maße beginnen und diesen schrittweise erhöhen, wenn Ihre Anwendung an Beliebtheit gewinnt. DynamoDB lässt sich nahtlos skalieren, um sehr große Datenmengen und sehr große Benutzerzahlen zu verarbeiten.

Weitere Informationen zur herkömmlichen relationalen Datenbankmodellierung und deren Anpassung an DynamoDB finden Sie unter Bewährte Methoden für die Modellierung relationaler Daten in DynamoDB.

In der folgenden Tabelle sind einige allgemeine Unterschiede zwischen einem relationalen Datenbankmanagementsystem (RDBMS) und DynamoDB aufgeführt.

Merkmal	Relationales Datenbank managementsystem (RDBMS)	Amazon DynamoDB
Optimale Workloads	Ad-hoc-Abfragen, Data Warehousing, OLAP (Online Analytical Processing).	Web-Scale-Anwendun gen, einschließlich sozialer Netzwerke, Gaming, gemeinsamer Mediennutzung und Internet of Things (IoT).
Datenmodell	Das relationale Modell erfordert ein eindeutig definiert es Schema, in dem Daten in Tabellen, Zeilen und Spalten standardisiert werden. Darüber hinaus werden alle Beziehungen zwischen Tabellen, Spalten, Indizes und sonstige Datenbankelementen definiert.	DynamoDB ist schemalos. Jede Tabelle muss über einen Primärschlüssel verfügen, um jedes Datenelement eindeutig zu identifizieren. Es gibt jedoch keine ähnlichen Einschränkungen bei anderen Nicht-Schlüsselattributen. DynamoDB kann strukturierte und halbstrukturierte Daten, einschließlich JSON-Doku menten, verwalten.
Datenzugriff	SQL ist der Standard für das Speichern und Abrufen von Daten. Relationale Datenbank en bieten eine umfassend e Auswahl an Tools, die die Entwicklung datenbank gestützter Anwendungen erleichtern. Diese Tools verwenden jedoch alle SQL.	Sie können die AWS-Manag ementkonsole, die oder NoSQL verwenden AWS CLI, um mit DynamoDB WorkBench zu arbeiten und Ad-hoc-Aufgaben auszuführen. Mit PartiQL, einer SQL-kompatiblen Abfragesprache, können

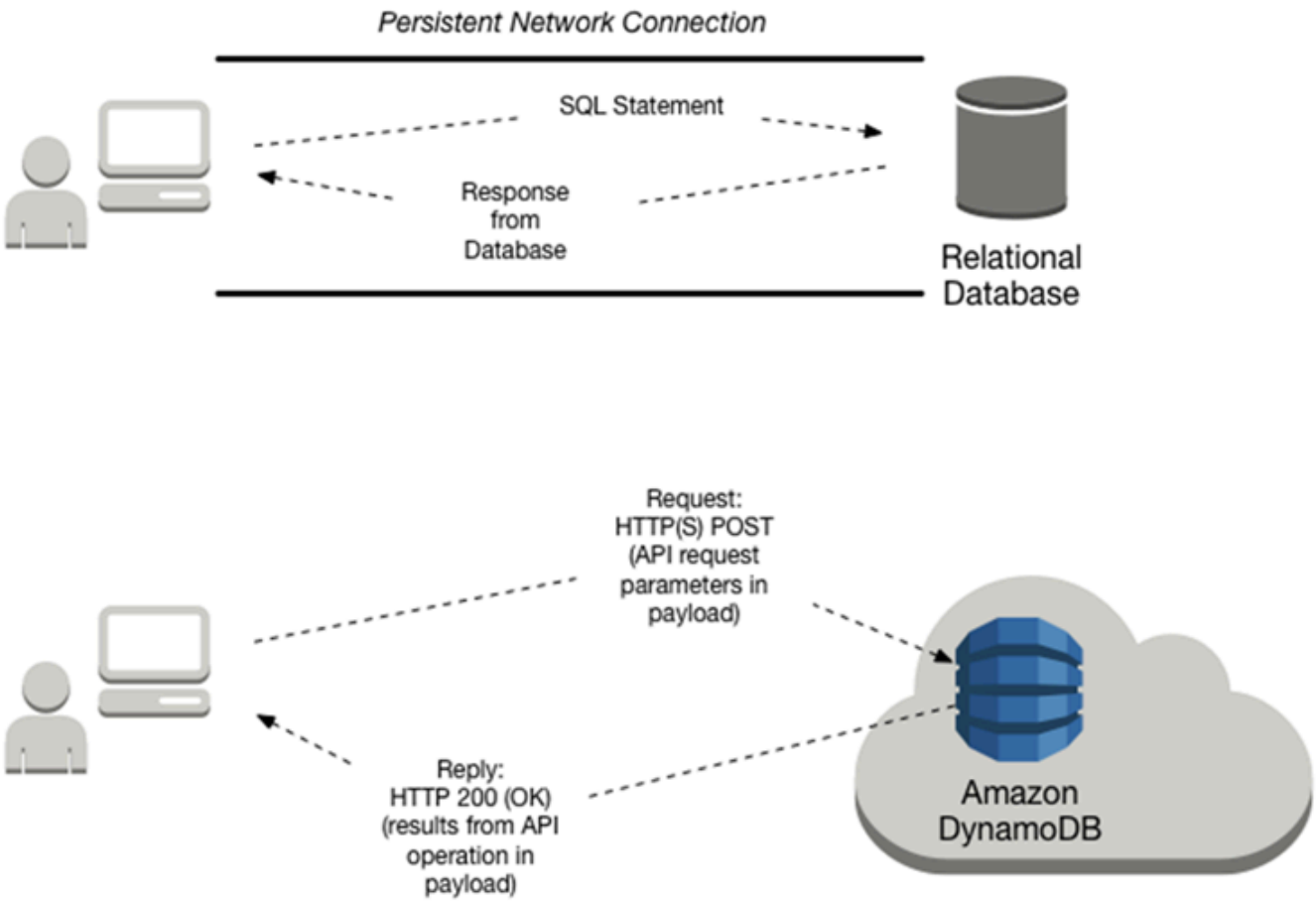
Merkmal	Relationales Datenbank managementsystem (RDBMS)	Amazon DynamoDB
		Sie Daten in DynamoDB auswählen, einfügen, aktualisieren und löschen. Anwendungen können die AWS Software Development Kits (SDKs) verwenden, um mit DynamoDB zu arbeiten und dabei objektbasierte, dokumentenzentrierte oder einfache Schnittstellen zu verwenden.
Leistung	Da relationale Datenbanken für das Speichern optimiert sind, hängt die Leistung in der Regel vom Datenträg ersubsystem ab. Developer und Datenbankadministratoren müssen Abfragen, Indizes und Tabellenstrukturen optimiere n, um Spitzenleistung zu erreichen.	Da DynamoDB für die Datenverarbeitung optimiert ist, stellt die Leistung hauptsächlich eine Funktion der zugrunde liegenden Hardware und Netzwerkl atenz dar. Als verwalteter Service schützt DynamoDB Sie und Ihre Anwendung en vor diesen Implement ierungsdetails, sodass Sie sich darauf konzentrieren können, robuste High-Performance-A nwendungen zu entwerfen und zu entwickeln.

Merkmal	Relationales Datenbank managementsystem (RDBMS)	Amazon DynamoDB
Skalierung	Eine Skalierung nach oben ist durch den Einsatz schneller er Hardware am einfachsten. Datenbanktabellen können sich auch über mehrere Hosts in einem verteilten System erstrecken. Dazu sind allerdings zusätzliche Investiti onen erforderlich. Relationa le Datenbanken unterliegen Höchstwerten im Hinblick auf die Anzahl und Größe der Dateien, wodurch der Skalierbarkeit Obergrenzen gesetzt werden.	DynamoDB wurde für die horizontale Skalierung mithilfe verteilter Hardwarecluster entwickelt. Dieses Design ermöglicht einen größeren Durchsatz ohne höhere Latenz. Kunden legen ihre Durchsatzanforderungen fest und DynamoDB weist ausreichende Ressourcen zu, um diesen Anforderungen gerecht zu werden. Es gibt weder in Bezug auf die Anzahl der Elemente pro Tabelle noch die Gesamtgröße dieser Tabelle Beschränkungen.

Unterschiede beim Zugriff auf eine relationale (SQL) Datenbank und DynamoDB

Um auf eine Datenbank zuzugreifen, muss sich Ihre Anwendung authentifizieren, um sicherzustellen, dass sie berechtigt ist, die Datenbank zu verwenden. Außerdem muss sie autorisiert sein, damit sie die Aktionen ausführen kann, zu denen sie berechtigt ist.

Das folgende Diagramm zeigt Interaktionen eines Clients mit einer relationalen Datenbank und mit Amazon DynamoDB.



In der folgenden Tabelle finden Sie weitere Informationen zu Interaktionsaufgaben von Clients:

Merkmal	Relationales Datenbank managementsystem (RDBMS)	Amazon DynamoDB
Tools für den Zugriff auf die Datenbank	Die meisten relationalen Datenbanken bieten eine Befehlszeilenschnittstelle (Command Line Interface, CLI), sodass Sie Ad-hoc-SQ L-Anweisungen eingeben und die Ergebnisse sofort sehen können.	In den meisten Fällen schreiben Sie Anwendung scode. Sie können auch die AWS-Managementkonsole AWS Command Line Interface (AWS CLI) oder NoSQL Workbench verwenden, um Ad-hoc-Anfragen an DynamoDB zu senden und die Ergebnisse anzuzeigen. Mit PartiQL, einer SQL-kompa

Merkmal	Relationales Datenbank managementsystem (RDBMS)	Amazon DynamoDB
		tiblen Abfragesprache, können Sie Daten in DynamoDB auswählen, einfügen, aktualisieren und löschen.
Verbinden mit der Datenbank	Ein Anwendungsprogramm stellt eine Netzwerkverbindung mit der Datenbank her. Wenn die Anwendung beendet ist, wird die Verbindung unterbrochen.	DynamoDB ist ein Web-Service. Interaktionen mit diesem Service sind zustandslos. Anwendungen müssen keine dauerhaften Netzwerkverbindungen unterhalten. Stattdessen erfolgt die Interaktion mit DynamoDB unter Verwendung von HTTP(S)-Anforderungen und -Antworten.
Authentifizierung	Eine Anwendung kann erst eine Verbindung mit der Datenbank herstellen, wenn sie authentifiziert ist. Das RDBMS kann die Authentifizierung selbst durchführen oder diese Aufgabe auf das Host-Betriebssystem oder einen Verzeichnisdienst auslagern.	Jede Anforderung an DynamoDB muss eine kryptografische Signatur enthalten, die die betreffende Anforderung authentifiziert. AWS SDKsSie bieten die gesamte Logik, die zum Erstellen von Signaturen und Signieranfragen erforderlich ist. Weitere Informationen finden Sie unter Signieren von AWS API-Anfragen im Allgemeine AWS-Referenz.

Merkmal	Relationales Datenbank managementsystem (RDBMS)	Amazon DynamoDB
Autorisierung	Anwendungen können nur die Aktionen ausführen, für die sie autorisiert sind. Datenbankadministratoren oder Anwendungseigentümer können mit den SQL-Anweisungen `GRANT` und `REVOKE` den Zugriff auf Datenbankobjekte (z. B. Tabellen), Daten (z. B. Zeilen in einer Tabelle) oder die Möglichkeit zum Durchführen bestimmter SQL-Anweisungen steuern.	In DynamoDB wird die Autorisierung von AWS Identity and Access Management (IAM) abgewickelt. Sie können eine IAM-Richtlinie schreiben, um Berechtigungen für eine DynamoDB-Ressource (z. B. eine Tabelle) zu erteilen und Benutzern und Rollen erlauben, die Richtlinie anzuwenden. IAM bietet außerdem eine differenzierte Zugriffskontrolle für die einzelnen Datenelemente in DynamoDB-Tabellen. Weitere Informationen finden Sie unter Identity and Access Management für Amazon DynamoDB.
Senden einer Anfrage	Die Anwendung sendet eine SQL-Anweisung für jede Datenbankoperation, die ausgeführt werden soll. Nach Erhalt der SQL-Anweisung überprüft das RDBMS seine Syntax, erstellt einen Plan zum Ausführen des Vorgangs und führt dann den Plan aus.	Die Anwendung sendet HTTP(S)-Anfragen an DynamoDB. Die Anfragen enthalten den Namen der auszuführenden DynamoDB-Operation sowie die entsprechenden Parameter. DynamoDB führt die Anfrage sofort aus.

Merkmal	Relationales Datenbank managementsystem (RDBMS)	Amazon DynamoDB
Empfangen einer Antwort	Das RDBMS gibt die Ergebnisse der SQL-Anwei sung zurück. Wenn ein Fehler auftritt, gibt das RDBMS einen Fehlerstatus und eine Fehlermeldung aus.	DynamoDB gibt eine HTTP(S)-Antwort mit den Ergebnissen der Operation zurück. Wenn ein Fehler auftritt, gibt DynamoDB einen HTTP-Fehlerstatus und Fehlermeldungen zurück.

Unterschiede zwischen einer relationalen (SQL) Datenbank und DynamoDB beim Erstellen einer Tabelle

Tabellen stellen die grundlegenden Datenstrukturen in relationalen Datenbanken und in Amazon DynamoDB dar. Bei relationalen Datenbankmanagementsystemen (RDBMS) müssen Sie das Tabellenschema beim Erstellen der Tabelle definieren. Im Gegensatz dazu sind DynamoDB-Tabellen schemalos mit Ausnahme des Primärschlüssels müssen Sie beim Erstellen einer Tabelle keine zusätzlichen Attribute oder Datentypen definieren.

Im folgenden Abschnitt wird verglichen, wie Sie eine Tabelle mit SQL erstellen würden und wie Sie sie mit DynamoDB erstellen würden.

Themen

- Erstellen einer Tabelle mit SQL
- Erstellen einer Tabelle mit DynamoDB

Erstellen einer Tabelle mit SQL

Mit SQL verwenden Sie die `CREATE TABLE`-Anweisung, um eine Tabelle zu erstellen, wie im folgenden Beispiel veranschaulicht.

```
CREATE TABLE Music (
    Artist VARCHAR(20) NOT NULL,
    SongTitle VARCHAR(30) NOT NULL,
    AlbumTitle VARCHAR(25),
```

```
    Year INT,
    Price FLOAT,
    Genre VARCHAR(10),
    Tags TEXT,
    PRIMARY KEY(Artist, SongTitle)
);
```

Der Primärschlüssel für diese Tabelle besteht aus Artist und. SongTitle

Definieren Sie alle Tabellenspalten und Datentypen sowie den Primärschlüssel der Tabelle. (Mit der `ALTER TABLE`-Anweisung können Sie diese Definitionen später ändern, falls erforderlich.)

Viele SQL-Implementierungen ermöglichen eine Definition der Speicherspezifikationen für Ihre Tabelle im Rahmen der `CREATE TABLE`-Anweisungen. Die Tabelle wird mit den Standardspeichereinstellungen erstellt, sofern Sie nichts anderes angeben. In einer Produktionsumgebung kann ein Datenbankadministrator Ihnen dabei helfen, die optimalen Speicherparameter festzulegen.

Erstellen einer Tabelle mit DynamoDB

Verwenden Sie die Aktion `CreateTable`, um eine Tabelle mit dem Modus bereitgestellter Kapazität zu erstellen und Parameter wie folgt anzugeben:

```
{
    TableName : "Music",
    KeySchema: [
        {
            AttributeName: "Artist",
            KeyType: "HASH" //Partition key
        },
        {
            AttributeName: "SongTitle",
            KeyType: "RANGE" //Sort key
        }
    ],
    AttributeDefinitions: [
        {
            AttributeName: "Artist",
            AttributeType: "S"
        },
        {
            AttributeName: "SongTitle",
```

```
            AttributeType: "S"
        }
    ],
    ProvisionedThroughput: {       // Only specified if using provisioned mode
        ReadCapacityUnits: 1,
        WriteCapacityUnits: 1
    }
}
```

Der Primärschlüssel für diese Tabelle besteht aus Artist (Partitionsschlüssel) und SongTitle(Sortierschlüssel).

Sie müssen die folgenden Parameter für `CreateTable` angeben:

- `TableName` – Name der Tabelle.
- `KeySchema` – Attribute, die für den Primärschlüssel verwendet werden. Weitere Informationen erhalten Sie unter Tabellen, Elemente und Attribute und Primärschlüssel.
- `AttributeDefinitions` – Datentypen für die Schlüsselschemaattribute.
- `ProvisionedThroughput (for provisioned tables)` – Anzahl der Lese- und Schreibvorgänge pro Sekunde, die Sie für diese Tabelle benötigen. DynamoDB reserviert ausreichend Speicher- und Systemressourcen, sodass Ihre Durchsatzanforderungen immer erfüllt werden. Mit der Aktion `UpdateTable` können Sie diese später ändern, falls erforderlich. Sie müssen die Speicheranforderungen einer Tabelle nicht angeben, da die Speicherzuweisung vollständig von DynamoDB verwaltet wird.

Unterschiede zwischen dem Abrufen von Tabelleninformationen aus einer relationalen (SQL) Datenbank und DynamoDB

Sie können überprüfen, ob eine Tabelle Ihren Spezifikationen entsprechend erstellt wurde. In einer relationalen Datenbank wird das gesamte Tabellenschema angezeigt. Amazon-DynamoDB-Tabellen sind schemalos, sodass nur die Primärschlüsselattribute angezeigt werden.

Themen

- Abrufen von Informationen zu einer Tabelle mit SQL
- Abrufen von Informationen zu einer Tabelle in DynamoDB

Abrufen von Informationen zu einer Tabelle mit SQL

Die meisten relationalen Datenbankmanagementsysteme (RDBMS) ermöglichen Ihnen, die Struktur einer Tabelle – Spalten, Datentypen, Primärschlüsseldefinition usw. – zu beschreiben. Dafür gibt es in SQL keine Standardmethode. Viele Datenbanksysteme stellen den Befehl `DESCRIBE` zur Verfügung. Das folgende Beispiel kommt aus MySQL.

```
DESCRIBE Music;
```

In diesem Beispiel wird die Struktur der Tabelle mit allen Spaltennamen, Datentypen und Größen zurückgegeben.

```
+------------+-------------+------+-----+---------+-------+
| Field      | Type        | Null | Key | Default | Extra |
+------------+-------------+------+-----+---------+-------+
| Artist     | varchar(20) | NO   | PRI | NULL    |       |
| SongTitle  | varchar(30) | NO   | PRI | NULL    |       |
| AlbumTitle | varchar(25) | YES  |     | NULL    |       |
| Year       | int(11)     | YES  |     | NULL    |       |
| Price      | float       | YES  |     | NULL    |       |
| Genre      | varchar(10) | YES  |     | NULL    |       |
| Tags       | text        | YES  |     | NULL    |       |
+------------+-------------+------+-----+---------+-------+
```

Der Primärschlüssel für diese Tabelle besteht aus Artist und. SongTitle

Abrufen von Informationen zu einer Tabelle in DynamoDB

DynamoDB verfügt über eine `DescribeTable`-Aktion, die ähnlich ist. Der einzige Parameter ist der Tabellenname.

```
{
    TableName : "Music"
}
```

Die Antwort von `DescribeTable` sieht wie folgt aus:

```
{
  "Table": {
```

```
    "AttributeDefinitions": [
      {
        "AttributeName": "Artist",
        "AttributeType": "S"
      },
      {
        "AttributeName": "SongTitle",
        "AttributeType": "S"
      }
    ],
    "TableName": "Music",
    "KeySchema": [
      {
        "AttributeName": "Artist",
        "KeyType": "HASH"  //Partition key
      },
      {
        "AttributeName": "SongTitle",
        "KeyType": "RANGE"  //Sort key
      }
    ],

    ...
```

`DescribeTable` gibt auch Informationen über Indexe in der Tabelle, Einstellungen für den bereitgestellten Durchsatz, eine ungefähre Elementanzahl und andere Metadaten zurück.

Unterschiede zwischen einer relationalen (SQL) Datenbank und DynamoDB beim Schreiben von Daten in eine Tabelle

Relationale Datenbanktabellen enthalten Datenzeilen. Die Zeilen bestehen aus Spalten. Amazon DynamoDB Tabellen enthalten Elemente. Elemente bestehen aus Attributen.

In diesem Abschnitt wird beschrieben, wie Sie eine Zeile (bzw. ein Element) in eine Tabelle schreiben.

Themen

- Schreiben von Daten in eine Tabelle mit SQL
- Schreiben von Daten in eine Tabelle in DynamoDB

Schreiben von Daten in eine Tabelle mit SQL

Eine Tabelle in einer relationalen Datenbank ist eine zweidimensionale Datenstruktur, die sich aus Zeilen und Spalten zusammensetzt. Einige Datenbankmanagementsysteme bieten auch Unterstützung für halbstrukturierte Daten, in der Regel mit nativen JSON- oder XML-Datentypen. Die Implementierungsdetails sind je nach Anbieter unterschiedlich.

In SQL verwenden Sie die `INSERT`-Anweisung zum Hinzufügen einer Zeile zu einer Tabelle.

```
INSERT INTO Music
    (Artist, SongTitle, AlbumTitle,
    Year, Price, Genre,
    Tags)
VALUES(
    'No One You Know', 'Call Me Today', 'Somewhat Famous',
    2015, 2.14, 'Country',
    '{"Composers": ["Smith", "Jones", "Davis"],"LengthInSeconds": 214}'
);
```

Der Primärschlüssel für diese Tabelle besteht aus Artist und. SongTitle Sie müssen Werte für diese Spalten angeben.

> **Note**
>
> In diesem Beispiel wird die Spalte Tags zum Speichern halbstrukturierter Daten zu den Songs in der Tabelle Musik verwendet. Die Spalte Tags ist als Typ TEXT definiert. Dieser kann bis zu 65 535 Zeichen in MySQL speichern.

Schreiben von Daten in eine Tabelle in DynamoDB

In Amazon DynamoDB können Sie entweder die DynamoDB-API oder PartiQL, eine SQL-kompatible Abfragesprache, verwenden, um einer Tabelle ein Element hinzuzufügen.

DynamoDB API

Mit der DynamoDB-API verwenden Sie die `PutItem`-Operation zum Hinzufügen eines Elements zu einer Tabelle.

```
{
    TableName: "Music",
```

```
    Item: {
        "Artist":"No One You Know",
        "SongTitle":"Call Me Today",
        "AlbumTitle":"Somewhat Famous",
        "Year": 2015,
        "Price": 2.14,
        "Genre": "Country",
        "Tags": {
            "Composers": [
                  "Smith",
                  "Jones",
                  "Davis"
            ],
            "LengthInSeconds": 214
        }
    }
}
```

Der Primärschlüssel für diese Tabelle besteht aus Artist und. SongTitle Sie müssen Werte für diese Attribute angeben.

Im Folgenden sind einige wichtige Fakten zu diesem `PutItem`-Beispiel aufgeführt:

- DynamoDB bietet native Unterstützung für Dokumente unter Verwendung von JSON. Dadurch ist DynamoDB ideal zum Speichern halbstrukturierter Daten wie Tags geeignet. Sie können Daten innerhalb von JSON-Dokumenten auch abrufen und bearbeiten.
- Die Musiktabelle hat außer dem Primärschlüssel (Künstler und SongTitle) keine vordefinierten Attribute.
- Die meisten SQL-Datenbanken sind transaktionsorientiert. Wenn Sie eine `INSERT`-Anweisung erstellen, sind die Datenänderungen erst dauerhaft, sobald Sie eine `COMMIT`-Anweisung generieren. Mit Amazon DynamoDB sind die Auswirkungen einer `PutItem`-Aktion dauerhaft, wenn DynamoDB mit einem HTTP 200-Statuscode (`OK`) antwortet.

Im Folgenden sind weitere `PutItem`-Beispiele aufgeführt.

```
{
    TableName: "Music",
    Item: {
        "Artist": "No One You Know",
        "SongTitle": "My Dog Spot",
```

```
        "AlbumTitle":"Hey Now",
        "Price": 1.98,
        "Genre": "Country",
        "CriticRating": 8.4
    }
}
```

```
{
    TableName: "Music",
    Item: {
        "Artist": "No One You Know",
        "SongTitle": "Somewhere Down The Road",
        "AlbumTitle":"Somewhat Famous",
        "Genre": "Country",
        "CriticRating": 8.4,
        "Year": 1984
    }
}
```

```
{
    TableName: "Music",
    Item: {
        "Artist": "The Acme Band",
        "SongTitle": "Still In Love",
        "AlbumTitle":"The Buck Starts Here",
        "Price": 2.47,
        "Genre": "Rock",
        "PromotionInfo": {
            "RadioStationsPlaying":[
                 "KHCR", "KBQX", "WTNR", "WJJH"
            ],
            "TourDates": {
                "Seattle": "20150625",
                "Cleveland": "20150630"
            },
            "Rotation": "Heavy"
        }
    }
}
```

```
{
    TableName: "Music",
```

```
    Item: {
        "Artist": "The Acme Band",
        "SongTitle": "Look Out, World",
        "AlbumTitle":"The Buck Starts Here",
        "Price": 0.99,
        "Genre": "Rock"
    }
}
```

> **Note**
>
> Neben `PutItem` unterstützt DynamoDB eine `BatchWriteItem`-Aktion zum Schreiben mehrerer Elemente gleichzeitig.

PartiQL for DynamoDB

Mit PartiQL verwenden Sie die `ExecuteStatement`-Operation zum Hinzufügen eines Elements zu einer Tabelle mit PartiQL `Insert`-Anweisung.

```
INSERT into Music value {
    'Artist': 'No One You Know',
    'SongTitle': 'Call Me Today',
    'AlbumTitle': 'Somewhat Famous',
    'Year' : '2015',
    'Genre' : 'Acme'
}
```

Der Primärschlüssel für diese Tabelle besteht aus Artist und SongTitle. Sie müssen Werte für diese Attribute angeben.

> **Note**
>
> Codebeispiele, die `Insert` und `ExecuteStatement` verwenden, finden Sie unter PartiQL-Insert-Anweisungen für DynamoDB.

Wichtige Unterschiede zwischen einer relationalen (SQL) Datenbank und DynamoDB beim Lesen von Daten aus einer Tabelle

Mit SQL verwenden Sie die `SELECT`-Anweisung, um eine oder mehrere Zeilen aus einer Tabelle abzurufen. Sie verwenden die `WHERE`-Klausel, um die Daten zu bestimmen, die an Sie zurückgesendet werden.

Dies unterscheidet sich von der Verwendung von Amazon DynamoDB, das die folgenden Operationen zum Lesen von Daten bereitstellt:

- `ExecuteStatement` ruft ein einzelnes oder mehrere Elemente aus einer Tabelle ab. `BatchExecuteStatement` ruft mehrere Elemente aus verschiedenen Tabellen in einem einzigen Vorgang ab. Beide dieser Operationen verwenden PartiQL, eine SQL-kompatible Abfragesprache.
- `GetItem` – Ruft ein einzelnes Element aus einer Tabelle ab. Dies ist die effizienteste Methode, um ein einzelnes Element zu lesen, da sie direkten Zugriff auf den physischen Standort des Elements ermöglicht. (DynamoDB bietet auch die `BatchGetItem`-Operation, mit der Sie bis zu 100 `GetItem`-Aufrufe in einer einzigen Operation ausführen können.)
- `Query` – Ruft alle Elemente mit einem bestimmten Partitionsschlüssel ab. Innerhalb dieser Elemente können Sie eine Bedingung auf den Sortierschlüssel anwenden und nur eine Teilmenge der Daten abrufen. `Query` ermöglicht schnellen und effizienten Zugriff auf die Partitionen, in denen die Daten gespeichert sind. (Weitere Informationen finden Sie unter Partitionen und Datenverteilung in DynamoDB.)
- `Scan` – Ruft alle Elemente in der angegebenen Tabelle ab. (Diese Operation sollte nicht über großen Tabellen ausgeführt werden, da sie große Mengen an Systemressourcen belegen kann.)

> **Note**
>
> Mit einer relationalen Datenbank können Sie die `SELECT`-Anweisung verwenden, um Daten aus mehreren Tabellen zu verknüpfen und die Ergebnisse zurückzugeben. Joins (Verknüpfungen) sind eine Voraussetzung für das relationale Modell. Um sicherzustellen, dass Joins effizient ausgeführt werden, sollten die Datenbank und die zugehörigen Anwendungen kontinuierlich leistungsoptimiert werden. DynamoDB ist eine nicht relationale NoSQL-Datenbank und unterstützt keine Tabellen-Joins. Stattdessen lesen die Anwendungen die Daten aus den Tabellen nacheinander aus.

In den folgenden Abschnitten werden die verschiedenen Anwendungsfälle für das Lesen von Daten sowie Anleitungen zum Ausführen dieser Aufgaben mit einer relationalen Datenbank und mit DynamoDB beschrieben.

Themen

- Unterschiede beim Lesen eines Elements mit dem zugehörigen Primärschlüssel
- Unterschiede beim Abfragen einer Tabelle
- Unterschiede beim Scannen einer Tabelle

Unterschiede beim Lesen eines Elements mit dem zugehörigen Primärschlüssel

Ein gängiges Zugriffsmuster für Datenbanken besteht darin, ein einzelnes Element aus einer Tabelle zu lesen. Sie müssen den Primärschlüssel des gewünschten Elements angeben.

Themen

- Lesen eines Elements mit dem zugehörigen Primärschlüssel mit SQL
- Lesen eines Elements mit dem zugehörigen Primärschlüssel in DynamoDB

Lesen eines Elements mit dem zugehörigen Primärschlüssel mit SQL

In SQL verwenden Sie die `SELECT`-Anweisung zum Abrufen von Daten aus einer Tabelle. Sie können eine oder mehrere Spalten im Ergebnis abfragen (oder alle Spalten, wenn Sie den *-Operator verwenden). Die `WHERE`-Klausel bestimmt, welche Zeilen zurückgegeben werden.

Das folgende Beispiel umfasst eine `SELECT`-Anweisung zum Abrufen einer einzelnen Zeile aus der Tabelle Music. Die `WHERE`-Klausel gibt die Primärschlüsselwerte an.

```
SELECT *
FROM Music
WHERE Artist='No One You Know' AND SongTitle = 'Call Me Today'
```

Sie können diese Abfrage ändern und nur eine Teilmenge der Spalten abrufen.

```
SELECT AlbumTitle, Year, Price
FROM Music
WHERE Artist='No One You Know' AND SongTitle = 'Call Me Today'
```

Beachten Sie, dass der Primärschlüssel für diese Tabelle aus Artist und besteht. SongTitle

Lesen eines Elements mit dem zugehörigen Primärschlüssel in DynamoDB

In Amazon DynamoDB können Sie entweder die DynamoDB-API oder PartiQL, eine SQL-kompatible Abfragesprache, verwenden, um ein Element aus einer Tabelle zu lesen.

DynamoDB API

Mit der DynamoDB-API verwenden Sie die `PutItem`-Operation zum Hinzufügen eines Elements zu einer Tabelle.

DynamoDB stellt die Aktion `GetItem` zum Abrufen eines Elements über dessen Primärschlüssel bereit. `GetItem` ist äußerst effizient, da die Aktion direkten Zugriff auf den physischen Speicherort des Elements ermöglicht. (Weitere Informationen finden Sie unter Partitionen und Datenverteilung in DynamoDB.)

Standardmäßig gibt `GetItem` das gesamte Element mit allen zugehörigen Attributen zurück.

```
{
    TableName: "Music",
    Key: {
        "Artist": "No One You Know",
        "SongTitle": "Call Me Today"
    }
}
```

Sie können einen `ProjectionExpression`-Parameter hinzufügen, um nur einige der Attribute zurückzugeben.

```
{
    TableName: "Music",
    Key: {
        "Artist": "No One You Know",
        "SongTitle": "Call Me Today"
    },
    "ProjectionExpression": "AlbumTitle, Year, Price"
}
```

Beachten Sie, dass der Primärschlüssel für diese Tabelle aus Artist und besteht. SongTitle

Die `GetItem`-DynamoDB-Aktion ist sehr effizient. Sie bestimmt anhand der Primärschlüsselwerte den genauen Speicherort des entsprechenden Elements und ruft es direkt von dort ab. Die SQL-Anweisung `SELECT` ist im Hinblick auf das Abrufen von Elementen mithilfe von Primärschlüsselwerten ähnlich effizient.

Die SQL-Anweisung `SELECT` unterstützt viele Arten von Abfragen und Tabellen-Scans. DynamoDB bietet eine ähnliche Funktionalität mit den Aktionen `Query` und `Scan`, die in Unterschiede beim Abfragen einer Tabelle und Unterschiede beim Scannen einer Tabelle beschrieben sind.

Die SQL-Anweisung `SELECT` kann Tabellen-Joins durchführen, sodass Sie Daten aus mehreren Tabellen gleichzeitig abrufen können. Joins sind am effektivsten, wenn die Datenbanktabellen normalisiert und die Beziehungen zwischen den Tabellen eindeutig sind. Wenn Sie allerdings zu viele Tabellen in einer `SELECT`-Anweisung verknüpfen, kann sich dies negativ auf die Anwendungsleistung auswirken. Sie können diese Probleme umgehen, indem Sie Datenbankreplikation, materialisierte Ansichten oder Abfrageumschreibungen verwenden.

DynamoDB ist eine nicht zusammenhängende Datenbank und unterstützt keine Tabellen-Joins. Wenn Sie eine vorhandene Anwendung von einer relationalen Datenbank nach DynamoDB migrieren, müssen Sie Ihr Datenmodell denormalisieren, damit keine Joins erforderlich sind.

PartiQL for DynamoDB

Mit PartiQL verwenden Sie die `ExecuteStatement`-Operation zum Lesen eines Elements aus einer Tabelle mit PartiQL `Select`-Anweisung.

```
SELECT AlbumTitle, Year, Price
FROM Music
WHERE Artist='No One You Know' AND SongTitle = 'Call Me Today'
```

Beachten Sie, dass der Primärschlüssel für diese Tabelle aus Artist und besteht SongTitle.

> **Note**
>
> Die select PartiQL-Anweisung kann auch verwendet werden, um eine DynamoDB-Tabelle abzufragen oder zu scannen

Codebeispiele, die `Select` und `ExecuteStatement` verwenden, finden Sie unter PartiQL-Select-Anweisungen für DynamoDB.

Unterschiede beim Abfragen einer Tabelle

Ein anderes gängiges Zugriffsmuster besteht darin, mehrere Elemente aus einer Tabelle basierend auf Ihren Abfragekriterien zu lesen.

Themen

- Abfragen einer Tabelle mit SQL
- Abfragen einer Tabelle in DynamoDB

Abfragen einer Tabelle mit SQL

Bei Verwendung von SQL können Sie mit der Anweisung `SELECT` Schlüsselspalten, Nicht-Schlüsselspalten oder eine beliebige Kombination daraus abfragen. Die `WHERE`-Klausel bestimmt, welche Zeilen zurückgegeben werden, wie in den folgenden Beispielen gezeigt.

```
/* Return a single song, by primary key */

SELECT * FROM Music
WHERE Artist='No One You Know' AND SongTitle = 'Call Me Today';
```

```
/* Return all of the songs by an artist */

SELECT * FROM Music
WHERE Artist='No One You Know';
```

```
/* Return all of the songs by an artist, matching first part of title */

SELECT * FROM Music
WHERE Artist='No One You Know' AND SongTitle LIKE 'Call%';
```

```
/* Return all of the songs by an artist, only if the price is less than 1.00 */

SELECT * FROM Music
WHERE Artist='No One You Know'
AND Price < 1.00;
```

Beachten Sie, dass der Primärschlüssel für diese Tabelle aus Artist und besteht SongTitle.

Abfragen einer Tabelle in DynamoDB

In Amazon DynamoDB können Sie entweder die DynamoDB-API oder PartiQL, eine SQL-kompatible Abfragesprache, verwenden, um ein Element aus einer Tabelle abzufragen.

DynamoDB API

Mit Amazon DynamoDB können Sie mit der Aktion `Query` Daten auf ähnliche Weise abrufen. Die Aktion `Query` bietet schnellen und effizienten Zugriff auf die physischen Orte, an denen die Daten gespeichert sind. Weitere Informationen finden Sie unter Partitionen und Datenverteilung in DynamoDB.

Sie können `Query` mit jeder Tabelle oder jedem sekundären Index verwenden. Sie müssen eine Gleichheitsbedingung für den Wert des Partitionsschlüssels festlegen und können optional eine andere Bedingung für das Sortierschlüsselattribut angeben, falls definiert.

Der Parameter `KeyConditionExpression` gibt die Schlüsselwerte an, die Sie abfragen möchten. Sie können einen optionalen `FilterExpression` verwenden, um bestimmte Elemente aus den Ergebnissen zu entfernen, bevor diese zurückgegeben werden.

In DynamoDB müssen Sie als Platzhalter `ExpressionAttributeValues` in Ausdrucksparametern (z. B. `KeyConditionExpression` und `FilterExpression`) verwenden. Dies entspricht der Verwendung von Bindungsvariablen in relationalen Datenbanken, in denen Sie die tatsächlichen Werte in der `SELECT`-Anweisung zur Laufzeit ersetzen.

Beachten Sie, dass der Primärschlüssel für diese Tabelle aus Artist und besteht SongTitle.

Im Folgenden sind weitere DynamoDB `Query`-Beispiele aufgeführt.

```
// Return a single song, by primary key

{
    TableName: "Music",
    KeyConditionExpression: "Artist = :a and SongTitle = :t",
    ExpressionAttributeValues: {
        ":a": "No One You Know",
        ":t": "Call Me Today"
    }
}
```

```
// Return all of the songs by an artist
```

```
{
    TableName: "Music",
    KeyConditionExpression: "Artist = :a",
    ExpressionAttributeValues: {
        ":a": "No One You Know"
    }
}
```

```
// Return all of the songs by an artist, matching first part of title

{
    TableName: "Music",
    KeyConditionExpression: "Artist = :a and begins_with(SongTitle, :t)",
    ExpressionAttributeValues: {
        ":a": "No One You Know",
        ":t": "Call"
    }
}
```

```
// Return all of the songs by an artist, only if the price is less than 1.00

{
    TableName: "Music",
    KeyConditionExpression: "Artist = :a",
    FilterExpression: "Price < :p",
    ExpressionAttributeValues: {
        ":a": "No One You Know",
        ":p": 1.00
    }
}
```

> **Note**
>
> A `FilterExpression` wird nach den `Query` Lesevorgängen angewendet, die den Elementen entsprechen, sodass die verbrauchte Lesekapazität nicht reduziert wird. Modellieren Sie Ihre Daten nach Möglichkeit so, dass Bereichsbedingungen für effiziente Abfragen `KeyConditionExpression` auf den Sortierschlüssel angewendet werden. Weitere Informationen finden Sie unter Abfragen von Tabellen in DynamoDB.

PartiQL for DynamoDB

Mit PartiQL können Sie eine Abfrage ausführen, indem Sie die `ExecuteStatement`-Aktion und die `Select`-Anweisung auf den Partitionsschlüssel anwenden.

```
SELECT AlbumTitle, Year, Price
FROM Music
WHERE Artist='No One You Know'
```

Verwendung der `SELECT`-Anweisung gibt auf diese Weise alle Songs zurück, die mit diesem bestimmten `Artist` assoziiert sind.

Codebeispiele, die `Select` und `ExecuteStatement` verwenden, finden Sie unter PartiQL-Select-Anweisungen für DynamoDB.

Unterschiede beim Scannen einer Tabelle

In SQL gibt eine `SELECT`-Anweisung ohne `WHERE`-Klausel alle Zeilen einer Tabelle zurück. In Amazon DynamoDB übernimmt die `Scan`-Operation diese Funktion. In beiden Fällen können Sie alle oder nur einige Elemente abrufen.

Unabhängig davon, ob Sie eine SQL- oder eine NoSQL-Datenbank verwenden, sollten Scans jedoch sparsam eingesetzt werden, da sie große Mengen an Systemressourcen belegen können. Manchmal ist ein Scan angebracht (z. B. Scannen einer kleinen Tabelle) oder unvermeidbar (wie die Ausführung eines Massenexports von Daten). Als allgemeine Regel sollten Sie Ihre Anwendungen so konzipieren, dass Scans vermieden werden. Weitere Informationen finden Sie unter Abfragen von Tabellen in DynamoDB.

> **Note**
>
> Bei einem Massenexport wird außerdem mindestens 1 Datei pro Partition erstellt. Alle Elemente in jeder Datei stammen aus dem Hash-Keyspace der betreffenden Partition.

Themen

- Scannen einer Tabelle mit SQL
- Scannen einer Tabelle in DynamoDB

Scannen einer Tabelle mit SQL

Bei Verwendung von SQL können Sie eine Tabelle scannen und alle Daten mit einer `SELECT`-Anweisung ohne Angabe einer `WHERE`-Bedingung abrufen. Sie können eine oder mehrere Spalten im Ergebnis abfragen. Sie können auch alle Spalten abfragen, wenn Sie das Platzhalterzeichen (*) verwenden.

Es folgen Beispiele für eine `SELECT`-Anweisung.

```
/* Return all of the data in the table */
SELECT * FROM Music;
```

```
/* Return all of the values for Artist and Title */
SELECT Artist, Title FROM Music;
```

Scannen einer Tabelle in DynamoDB

In Amazon DynamoDB können Sie entweder die DynamoDB-API oder PartiQL, eine SQL-kompatible Abfragesprache, verwenden, um einen Scan für eine Tabelle durchzuführen.

DynamoDB API

Mit der DynamoDB-API verwenden Sie die `Scan`-Operation, um ein oder mehrere Elemente und Elementattribute zurückzugeben, indem Sie auf jedes Element in einer Tabelle oder einem sekundären Index zugreifen.

```
// Return all of the data in the table
{
    TableName:  "Music"
}
```

```
// Return all of the values for Artist and Title
{
    TableName:  "Music",
    ProjectionExpression: "Artist, Title"
}
```

Die `Scan`-Aktion bietet auch einen Parameter `FilterExpression`, der es ermöglicht, Elemente zu verwerfen, die nicht in den Ergebnissen enthalten sein sollen. `FilterExpression` wird

angewendet, nachdem der Scan durchgeführt wurde, aber bevor die Ergebnisse an Sie zurückgegeben werden. (Für große Tabellen wird dies nicht empfohlen: Sie werden für den gesamten `Scan` belastet, auch wenn nur einige übereinstimmende Elemente zurückgegeben werden.)

PartiQL for DynamoDB

Mit PartiQL führen Sie einen Scan durch, indem Sie den `ExecuteStatement`-Vorgang verwenden, um den gesamten Inhalt einer Tabelle mit der `Select`-Anweisung zurückzugeben.

```
SELECT AlbumTitle, Year, Price
FROM Music
```

Beachten Sie, dass diese Anweisung alle Elemente für in der Tabelle Musik zurückgibt.

Codebeispiele mit `Select` und `ExecuteStatement` finden Sie unter PartiQL-Select-Anweisungen für DynamoDB

Unterschiede zwischen einer relationalen (SQL) Datenbank und DynamoDB beim Verwalten von Indizes

Indexe ermöglichen Ihnen den Zugriff auf alternative Abfragemuster und können Abfragen beschleunigen. In diesem Abschnitt werden die Indexerstellung und -verwendung in SQL und Amazon DynamoDB verglichen und gegenübergestellt.

Ganz gleich, ob Sie eine relationale Datenbank oder DynamoDB verwenden, sollten Sie bei der Indexerstellung mit Bedacht vorgehen. Bei jedem Schreibvorgang in einer Tabelle müssen alle Indexe der Tabelle aktualisiert werden. In einer Umgebung mit vielen Schreibvorgängen und großen Tabellen können dadurch große Mengen von Systemressourcen belegt werden. In schreibgeschützten Umgebungen (bzw. Umgebungen, indenen Daten vor allem gelesen werden) stellt dies kein Problem dar. Sie sollten jedoch sicherstellen, dass die Indexe tatsächlich von der Anwendung verwendet werden und nicht nur Speicherplatz belegen.

Themen

- Unterschiede zwischen einer relationalen (SQL) Datenbank und DynamoDB beim Erstellen eines Index
- Unterschiede zwischen einer relationalen (SQL) Datenbank und DynamoDB beim Abfragen und Scannen eines Index

Unterschiede zwischen einer relationalen (SQL) Datenbank und DynamoDB beim Erstellen eines Index

Vergleichen Sie die Anweisung `CREATE INDEX` in SQL mit der Operation `UpdateTable` in Amazon DynamoDB.

Themen

- Erstellen eines Index mit SQL
- Erstellen eines Index in DynamoDB

Erstellen eines Index mit SQL

In einer relationalen Datenbank ist ein Index eine Datenstruktur, mit der Sie schnelle Abfragen für verschiedene Spalten einer Tabelle ausführen können. Mit der SQL-Anweisung `CREATE INDEX` können Sie einer vorhandenen Tabelle einen Index hinzufügen, indem Sie die zu indizierenden Spalten angeben. Nachdem der Index erstellt wurde, können Sie die Daten in der Tabelle wie üblich abfragen. Die Datenbank kann die angegebenen Zeilen in der Tabelle anhand des Index schnell finden, sodass nicht die gesamte Tabelle gescannt werden muss.

Nachdem Sie einen Index erstellt haben, wird dieser von der Datenbank gepflegt. Sobald Sie Daten in der Tabelle ändern, wird der Index automatisch den Änderungen in der Tabelle entsprechend angepasst.

In MySQL können Sie einen Index wie den folgenden erstellen:

```
CREATE INDEX GenreAndPriceIndex
ON Music (genre, price);
```

Erstellen eines Index in DynamoDB

In DynamoDB können Sie einen sekundären Index für ähnliche Zwecke erstellen und verwenden.

Indexe in DynamoDB unterscheiden sich von ihren relationalen Gegenstücken. Wenn Sie einen sekundären Index erstellen, müssen Sie dessen Schlüsselattribute angeben – einen Partitionsschlüssel und einen Sortierschlüssel. Nachdem Sie den sekundären Index erstellt haben, können Sie `Query` oder `Scan` wie bei einer Tabelle erstellen. DynamoDB bietet keinen Abfrageoptimierer, sodass ein sekundärer Index nur bei der `Query`- oder `Scan`-Aktion verwendet wird.

DynamoDB unterstützt zwei verschiedene Arten von Indexen:

- Globale sekundäre Indizes – Der Primärschlüssel des Index kann aus zwei beliebigen Attributen der Tabelle bestehen.
- Lokale sekundäre Indizes – Der Partitionsschlüssel des Index muss mit dem Partitionsschlüssel der Tabelle übereinstimmen. Der Sortierschlüssel kann ein beliebiges, anderes Attribut sein.

DynamoDB stellt sicher, dass die Daten in einem sekundären Index schließlich mit seiner Tabelle konsistent sind. Sie können stark konsistente `Query`- oder `Scan`-Aktionen für eine Tabelle oder einen lokalen sekundären Index anfordern. Globale sekundäre Indizes unterstützen jedoch nur die letztendliche Datenkonsistenz.

Sie können einer vorhandenen Tabelle einen globalen sekundären Index hinzufügen, indem Sie die `UpdateTable`-Aktion verwenden und `GlobalSecondaryIndexUpdates` angeben.

```
{
    TableName: "Music",
    AttributeDefinitions:[
        {AttributeName: "Genre", AttributeType: "S"},
        {AttributeName: "Price", AttributeType: "N"}
    ],
    GlobalSecondaryIndexUpdates: [
        {
            Create: {
                IndexName: "GenreAndPriceIndex",
                KeySchema: [
                    {AttributeName: "Genre", KeyType: "HASH"}, //Partition key
                    {AttributeName: "Price", KeyType: "RANGE"}, //Sort key
                ],
                Projection: {
                    "ProjectionType": "ALL"
                },
                ProvisionedThroughput: {                                // Only
 specified if using provisioned mode
                    "ReadCapacityUnits": 1,"WriteCapacityUnits": 1
                }
            }
        }
    ]
}
```

Sie müssen die folgenden Parameter für `UpdateTable` angeben:

- `TableName` – Die Tabelle, mit der der Index verknüpft wird.
- `AttributeDefinitions` – Die Datentypen für die Schlüsselschemaattribute des Index.
- `GlobalSecondaryIndexUpdates` – Details zu dem Index, den Sie erstellen möchten:
  - `IndexName` – Ein Name für den Index.
  - `KeySchema` – Die Attribute, die für den Indexprimärschlüssel verwendet werden.
  - `Projection` – Attribute aus der Tabelle, die in den Index kopiert werden. In diesem Fall bedeutet `ALL`, dass alle Attribute kopiert werden.
  - `ProvisionedThroughput (for provisioned tables)` – Die Anzahl der Lese- und Schreibvorgänge pro Sekunde, die Sie für diesen Index benötigen. (Dieser Wert steht nicht mit den Einstellungen für den bereitgestellten Durchsatz der Tabelle in Zusammenhang.)

Ein Teil dieser Operation umfasst das Abgleichen von Daten aus der Tabelle mit dem neuen Index. Während des Abgleichs ist die Tabelle weiterhin verfügbar. Der Index ist allerdings erst bereit, wenn sich das Attribut `Backfilling` von TRUE in FALSE ändert. Sie können die Aktion `DescribeTable` zum Anzeigen dieses Attributs verwenden.

Unterschiede zwischen einer relationalen (SQL) Datenbank und DynamoDB beim Abfragen und Scannen eines Index

Vergleichen Sie das Abfragen und Scannen eines Index mithilfe der SELECT-Anweisung in SQL mit den Operationen `Query` und `Scan` in Amazon DynamoDB.

Themen

- Abfragen und Scannen eines Index mit SQL
- Abfragen und Scannen eines Index in DynamoDB

Abfragen und Scannen eines Index mit SQL

In einer relationalen Datenbank arbeiten Sie nicht direkt mit Indexen. Stattdessen fragen Sie Tabellen mithilfe von `SELECT`-Anweisungen ab. Der Abfrageoptimierer kann jeden beliebigen Index nutzen.

Ein Abfrageoptimierer ist eine Komponente eines relationalen Datenbankmanagementsystems (RDBMS), die verfügbare Indexe auswertet und bestimmt, ob diese zum Beschleunigen einer Abfrage

verwendet werden können. Wenn die Indexe zum Beschleunigen einer Abfrage verwendet werden können, greift das RDBMS zuerst auf den Index zu und sucht damit die Daten in der Tabelle.

Im Folgenden finden Sie einige SQL-Anweisungen, mit denen Sie die Leistung verbessern können GenreAndPriceIndex. Wir setzen voraus, dass die Tabelle Musik genügend Daten enthält, damit der Abfrageoptimierer diesen Index verwendet, statt die gesamte Tabelle einfach zu scannen.

```
/* All of the rock songs */

SELECT * FROM Music
WHERE Genre = 'Rock';
```

```
/* All of the cheap country songs */

SELECT Artist, SongTitle, Price FROM Music
WHERE Genre = 'Country' AND Price < 0.50;
```

Abfragen und Scannen eines Index in DynamoDB

In DynamoDB führen Sie `Query` und `Scan`-Operationen direkt für den Index aus, genauso wie für eine Tabelle. Sie können entweder die DynamoDB-API verwenden oder PartiQL, eine SQL-kompatiblen Abfragesprache, um den Index abzufragen oder zu scannen. Sie müssen sowohl `TableName` als auch `IndexName` angeben.

Im Folgenden finden Sie einige Abfragen GenreAndPriceIndexin DynamoDB. (Das Schlüsselschema für diesen Index besteht aus Genre und Preis.)

DynamoDB API

```
// All of the rock songs

{
    TableName: "Music",
    IndexName: "GenreAndPriceIndex",
    KeyConditionExpression: "Genre = :genre",
    ExpressionAttributeValues: {
        ":genre": "Rock"
    },
};
```

In diesem Beispiel wird ein `ProjectionExpression` verwendet, um anzugeben, dass nicht alle, sondern nur einige Attribute in den Ergebnissen enthalten sein sollen.

```
// All of the cheap country songs

{
    TableName: "Music",
    IndexName: "GenreAndPriceIndex",
    KeyConditionExpression: "Genre = :genre and Price < :price",
    ExpressionAttributeValues: {
        ":genre": "Country",
        ":price": 0.50
    },
    ProjectionExpression: "Artist, SongTitle, Price"
};
```

Im Folgenden ist ein Scan aktiviert. GenreAndPriceIndex

```
// Return all of the data in the index

{
    TableName:  "Music",
    IndexName: "GenreAndPriceIndex"
}
```

PartiQL for DynamoDB

Mit PartiQL verwenden Sie die PartiQL `Select`-Anweisung zur Durchführung von Abfragen und Scans für den Index.

```
// All of the rock songs

SELECT *
FROM Music.GenreAndPriceIndex
WHERE Genre = 'Rock'
```

```
// All of the cheap country songs

SELECT *
FROM Music.GenreAndPriceIndex
WHERE Genre = 'Rock' AND Price < 0.50
```

Das Folgende ist ein Scan aktiviert GenreAndPriceIndex.

```
// Return all of the data in the index

SELECT *
FROM Music.GenreAndPriceIndex
```

> **Note**
>
> Codebeispiele, die `Select` verwenden, finden Sie unter PartiQL-Select-Anweisungen für DynamoDB.

Unterschiede zwischen einer relationalen (SQL) Datenbank und DynamoDB beim Ändern von Daten in einer Tabelle

Die SQL-Sprache bietet die `UPDATE`-Anweisung zum Ändern von Daten. Amazon DynamoDB verwendet die `UpdateItem`-Operation für ähnliche Aufgaben.

Themen

- Ändern von Daten in einer Tabelle mit SQL
- Ändern von Daten in einer Tabelle in DynamoDB

Ändern von Daten in einer Tabelle mit SQL

In SQL verwenden Sie die `UPDATE`-Anweisung, um eine oder mehrere Zeilen zu ändern. Die `SET`-Klausel gibt neue Werte für eine oder mehrere Spalten an und mit der `WHERE`-Klausel wird bestimmt, welche Zeilen geändert werden. Im Folgenden wird ein -Beispiel gezeigt.

```
UPDATE Music
SET RecordLabel = 'Global Records'
WHERE Artist = 'No One You Know' AND SongTitle = 'Call Me Today';
```

Wenn keine Zeilen mit der `WHERE`-Klausel übereinstimmen, ist die `UPDATE`-Anweisung wirkungslos.

Ändern von Daten in einer Tabelle in DynamoDB

In DynamoDB können Sie entweder die DynamoDB-API oder PartiQL, eine SQL-kompatible Abfragesprache, verwenden, um ein einzelnes Element zu ändern. Wenn Sie mehrere Elemente ändern möchten, müssen Sie mehrere Operationen verwenden.

DynamoDB API

In der DynamoDB-API verwenden Sie die `UpdateItem`-Aktion, um ein einzelnes Element zu ändern.

```
{
    TableName: "Music",
    Key: {
        "Artist":"No One You Know",
        "SongTitle":"Call Me Today"
    },
    UpdateExpression: "SET RecordLabel = :label",
    ExpressionAttributeValues: {
        ":label": "Global Records"
    }
}
```

Sie müssen die `Key`-Attribute des zu ändernden Elements angeben sowie einen `UpdateExpression` zur Angabe der Attributwerte. `UpdateItem` verhält sich wie eine „upsert“-Operation. Das Element wird aktualisiert, wenn es in der Tabelle vorhanden ist. Andernfalls wird ein neues Element hinzugefügt (eingefügt).

`UpdateItem` unterstützt bedingte Schreibvorgänge, in denen die Operation nur erfolgreich abgeschlossen wird, wenn ein bestimmter `ConditionExpression` mit TRUE ausgewertet wird. Die folgende `UpdateItem`-Aktion führt die Aktualisierung nur durch, wenn der Preis des Songs größer oder gleich 2,00 ist.

```
{
    TableName: "Music",
    Key: {
        "Artist":"No One You Know",
        "SongTitle":"Call Me Today"
    },
    UpdateExpression: "SET RecordLabel = :label",
    ConditionExpression: "Price >= :p",
```

```
    ExpressionAttributeValues: {
        ":label": "Global Records",
        ":p": 2.00
    }
}
```

`UpdateItem` unterstützt außerdem unteilbare Zähler, dies sind Attribute vom Typ `Number`, die schrittweise erhöht oder verringert werden können. Unteilbare Zähler ähneln in vielerlei Hinsicht Sequenzgeneratoren, Identitätsspalten oder Feldern für die automatische Inkrementierung in SQL-Datenbanken.

Das folgende Beispiel ist eine `UpdateItem`-Operation zur Initialisierung eines neuen Attributs (Plays), um zu verfolgen, wie oft der Song abgespielt wurde.

```
{
    TableName: "Music",
    Key: {
        "Artist":"No One You Know",
        "SongTitle":"Call Me Today"
    },
    UpdateExpression: "SET Plays = :val",
    ExpressionAttributeValues: {
        ":val": 0
    },
    ReturnValues: "UPDATED_NEW"
}
```

Der Parameter `ReturnValues` wird auf `UPDATED_NEW` festgelegt. Hiermit werden die neuen Werte aktualisierter Attribute zurückgegeben. In diesem Fall wird 0 (Null) zurückgegeben.

Sobald dieser Song abgespielt wird, können wir die folgende `UpdateItem`-Operation verwenden, um das Attribut Plays um eins zu erhöhen.

```
{
    TableName: "Music",
    Key: {
        "Artist":"No One You Know",
        "SongTitle":"Call Me Today"
    },
    UpdateExpression: "SET Plays = Plays + :incr",
    ExpressionAttributeValues: {
        ":incr": 1
```

```
    },
    ReturnValues: "UPDATED_NEW"
}
```

PartiQL for DynamoDB

Mit PartiQL verwenden Sie die `ExecuteStatement`-Operation zum Ändern eines Elements in einer Tabelle mit PartiQL `Update`-Anweisung.

Der Primärschlüssel für diese Tabelle besteht aus Artist und. SongTitle Sie müssen Werte für diese Attribute angeben.

```
UPDATE Music
SET RecordLabel ='Global Records'
WHERE Artist='No One You Know' AND SongTitle='Call Me Today'
```

Sie können wie im folgenden Beispiel auch mehrere Felder gleichzeitig ändern.

```
UPDATE Music
SET RecordLabel = 'Global Records'
SET AwardsWon = 10
WHERE Artist ='No One You Know' AND SongTitle='Call Me Today'
```

`Update` unterstützt außerdem unteilbare Zähler, dies sind Attribute vom Typ `Number`, die schrittweise erhöht oder verringert werden können. Unteilbare Zähler ähneln in vielerlei Hinsicht Sequenzgeneratoren, Identitätsspalten oder Feldern für die automatische Inkrementierung in SQL-Datenbanken.

Das folgende Beispiel ist eine `Update`-Anweisung zur Initialisierung eines neuen Attributs (Plays), um zu verfolgen, wie oft der Song abgespielt wurde.

```
UPDATE Music
SET Plays = 0
WHERE Artist='No One You Know' AND SongTitle='Call Me Today'
```

Sobald dieser Song abgespielt wird, können wir die folgende `Update`-Anweisung verwenden, um das Attribut Spielt um eins zu erhöhen.

```
UPDATE Music
```

```
SET Plays = Plays + 1
WHERE Artist='No One You Know' AND SongTitle='Call Me Today'
```

> **Note**
>
> Codebeispiele, die `Update` und `ExecuteStatement` verwenden, finden Sie unter Aktualisierungen für PartiQL-Anweisungen für DynamoDB.

Unterschiede zwischen einer relationalen (SQL) Datenbank und DynamoDB beim Löschen von Daten aus einer Tabelle

In SQL wird mit der `DELETE`-Anweisung eine oder mehrere Zeilen aus einer Tabelle entfernt. Amazon DynamoDB verwendet die `DeleteItem`-Operation, um jeweils ein Element zu löschen.

Themen

- Löschen von Daten aus einer Tabelle mit SQL
- Löschen von Daten aus einer Tabelle in DynamoDB

Löschen von Daten aus einer Tabelle mit SQL

In SQL verwenden Sie die `DELETE`-Anweisung, um eine oder mehrere Zeilen zu löschen. Die `WHERE`-Klausel bestimmt, welche Zeilen geändert werden sollen. Im Folgenden wird ein -Beispiel gezeigt.

```
DELETE FROM Music
WHERE Artist = 'The Acme Band' AND SongTitle = 'Look Out, World';
```

Sie können die `WHERE`-Klausel ändern, um mehrere Zeilen zu löschen. Sie können beispielsweise alle Songs eines bestimmten Interpreten löschen, wie in dem folgenden Beispiel gezeigt wird.

```
DELETE FROM Music WHERE Artist = 'The Acme Band'
```

Löschen von Daten aus einer Tabelle in DynamoDB

In DynamoDB können Sie entweder die DynamoDB-API oder PartiQL, eine SQL-kompatible Abfragesprache, verwenden, um ein einzelnes Element zu löschen. Wenn Sie mehrere Elemente ändern möchten, müssen Sie mehrere Operationen verwenden.

DynamoDB API

In der DynamoDB-API verwenden Sie die Aktion `DeleteItem`, um Daten elementweise aus einer Tabelle zu löschen. Sie müssen die Primärschlüsselwerte des Elements angeben.

```
{
    TableName: "Music",
    Key: {
        Artist: "The Acme Band",
        SongTitle: "Look Out, World"
    }
}
```

> **Note**
>
> Neben `DeleteItem` unterstützt Amazon DynamoDB eine `BatchWriteItem`-Aktion zum gleichzeitigen Löschen mehrerer Elemente.

`DeleteItem` unterstützt bedingte Schreibvorgänge, in denen die Operation nur erfolgreich abgeschlossen wird, wenn ein bestimmter `ConditionExpression` mit TRUE ausgewertet wird. Beispielsweise löscht der folgende `DeleteItem` Vorgang das Element nur, wenn es über ein Attribut verfügt. RecordLabel

```
{
    TableName: "Music",
    Key: {
        Artist: "The Acme Band",
        SongTitle: "Look Out, World"
    },
   ConditionExpression: "attribute_exists(RecordLabel)"
}
```

PartiQL for DynamoDB

Mit PartiQL verwenden Sie die `Delete`-Anweisung durch die`ExecuteStatement`-Operation zum Löschen von Daten aus einer Tabelle, jeweils ein Element. Sie müssen die Primärschlüsselwerte des Elements angeben.

Der Primärschlüssel für diese Tabelle besteht aus Artist und SongTitle. Sie müssen Werte für diese Attribute angeben.

```
DELETE FROM Music
WHERE Artist = 'Acme Band' AND SongTitle = 'PartiQL Rocks'
```

Sie können auch zusätzliche Bedingungen für die Operation angeben. Die folgende DELETE-Operation löscht das Element nur, wenn es mehr als 11 Auszeichnungen hat.

```
DELETE FROM Music
WHERE Artist = 'Acme Band' AND SongTitle = 'PartiQL Rocks' AND Awards > 11
```

> **Note**
>
> Codebeispiele, die DELETE und ExecuteStatement verwenden, finden Sie unter PartiQL-Delete-Anweisungen für DynamoDB.

Unterschiede zwischen einer relationalen (SQL) Datenbank und DynamoDB beim Entfernen einer Tabelle

In SQL verwenden Sie die DROP TABLE-Anweisung zum Entfernen einer Tabelle. In Amazon DynamoDB verwenden Sie die DeleteTable-Operation.

Themen

- Entfernen einer Tabelle mit SQL
- Entfernen einer Tabelle in DynamoDB

Entfernen einer Tabelle mit SQL

Wenn Sie eine Tabelle nicht mehr benötigen und sie dauerhaft löschen möchten, verwenden Sie die DROP TABLE-Anweisung in SQL.

```
DROP TABLE Music;
```

Nachdem eine Tabelle entfernt wurde, kann sie nicht wiederhergestellt werden. (Bei einigen relationalen Datenbanken ist es möglich, eine DROP TABLE-Operation rückgängig zu machen. Hierbei handelt es sich jedoch um eine anbieterspezifische Funktionalität, die nicht gängig ist.)

Entfernen einer Tabelle in DynamoDB

In DynamoDB ist die `DeleteTable`-Aktion ähnlich. Im folgenden Beispiel wird die Tabelle dauerhaft gelöscht.

```
{
    TableName: "Music"
}
```

Lernressourcen und Tools für Amazon DynamoDB

Sie können die folgenden weiteren Ressourcen nutzen, um DynamoDB zu verstehen und damit zu arbeiten.

Themen

- Tools für die Codierung und Visualisierung
- Artikel über Prescriptive Guidance
- Wissenscenter-Artikel
- Blogbeiträge, Repositorys und Leitfäden
- Datenmodellierung und Designmusterpräsentationen
- Schulungskurse

Tools für die Codierung und Visualisierung

Sie können die folgenden Codierungs- und Visualisierungstools für die Arbeit mit DynamoDB verwenden:

- NoSQL Workbench für Amazon DynamoDB – ein einheitliches, visuelles Tool, mit dem Sie DynamoDB-Tabellen entwerfen, erstellen, abfragen und verwalten können. Es bietet Funktionen zur Datenmodellierung, Datenvisualisierung und Abfrageentwicklung.
- Dynobase – ein Desktop-Tool, das das Anzeigen Ihrer DynamoDB-Tabellen und die Arbeit damit sowie das Erstellen von Anwendungscode und das Bearbeiten von Datensätzen mit Echtzeit-Validierung erleichtert.
- DynamoDB Toolbox — Ein Projekt von Jeremy Daly, das hilfreiche Hilfsprogramme für die Arbeit mit Datenmodellierung und JavaScript Node.js bietet.

- DynamoDB-Streams-Prozessor – ein einfaches Tool für die Arbeit mit DynamoDB Streams.

Artikel über Prescriptive Guidance

AWS Prescriptive Guidance bietet bewährte Strategien, Leitfäden und Muster, mit denen Sie Ihre Projekte beschleunigen können. Diese Ressourcen wurden von AWS Technologieexperten und der weltweiten AWS Partnergemeinschaft auf der Grundlage ihrer jahrelangen Erfahrung bei der Unterstützung von Kunden beim Erreichen ihrer Geschäftsziele entwickelt.

Datenmodellierung und -migration

- Ein hierarchisches Datenmodell in DynamoDB
- Modellieren von Daten mit DynamoDB
- Migrieren Sie eine Oracle-Datenbank zu DynamoDB mit AWS DMS

Globale Tabellen

- Verwenden der globalen Amazon-DynamoDB-Tabellen

Serverless

- Implementieren Sie das serverlose Saga-Muster mit AWS Step Functions

SaaS-Architektur

- Mandanten für mehrere SaaS-Produkte auf einer einzigen Steuerebene verwalten
- Mandanten-Onboarding in SaaS-Architektur für das Silomodell mit C# und AWS -CDK

Datenschutz und Datenverschiebung

- Kontenübergreifenden Zugriff auf Amazon DynamoDB konfigurieren
- Optionen zum Kopieren ganzer Tabellen für DynamoDB
- Strategie zur Notfallwiederherstellung für Datenbanken in AWS

Sonstiges

- Bei der Durchsetzung von Tagging in DynamoDB helfen

Beispielhafte Videoanleitungen mit präskriptiven Anleitungen

- Verwenden einer Serverless-Architektur zur Erstellung von Daten-Pipelines
- Novartis – Einkaufs-Engine: KI-gestütztes Einkaufsportal
- Veritiv: Ermöglichen Sie Einblicke, um die Umsatznachfrage auf AWS Data Lakes zu Forecast
- mimik: Hybrid-Edge-Cloud-Nutzung AWS zur Support von Edge Microservice Mesh
- Ändern der Datenerfassung mit Amazon DynamoDB

Weitere Artikel und Videos zu Prescriptive Guidance für DynamoDB finden Sie unter Prescriptive Guidance.

Wissenscenter-Artikel

Die Artikel und Videos im AWS Knowledge Center behandeln die häufigsten Fragen und Anfragen, die wir von AWS Kunden erhalten. Im Folgenden finden Sie einige aktuelle Wissenscenter-Artikel zu bestimmten Aufgaben, die sich auf DynamoDB beziehen:

Kostenoptimierung

- Wie optimiere ich die Kosten mit Amazon DynamoDB?

Drosselung und Latenz

- Wie kann ich Probleme mit hoher Latenz in einer Amazon DynamoDB-Tabelle beheben?
- Warum wird meine DynamoDB-Tabelle gedrosselt?
- Warum wird meine On-Demand-DynamoDB-Tabelle gedrosselt?

Paginierung

- Wie implementiere ich die Paginierung in DynamoDB

Transaktionen

- Warum schlägt mein `TransactWriteItems`-API-Aufruf in DynamoDB fehl?

Fehlersuche

- Wie löse ich Probleme mit DynamoDB Auto Scaling?
- Wie behebe ich HTTP-4XX-Fehler in DynamoDB

Weitere Artikel und Videos zu DynamoDB finden Sie in den Wissenscenter-Artikeln.

Blogbeiträge, Repositorys und Leitfäden

Zusätzlich zum DynamoDB-Entwicklerhandbuch gibt es viele nützliche Ressourcen für die Arbeit mit DynamoDB. Hier finden Sie einige ausgewählte Blogbeiträge, Repositorys und Anleitungen für die Arbeit mit DynamoDB:

- AWS Repository mit DynamoDB-Codebeispielen in verschiedenen AWS SDK-Sprachen: Node.js, Java, Python, .Net, Go und Rust.
- Das DynamoDB-Buch — Ein umfassender Leitfaden von Alex DeBrie, der einen strategieorientierten Ansatz für die Datenmodellierung mit DynamoDB vermittelt.
- DynamoDB-Leitfaden — Ein offener Leitfaden von Alex DeBrie, der die grundlegenden Konzepte und erweiterten Funktionen der DynamoDB-NoSQL-Datenbank erläutert.
- So wechseln Sie in nur 20 einfachen Schritten von RDBMS zu DynamoDB – eine Liste nützlicher Schritte zum Erlernen der Datenmodellierung von Jeremy Daly.
- JavaScript DocumentClient DynamoDB-Spickzettel — Ein Spickzettel, der Ihnen bei den ersten Schritten beim Erstellen von Anwendungen mit DynamoDB in einer Node.js oder Umgebung hilft. JavaScript
- DynamoDB-Core-Konzeptvideos – diese Wiedergabeliste deckt viele der Kernkonzepte von DynamoDB ab.

Datenmodellierung und Designmusterpräsentationen

Mit den folgenden Ressourcen zu Datenmodellierung und Entwurfsmustern können Sie DynamoDB optimal nutzen:

- AWS re:Invent 2019: Datenmodellierung mit DynamoDB
  - Ein Vortrag von Alex DeBrie, der Ihnen hilft, mit den Prinzipien der DynamoDB-Datenmodellierung zu beginnen.

- AWS re:Invent 2020: Datenmodellierung mit DynamoDB — Teil 1
- AWS re:Invent 2020: Datenmodellierung mit DynamoDB — Teil 2
- AWS re:Invent 2017: Fortgeschrittene Entwurfsmuster
- AWS re:Invent 2018: Fortschrittliche Entwurfsmuster
- AWS re:Invent 2019: Fortschrittliche Entwurfsmuster
  - Jeremy Daly berichtet über seine 12 wichtigsten Erkenntnisse aus diesem Vortrag.
- AWS re:Invent 2020: DynamoDB Fortgeschrittene Designmuster – Teil 1
- AWS re:Invent 2020: DynamoDB-Entwurfsmuster für Fortgeschrittene — Teil 2
- DynamoDB Sprechstunden auf Twitch

> **Note**
>
> In jedem Vortrag werden verschiedene Anwendungsfälle und Beispiele behandelt.

Schulungskurse

Es gibt viele verschiedene Trainings und Bildungsmöglichkeiten, um mehr über DynamoDB zu erfahren. Hier sind einige aktuelle Beispiele:

- Entwickeln mit Amazon DynamoDB — Entwickelt von AWS , um Sie vom Anfänger zum Experten für die Entwicklung realer Anwendungen mit Datenmodellierung für Amazon DynamoDB zu machen.
- DynamoDB-Vertiefungskurs – ein Kurs von Pluralsight.
- Amazon DynamoDB: Entwicklung datenbankgestützter NoSQL-Anwendungen — Ein Kurs des AWS Schulungs- und Zertifizierungsteams, der auf edX gehostet wird.

Lese- und Schreibvorgänge in DynamoDB

DynamoDB-Lese- und Schreibvorgänge beziehen sich auf Operationen, die Daten aus einer Tabelle abrufen (Lesevorgänge) und Daten in einer Tabelle einfügen, aktualisieren oder löschen (Schreibvorgänge). Wenn Sie mit DynamoDB arbeiten, ist es wichtig, die Konzepte von Lese- und Schreibvorgängen zu verstehen, da sie sich direkt auf die Leistung und die Kosten Ihrer Anwendung auswirken.

In diesem Thema werden Einzelheiten zu den verschiedenen Arten der Lesekonsistenz vorgestellt, die für DynamoDB gelten. Weiterhin wird in diesem Thema der Einheitenverbrauch für verschiedene Lese- und Schreibvorgänge beschrieben, die Sie möglicherweise ausführen werden.

Themen

- DynamoDB-Lesekonsistenz
- Lese- und Schreibvorgänge in DynamoDB

DynamoDB-Lesekonsistenz

Amazon DynamoDB liest Daten aus Tabellen, lokalen Sekundärindizes (LSIs), globalen Sekundärindizes (GSIs) und Streams. Weitere Informationen finden Sie unter Kernkomponenten von Amazon DynamoDB. Beide Tabellen LSIs bieten zwei Optionen für die Lesekonsistenz: eventuell konsistente (Standard) und stark konsistente Lesevorgänge. Alle Lesevorgänge GSIs und Streams sind letztlich konsistent.

Wenn Ihre Anwendung Daten in eine DynamoDB-Tabelle schreibt und eine HTTP-200-Antwort (OK) empfängt, bedeutet dies, dass der Schreibvorgang erfolgreich abgeschlossen wurde und dauerhaft persistent ist. DynamoDB stellt die Isolationsstufe read-committed bereit und stellt sicher, dass Lesevorgänge immer mit Commit gespeicherte Werte für ein Element zurückgeben. Beim Lesevorgang wird niemals eine Ansicht des Elements aus einem Schreibvorgang angezeigt, der nicht letztendlich erfolgreich war. Die Isolation "Read-committed" verhindert keine Änderungen am Element direkt nach dem Lesevorgang.

Letzendlich konsistente Lesevorgänge

Letztendlich konsistent ist das standardmäßige Konsistenzmodell für alle Lesevorgänge. Wenn letztendlich konsistente Lesevorgänge an eine DynamoDB-Tabelle oder einen Index ausgegeben

werden, geben die Antworten möglicherweise nicht die Ergebnisse eines kürzlich abgeschlossenen Schreibvorgangs wider. Wenn Sie die Leseanforderung nach kurzer Zeit wiederholen, sollte in der Antwort letztendlich das neueste Element enthalten sein. Letztendlich konsistente Lesevorgänge werden für Tabellen, lokale sekundäre Indizes und globale sekundäre Indizes unterstützt. Beachten Sie außerdem, dass alle Lesevorgänge aus einem DynamoDB-Stream ebenfalls letztendlich konsistent sind.

Letztendlich konsistente Lesevorgänge kosten nur halb so viel wie strikt konsistente Lesevorgänge. Weitere Informationen finden Sie unter Amazon DynamoDB – Preise.

Strikt konsistente Lesevorgänge

Lesevorgänge wie beispielsweise `GetItem`, `Query` und `Scan` stellen einen `ConsistentRead`-Parameter bereit. Wenn Sie `ConsistentRead` auf true setzen, gibt DynamoDB eine Antwort mit den meisten up-to-date Daten zurück, die die Aktualisierungen aller vorherigen Schreibvorgänge widerspiegelt, die erfolgreich waren. Strikt konsistente Lesevorgänge werden nur für Tabellen und lokale sekundäre Indizes unterstützt. Strikt konsistente Lesevorgänge aus einem globalen sekundären Index oder einem DynamoDB-Stream werden nicht unterstützt.

Lesekonsistenz in globalen Tabellen

DynamoDB unterstützt auch globale Tabellen für die multiaktive und regionsübergreifende Replikation. Eine globale Tabelle besteht aus mehreren Replikattabellen in verschiedenen Regionen. AWS Alle Änderungen an einem Element einer Replikattabelle werden, in der Regel innerhalb einer Sekunde, auf alle anderen Replikate innerhalb derselben globalen Tabelle repliziert und sind letztendlich konsistent. Weitere Informationen finden Sie unter Konsistenzmodi.

Lese- und Schreibvorgänge in DynamoDB

DynamoDB-Leseoperationen ermöglichen es Ihnen, ein oder mehrere Elemente aus einer Tabelle abzurufen, indem Sie den Partitionsschlüsselwert und optional den Sortierschlüsselwert angeben. Mithilfe von DynamoDB-Schreibvorgänge können Sie Elemente in eine Tabelle einfügen, aktualisieren oder löschen. In diesem Thema wird der Verbrauch von Kapazitätseinheiten für diese beiden Operationen erklärt.

Themen

- Verbrauch der Kapazitätseinheit für Leseoperationen
- Verbrauch von Kapazitätseinheiten für Schreibvorgänge

Verbrauch der Kapazitätseinheit für Leseoperationen

DynamoDB-Leseanforderungen können entweder stark konsistent, eventuell konsistent oder transaktional sein.

- Die Anforderung eines strikt konsistenten Lesevorgangs für ein Element bis zu 4 KB verbraucht eine Leseeinheit.
- Die Anforderung eines letztendlich konsistenten Lesevorgangs für ein Element bis 4 KB verbraucht eine halbe Leseeinheit.
- Die Anforderung eines transaktionalen Lesevorgang für ein Element bis 4 KB verbraucht zwei Leseeinheiten.

Weitere Informationen über die DynamoDB-Lesekonsistenzmodelle finden Sie unter DynamoDB-Lesekonsistenz.

Elementgrößen für Lesevorgänge werden auf das nächste Vielfache von 4 KB aufgerundet. Beispiel: Das Lesen eines 3 500-Byte-Elements verbraucht den gleichen Durchsatz wie das Lesen eines 4 KB-Elements.

Wenn ein Element gelesen werden soll, das größer als 4 KB ist, verbraucht DynamoDB zusätzliche Leseeinheiten. Die Gesamtanzahl der erforderlichen Leseeinheiten ist von der Elementgröße und davon abhängig, ob Sie einen letztendlich konsistenten oder strikt konsistenten Lesevorgang benötigen. Wenn Ihr Element beispielsweise 8 KB groß ist, benötigen Sie 2 Leseeinheiten, um einen strikt konsistenten Lesevorgang zu gewährleisten. Sie benötigen nur 1 Leseeinheit, wenn Sie letzendlich konsistente Lesevorgänge wählen, oder 4 Leseeinheiten für eine transaktionale Leseanforderung.

In der folgenden Liste wird beschrieben, wie DynamoDB-Lesevorgänge Leseeinheiten verbrauchen:

- GetItem: Liest ein einzelnes Element aus einer Tabelle. Um die Anzahl der Kapazitätseinheiten zu bestimmen, die `GetItem` verbrauchen wird, nehmen Sie die Elementgröße und runden sie auf den nächsten 4 KB-Grenzwert auf. Wenn Sie einen strikt konsistenter Lesevorgang angegeben haben, ist dies die Anzahl der erforderlichen Leseeinheiten. Teilen Sie diese Zahl für einen letzendlich konsistenten Lesevorgang (Standard) durch zwei.

  Wenn Sie beispielsweise ein Element lesen, das 3,5 KB groß ist, rundet DynamoDB die Elementgröße auf 4 KB. Wenn Sie ein Element von 10 KB lesen, rundet DynamoDB die Elementgröße auf 12 KB.

- BatchGetItem: Liest bis zu 100 Elemente aus einer oder mehreren Tabellen. DynamoDB verarbeitet jedes Element im Stapel als eine individuelle `GetItem`-Anforderung. DynamoDB rundet zunächst die Größe jedes Elements auf den nächsten 4 KB-Grenzwert auf und berechnet dann die Gesamtgröße. Das Ergebnis ist nicht unbedingt dasselbe wie die Gesamtgröße aller Elemente. Wenn beispielsweise zwei Elemente mit den Größen 1,5 KB und 6,5 KB von `BatchGetItem` gelesen werden, berechnet DynamoDB die Größe als 12 KB (4 KB + 8 KB). DynamoDB berechnet die Größe also nicht als 8 KB (1,5 KB + 6,5 KB).
- Query: Liest mehrere Elemente, die denselben Partitionsschlüsselwert haben. Alle zurückgegebenen Elemente werden als ein einzelner Lesevorgang behandelt, wobei DynamoDB die Gesamtgröße aller Elemente berechnet. Anschließend rundet DynamoDB die Größe auf den nächsten 4 KB-Grenzwert auf. Beispiel: Angenommen, Ihre Abfrage gibt 10 Elemente zurück, deren Gesamtgröße 40,8 KB ist. DynamoDB rundet die Elementgröße für den Vorgang auf 44 KB. Wenn eine Abfrage 1 500 Elemente mit jeweils 64 Bytes zurückgibt, ist die kumulative Größe 96 KB.
- Scan: Liest alle Elemente in einer Tabelle. DynamoDB betrachtet die Größe der Elemente, die geprüft werden und nicht die Größe der Elemente, die von dem Scan zurückgegeben werden. Weitere Informationen zu Scan-Operationen finden Sie unter Scannen von Tabellen in DynamoDB.

> **Important**
>
> Wenn Sie einen Lesevorgang für ein Element ausführen, das nicht vorhanden ist, verbraucht DynamoDB trotzdem den Lesedurchsatz wie oben beschrieben. Bei `Query`/`Scan`-Operationen wird Ihnen trotzdem zusätzlicher Lesedurchsatz auf Basis der Lesekonsistenz und Anzahl der Partitionen, die zur Bearbeitung der Anfrage durchsucht werden in Rechnung gestellt, selbst wenn keine Daten vorhanden sind.

Für jede Operation, die Elemente zurückgibt, können Sie eine Teilmenge der abzurufenden Attribute anfordern. Allerdings wirkt sich dies nicht auf die Berechnungen der Elementgrößen aus. Darüber hinaus können `Query` und `Scan` die Elementanzahl anstatt der Attributwerte zurückgeben. Das Abrufen der Elementanzahl verwendet dieselbe Menge an Leseeinheiten und unterliegt den gleichen Berechnungen der Elementgröße. Grund hierfür ist, dass DynamoDB jedes Element lesen muss, um die Anzahl erhöhen zu können.

Verbrauch von Kapazitätseinheiten für Schreibvorgänge

Eine Schreibeinheit entspricht einem Schreibvorgang für ein Element mit einer Größe bis 1 KB. Wenn Sie ein Element schreiben möchten, das größer als 1 KB ist, muss DynamoDB zusätzliche Schreibeinheiten verbrauchen. Transaktionale Schreibforderungen erfordern 2 Schreibeinheiten, um einen Schreibvorgang für Elemente mit einer Größe bis 1 KB durchzuführen. Die Gesamtanzahl der Schreibanforderungseinheiten ist abhängig von der Elementgröße. Beispiel: Bei einer Elementgröße von 2 KB benötigen Sie 2 Schreibeinheiten für eine Schreibanforderung oder 4 Schreibeinheiten für eine transaktionale Schreibanforderung.

Elementgrößen für Schreibvorgänge werden auf die nächsten mehreren 1 KB aufgerundet. Beim Schreiben eines 500-Byte-Elements wird derselbe Durchsatz verbraucht wie beim Schreiben eines 1 KB-Elements.

In der folgenden Liste wird beschrieben, wie DynamoDB-Schreibvorgänge Schreibeinheiten verbrauchen:

- PutItem: Schreibt ein einzelnes Element in eine Tabelle. Wenn ein Element mit demselben Primärschlüssel in der Tabelle vorhanden ist, ersetzt die Operation das Element. Bei der Berechnung des bereitgestellten Durchsatzes ist die Elementgröße, auf die es ankommt, die größere von beiden.
- UpdateItem: Ändert ein einzelnes Element in der Tabelle. DynamoDB betrachtet die Größe des Elements vor und nach der Aktualisierung. Der bereitgestellte Durchsatz, der verbraucht wurde, spiegelt die größere dieser Elementgrößen wider. Auch wenn Sie nur eine Teilmenge der Attribute des Elements aktualisieren, verbraucht `UpdateItem` weiterhin den gesamten bereitgestellten Durchsatz (die größere der "vorher"- und "nachher"-Elementgrößen).
- DeleteItem: Entfernt ein einzelnes Element aus einer Tabelle. Der Verbrauch des bereitgestellten Durchsatzes basiert auf der Größe des gelöschten Elements.
- BatchWriteItem: Schreibt bis zu 25 Elemente in eine oder mehrere Tabellen. DynamoDB verarbeitet jedes Element im Batch als einzelne `PutItem` oder `DeleteItem`-Anforderung (Aktualisierungen werden nicht unterstützt). DynamoDB rundet zunächst die Größe jedes Elements auf den nächsten 1 KB-Grenzwert auf und berechnet dann die Gesamtgröße. Das Ergebnis ist nicht unbedingt dasselbe wie die Gesamtgröße aller Elemente. Wenn beispielsweise `BatchWriteItem` zwei Elemente mit den Größen 500 Byte und 3,5 KB schreibt, berechnet DynamoDB die Größe als 5 KB (1 KB + 4 KB). DynamoDB berechnet die Größe also nicht als 4 KB (500 Byte + 3,5 KB).

Für die Operationen `PutItem`, `UpdateItem` und `DeleteItem` rundet DynamoDB die Elementgröße auf das nächste 1 KB. Wenn Sie z. B. ein 1,6 KB-Element setzen oder löschen, rundet DynamoDB die Elementgröße auf 2 KB auf.

`PutItem`-, `UpdateItem`- und `DeleteItem`-Operationen erlauben bedingte Schreibvorgänge, bei denen Sie einen Ausdruck angeben, der als True ausgewertet werden muss, damit die Operation erfolgreich ausgeführt werden kann. Wenn der Ausdruck als falsch ausgewertet wird, verbraucht DynamoDB dennoch Schreibkapazitätseinheiten aus der Tabelle. Die Anzahl der verbrauchten Schreibkapazitätseinheiten hängt von der Größe des Elements ab. Dieses Element kann entweder ein bereits vorhandenes Element in der Tabelle sein oder ein neues, das Sie zu erstellen oder zu aktualisieren versuchen. Nehmen wir beispielsweise an, dass ein vorhandenes Element 300 KB groß ist. Das neue Element, das Sie zu erstellen oder aktualisieren versuchen, hat eine Größe von 310 KB. Die verbrauchten Schreibkapazitätseinheiten für das neue Element betragen 310 KB.

DynamoDB-Durchsatzkapazität

Dieser Abschnitt enthält einen Überblick über die beiden Durchsatzmodi, die für Ihre DynamoDB-Tabelle verfügbar sind, sowie Überlegungen zur Auswahl des geeigneten Kapazitätsmodus für Ihre Anwendung. Der Durchsatzmodus einer Tabelle bestimmt, wie die Kapazität einer Tabelle verwaltet wird. Der Durchsatzmodus bestimmt auch, wie Ihnen die Lese- und Schreibvorgänge in Ihren Tabellen in Rechnung gestellt werden. In Amazon DynamoDB können Sie zwischen dem On-Demand-Modus und dem Modus bereitgestellter Kapazität für Ihre Tabellen wählen, um unterschiedlichen Workload-Anforderungen gerecht zu werden.

Themen

- On-Demand-Modus
- Modus bereitgestellter Kapazität
- On-Demand-Kapazitätsmodus von DynamoDB
- DynamoDB – Modus mit bereitgestellter Kapazität
- Grundlegendes zum Warmdurchsatz von DynamoDB
- DynamoDB – Burst- und adaptive Kapazität
- Überlegungen beim Umstellen der Kapazitätsmodi in DynamoDB

On-Demand-Modus

Der On-Demand-Modus von Amazon DynamoDB ist eine Serverless-Durchsatzoption, die das Datenbankmanagement vereinfacht und automatisch skaliert, um die anspruchsvollsten Anwendungen der Kunden zu unterstützen. Mit DynamoDB On-Demand können Sie eine Tabelle erstellen, ohne sich Gedanken über Kapazitätsplanung, Nutzungsüberwachung und die Konfiguration von Skalierungsrichtlinien machen zu müssen. DynamoDB On-Demand bietet pay-per-request Preise für Lese- und Schreibanforderungen, sodass Sie nur für das bezahlen, was Sie tatsächlich nutzen. Für On-Demand-Modustabellen müssen Sie nicht angeben, wie viel Lese- und Schreibdurchsatz Sie von Ihrer Anwendung erwarten.

Der On-Demand-Modus ist die standardmäßige und empfohlene Durchsatzoption für die meisten DynamoDB-Workloads. DynamoDB kümmert sich um alle Aspekte der Durchsatzverwaltung und -skalierung, um Workloads zu unterstützen, die klein beginnen und auf Millionen von Anforderungen pro Sekunde skaliert werden. Sie können nach Bedarf Lese- und Schreibvorgänge in Ihrer

DynamoDB-Tabellen ausführen, ohne die Durchsatzkapazität der Tabelle verwalten zu müssen. Weitere Informationen finden Sie unter On-Demand-Kapazitätsmodus von DynamoDB.

Modus bereitgestellter Kapazität

Im Modus bereitgestellter Kapazität müssen Sie die Anzahl der Lese- und Schreibvorgänge pro Sekunde festlegen, die Sie für Ihre Anwendung benötigen. Die Abrechnung erfolgt auf der Grundlage der von Ihnen bereitgestellten stündlichen Lese- und Schreibkapazität. Sie basiert nicht darauf, wie viel von der bereitgestellten Kapazität Sie tatsächlich verbraucht haben. Auf diese Weise können Sie leicht regeln, dass Ihre DynamoDB-Nutzung eine definierte Anforderungsrate nicht überschreitet, um planbare Kosten zu erzielen.

Sie können die bereitgestellte Kapazität nutzen, wenn Sie über konstante Workloads mit vorhersehbarem Wachstum verfügen und Sie die Kapazitätsanforderungen für Ihre Anwendung zuverlässig prognostizieren können. Weitere Informationen finden Sie unter DynamoDB – Modus mit bereitgestellter Kapazität.

On-Demand-Kapazitätsmodus von DynamoDB

Amazon DynamoDB On-Demand bietet ein echtes Serverless-Datenbankerlebnis, das automatisch skaliert wird, um die anspruchsvollsten Workloads ohne Kapazitätsplanung zu bewältigen. Der On-Demand-Modus vereinfacht den Einrichtungsprozess, macht die Kapazitätsverwaltung und -überwachung überflüssig und ermöglicht eine schnelle, automatische Skalierung. Bei der pay-per-request Preisgestaltung müssen Sie sich keine Gedanken über ungenutzte Kapazitäten machen, da Sie nur für den Durchsatz zahlen, den Sie tatsächlich nutzen. Die Abrechnung erfolgt pro Lese- oder Schreibanforderung, sodass Ihre Kosten direkt Ihre tatsächliche Nutzung widerspiegeln.

Wenn Sie den On-Demand-Modus wählen, passt DynamoDB Ihre Workloads, wenn sie steigen oder sinken, sofort an einen beliebigen zuvor erreichten Stand des Datenverkehrs an. Wenn der Datenverkehr eines Workloads einen neuen Höchststand erreicht, skaliert DynamoDB automatisch, um den erhöhten Durchsatzanforderungen gerecht zu werden. Der On-Demand-Modus ist die standardmäßige und empfohlene Durchsatzoption, da er die Erstellung moderner Serverless-Anwendungen vereinfacht, die klein beginnen und auf Millionen von Anforderungen pro Sekunde skaliert werden. Sobald Ihre On-Demand-Tabelle aufskaliert ist, können Sie künftig sofort wieder denselben Durchsatz erzielen, ohne dass eine Drosselung erforderlich ist. Wenn Sie keinen Traffic zu Ihrer Tabelle lenken, wird Ihnen bei On-Demand-Modus kein Durchsatz in Rechnung gestellt. Weitere Informationen zu den Skalierungseigenschaften des On-Demand-Modus finden Sie unter Anfänglicher Durchsatz und Skalierungseigenschaften.

Tabellen, die den On-Demand-Modus nutzen, erzielen dieselbe Latenz im einstelligen Millisekundenbereich, das Service Level Agreement (SLA) und die Sicherheit, die vom DynamoDB-Modus bereitgestellter Kapazität geboten werden.

> **Note**
>
> Standardmäßig schützt DynamoDB Sie vor unbeabsichtigter, unkontrollierter Nutzung. Um die Durchsatzgrenzen von 40 000 Lese- und Schreibvorgängen auf Tabellenebene für alle Tabellen in Ihrem Konto zu überschreiten, können Sie eine Erhöhung dieses Kontingents anfordern. Durchsatzanforderungen, die das standardmäßige Durchsatzkontingent der Tabelle überschreiten, werden gedrosselt. Weitere Informationen finden Sie unter Standardkontingente für den Durchsatz.

Optional können Sie auch den maximalen Lese- oder Schreibdurchsatz (oder beides) pro Sekunde für einzelne On-Demand-Tabellen und globale sekundäre Indizes konfigurieren. Durch die Konfiguration des Durchsatzes können Sie die Nutzung und Kosten auf Tabellenebene begrenzen, sich vor einem unbeabsichtigten Anstieg verbrauchter Ressourcen schützen und eine übermäßige Nutzung verhindern, um ein vorhersehbares Kostenmanagement zu gewährleisten. Durchsatzanforderungen, die den maximalen Tabellendurchsatz überschreiten, werden gedrosselt. Sie können den tabellenspezifischen maximalen Durchsatz jederzeit an Ihre Anwendungsanforderungen anpassen. Weitere Informationen finden Sie unter Maximaler DynamoDB-Durchsatz für On-Demand-Tabellen.

Erstellen oder aktualisieren Sie zunächst eine Tabelle zur Verwendung des On-Demand-Modus. Weitere Informationen finden Sie unter Grundlegende Operationen für DynamoDB-Tabellen.

Sie können Tabellen innerhalb eines fortlaufenden 24-stündigen Zeitfensters bis zu viermal vom Modus mit bereitgestellter Kapazität auf den On-Demand-Modus umstellen. Sie können Tabellen jederzeit vom On-Demand-Modus auf den Modus mit bereitgestellter Kapazität umstellen.

Weitere Informationen zum Wechseln zwischen Lese- und Schreibkapazitätsmodus finden Sie unter Überlegungen beim Umstellen der Kapazitätsmodi in DynamoDB. Informationen zu On-Demand-Tabellenkontingenten finden Sie unter Lese- und Schreibdurchsatz.

Themen

- Leseanforderungseinheiten und Schreibanforderungseinheiten
- Anfänglicher Durchsatz und Skalierungseigenschaften

- Maximaler DynamoDB-Durchsatz für On-Demand-Tabellen

Leseanforderungseinheiten und Schreibanforderungseinheiten

DynamoDB berechnet Ihnen die Lese- und Schreibvorgänge, die Ihre Anwendung bei Ihren Tabellen durchführt, als Leseanforderungseinheiten und als Schreibanforderungseinheiten.

Eine Leseanforderungseinheit entspricht einem strikt konsistenten Lesevorgang pro Sekunde oder zwei letztendlich konsistenten Lesevorgängen pro Sekunde für ein Element mit einer Größe von bis zu 4 KB. Weitere Informationen über die DynamoDB-Lesekonsistenzmodelle finden Sie unter DynamoDB-Lesekonsistenz.

Eine Schreibanforderungseinheit entspricht einem Schreibvorgang pro Sekunde für ein Element mit einer Größe von bis zu 1 KB.

Weitere Informationen zum Verbrauch von Lese- und Schreibeinheiten finden Sie unter Lese- und Schreibvorgänge in DynamoDB.

Anfänglicher Durchsatz und Skalierungseigenschaften

DynamoDB-Tabellen mit dem On-Demand-Kapazitätsmodus passen sich automatisch an das Datenverkehrsaufkommen Ihrer Anwendung an. Neue On-Demand-Tabellen können bis zu 4 000 Schreibvorgänge pro Sekunde und 12 000 Lesevorgänge pro Sekunde verarbeiten. Der On-Demand-Kapazitätsmodus passt sich sofort an bis zu das Doppelte des vorherigen Höchststands des Datenverkehrs für eine Tabelle an. Nehmen wir z. B. an, dass das Datenverkehrsmuster Ihrer Anwendung zwischen 25 000 und 50 000 strikt konsistenten Lesevorgängen pro Sekunde variiert. Der vorherige Höchststand des Datenverkehrs liegt bei 50 000 Lesevorgängen pro Sekunde. Der On-Demand-Kapazitätsmodus ermöglicht sofort einen anhaltenden Datenverkehr von bis zu 100 000 Lesevorgängen pro Sekunde. Wenn Ihre Anwendung einen Datenverkehr von 100 000 Lesevorgängen pro Sekunde aufrechterhält, wird dieser Höchststand zum neuen vorherigen Höchststand. Dieser Höchststand ermöglicht es, dass ein nachfolgender Datenverkehr von bis zu 200 000 Lesevorgängen pro Sekunde erreicht werden kann.

Wenn Ihr Workload mehr als das Doppelte des vorherigen Höchststands für eine Tabelle generiert, weist DynamoDB automatisch mehr Kapazität zu, während sich Ihr Verkehrsaufkommen erhöht. Diese Kapazitätszuweisung trägt dazu bei, dass Ihr Workload nicht gedrosselt wird. Eine Drosselung kann jedoch eintreten, wenn Sie innerhalb von 30 Minuten das Doppelte Ihres vorherigen Höchststands überschreiten. Nehmen wir z. B. an, dass das Datenverkehrsmuster Ihrer Anwendung

zwischen 25 000 und 50 000 strikt konsistenten Lesevorgängen pro Sekunde variiert. Der vorherige Höchststand des Datenverkehrs liegt bei 50 000 Lesevorgängen pro Sekunde. Wir empfehlen, Ihre Tabelle entsprechend vorzubereiten oder Ihr Datenverkehrswachstum auf mindestens 30 Minuten zu verteilen, bevor Sie mehr als 100 000 Lesevorgänge pro Sekunde erreichen. Weitere Informationen zum Vorwärmen finden Sie unter Grundlegendes zum Warmdurchsatz von DynamoDB.

DynamoDB legt die 30-minütige Drosselungsbeschränkung nicht fest, wenn der Höchststand des Datenverkehrs Ihres Workloads innerhalb des Doppelten des vorherigen Höchststands bleibt. Wenn der Höchststand Ihres Datenverkehrs das Doppelte des Höchststands überschreitet, stellen Sie sicher, dass dieses Wachstum 30 Minuten nach dem letzten Erreichen des Höchststands erfolgt.

Maximaler DynamoDB-Durchsatz für On-Demand-Tabellen

Für On-Demand-Tabellen können Sie optional den maximalen Lese- oder Schreibdurchsatz (oder beides) pro Sekunde für einzelne Tabellen und zugehörige globale Sekundärindizes (GSIs) angeben. Die Angabe eines maximalen On-Demand-Durchsatzes trägt dazu bei, die Nutzung und die Kosten auf Tabellenebene zu begrenzen. Standardmäßig gelten die Einstellungen für den maximalen Durchsatz nicht. Ihre On-Demand-Durchsatzrate ist durch ein AWS -Servicekontingent auf einen Durchsatz von 40 000 Lese- und Schreibvorgängen auf Tabellenebene für alle Tabellen im Konto begrenzt. Bei Bedarf können Sie eine Erhöhung des Servicekontingents beantragen.

Wenn Sie den maximalen Durchsatz für eine On-Demand-Tabelle konfigurieren, werden Durchsatzanforderungen, die den angegebenen Höchstwert überschreiten, gedrosselt. Sie können die Einstellungen für den maximalen Durchsatz auf Tabellenebene jederzeit an Ihre Anwendungsanforderungen anpassen.

Im Folgenden sind einige gängige Anwendungsfälle aufgeführt, für die der maximale Durchsatz für On-Demand-Tabellen von Vorteil sein kann:

- Optimierung der Durchsatzkosten – Die Verwendung des maximalen Durchsatzes für On-Demand-Tabellen bietet eine zusätzliche Ebene der Kostenvorhersehbarkeit und Verwaltbarkeit. Darüber hinaus profitieren Sie von mehr Flexibilität mit dem On-Demand-Modus zur Unterstützung von Workloads mit unterschiedlichen Datenverkehrsmustern und Budgets.
- Schutz vor übermäßiger Nutzung – Wenn Sie den maximalen Durchsatz festlegen, können Sie verhindern, dass bei einer On-Demand-Tabelle ein unbeabsichtigter Anstieg des Lese- oder Schreibverbrauchs auftritt, der durch nicht optimierten Code oder nicht autorisierte Prozesse entstehen könnte. Diese Einstellung auf Tabellenebene kann Unternehmen davor schützen, innerhalb eines bestimmten Zeitraums übermäßig viele Ressourcen zu verbrauchen.

- Schutz nachgelagerter Dienste – Eine Kundenanwendung kann Serverless- und Nicht-Serverless-Technologien beinhalten. Der Serverless-Teil der Architektur kann schnell skaliert werden, um den Anforderungen gerecht zu werden. Nachgelagerte Komponenten mit festen Kapazitäten könnten jedoch überlastet sein. Durch die Implementierung von Einstellungen für den maximalen Durchsatz für On-Demand-Tabellen kann verhindert werden, dass große Mengen von Ereignissen mit unerwarteten negativen Auswirkungen auf mehrere nachgelagerte Komponenten übertragen werden.

Sie können den maximalen Durchsatz für den On-Demand-Modus für neue und bestehende Tabellen mit einer Region und globale Tabellen und konfigurieren. GSIs Sie können den maximalen Durchsatz außerdem bei der Tabellenwiederherstellung und beim Datenimport aus Amazon-S3-Workflows konfigurieren.

Sie können die Einstellungen für den maximalen Durchsatz für On-Demand-Tabellen über die DynamoDB-Konsole, die AWS CLI, AWS CloudFormation oder die DynamoDB-API festlegen.

> **Note**
>
> Der maximale Durchsatz für eine On-Demand-Tabelle wird mit bestem Bemühen angewendet und sollte als Ziel und nicht als garantierte Anforderungsobergrenze betrachtet werden. Ihr Workload kann aufgrund der Burst-Kapazität vorübergehend den angegebenen maximalen Durchsatz überschreiten. In einigen Fällen verwendet DynamoDB die Burst-Kapazität, um Lese- und Schreibvorgänge zu bewältigen, die Ihre Einstellungen für den maximalen Durchsatz der Tabelle überschreiten. Mit Burst-Kapazität können unerwartete Lese- oder Schreibanforderungen erfolgreich sein, wo sie andernfalls gedrosselt werden würden.

Themen

- Überlegungen zur Verwendung des maximalen Durchsatzes für den On-Demand-Modus
- Drosselung und Metriken anfordern CloudWatch

Überlegungen zur Verwendung des maximalen Durchsatzes für den On-Demand-Modus

Wenn Sie den maximalen Durchsatz für Tabellen im On-Demand-Modus verwenden, gelten die folgenden Überlegungen:

- Sie können unabhängig voneinander den maximalen Durchsatz für Lese- und Schreibvorgänge für jede On-Demand-Tabelle oder einen einzelnen globalen sekundären Index innerhalb dieser Tabelle festlegen, um Ihren Ansatz auf der Grundlage spezifischer Anforderungen zu optimieren.
- Sie können Amazon verwenden CloudWatch , um DynamoDB-Nutzungsmetriken auf Tabellenebene zu überwachen und zu verstehen und die geeigneten Einstellungen für den maximalen Durchsatz für den On-Demand-Modus zu ermitteln. Weitere Informationen finden Sie unter DynamoDB-Metriken und -Dimensionen.
- Wenn Sie die Einstellungen für den maximalen Lese- oder Schreibdurchsatz (oder beide) für ein Replikat einer globalen Tabelle angeben, werden dieselben Einstellungen für den maximalen Durchsatz automatisch auf alle Replikattabellen angewendet. Es ist wichtig, dass die Replikattabellen und sekundären Indizes in einer globalen Tabelle über identische Schreibdurchsatzeinstellungen verfügen, um eine ordnungsgemäße Replikation der Daten sicherzustellen. Weitere Informationen finden Sie unter Bewährte Methoden für globale Tabellen.
- Der kleinste maximale Lese- oder Schreibdurchsatz, den Sie angeben können, ist eine Anforderungseinheit pro Sekunde.
- Der von Ihnen angegebene maximale Durchsatz muss unter dem Standard-Durchsatzkontingent liegen, das für jede On-Demand-Tabelle oder einen einzelnen globalen sekundären Index innerhalb dieser Tabelle verfügbar ist.

Drosselung und Metriken anfordern CloudWatch

Wenn Ihre Anwendung den maximalen Lese- oder Schreibdurchsatz überschreitet, den Sie für Ihre On-Demand-Tabelle festgelegt haben, beginnt DynamoDB, diese Anforderungen zu drosseln. Wenn DynamoDB einen Lese- oder Schreibvorgang drosselt, gibt er eine `ThrottlingException` an den Aufrufer zurück. Sie können dann, falls erforderlich, entsprechende Maßnahmen ergreifen. Sie können beispielsweise die Einstellung für den maximalen Tabellendurchsatz erhöhen oder deaktivieren oder kurz warten, bevor Sie die Anforderung erneut versuchen.

Um die Überwachung des für eine Tabelle oder einen globalen sekundären Index konfigurierten maximalen Durchsatzes zu vereinfachen, CloudWatch bietet die folgenden Metriken: OnDemandMaxReadRequestUnits und. OnDemandMaxWriteRequestUnits

DynamoDB – Modus mit bereitgestellter Kapazität

Wenn Sie eine neue bereitgestellte Tabelle in DynamoDB erstellen, müssen Sie ihre bereitgestellte Durchsatzkapazität angeben. Dies ist der Umfang des Lese- und Schreibdurchsatzes, der von

der Tabelle unterstützt werden kann. Die Abrechnung erfolgt auf der Grundlage der von Ihnen bereitgestellten stündlichen Lese- und Schreibkapazität. Sie basiert nicht darauf, wie viel von der bereitgestellten Kapazität Sie tatsächlich verbraucht haben.

Da sich Ihre Anwendungsdaten und Zugriffsanforderungen ändern, müssen Sie möglicherweise die Durchsatzeinstellungen der Tabelle anpassen. Die bereitgestellte Kapazität Ihrer Tabelle kann mithilfe von Auto Scaling als Reaktion auf Datenverkehrsänderungen automatisch angepasst werden. DynamoDB Auto Scaling verwendet eine Skalierungsrichtlinie in Application Auto Scaling. Um Auto Scaling in DynamoDB zu konfigurieren, legen Sie zusätzlich zum Zielauslastungsprozentsatz die Mindest- und Höchstwerte der Lese- und Schreibkapazität fest. Application Auto Scaling erstellt und verwaltet die CloudWatch Alarme, die Skalierungsereignisse auslösen, wenn die Metrik vom Ziel abweicht. Auto Scaling überwacht die Aktivität Ihrer Tabelle und passt ihre Kapazitätseinstellungen basierend auf vorkonfigurierten Schwellenwerten nach oben oder unten an. Auto Scaling wird ausgelöst, wenn Ihre verbrauchte Kapazität für zwei aufeinanderfolgende Minuten die konfigurierte Zielauslastung überschreitet. CloudWatch Alarme können eine kurze Verzögerung von bis zu einigen Minuten haben, bevor sie die auto Skalierung auslösen. Weitere Informationen finden Sie unter Automatische Verwaltung der Durchsatzkapazität mit DynamoDB-Auto-Scaling.

Wenn Sie DynamoDB-Auto-Scaling verwenden, werden die Durchsatzeinstellungen automatisch, entsprechend der tatsächlichen Workloads, angepasst. Sie können auch die UpdateTable-Operation verwenden, um die Durchsatzkapazität der Tabelle manuell anzupassen. Dafür können Sie sich beispielsweise entscheiden, wenn Sie Daten per Massenladevorgang aus einem vorhandenen Datenspeicher in die neue DynamoDB-Tabelle übertragen müssen. Sie könnten die Tabelle mit einer Einstellung für einen großen Schreibdurchsatz erstellen und anschließend diese Einstellung, nachdem das Massenladen der Daten beendet ist, reduzieren.

> **Note**
>
> Standardmäßig schützt DynamoDB Sie vor unbeabsichtigter, unkontrollierter Nutzung. Um die Durchsatzgrenzen von 40 000 Lese- und Schreibvorgängen auf Tabellenebene für alle Tabellen in Ihrem Konto zu überschreiten, können Sie eine Erhöhung dieses Kontingents anfordern. Durchsatzanforderungen, die das standardmäßige Durchsatzkontingent der Tabelle überschreiten, werden gedrosselt. Weitere Informationen finden Sie unter Standardkontingente für den Durchsatz.

Sie können Tabellen innerhalb eines fortlaufenden 24-stündigen Zeitfensters bis zu viermal vom Modus mit bereitgestellter Kapazität auf den On-Demand-Modus umstellen. Sie können Tabellen jederzeit vom On-Demand-Modus auf den Modus mit bereitgestellter Kapazität umstellen.

Weitere Informationen zum Wechseln zwischen Lese- und Schreibkapazitätsmodus finden Sie unter Überlegungen beim Umstellen der Kapazitätsmodi in DynamoDB.

Themen

- Lesekapazitätseinheiten und Schreibkapazitätseinheiten
- Auswählen der ersten Durchsatzeinstellungen
- DynamoDB Auto Scaling
- Automatische Verwaltung der Durchsatzkapazität mit DynamoDB-Auto-Scaling
- DynamoDB – reservierte Kapazität

Lesekapazitätseinheiten und Schreibkapazitätseinheiten

Für Tabellen im bereitgestellten Modus geben Sie die Durchsatzanforderungen in Form von Kapazitätseinheiten an. Diese Einheiten stellen die Datenmenge dar, die Ihre Anwendung zum Lesen oder Schreiben pro Sekunde benötigt. Sie können diese Einstellungen später bei Bedarf ändern oder DynamoDB-Auto-Scaling aktivieren, um sie automatisch zu ändern.

Für ein Elemente mit einer Größe von bis zu 4 KB entspricht eine Leseanforderungseinheit (RCU) einem strikt konsistenten Lesevorgang pro Sekunde oder zwei letztendlich konsistenten Lesevorgängen pro Sekunde. Weitere Informationen über die DynamoDB-Lesekonsistenzmodelle finden Sie unter DynamoDB-Lesekonsistenz.

Eine Schreibkapazitätseinheit (WCU) stellt einen Schreibvorgang pro Sekunde für ein Element mit einer Größe von bis zu 1 KB dar. Weitere Informationen über die verschiedenen Lese- und Schreibvorgänge finden Sie unter Lese- und Schreibvorgänge in DynamoDB.

Auswählen der ersten Durchsatzeinstellungen

Für jede Anwendung gelten unterschiedliche Anforderungen für Lese- und Schreibvorgänge in einer Datenbank. Wenn Sie die ersten Durchsatzeinstellungen für eine DynamoDB-Tabelle festlegen, berücksichtigen Sie Folgendes:

- Erwartete Lese- und Schreibanforderungsraten – Sie müssen die Anzahl der Lese- und Schreibvorgänge schätzen, die Sie pro Sekunde ausführen müssen.

- Größe der Elemente – Einige Elemente sind so klein, dass sie mit einer einzigen Kapazitätseinheit gelesen oder geschrieben werden können. Größere Elemente erfordern mehrere Kapazitätseinheiten. Durch die Schätzung der durchschnittlichen Größe der Elemente in der Tabelle können Sie genaue Einstellungen für den bereitgestellten Durchsatz der Tabelle angeben.
- Anforderungen an die Lesekonsistenz – Lesekapazitätseinheiten basieren auf strikt konsistenten Lesevorgängen, die doppelt so viele Datenbankressourcen verbrauchen wie letztendlich konsistente Lesevorgänge. Sie müssen selbst entscheiden, ob Ihre Anwendung Strongly Consistent-Lesevorgänge erfordert oder ob sie diese Anforderung lockern kann und stattdessen Eventually Consistent-Lesevorgänge verwendet. Lesevorgänge in DynamoDB sind standardmäßig letztendlich konsistent. Sofern erforderlich, können Sie strikt konsistente Lesevorgänge für diese Operationen anfordern.

Angenommen, Sie möchten 80 Elemente pro Sekunde aus einer Tabelle lesen. Die Elemente haben eine Größe von 3 KB und Sie benötigen strikt konsistente Lesevorgänge. In diesem Szenario ist für jeden Lesevorgang eine bereitgestellte Lesekapazitätseinheit erforderlich. Um die entsprechende Zahl zu bestimmen, teilen Sie die Elementgröße der Operation durch 4 KB. Runden Sie dann auf die nächste ganze Zahl auf, wie im folgenden Beispiel dargestellt:

- 3 KB / 4 KB = 0,75 oder 1 Lesekapazitätseinheit

Um also 80 Elemente pro Sekunde aus einer Tabelle zu lesen, legen Sie den bereitgestellten Lesedurchsatz der Tabelle auf 80 Lesekapazitätseinheiten fest, wie im folgenden Beispiel gezeigt:

- 1 Lesekapazitätseinheit pro Element × 80 Lesevorgänge pro Sekunde = 80 Lesekapazitätseinheiten

Nehmen wir jetzt an, Sie möchten 100 Elemente pro Sekunde in Ihre Tabelle schreiben und die Elemente haben jeweils eine Größe von 512 Byte. In diesem Fall ist für jeden Schreibvorgang eine bereitgestellte Schreibkapazitätseinheit erforderlich. Um die entsprechende Zahl zu bestimmen, teilen Sie die Elementgröße der Operation durch 1 KB. Runden Sie dann auf die nächste ganze Zahl auf, wie im folgenden Beispiel dargestellt:

- 512 Byte / 1 KB = 0,5 oder 1 Schreibkapazitätseinheit

Um 100 Elemente pro Sekunde in Ihre Tabelle zu schreiben, legen Sie den bereitgestellten Schreibdurchsatz der Tabelle auf 100 Schreibkapazitätseinheiten fest:

- 1 Schreibkapazitätseinheit pro Element × 100 Schreibvorgänge pro Sekunde = 100 Schreibkapazitätseinheiten

DynamoDB Auto Scaling

DynamoDB Auto Scaling verwaltet die bereitgestellte Durchsatzkapazität für Tabellen und globale sekundäre Indizes. Mit Auto Scaling definieren Sie einen Bereich (Unter- und Obergrenze) für Lese- und Schreibkapazitätseinheiten. Außerdem legen Sie einen Zielauslastungsprozentsatz in diesem Bereich fest. DynamoDB-Auto-Scaling versucht, Ihre Zielauslastung aufrechtzuerhalten, auch wenn der Workload Ihrer Anwendung steigt oder sinkt.

Mit Auto Scaling von DynamoDB kann eine Tabelle oder ein globaler sekundärer Index ihre bereitgestellte Lese- und Schreibkapazität erhöhen, um plötzliche Erhöhungen des Datenverkehrs ohne Anforderungsdrosselung zu bewältigen. Wenn die Workload abnimmt, kann Auto Scaling von DynamoDB den Durchsatz verringern, sodass Sie nicht für ungenutzte bereitgestellte Kapazität bezahlen.

> **Note**
>
> Wenn Sie den verwenden AWS-Managementkonsole , um eine Tabelle oder einen globalen sekundären Index zu erstellen, ist DynamoDB Auto Scaling standardmäßig aktiviert.
> Sie können die Auto Scaling-Einstellungen jederzeit über die Konsole AWS CLI, die oder eine der verwalten AWS SDKs. Weitere Informationen finden Sie unter Automatische Verwaltung der Durchsatzkapazität mit DynamoDB-Auto-Scaling.

Nutzungsrate

Mithilfe der Nutzungsrate können Sie feststellen, ob Sie die zu viel Kapazität bereitstellen. In diesem Fall sollten Sie die Kapazität Ihrer Tabelle reduzieren, um Kosten zu sparen. Umgekehrt kann sie mit der Nutzungsrate auch feststellen, ob Sie zu wenig Kapazität bereitstellen. In diesem Fall sollten Sie die Tabellenkapazität erhöhen, um eine mögliche Drosselung von Anforderung bei Instances mit unerwartet hohem Datenverkehr zu verhindern. Weitere Informationen finden Sie unter Amazon DynamoDB auto scaling: Performance and cost optimization at any scale.

Wenn Sie DynamoDB Auto Scaling verwenden, müssen Sie auch einen Zielauslastungsprozentsatz festlegen. Auto Scaling verwendet diesen Prozentsatz als Ziel, um die Kapazität nach oben oder

unten anzupassen. Wir empfehlen, die Zielauslastung auf 70 % festzulegen. Weitere Informationen finden Sie unter Automatische Verwaltung der Durchsatzkapazität mit DynamoDB-Auto-Scaling.

Automatische Verwaltung der Durchsatzkapazität mit DynamoDB-Auto-Scaling

Viele Datenbank-Workloads sind zyklischer Natur, während andere sich nur schwer vorhersagen lassen. Nehmen wir als Beispiel eine Social-Networking-App, bei der die meisten Benutzer tagsüber aktiv sind. Die Datenbank muss in der Lage sein, die Aktivitäten am Tag zu verarbeiten, während in der Nacht nicht so viel Durchsatz erforderlich ist. Ein weiteres Beispiel wäre eine neue Mobile-Gaming-App, die sich unerwartet schnell großer Beliebtheit erfreut. Wenn das Spiel jedoch zu beliebt wird, kann dies dazu führen, dass die verfügbaren Datenbankressourcen überschritten werden und sich dies negativ auf die Leistung und Kundenzufriedenheit auswirkt. Diese Arten von Workloads erfordern häufig manuelle Eingriffe, um die Datenbankressourcen nach oben oder unten zu skalieren, um sie an die jeweiligen Nutzungsschwankungen anzupassen.

Amazon DynamoDB Auto Scaling verwendet den AWS Application Auto Scaling-Service, um die bereitgestellte Durchsatzkapazität in Ihrem Namen dynamisch an die tatsächlichen Verkehrsmuster anzupassen. Auf diese Weise kann eine Tabelle oder ein globaler sekundärer Index (GSI) die bereitgestellte Lese- und Schreibkapazität erhöhen, um einen plötzlichen Anstieg des Datenverkehrs ohne Drosselung zu bewältigen. Wenn der Workload abnimmt, senkt Application Auto Scaling Auto Scaling den Durchsatz, sodass Sie für ungenutzte Kapazität nicht zahlen müssen.

> **Note**
>
> Wenn Sie den verwenden AWS-Managementkonsole , um eine Tabelle oder einen globalen sekundären Index zu erstellen, ist DynamoDB Auto Scaling standardmäßig aktiviert. Sie können Ihre Auto Scaling-Einstellungen jederzeit ändern. Weitere Informationen finden Sie unter Verwenden der auto AWS-Managementkonsole Skalierung mit DynamoDB.
> Wenn Sie eine Tabelle oder ein globales Tabellenreplikat löschen, werden alle zugehörigen skalierbaren Ziele, Skalierungsrichtlinien oder CloudWatch Alarme nicht automatisch mit gelöscht.

Mit Application Auto Scaling erstellen Sie eine Skalierungsrichtlinie für eine Tabelle oder einen globalen sekundären Index. Die Skalierungsrichtlinie gibt an, ob die Lese- oder Schreibkapazität (oder beides) sowie die Einstellungen für die minimalen und maximalen bereitgestellten Kapazitätseinheiten für die Tabelle oder den Index skaliert werden sollen.

Die Skalierungsrichtlinie enthält außerdem eine Zielauslastung, d. h. den zu einem bestimmten Zeitpunkt verbrauchten bereitgestellten Durchsatz in Prozent. Application Auto Scaling verwendet einen Zielverfolgungsalgorithmus, um den bereitgestellten Durchsatz der Tabelle (oder des Indexes) als Reaktion auf die tatsächlichen Workloads nach oben oder unten anzupassen, sodass die tatsächliche Kapazitätsauslastung bei oder in der Nähe Ihrer Zielauslastung bleibt.

DynamoDB-Ausgaben verbrauchten den bereitgestellten Durchsatz für Zeiträume von einer Minute. Auto Scaling wird ausgelöst, wenn Ihre verbrauchte Kapazität für zwei aufeinanderfolgende Minuten die konfigurierte Zielauslastung überschreitet. CloudWatch Alarme können eine kurze Verzögerung von bis zu einigen Minuten haben, bevor sie die auto Skalierung auslösen. Diese Verzögerung gewährleistet eine genaue CloudWatch metrische Auswertung. Wenn die Spitzen des verbrauchten Durchsatzes in Abständen von mehr als einer Minute auftreten, wird Auto Scaling möglicherweise nicht ausgelöst. In ähnlicher Weise kann es zu einer Herunterskalierung kommen, wenn 15 aufeinanderfolgende Datenpunkte unter der Zielauslastung liegen. In beiden Fällen wird die UpdateTableAPI aufgerufen, nachdem Auto Scaling ausgelöst wurde. Die Aktualisierung der bereitgestellten Kapazität für die Tabelle oder den Index kann dann einige Minuten dauern. Während dieses Zeitraums werden alle Anforderungen gedrosselt, die die zuvor bereitgestellte Kapazität der Tabellen überschreiten.

> **Important**
>
> Sie können die Anzahl der Datenpunkte, die verletzt werden müssen, um den zugrunde liegenden Alarm auszulösen, nicht anpassen (obwohl sich die aktuelle Anzahl künftig ändern könnte).

Sie können die Zielauslastungswerte von Auto Scaling auf 20 bis 90 % für Ihre Lese- und Schreibkapazität festlegen.

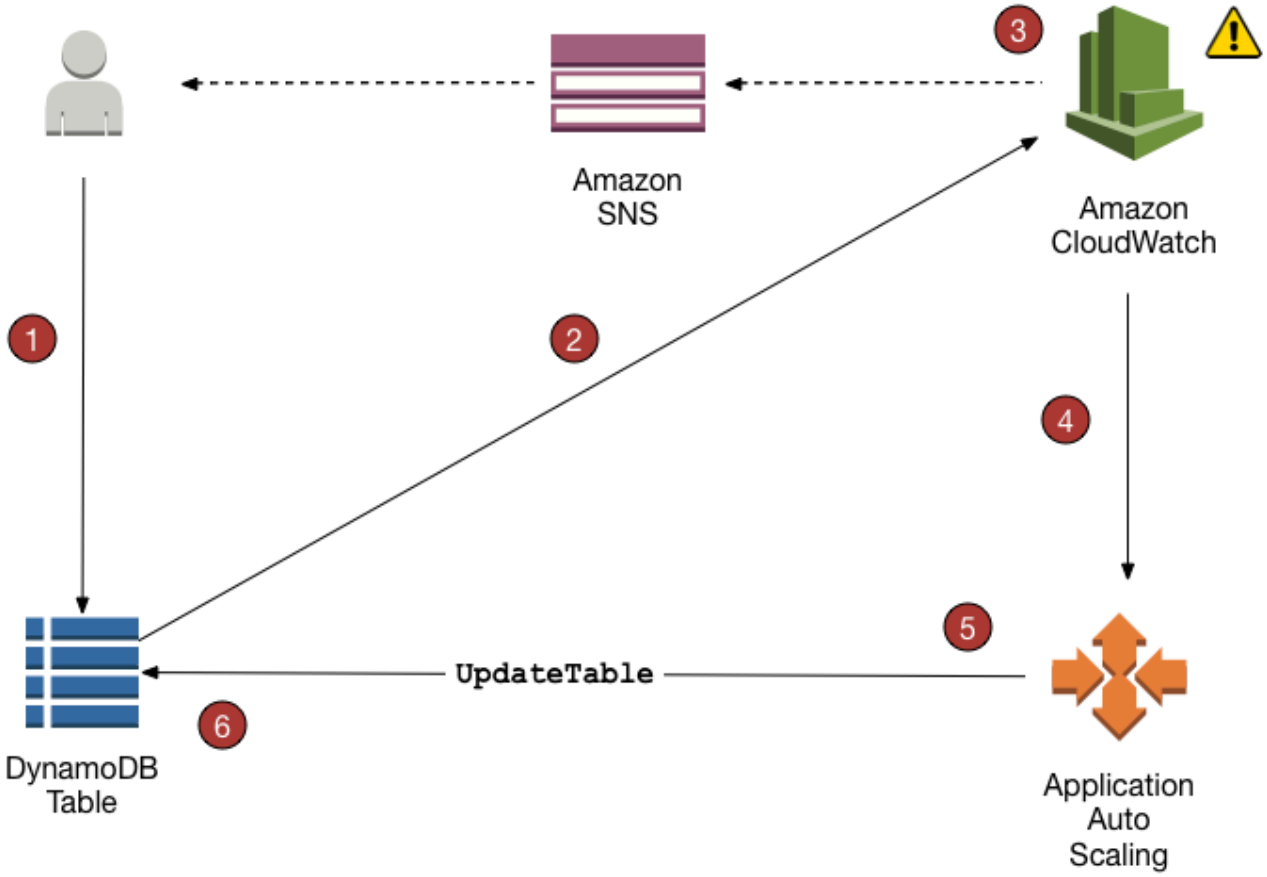
> **Note**
>
> Zusätzlich zu den Tabellen unterstützt DynamoDB-Auto-Scaling auch globale sekundäre Indizes. Jeder globale sekundäre Index verfügt über eine eigene bereitgestellte Durchsatzkapazität, unabhängig von der seiner Basistabelle. Wenn Sie eine Skalierungsrichtlinie für einen globalen sekundären Index erstellen, passt Application Auto Scaling die Einstellungen für den bereitgestellten Durchsatz für den Index an, um

sicherzustellen, dass die tatsächliche Auslastung auf oder in der Nähe des gewünschten Auslastungsverhältnisses bleibt.

Funktionsweise von DynamoDB-Auto-Scaling

> **Note**
>
> Informationen für einen schnellen Einstieg in DynamoDB-Auto-Scaling finden Sie unter Verwenden der auto AWS-Managementkonsole Skalierung mit DynamoDB.

Die folgende Abbildung bietet einen allgemeinen Überblick darüber, wie DynamoDB-Auto-Scaling die Durchsatzkapazität für eine Tabelle verwaltet:



Die folgenden Schritte fassen den Prozess der automatischen Skalierung zusammen, wie in der vorherigen Abbildung gezeigt:

1. Sie erstellen eine Application-Auto-Scaling-Richtlinie für die DynamoDB-Tabelle.
2. DynamoDB veröffentlicht Kennzahlen zur verbrauchten Kapazität auf Amazon. CloudWatch
3. Wenn die verbrauchte Kapazität der Tabelle Ihre Zielauslastung für einen bestimmten Zeitraum überschreitet (oder unter das Ziel fällt), CloudWatch löst Amazon einen Alarm aus. Sie können den Alarm in der Konsole anzeigen und Benachrichtigungen über Amazon Simple Notification Service (Amazon SNS) erhalten.
4. Der CloudWatch Alarm ruft Application Auto Scaling auf, um Ihre Skalierungsrichtlinie auszuwerten.
5. Application Auto Scaling gibt eine `UpdateTable`-Anforderung aus, um den bereitgestellten Durchsatz der Tabelle anzupassen.
6. DynamoDB verarbeitet die `UpdateTable`-Anforderung und erhöht (bzw. senkt) die bereitgestellte Durchsatzkapazität der Tabelle dynamisch, sodass sie sich der Zielauslastung nähert.

Wir wollen die Funktionsweise von DynamoDB-Auto-Scaling anhand einer Beispieltabelle namens `ProductCatalog` erläutern. In die Tabelle werden selten Daten massenweise geladen, es erfolgt also nicht sehr viel Schreibaktivität. Es gibt jedoch ein hohes Maß an Leseaktivität, die im Laufe der Zeit variiert. Durch die Überwachung der CloudWatch Amazon-Metriken für stellen Sie fest`ProductCatalog`, dass die Tabelle 1.200 Lesekapazitätseinheiten benötigt (um zu verhindern, dass DynamoDB Leseanforderungen drosselt, wenn die Aktivität ihren Höhepunkt erreicht). Sie stellen außerdem fest, dass `ProductCatalog` mindestens 150 Lesekapazitätseinheiten erfordert, wenn der Leseverkehr am niedrigsten ist. Weitere Informationen zur Vermeidung einer Drosselung finden Sie unter Beheben von Drosselungsereignissen in Amazon DynamoDB.

Angesichts der Spanne von 150 bis 1.200 Lesekapazitätseinheiten entscheiden Sie, dass eine Zielauslastung von 70 Prozent für die Tabelle `ProductCatalog` angemessen wäre. Die Zielauslastung ist das Verhältnis der verbrauchten Kapazitätseinheiten zu den bereitgestellten Kapazitätseinheiten in Prozent. Application Auto Scaling verwendet einen eigenen Ziel-Tracking-Algorithmus, um sicherzustellen, dass die bereitgestellte Lesekapazität von `ProductCatalog` wie erforderlich angepasst wird, damit die Auslastung nahezu 70 Prozent beträgt.

> **Note**
>
> DynamoDB Auto Scaling ändert die Einstellungen für den bereitgestellten Durchsatz nur, wenn die tatsächliche Workload über einen anhaltenden Zeitraum von mehreren Minuten erhöht oder niedrig bleibt. Der Ziel-Tracking-Algorithmus von Application Auto Scaling

versucht, dafür zu sorgen, dass die Zielauslastung langfristig Ihrem gewählten Wert nahezu entspricht.
Plötzliche Aktivitätsspitzen werden von der integrierten Burst-Kapazität der Tabelle bewältigt. Weitere Informationen finden Sie unter Burst-Kapazität.

Um DynamoDB-Auto-Scaling für die Tabelle `ProductCatalog` zu aktivieren, erstellen Sie eine Skalierungsrichtlinie. Diese Richtlinie legt Folgendes fest:

- Die Tabelle oder der globale sekundäre Index, die Sie verwalten möchten
- Den Kapazitätstyp, der verwaltet werden soll (Lese- oder Schreibkapazität)
- Die Ober- und Untergrenze für die bereitgestellten Durchsatzeinstellungen
- Ihre Zielauslastung

Wenn Sie eine Skalierungsrichtlinie erstellen, erstellt Application Auto Scaling in Ihrem Namen zwei CloudWatch Amazon-Alarme. Jedes Paar stellt die Ober- und Untergrenze für Ihren bereitgestellten Durchsatz dar. Diese CloudWatch Alarme werden ausgelöst, wenn die tatsächliche Auslastung der Tabelle über einen längeren Zeitraum von Ihrer Zielauslastung abweicht.

Wenn einer der CloudWatch Alarme ausgelöst wird, sendet Ihnen Amazon SNS eine Benachrichtigung (sofern Sie ihn aktiviert haben). Der CloudWatch Alarm ruft dann Application Auto Scaling auf, das wiederum DynamoDB auffordert, die bereitgestellte Kapazität der `ProductCatalog` Tabelle nach Bedarf nach oben oder unten anzupassen.

Während eines Skalierungsereignisses AWS Config wird pro aufgezeichnetem Konfigurationselement abgerechnet. Wenn ein Skalierungsereignis auftritt, werden für jedes auto-scaling Skalierungsereignis beim Lesen und Schreiben vier CloudWatch ProvisionedCapacity Alarme erstellt: ProvisionedCapacityLow alarms: ProvisionedCapacityHigh und ConsumedCapacity alarms: AlarmHigh, AlarmLow. Das bedeutet insgesamt acht Alarme. AWS Config Zeichnet daher acht Konfigurationselemente für jedes Skalierungsereignis auf.

> **Note**
>
> Sie können die DynamoDB-Skalierung auch so planen, dass sie zu bestimmten Zeiten erfolgt. Erfahren Sie hier mehr über die grundlegenden Schritte.

Nutzungshinweise

Bevor Sie DynamoDB-Auto-Scaling verwenden, beachten Sie Folgendes:

- Bei DynamoDB-Auto-Scaling kann die Lese- bzw. Schreibkapazität in Übereinstimmung mit Ihrer Auto-Scaling-Richtlinie so oft wie nötig erhöht werden. Alle DynamoDB-Kontingente bleiben in Kraft, wie unter Kontingente in Amazon DynamoDB beschrieben.
- DynamoDB-Auto-Scaling hindert Sie nicht daran, die Einstellungen für den bereitgestellten Durchsatz manuell zu ändern. Diese manuellen Anpassungen wirken sich nicht auf bestehende CloudWatch Alarme aus, die sich auf DynamoDB Auto Scaling beziehen.
- Wenn Sie die automatische DynamoDB-Skalierung für eine Tabelle aktivieren, die über einen oder mehrere globale sekundäre Indizes verfügt, wird dringend empfohlen, die automatische Skalierung auch einheitlich auf diese Indexe anzuwenden. Dies trägt dazu bei, eine bessere Leistung beim Schreiben und Lesen von Tabellen zu gewährleisten und eine Drosselung zu vermeiden. Sie können Auto Scaling aktivieren, indem Sie Apply same settings to global secondary indexes (Die gleichen Einstellungen für globale sekundäre Indizes anwenden) in der AWS-Managementkonsole auswählen. Weitere Informationen finden Sie unter Aktivieren von DynamoDB-Auto-Scaling in bestehenden Tabellen.
- Wenn Sie eine Tabelle oder ein globales Tabellenreplikat löschen, werden alle zugehörigen skalierbaren Ziele, Skalierungsrichtlinien oder CloudWatch Alarme nicht automatisch mit gelöscht.
- Beim Erstellen einer GSI für eine vorhandene Tabelle ist Auto Scaling für die GSI nicht aktiviert. Sie müssen die Kapazität während der Erstellung der GSI manuell verwalten. Sobald der Backfill auf der GSI abgeschlossen ist und sie den aktiven Status erreicht, funktioniert Auto Scaling wie gewohnt.

Verwenden der auto AWS-Managementkonsole Skalierung mit DynamoDB

Wenn Sie das verwenden, AWS-Managementkonsole um eine neue Tabelle zu erstellen, ist Amazon DynamoDB Auto Scaling für diese Tabelle standardmäßig aktiviert. Sie können die Konsole auch verwenden, um Auto Scaling für vorhandene Tabellen zu aktivieren, Auto Scaling-Einstellungen zu ändern oder Auto Scaling zu deaktivieren.

> **Note**
>
> Für erweiterte Funktionen wie das Einstellen der Abklingzeiten für Scale-In und Scale-Out verwenden Sie die AWS Command Line Interface (AWS CLI), um die auto Skalierung von

DynamoDB zu verwalten. Weitere Informationen finden Sie unter Verwenden der auto AWS CLI Skalierung von DynamoDB zur Verwaltung.

Themen

- Bevor Sie beginnen: Erteilen von Benutzerberechtigungen für DynamoDB-Auto-Scaling
- Erstellen einer neuen Tabelle mit aktiviertem Auto Scaling
- Aktivieren von DynamoDB-Auto-Scaling in bestehenden Tabellen
- Anzeigen von Auto-Scaling-Aktivitäten in der Konsole
- Ändern oder Deaktivieren der DynamoDB-Auto-Scaling-Einstellungen

Bevor Sie beginnen: Erteilen von Benutzerberechtigungen für DynamoDB-Auto-Scaling

In AWS Identity and Access Management (IAM) `DynamoDBFullAccess` stellt die AWS verwaltete Richtlinie die erforderlichen Berechtigungen für die Verwendung der DynamoDB-Konsole bereit. Jedoch benötigen Benutzer für DynamoDB Auto Scaling einige zusätzliche Berechtigungen.

Important

Zum Löschen einer Tabelle mit Auto Scaling sind `application-autoscaling:*`-Berechtigungen erforderlich. Die AWS verwaltete Richtlinie `DynamoDBFullAccess` umfasst solche Berechtigungen.

Um einen Benutzer für DynamoDB-Konsolenzugriff und DynamoDB Auto Scaling einzurichten, erstellen Sie eine Rolle und fügen Sie die AmazonDynamoDBFullAccess-Richtlinie zu dieser Rolle hinzu. Weisen Sie die Rolle dann einem Benutzer zu.

Erstellen einer neuen Tabelle mit aktiviertem Auto Scaling

Note

DynamoDB Auto Scaling erfordert das Vorhandensein einer serviceverknüpften Rolle (`AWSServiceRoleForApplicationAutoScaling_DynamoDBTable`), die in Ihrem Namen Auto-Scaling-Aktionen durchführt. Diese Rolle wird automatisch für Sie erstellt. Weitere Informationen finden Sie unter Serviceverknüpfte Rollen für Application Auto Scaling im Benutzerhandbuch zu Application Auto Scaling.

So erstellen Sie eine neue Tabelle mit aktiviertem Auto Scaling

1. Öffnen Sie die DynamoDB-Konsole unter. https://console.aws.amazon.com/dynamodb/
2. Wählen Sie Create table (Tabelle erstellen) aus.
3. Geben Sie auf der Seite Tabelle erstellen einen Tabellennamen und Details zum Primärschlüssel ein.
4. Wenn Sie Standardeinstellungen auswählen, wird Auto Scaling in der neuen Tabelle aktiviert.

   Wählen Sie andernfalls Einstellungen anpassen aus und gehen Sie wie folgt vor, um benutzerdefinierte Einstellungen für die Tabelle anzugeben:

   a. Behalten Sie für Tabellenklasse die Standardauswahl DynamoDB Standard bei.

   b. Behalten Sie für Lese-/Schreibkapazitätseinstellungen die Standardauswahl Bereitgestellt bei und gehen Sie dann wie folgt vor:

      i. Stellen Sie sicher, dass unter Lesekapazität die Option Auto Scaling auf An festgelegt ist.

      ii. Stellen Sie sicher, dass unter Schreibkapazität die Option Auto Scaling auf An festgelegt ist.

      iii. Legen Sie unter Lesekapazität und Schreibkapazität die gewünschte Skalierungsrichtlinie für die Tabelle und optional alle globalen sekundären Indizes der Tabelle fest.

      - Minimale Kapazitätseinheiten – Geben Sie den unteren Grenzwert für den Auto-Scaling-Bereich ein.
      - Maximale Kapazitätseinheiten – Geben Sie den oberen Grenzwert für den Auto-Scaling-Bereich ein.
      - Zielauslastung – Geben Sie den Zielauslastungsprozentsatz für die Tabelle ein.

      > **Note**
      >
      > Wenn Sie einen globalen Sekundärindex für die neue Tabelle erstellen, entspricht die Kapazität des Index zum Zeitpunkt der Erstellung der Kapazität Ihrer Basistabelle. Sie können die Kapazität des Index in den Einstellungen der Tabelle ändern, nachdem Sie die Tabelle erstellt haben.

5. Wählen Sie Create table (Tabelle erstellen) aus. Die Tabelle wird mit den von Ihnen angegebenen Auto-Scaling-Parametern erstellt.

Aktivieren von DynamoDB-Auto-Scaling in bestehenden Tabellen

> **Note**
>
> DynamoDB Auto Scaling erfordert das Vorhandensein einer serviceverknüpften Rolle (`AWSServiceRoleForApplicationAutoScaling_DynamoDBTable`), die in Ihrem Namen Auto-Scaling-Aktionen durchführt. Diese Rolle wird automatisch für Sie erstellt. Weitere Informationen finden Sie unter Serviceverknüpfte Rollen für Application Auto Scaling.

So aktivieren Sie DynamoDB-Auto-Scaling für eine vorhandene Tabelle

1. Öffnen Sie die DynamoDB-Konsole unter. https://console.aws.amazon.com/dynamodb/
2. Klicken Sie im Navigationsbereich auf der linken Seite der Konsole auf Tabellen.
3. Wählen Sie die Tabelle aus, für die Sie Auto Scaling aktivieren möchten, und gehen Sie dann wie folgt vor:
   a. Wählen Sie die Registerkarte Zusätzliche Einstellungen aus.
   b. Wählen Sie im Abschnitt Lese-/Schreibkapazität die Option Bearbeiten aus.
   c. Wählen Sie im Abschnitt Kapazitätsmodus die Option Bereitgestellt aus.
   d. Setzen Sie im Abschnitt Kapazität der Tabelle Auto Scaling auf Ein für Lesekapazität, Schreibkapazität oder beides. Legen Sie für jeden dieser Punkte die gewünschte Skalierungsrichtlinie für die Tabelle und optional alle globalen sekundären Indizes der Tabelle fest.
      - Minimale Kapazitätseinheiten – Geben Sie den unteren Grenzwert für den Auto-Scaling-Bereich ein.
      - Maximale Kapazitätseinheiten – Geben Sie den oberen Grenzwert für den Auto-Scaling-Bereich ein.
      - Zielauslastung – Geben Sie den Zielauslastungsprozentsatz für die Tabelle ein.
      - Dieselben read/write Kapazitätseinstellungen für alle globalen sekundären Indizes verwenden — Wählen Sie aus, ob globale sekundäre Indizes dieselbe Auto Scaling-Richtlinie wie die Basistabelle verwenden sollen.

> **Note**
>
> Um eine optimale Leistung zu erzielen, empfehlen wir, die Option Dieselben read/write Kapazitätseinstellungen für alle globalen sekundären Indizes verwenden zu aktivieren. Mit dieser Option kann die automatische DynamoDB-Skalierung alle globalen sekundären Indexe in der Basistabelle einheitlich skalieren. Dazu gehören vorhandene globale sekundäre Indizes und alle anderen, die Sie in Zukunft für diese Tabelle erstellen.
> Wenn diese Option aktiviert ist, können Sie keine Skalierungsrichtlinie für einen einzelnen globale sekundäre Index festlegen.

4. Wenn Sie die gewünschten Einstellungen vorgenommen haben, wählen Sie Save (Speichern) aus.

Anzeigen von Auto-Scaling-Aktivitäten in der Konsole

Wenn Ihre Anwendung Lese- und Schreibverkehr in die Tabellen führt, modifiziert DynamoDB-Auto-Scaling die Durchsatzeinstellungen der Tabelle dynamisch. Amazon CloudWatch verfolgt die bereitgestellte und verbrauchte Kapazität, gedrosselte Ereignisse, Latenz und andere Metriken für all Ihre DynamoDB-Tabellen und sekundären Indizes.

Wenn Sie diese Metriken in der DynamoDB-Konsole anzeigen möchten, wählen Sie die Tabelle aus, mit der Sie arbeiten möchten, und öffnen Sie die Registerkarte Überwachen. Um eine anpassbare Ansicht der Tabellenmetriken zu erstellen, wählen Sie Alle anzeigen in. CloudWatch

Ändern oder Deaktivieren der DynamoDB-Auto-Scaling-Einstellungen

Sie können den verwenden AWS-Managementkonsole , um Ihre DynamoDB-Auto-Scaling-Einstellungen zu ändern. Gehen Sie dazu zur Registerkarte Zusätzliche Einstellungen für Ihre Tabelle und wählen Sie Bearbeiten im Abschnitt Lese-/Schreibkapazität aus. Weitere Informationen zu diesen Einstellungen finden Sie unter Aktivieren von DynamoDB-Auto-Scaling in bestehenden Tabellen.

Verwenden der auto AWS CLI Skalierung von DynamoDB zur Verwaltung

Anstatt das zu verwenden AWS-Managementkonsole, können Sie das AWS Command Line Interface (AWS CLI) verwenden, um die auto Skalierung von Amazon DynamoDB zu verwalten. Das Tutorial in diesem Abschnitt erläutert, wie der AWS CLI installiert und konfiguriert wird, um DynamoDB-Auto-Scaling zu verwalten. In diesem Tutorial führen Sie folgende Aufgaben aus:

- Erstellen einer DynamoDB-Tabelle mit dem Namen `TestTable`. Die Ersteinstellungen des Durchsatzes sind 5 Lesekapazitätseinheiten und 5 Schreibkapazitätseinheiten.
- Erstellen Sie die Application-Auto-Scaling-Richtlinie für `TestTable`. Die Richtlinie versucht ein 50 %-Zielverhältnis zwischen verbrauchter und bereitgestellter Schreibkapazität beizubehalten. Der Bereich für diese Metrik liegt zwischen 5 und 10 Schreibkapazitätseinheiten. (Application Auto Scaling darf den Durchsatz außerhalb dieses Bereichs nicht anpassen.)
- Ausführen eines Python-Programms, um den Datenverkehr zu `TestTable` zu leiten. Wenn das Zielverhältnis 50 Prozent über einen anhaltenden Zeitraum überschreitet, benachrichtigt Application Auto Scaling DynamoDB, um den Durchsatz von `TestTable` nach oben anzupassen, sodass die 50 % Zielauslastung aufrechterhalten werden kann.
- Überprüfen, ob DynamoDB die bereitgestellte Schreibkapazität für `TestTable` erfolgreich angepasst hat.

> **Note**
>
> Sie können die DynamoDB-Skalierung auch so planen, dass sie zu bestimmten Zeiten erfolgt. Erfahren Sie hier mehr über die grundlegenden Schritte.

Themen

- Bevor Sie beginnen
- Schritt 1: Erstellen einer DynamoDB-Tabelle
- Schritt 2: Registrieren eines skalierbaren Ziels
- Schritt 3: Erstellen einer Skalierungsrichtlinie
- Schritt 4: Leiten Sie den Schreibverkehr weiter zu TestTable
- Schritt 5: Anzeigen der Application Auto Scaling
- (Optional) Schritt 6: Bereinigen

Bevor Sie beginnen

Führen Sie folgende Schritte durch, bevor Sie das Tutorial starten.

Installieren Sie das AWS CLI

Wenn Sie es noch nicht getan haben, müssen Sie AWS CLI installieren und konfigurieren. Befolgen Sie hierzu die Anweisungen im AWS Command Line Interface Benutzerhandbuch:

- Installieren des AWS CLI
- Konfigurieren von AWS CLI

Installieren von Python

Ein Teil dieses Tutorials erfordert von Ihnen das Ausführen eines Python-Programms (siehe Schritt 4: Leiten Sie den Schreibverkehr weiter zu TestTable). Wenn Sie das noch nicht installiert haben, können Sie Python herunterladen.

Schritt 1: Erstellen einer DynamoDB-Tabelle

In diesem Schritt verwenden Sie die, AWS CLI um zu erstellen`TestTable`. Der Primärschlüssel besteht aus `pk` (Partitionsschlüssel) und `sk` (Sortierschlüssel). Beide Attribute sind vom Typ `Number`. Die Ersteinstellungen des Durchsatzes sind 5 Lesekapazitätseinheiten und 5 Schreibkapazitätseinheiten.

1. Verwenden Sie den folgenden AWS CLI Befehl, um die Tabelle zu erstellen.

```
aws dynamodb create-table \
    --table-name TestTable \
    --attribute-definitions \
        AttributeName=pk,AttributeType=N \
        AttributeName=sk,AttributeType=N \
    --key-schema \
        AttributeName=pk,KeyType=HASH \
        AttributeName=sk,KeyType=RANGE \
    --provisioned-throughput ReadCapacityUnits=5,WriteCapacityUnits=5
```

2. Verwenden Sie den folgenden Befehl, um den Status der Tabelle zu überprüfen.

```
aws dynamodb describe-table \
    --table-name TestTable \
    --query "Table.[TableName,TableStatus,ProvisionedThroughput]"
```

   Die Tabelle ist betriebsbereit, wenn ihr Status `ACTIVE` ist.

Schritt 2: Registrieren eines skalierbaren Ziels

Jetzt registrieren Sie die Schreibkapazität der Tabelle als ein skalierbares Ziel mit Application Auto Scaling. Auf diese Weise kann Application Auto Scaling die bereitgestellte Schreibkapazität für TestTable, jedoch nur im Bereich von 5—10 Kapazitätseinheiten, anpassen.

> **Note**
>
> DynamoDB-Auto-Scaling erfordert das Vorhandensein einer Rolle (`AWSServiceRoleForApplicationAutoScaling_DynamoDBTable`), die in Ihrem Namen Auto-Scaling-Aktionen durchführt. Diese Rolle wird automatisch für Sie erstellt. Weitere Informationen finden Sie unter Serviceverknüpfte Rollen für Application Auto Scaling im Benutzerhandbuch zu Application Auto Scaling.

1. Geben Sie den folgenden Befehl ein, um das skalierbare Ziel zu registrieren.

```
aws application-autoscaling register-scalable-target \
    --service-namespace dynamodb \
    --resource-id "table/TestTable" \
    --scalable-dimension "dynamodb:table:WriteCapacityUnits" \
    --min-capacity 5 \
    --max-capacity 10
```

2. Verwenden Sie den folgenden Befehl, um die Registrierung zu überprüfen.

```
aws application-autoscaling describe-scalable-targets \
    --service-namespace dynamodb \
    --resource-id "table/TestTable"
```

> **Note**
>
> Sie können ein skalierbares Ziel auch für einen globalen sekundären Index registrieren. Beispielsweise werden für einen globalen sekundären Index („Testindex“) die Ressourcen-ID und die skalierbaren Dimensionsargumente entsprechend aktualisiert.
>
> ```
> aws application-autoscaling register-scalable-target \
>     --service-namespace dynamodb \
>     --resource-id "table/TestTable/index/test-index" \
>     --scalable-dimension "dynamodb:index:WriteCapacityUnits" \
> ```

```
    --min-capacity 5 \
    --max-capacity 10
```

Schritt 3: Erstellen einer Skalierungsrichtlinie

In diesem Schritt erstellen Sie eine Skalierungsrichtlinie für `TestTable`. Die Richtlinie definiert die Details, unter denen Application Auto Scaling den bereitgestellten Durchsatz Ihrer Tabelle anpassen kann und welche Aktionen erfolgen müssen, wenn dies geschieht. Sie ordnen diese Richtlinie dem skalierbaren Ziel zu, das Sie im vorherigen Schritt definiert haben (Schreibkapazitätseinheiten für die `TestTable`-Tabelle).

Die Richtlinie enthält die folgenden Elemente:

- `PredefinedMetricSpecification` – Die Metrik, die Application Auto Scaling anpassen darf. Für DynamoDB sind die folgenden Werte gültige Werte für `PredefinedMetricType`:
  - `DynamoDBReadCapacityUtilization`
  - `DynamoDBWriteCapacityUtilization`
- `ScaleOutCooldown` – Die Mindestzeit (in Sekunden) zwischen jedem Application-Auto-Scaling-Ereignis, das den bereitgestellten Durchsatz erhöht. Dieser Parameter ermöglicht Application Auto Scaling den Durchsatz als Reaktion auf reale Workloads fortlaufend, aber nicht aggressiv, zu erhöhen. Die Standardeinstellung für `ScaleOutCooldown` lautet 0.
- `ScaleInCooldown` – Die Mindestzeit (in Sekunden) zwischen jedem Application-Auto-Scaling-Ereignis, das den bereitgestellten Durchsatz verringert. Dieser Parameter ermöglicht Application Auto Scaling den Durchsatz schrittweise und berechenbar zu verringern. Die Standardeinstellung für `ScaleInCooldown` lautet 0.
- `TargetValue`—Application Auto Scaling stellt sicher, dass das Verhältnis von verbrauchter Kapazität zu bereitgestellter Kapazität diesem Wert nahezu entspricht. Sie definieren `TargetValue` als Prozentsatz.

> **Note**
>
> Um besser zu verstehen, wie der `TargetValue` funktioniert, nehmen Sie an, dass Sie über eine Tabelle mit einer Einstellung des bereitgestellten Durchsatzes von 200 Schreibkapazitätseinheiten verfügen. Sie entscheiden sich dafür, eine Skalierungsrichtlinie für diese Tabelle mit einem `TargetValue` von 70 % zu erstellen.

Angenommen, Sie beginnen den Schreibverkehr zu der Tabelle zu leiten, damit der tatsächliche Schreibdurchsatz bei 150 Kapazitätseinheiten liegt. Das consumed-to-provisioned Verhältnis liegt jetzt bei (150/ 200) oder 75 Prozent. Dieses Verhältnis überschreitet Ihr Ziel, sodass Application Auto Scaling die bereitgestellte Schreibkapazität auf 215 erhöht, damit das Verhältnis (150 / 215) oder 69,77 Prozent ist – so nah an Ihrem `TargetValue` wie möglich, jedoch nicht darüber.

Für `TestTable`, setzen Sie `TargetValue` 50 Prozent. Application Auto Scaling passt den bereitgestellten Durchsatz der Tabelle im Bereich von 5 bis 10 Kapazitätseinheiten (sieheSchritt 2: Registrieren eines skalierbaren Ziels) an, sodass das consumed-to-provisioned Verhältnis bei oder nahe 50 Prozent bleibt. Sie legen die Werte für `ScaleOutCooldown` und `ScaleInCooldown` auf 60 Sekunden fest.

1. Erstellen Sie eine Datei mit dem Namen `scaling-policy.json` und dem folgenden Inhalt.

```
{
    "PredefinedMetricSpecification": {
        "PredefinedMetricType": "DynamoDBWriteCapacityUtilization"
    },
    "ScaleOutCooldown": 60,
    "ScaleInCooldown": 60,
    "TargetValue": 50.0
}
```

2. Verwenden Sie den folgenden AWS CLI Befehl, um die Richtlinie zu erstellen.

```
aws application-autoscaling put-scaling-policy \
    --service-namespace dynamodb \
    --resource-id "table/TestTable" \
    --scalable-dimension "dynamodb:table:WriteCapacityUnits" \
    --policy-name "MyScalingPolicy" \
    --policy-type "TargetTrackingScaling" \
    --target-tracking-scaling-policy-configuration file://scaling-policy.json
```

3. Beachten Sie in der Ausgabe, dass Application Auto Scaling zwei CloudWatch Amazon-Alarme generiert hat — jeweils einen für die obere und untere Grenze des Skalierungszielbereichs.
4. Verwenden Sie den folgenden AWS CLI Befehl, um weitere Details zur Skalierungsrichtlinie anzuzeigen.

```
aws application-autoscaling describe-scaling-policies \
    --service-namespace dynamodb \
    --resource-id "table/TestTable" \
    --policy-name "MyScalingPolicy"
```

5. Überprüfen Sie in der Ausgabe, dass die Richtlinieneinstellungen mit Ihren Spezifikationen von Schritt 2: Registrieren eines skalierbaren Ziels und Schritt 3: Erstellen einer Skalierungsrichtlinie übereinstimmen.

Schritt 4: Leiten Sie den Schreibverkehr weiter zu TestTable

Jetzt können Sie Ihre Skalierungsrichtlinie testen, indem Sie Daten in die `TestTable` schreiben. Führen Sie dafür ein Python-Programm aus.

1. Erstellen Sie eine Datei mit dem Namen `bulk-load-test-table.py` und dem folgenden Inhalt.

```
import boto3
dynamodb = boto3.resource('dynamodb')

table = dynamodb.Table("TestTable")

filler = "x" * 100000

i = 0
while (i < 10):
    j = 0
    while (j < 10):
        print (i, j)

        table.put_item(
            Item={
                'pk':i,
                'sk':j,
                'filler':{"S":filler}
            }
        )
        j += 1
    i += 1
```

2. Geben Sie den folgenden Befehl ein, um das Programm auszuführen.

```
python bulk-load-test-table.py
```

Die bereitgestellte Schreibkapazität für `TestTable` ist sehr niedrig (5 Schreibkapazitätseinheiten), sodass das Programm aufgrund von Schreibeinschränkungen gelegentlich stockt. Dieses Verhalten wird erwartet.

Lassen Sie das Programm weiter ausführen, während Sie mit dem nächsten Schritt fortfahren.

Schritt 5: Anzeigen der Application Auto Scaling

In diesem Schritt sehen Sie sich die Application-Auto-Scaling-Aktionen an, die in Ihrem Namen initiiert werden. Sie überprüfen ebenfalls, ob Application Auto Scaling die bereitgestellte Schreibkapazität für `TestTable` aktualisiert hat.

1. Geben Sie den folgenden Befehl ein, um die Application Auto Scaling-Aktionen anzuzeigen.

   ```
   aws application-autoscaling describe-scaling-activities \
       --service-namespace dynamodb
   ```

   Führen Sie diesen Befehl gelegentlich erneut aus, während das Python-Programm ausgeführt wird. (Es kann einige Minuten dauern, bevor die Skalierungsrichtlinie aufgerufen wird.) Sie sollten schließlich die folgende Ausgabe sehen.

   ```
   ...
   {
       "ScalableDimension": "dynamodb:table:WriteCapacityUnits",
       "Description": "Setting write capacity units to 10.",
       "ResourceId": "table/TestTable",
       "ActivityId": "0cc6fb03-2a7c-4b51-b67f-217224c6b656",
       "StartTime": 1489088210.175,
       "ServiceNamespace": "dynamodb",
       "EndTime": 1489088246.85,
       "Cause": "monitor alarm AutoScaling-table/TestTable-
   AlarmHigh-1bb3c8db-1b97-4353-baf1-4def76f4e1b9 in state ALARM triggered policy
    MyScalingPolicy",
       "StatusMessage": "Successfully set write capacity units to 10. Change
    successfully fulfilled by dynamodb.",
       "StatusCode": "Successful"
   },
   ```

```
...
```

Dies weist darauf hin, dass Application Auto Scaling eine `UpdateTable`Anforderung an DynamoDB ausgegeben hat.

2. Geben Sie den folgenden Befehl ein, um zu überprüfen, ob DynamoDB die Schreibkapazität der Tabelle erhöht hat:

```
aws dynamodb describe-table \
    --table-name TestTable \
    --query "Table.[TableName,TableStatus,ProvisionedThroughput]"
```

Die `WriteCapacityUnits` hätten von `5` auf `10` skaliert werden sollen.

(Optional) Schritt 6: Bereinigen

In diesem Tutorial haben Sie einige Ressourcen erstellt. Sie können diese Ressourcen löschen, wenn Sie sie nicht mehr benötigen.

1. Löschen Sie die Skalierungsrichtlinie für `TestTable`:

```
aws application-autoscaling delete-scaling-policy \
    --service-namespace dynamodb \
    --resource-id "table/TestTable" \
    --scalable-dimension "dynamodb:table:WriteCapacityUnits" \
    --policy-name "MyScalingPolicy"
```

2. Melden Sie das skalierbare Ziel ab.

```
aws application-autoscaling deregister-scalable-target \
    --service-namespace dynamodb \
    --resource-id "table/TestTable" \
    --scalable-dimension "dynamodb:table:WriteCapacityUnits"
```

3. Löschen Sie die `TestTable`-Tabelle.

```
aws dynamodb delete-table --table-name TestTable
```

Verwenden des AWS SDK zur Konfiguration von Auto Scaling für Amazon DynamoDB-Tabellen

Zusätzlich zur Verwendung von AWS-Managementkonsole und AWS Command Line Interface (AWS CLI) können Sie Anwendungen schreiben, die mit Amazon DynamoDB Auto Scaling interagieren. Dieser Abschnitt enthält zwei Java-Programme, die Sie verwenden können, um diese Funktionalität zu testen:

- `EnableDynamoDBAutoscaling.java`
- `DisableDynamoDBAutoscaling.java`

Aktivieren von Application Auto Scaling für eine Tabelle

Das folgende Programm zeigt ein Beispiel für die Einrichtung einer Auto-Scaling-Richtlinie für eine DynamoDB-Tabelle (`TestTable`). Es fährt wie folgt fort:

- Das Programm registriert Schreibkapazitätseinheiten als skalierbares Ziel für `TestTable`. Der Bereich für diese Metrik liegt zwischen 5 und 10 Schreibkapazitätseinheiten.
- Nachdem das skalierbare Ziel erstellt wird, erstellt das Programm eine Ziel-Tracking-Konfiguration. Die Richtlinie versucht ein 50 %-Zielverhältnis zwischen verbrauchter und bereitgestellter Schreibkapazität beizubehalten.
- Das Programm erstellt dann die Skalierungsrichtlinie, basierend auf der Ziel-Tracking-Konfiguration.

> **Note**
>
> Wenn Sie eine Tabelle oder ein globales Tabellenreplikat manuell entfernen, entfernen Sie nicht automatisch alle zugehörigen skalierbaren Ziele, Skalierungsrichtlinien oder Alarme. CloudWatch

Java v2

```
import software.amazon.awssdk.regions.Region;
import
 software.amazon.awssdk.services.applicationautoscaling.ApplicationAutoScalingClient;
```

```
import
 software.amazon.awssdk.services.applicationautoscaling.model.ApplicationAutoScalingExceptio
import
 software.amazon.awssdk.services.applicationautoscaling.model.DescribeScalableTargetsRequest
import
 software.amazon.awssdk.services.applicationautoscaling.model.DescribeScalableTargetsRespons
import
 software.amazon.awssdk.services.applicationautoscaling.model.DescribeScalingPoliciesRequest
import
 software.amazon.awssdk.services.applicationautoscaling.model.DescribeScalingPoliciesRespons
import software.amazon.awssdk.services.applicationautoscaling.model.PolicyType;
import
 software.amazon.awssdk.services.applicationautoscaling.model.PredefinedMetricSpecification;
import
 software.amazon.awssdk.services.applicationautoscaling.model.PutScalingPolicyRequest;
import
 software.amazon.awssdk.services.applicationautoscaling.model.RegisterScalableTargetRequest;
import software.amazon.awssdk.services.applicationautoscaling.model.ScalingPolicy;
import
 software.amazon.awssdk.services.applicationautoscaling.model.ServiceNamespace;
import
 software.amazon.awssdk.services.applicationautoscaling.model.ScalableDimension;
import software.amazon.awssdk.services.applicationautoscaling.model.MetricType;
import
 software.amazon.awssdk.services.applicationautoscaling.model.TargetTrackingScalingPolicyCon
import java.util.List;

/**
 * Before running this Java V2 code example, set up your development environment,
 including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-started.html
 */
public class EnableDynamoDBAutoscaling {
    public static void main(String[] args) {
        final String usage = """

            Usage:
               <tableId> <roleARN> <policyName>\s

            Where:
               tableId - The table Id value (for example, table/Music).
```

```
                roleARN - The ARN of the role that has ApplicationAutoScaling
permissions.
                policyName - The name of the policy to create.

            """;

        if (args.length != 3) {
            System.out.println(usage);
            System.exit(1);
        }

        System.out.println("This example registers an Amazon DynamoDB table, which
is the resource to scale.");
        String tableId = args[0];
        String roleARN = args[1];
        String policyName = args[2];
        ServiceNamespace ns = ServiceNamespace.DYNAMODB;
        ScalableDimension tableWCUs =
ScalableDimension.DYNAMODB_TABLE_WRITE_CAPACITY_UNITS;
        ApplicationAutoScalingClient appAutoScalingClient =
ApplicationAutoScalingClient.builder()
            .region(Region.US_EAST_1)
            .build();

        registerScalableTarget(appAutoScalingClient, tableId, roleARN, ns,
tableWCUs);
        verifyTarget(appAutoScalingClient, tableId, ns, tableWCUs);
        configureScalingPolicy(appAutoScalingClient, tableId, ns, tableWCUs,
policyName);
    }

    public static void registerScalableTarget(ApplicationAutoScalingClient
appAutoScalingClient, String tableId, String roleARN, ServiceNamespace ns,
ScalableDimension tableWCUs) {
        try {
            RegisterScalableTargetRequest targetRequest =
RegisterScalableTargetRequest.builder()
                .serviceNamespace(ns)
                .scalableDimension(tableWCUs)
                .resourceId(tableId)
                .roleARN(roleARN)
                .minCapacity(5)
                .maxCapacity(10)
                .build();
```

```
            appAutoScalingClient.registerScalableTarget(targetRequest);
            System.out.println("You have registered " + tableId);

        } catch (ApplicationAutoScalingException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
        }
    }

    // Verify that the target was created.
    public static void verifyTarget(ApplicationAutoScalingClient
appAutoScalingClient, String tableId, ServiceNamespace ns, ScalableDimension
tableWCUs) {
        DescribeScalableTargetsRequest dscRequest =
DescribeScalableTargetsRequest.builder()
            .scalableDimension(tableWCUs)
            .serviceNamespace(ns)
            .resourceIds(tableId)
            .build();

        DescribeScalableTargetsResponse response =
appAutoScalingClient.describeScalableTargets(dscRequest);
        System.out.println("DescribeScalableTargets result: ");
        System.out.println(response);
    }

    // Configure a scaling policy.
    public static void configureScalingPolicy(ApplicationAutoScalingClient
appAutoScalingClient, String tableId, ServiceNamespace ns, ScalableDimension
tableWCUs, String policyName) {
        // Check if the policy exists before creating a new one.
        DescribeScalingPoliciesResponse describeScalingPoliciesResponse =
appAutoScalingClient.describeScalingPolicies(DescribeScalingPoliciesRequest.builder()
            .serviceNamespace(ns)
            .resourceId(tableId)
            .scalableDimension(tableWCUs)
            .build());

        if (!describeScalingPoliciesResponse.scalingPolicies().isEmpty()) {
            // If policies exist, consider updating an existing policy instead of
creating a new one.
            System.out.println("Policy already exists. Consider updating it
instead.");
```

```
            List<ScalingPolicy> polList =
 describeScalingPoliciesResponse.scalingPolicies();
            for (ScalingPolicy pol : polList) {
                System.out.println("Policy name:" +pol.policyName());
            }
        } else {
            // If no policies exist, proceed with creating a new policy.
            PredefinedMetricSpecification specification =
 PredefinedMetricSpecification.builder()

 .predefinedMetricType(MetricType.DYNAMO_DB_WRITE_CAPACITY_UTILIZATION)
                .build();

            TargetTrackingScalingPolicyConfiguration policyConfiguration =
 TargetTrackingScalingPolicyConfiguration.builder()
                .predefinedMetricSpecification(specification)
                .targetValue(50.0)
                .scaleInCooldown(60)
                .scaleOutCooldown(60)
                .build();

            PutScalingPolicyRequest putScalingPolicyRequest =
 PutScalingPolicyRequest.builder()
                .targetTrackingScalingPolicyConfiguration(policyConfiguration)
                .serviceNamespace(ns)
                .scalableDimension(tableWCUs)
                .resourceId(tableId)
                .policyName(policyName)
                .policyType(PolicyType.TARGET_TRACKING_SCALING)
                .build();

            try {
                appAutoScalingClient.putScalingPolicy(putScalingPolicyRequest);
                System.out.println("You have successfully created a scaling policy
 for an Application Auto Scaling scalable target");
            } catch (ApplicationAutoScalingException e) {
                System.err.println("Error: " + e.awsErrorDetails().errorMessage());
            }
        }
    }
}
```

Java v1

Das Programm erfordert, dass Sie einenn Amazon-Ressourcennamen (ARN) für eine gültige serviceverknüpfte Application-Auto-Scaling-Rolle angeben. (Zum Beispiel: `arn:aws:iam::122517410325:role/AWSServiceRoleForApplicationAutoScaling_DynamoDBTable`.) Ersetzen Sie im folgenden Programm `SERVICE_ROLE_ARN_GOES_HERE` mit dem tatsächlichen ARN.

```
package com.amazonaws.codesamples.autoscaling;

import
 com.amazonaws.services.applicationautoscaling.AWSApplicationAutoScalingClient;
import
 com.amazonaws.services.applicationautoscaling.AWSApplicationAutoScalingClientBuilder;
import
 com.amazonaws.services.applicationautoscaling.model.DescribeScalableTargetsRequest;
import
 com.amazonaws.services.applicationautoscaling.model.DescribeScalableTargetsResult;
import
 com.amazonaws.services.applicationautoscaling.model.DescribeScalingPoliciesRequest;
import
 com.amazonaws.services.applicationautoscaling.model.DescribeScalingPoliciesResult;
import com.amazonaws.services.applicationautoscaling.model.MetricType;
import com.amazonaws.services.applicationautoscaling.model.PolicyType;
import
 com.amazonaws.services.applicationautoscaling.model.PredefinedMetricSpecification;
import com.amazonaws.services.applicationautoscaling.model.PutScalingPolicyRequest;
import
 com.amazonaws.services.applicationautoscaling.model.RegisterScalableTargetRequest;
import com.amazonaws.services.applicationautoscaling.model.ScalableDimension;
import com.amazonaws.services.applicationautoscaling.model.ServiceNamespace;
import
 com.amazonaws.services.applicationautoscaling.model.TargetTrackingScalingPolicyConfiguratio

public class EnableDynamoDBAutoscaling {

 static AWSApplicationAutoScalingClient aaClient = (AWSApplicationAutoScalingClient)
 AWSApplicationAutoScalingClientBuilder
   .standard().build();

 public static void main(String args[]) {

  ServiceNamespace ns = ServiceNamespace.Dynamodb;
```

```
 ScalableDimension tableWCUs = ScalableDimension.DynamodbTableWriteCapacityUnits;
 String resourceID = "table/TestTable";

 // Define the scalable target
 RegisterScalableTargetRequest rstRequest = new RegisterScalableTargetRequest()
   .withServiceNamespace(ns)
   .withResourceId(resourceID)
   .withScalableDimension(tableWCUs)
   .withMinCapacity(5)
   .withMaxCapacity(10)
   .withRoleARN("SERVICE_ROLE_ARN_GOES_HERE");

 try {
  aaClient.registerScalableTarget(rstRequest);
 } catch (Exception e) {
  System.err.println("Unable to register scalable target: ");
  System.err.println(e.getMessage());
 }

 // Verify that the target was created
 DescribeScalableTargetsRequest dscRequest = new DescribeScalableTargetsRequest()
   .withServiceNamespace(ns)
   .withScalableDimension(tableWCUs)
   .withResourceIds(resourceID);
 try {
  DescribeScalableTargetsResult dsaResult =
aaClient.describeScalableTargets(dscRequest);
  System.out.println("DescribeScalableTargets result: ");
  System.out.println(dsaResult);
  System.out.println();
 } catch (Exception e) {
  System.err.println("Unable to describe scalable target: ");
  System.err.println(e.getMessage());
 }

 System.out.println();

 // Configure a scaling policy
 TargetTrackingScalingPolicyConfiguration targetTrackingScalingPolicyConfiguration
= new TargetTrackingScalingPolicyConfiguration()
   .withPredefinedMetricSpecification(
     new PredefinedMetricSpecification()
       .withPredefinedMetricType(MetricType.DynamoDBWriteCapacityUtilization))
   .withTargetValue(50.0)
```

```
    .withScaleInCooldown(60)
    .withScaleOutCooldown(60);

  // Create the scaling policy, based on your configuration
  PutScalingPolicyRequest pspRequest = new PutScalingPolicyRequest()
    .withServiceNamespace(ns)
    .withScalableDimension(tableWCUs)
    .withResourceId(resourceID)
    .withPolicyName("MyScalingPolicy")
    .withPolicyType(PolicyType.TargetTrackingScaling)

 .withTargetTrackingScalingPolicyConfiguration(targetTrackingScalingPolicyConfiguration);

  try {
   aaClient.putScalingPolicy(pspRequest);
  } catch (Exception e) {
   System.err.println("Unable to put scaling policy: ");
   System.err.println(e.getMessage());
  }

  // Verify that the scaling policy was created
  DescribeScalingPoliciesRequest dspRequest = new DescribeScalingPoliciesRequest()
    .withServiceNamespace(ns)
    .withScalableDimension(tableWCUs)
    .withResourceId(resourceID);

  try {
   DescribeScalingPoliciesResult dspResult =
 aaClient.describeScalingPolicies(dspRequest);
   System.out.println("DescribeScalingPolicies result: ");
   System.out.println(dspResult);
  } catch (Exception e) {
   e.printStackTrace();
   System.err.println("Unable to describe scaling policy: ");
   System.err.println(e.getMessage());
  }

 }

}
```

Deaktivieren der Application Auto Scaling für eine Tabelle

Das folgende Programm macht den vorherigen Prozess rückgängig. Es entfernt die Auto Scaling-Richtlinie und hebt die Registrierung des skalierbaren Ziels auf.

Java v2

```
import software.amazon.awssdk.regions.Region;
import
 software.amazon.awssdk.services.applicationautoscaling.ApplicationAutoScalingClient;
import
 software.amazon.awssdk.services.applicationautoscaling.model.ApplicationAutoScalingExceptio
import
 software.amazon.awssdk.services.applicationautoscaling.model.DeleteScalingPolicyRequest;
import
 software.amazon.awssdk.services.applicationautoscaling.model.DeregisterScalableTargetReques
import
 software.amazon.awssdk.services.applicationautoscaling.model.DescribeScalableTargetsRequest
import
 software.amazon.awssdk.services.applicationautoscaling.model.DescribeScalableTargetsRespons
import
 software.amazon.awssdk.services.applicationautoscaling.model.DescribeScalingPoliciesRequest
import
 software.amazon.awssdk.services.applicationautoscaling.model.DescribeScalingPoliciesRespons
import
 software.amazon.awssdk.services.applicationautoscaling.model.ScalableDimension;
import
 software.amazon.awssdk.services.applicationautoscaling.model.ServiceNamespace;

/**
 * Before running this Java V2 code example, set up your development environment,
 including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-started.html
 */

public class DisableDynamoDBAutoscaling {
    public static void main(String[] args) {
        final String usage = """

            Usage:
               <tableId> <policyName>\s
```

```
            Where:
               tableId - The table Id value (for example, table/Music).\s
               policyName - The name of the policy (for example, $Music5-scaling-
policy).

            """;
        if (args.length != 2) {
            System.out.println(usage);
            System.exit(1);
        }

        ApplicationAutoScalingClient appAutoScalingClient =
 ApplicationAutoScalingClient.builder()
            .region(Region.US_EAST_1)
            .build();

        ServiceNamespace ns = ServiceNamespace.DYNAMODB;
        ScalableDimension tableWCUs =
 ScalableDimension.DYNAMODB_TABLE_WRITE_CAPACITY_UNITS;
        String tableId = args[0];
        String policyName = args[1];

        deletePolicy(appAutoScalingClient, policyName, tableWCUs, ns, tableId);
        verifyScalingPolicies(appAutoScalingClient, tableId, ns, tableWCUs);
        deregisterScalableTarget(appAutoScalingClient, tableId, ns, tableWCUs);
        verifyTarget(appAutoScalingClient, tableId, ns, tableWCUs);
    }

    public static void deletePolicy(ApplicationAutoScalingClient
 appAutoScalingClient, String policyName, ScalableDimension tableWCUs,
 ServiceNamespace ns, String tableId) {
        try {
            DeleteScalingPolicyRequest delSPRequest =
 DeleteScalingPolicyRequest.builder()
                .policyName(policyName)
                .scalableDimension(tableWCUs)
                .serviceNamespace(ns)
                .resourceId(tableId)
                .build();

            appAutoScalingClient.deleteScalingPolicy(delSPRequest);
            System.out.println(policyName +" was deleted successfully.");
```

```
        } catch (ApplicationAutoScalingException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
        }
    }

    // Verify that the scaling policy was deleted
    public static void verifyScalingPolicies(ApplicationAutoScalingClient
appAutoScalingClient, String tableId, ServiceNamespace ns, ScalableDimension
tableWCUs) {
        DescribeScalingPoliciesRequest dscRequest =
DescribeScalingPoliciesRequest.builder()
            .scalableDimension(tableWCUs)
            .serviceNamespace(ns)
            .resourceId(tableId)
            .build();

        DescribeScalingPoliciesResponse response =
appAutoScalingClient.describeScalingPolicies(dscRequest);
        System.out.println("DescribeScalableTargets result: ");
        System.out.println(response);
    }

    public static void deregisterScalableTarget(ApplicationAutoScalingClient
appAutoScalingClient, String tableId, ServiceNamespace ns, ScalableDimension
tableWCUs) {
        try {
            DeregisterScalableTargetRequest targetRequest =
DeregisterScalableTargetRequest.builder()
                .scalableDimension(tableWCUs)
                .serviceNamespace(ns)
                .resourceId(tableId)
                .build();

            appAutoScalingClient.deregisterScalableTarget(targetRequest);
            System.out.println("The scalable target was deregistered.");

        } catch (ApplicationAutoScalingException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
        }
    }

    public static void verifyTarget(ApplicationAutoScalingClient
appAutoScalingClient, String tableId, ServiceNamespace ns, ScalableDimension
tableWCUs) {
```

```
        DescribeScalableTargetsRequest dscRequest =
 DescribeScalableTargetsRequest.builder()
            .scalableDimension(tableWCUs)
            .serviceNamespace(ns)
            .resourceIds(tableId)
            .build();

        DescribeScalableTargetsResponse response =
 appAutoScalingClient.describeScalableTargets(dscRequest);
        System.out.println("DescribeScalableTargets result: ");
        System.out.println(response);
    }
}
```

Java v1

```
package com.amazonaws.codesamples.autoscaling;

import
 com.amazonaws.services.applicationautoscaling.AWSApplicationAutoScalingClient;
import
 com.amazonaws.services.applicationautoscaling.model.DeleteScalingPolicyRequest;
import
 com.amazonaws.services.applicationautoscaling.model.DeregisterScalableTargetRequest;
import
 com.amazonaws.services.applicationautoscaling.model.DescribeScalableTargetsRequest;
import
 com.amazonaws.services.applicationautoscaling.model.DescribeScalableTargetsResult;
import
 com.amazonaws.services.applicationautoscaling.model.DescribeScalingPoliciesRequest;
import
 com.amazonaws.services.applicationautoscaling.model.DescribeScalingPoliciesResult;
import com.amazonaws.services.applicationautoscaling.model.ScalableDimension;
import com.amazonaws.services.applicationautoscaling.model.ServiceNamespace;

public class DisableDynamoDBAutoscaling {

 static AWSApplicationAutoScalingClient aaClient = new
 AWSApplicationAutoScalingClient();

 public static void main(String args[]) {

  ServiceNamespace ns = ServiceNamespace.Dynamodb;
```

```
 ScalableDimension tableWCUs = ScalableDimension.DynamodbTableWriteCapacityUnits;
 String resourceID = "table/TestTable";

 // Delete the scaling policy
 DeleteScalingPolicyRequest delSPRequest = new DeleteScalingPolicyRequest()
   .withServiceNamespace(ns)
   .withScalableDimension(tableWCUs)
   .withResourceId(resourceID)
   .withPolicyName("MyScalingPolicy");

 try {
  aaClient.deleteScalingPolicy(delSPRequest);
 } catch (Exception e) {
  System.err.println("Unable to delete scaling policy: ");
  System.err.println(e.getMessage());
 }

 // Verify that the scaling policy was deleted
 DescribeScalingPoliciesRequest descSPRequest = new
DescribeScalingPoliciesRequest()
   .withServiceNamespace(ns)
   .withScalableDimension(tableWCUs)
   .withResourceId(resourceID);

 try {
  DescribeScalingPoliciesResult dspResult =
aaClient.describeScalingPolicies(descSPRequest);
  System.out.println("DescribeScalingPolicies result: ");
  System.out.println(dspResult);
 } catch (Exception e) {
  e.printStackTrace();
  System.err.println("Unable to describe scaling policy: ");
  System.err.println(e.getMessage());
 }

 System.out.println();

 // Remove the scalable target
 DeregisterScalableTargetRequest delSTRequest = new
DeregisterScalableTargetRequest()
   .withServiceNamespace(ns)
   .withScalableDimension(tableWCUs)
   .withResourceId(resourceID);
```

```
  try {
   aaClient.deregisterScalableTarget(delSTRequest);
  } catch (Exception e) {
   System.err.println("Unable to deregister scalable target: ");
   System.err.println(e.getMessage());
  }

  // Verify that the scalable target was removed
  DescribeScalableTargetsRequest dscRequest = new DescribeScalableTargetsRequest()
    .withServiceNamespace(ns)
    .withScalableDimension(tableWCUs)
    .withResourceIds(resourceID);

  try {
   DescribeScalableTargetsResult dsaResult =
 aaClient.describeScalableTargets(dscRequest);
   System.out.println("DescribeScalableTargets result: ");
   System.out.println(dsaResult);
   System.out.println();
  } catch (Exception e) {
   System.err.println("Unable to describe scalable target: ");
   System.err.println(e.getMessage());
  }

 }

}
```

DynamoDB – reservierte Kapazität

Für Tabellen mit bereitgestellter Kapazität, die die Standard-Tabellenklasse verwenden, bietet DynamoDB die Möglichkeit, reservierte Kapazität für Ihre Lese- und Schreibkapazität zu erwerben. Der Erwerb reservierter Kapazität ist eine Vereinbarung zur Zahlung für eine Mindestmenge an bereitgestellter Durchsatzkapazität für die Dauer der Vereinbarung zu vergünstigten Preisen.

> **Note**
>
> Sie können keine reservierte Kapazität für replizierte Schreibkapazitätseinheiten (rWCUs) erwerben. Reservierte Kapazität gilt nur für die Region, in der sie erworben wurde.

> Reservierte Kapazität ist auch nicht für Tabellen verfügbar, die die DynamoDB Standard-IA-Tabellenklasse oder den On-Demand-Kapazitätsmodus verwenden.

Reservierte Kapazität wird in Zuordnungen von 100 WCUs oder 100 erworben. RCUs Das kleinste Angebot an reservierter Kapazität umfasst 100 Kapazitätseinheiten (Lese- oder Schreibvorgänge). Reservierte DynamoDB-Kapazität wird entweder als einjähriges Abonnement oder in ausgewählten Regionen als dreijähriges Abonnement angeboten. Bei einer Laufzeit von einem Jahr können Sie bis zu 54 % gegenüber den Standardtarifen und bei einer Laufzeit von drei Jahren bis zu 77 % gegenüber den Standardtarifen sparen. Weitere Informationen darüber, wie und wann Sie kaufen sollten, finden Sie unter Amazon DynamoDB – reservierte Kapazität.

Wenn Sie reservierte DynamoDB-Kapazität erwerben, zahlen Sie eine einmalige Teilvorauszahlung und erhalten einen vergünstigten Stundensatz für die zugesicherte, bereitgestellte Nutzung. Sie zahlen für die gesamte zugesagte bereitgestellte Nutzung, unabhängig von der tatsächlichen Nutzung, sodass Ihre Kosteneinsparungen eng mit der Nutzung verknüpft sind. Kapazitäten, die Sie über die reservierte Kapazität hinaus nutzen, werden zu Standardpreisen für bereitgestellte Kapazität verrechnet. Durch die Reservierung der Lese- und Schreibkapazitätseinheiten im Voraus erzielen Sie erhebliche Einsparungen Ihrer Kosten für bereitgestellte Kapazitäten.

Sie können reservierte Kapazität nicht verkaufen, kündigen oder in eine andere Region oder ein anderes Konto übertragen.

Grundlegendes zum Warmdurchsatz von DynamoDB

Warmdurchsatz bezieht sich auf die Anzahl der Lese- und Schreibvorgänge, die Ihre DynamoDB-Tabelle sofort unterstützen kann. Diese Werte sind standardmäßig für alle Tabellen und globalen sekundären Indizes (GSI) verfügbar und geben an, wie stark sie auf der Grundlage der historischen Nutzung skaliert wurden. Wenn Sie den On-Demand-Modus verwenden oder den bereitgestellten Durchsatz auf diese Werte aktualisieren, kann Ihre Anwendung sofort Anforderungen bis zu diesen Werten ausgeben.

DynamoDB passt die Werte für den Warmdurchsatz automatisch an, wenn Ihre Nutzung zunimmt. Sie können diese Werte jedoch bei Bedarf auch proaktiv erhöhen, was besonders bei bevorstehenden Spitzenereignissen wie Produkteinführungen oder Sonderverkäufen nützlich ist. Bei geplanten Spitzenereignissen, bei denen die Anforderungsraten an Ihre DynamoDB-Tabelle um das Zehn-, Hundertfache oder mehr steigen könnten, können Sie jetzt bewerten, ob der aktuelle

Warmdurchsatz ausreicht, um den erwarteten Datenverkehr zu bewältigen. Ist dies nicht der Fall, können Sie den Wert für den Warmdurchsatz erhöhen, ohne Ihre Durchsatzeinstellungen oder den Fakturierungsmodus zu ändern. Dieser Vorgang wird als Vorwärmen einer Tabelle bezeichnet, sodass Sie eine Baseline festlegen können, den Ihre Tabellen sofort unterstützen können. Dadurch wird sichergestellt, dass Ihre Anwendungen höhere Anforderungsraten von dem Moment an bewältigen können, in dem sie auftreten.

Sie können den Wert für den Warmdurchsatz für Lesevorgänge, Schreibvorgänge oder beides erhöhen. Sie können diesen Wert für neue und bestehende Tabellen mit einer Region, globale Tabellen und erhöhen. GSIs Bei globalen Tabellen ist diese Funktion für Version 2019.11.21 (aktuell) verfügbar. Die von Ihnen festgelegten Einstellungen für den Warmdurchsatz gelten automatisch für alle Replikattabellen in der globalen Tabelle. Es gibt keine Begrenzung der Anzahl von DynamoDB-Tabellen, die Sie jederzeit vorwärmen können. Die Dauer des Vorwärmens hängt von den von Ihnen eingestellten Werten und der Größe der Tabelle oder des Index ab. Sie können Anforderungen zum Vorwärmen gleichzeitig einreichen, ohne dass diese Anforderungen die Tabellenoperationen beeinträchtigen. Sie können Ihre Tabelle vorwärmen, bis das Tabellen- oder Indexkontingentlimit für Ihr Konto in dieser Region erreicht ist. Verwenden Sie die Service-Quotas-Konsole, um Ihre aktuellen Limits zu überprüfen und sie bei Bedarf zu erhöhen.

Werte für den Warmdurchsatz sind standardmäßig für alle Tabellen und sekundären Indizes kostenlos verfügbar. Wenn Sie diese Standardwerte für den Warmdurchsatz jedoch proaktiv erhöhen, um die Tabellen vorzuwärmen, werden Ihnen diese Anforderungen in Rechnung gestellt. Weitere Informationen finden Sie unter Amazon DynamoDB – Preise.

Weitere Informationen zum Warmdurchsatz finden Sie in den folgenden Themen:

Themen

- Überprüfen des aktuellen Warmdurchsatzes Ihrer DynamoDB-Tabelle
- Erhöhen des Warmdurchsatzes für die vorhandene DynamoDB-Tabelle
- Erstellen einer neuen DynamoDB-Tabelle mit einem höheren Warmdurchsatz
- Grundlegendes zum DynamoDB-Warmdurchsatz in verschiedenen Szenarien

Überprüfen des aktuellen Warmdurchsatzes Ihrer DynamoDB-Tabelle

Verwenden Sie die folgenden Anweisungen AWS CLI und die Anweisungen in der AWS Konsole, um den aktuellen Warmdurchsatzwert Ihrer Tabelle oder Ihres Indexes zu überprüfen.

AWS-Managementkonsole

So überprüfen Sie den Warmdurchsatz Ihrer DynamoDB-Tabelle mithilfe der DynamoDB-Konsole:

1. Melden Sie sich bei der an AWS-Managementkonsole und öffnen Sie die DynamoDB-Konsole unter. https://console.aws.amazon.com/dynamodb/
2. Wählen Sie im linken Navigationsbereich Tables (Tabellen) aus.
3. Wählen Sie auf der Seite Tabellen die gewünschte Tabelle aus.
4. Wählen Sie Zusätzliche Einstellungen aus, um den aktuellen Warmdurchsatzwert anzuzeigen. Dieser Wert wird als Leseeinheiten pro Sekunde und Schreibeinheiten pro Sekunde angezeigt.

AWS CLI

Das folgende AWS CLI Beispiel zeigt Ihnen, wie Sie den Warmdurchsatz Ihrer DynamoDB-Tabelle überprüfen können.

1. Führen Sie die `describe-table`-Operation für die DynamoDB-Tabelle aus.

```
aws dynamodb describe-table --table-name GameScores
```

2. Sie erhalten eine ähnliche Antwort wie die folgende. Ihre `WarmThroughput`-Einstellungen werden als `ReadUnitsPerSecond` und `WriteUnitsPerSecond` angezeigt. `Status` lautet `UPDATING`, wenn der Wert für den Warmdurchsatz aktualisiert wird, und `ACTIVE`, wenn der neue Warmdurchsatzwert festgelegt wird.

```
{
    "Table": {
        "AttributeDefinitions": [
            {
                "AttributeName": "GameTitle",
                "AttributeType": "S"
            },
            {
                "AttributeName": "TopScore",
                "AttributeType": "N"
            },
            {
                "AttributeName": "UserId",
                "AttributeType": "S"
            }
```

```
        ],
        "TableName": "GameScores",
        "KeySchema": [
            {
                "AttributeName": "UserId",
                "KeyType": "HASH"
            },
            {
                "AttributeName": "GameTitle",
                "KeyType": "RANGE"
            }
        ],
        "TableStatus": "ACTIVE",
        "CreationDateTime": 1726128388.729,
        "ProvisionedThroughput": {
            "NumberOfDecreasesToday": 0,
            "ReadCapacityUnits": 0,
            "WriteCapacityUnits": 0
        },
        "TableSizeBytes": 0,
        "ItemCount": 0,
        "TableArn": "arn:aws:dynamodb:us-east-1:XXXXXXXXXXXX:table/GameScores",
        "TableId": "XXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXXX",
        "BillingModeSummary": {
            "BillingMode": "PAY_PER_REQUEST",
            "LastUpdateToPayPerRequestDateTime": 1726128388.729
        },
        "GlobalSecondaryIndexes": [
            {
                "IndexName": "GameTitleIndex",
                "KeySchema": [
                    {
                        "AttributeName": "GameTitle",
                        "KeyType": "HASH"
                    },
                    {
                        "AttributeName": "TopScore",
                        "KeyType": "RANGE"
                    }
                ],
                "Projection": {
                    "ProjectionType": "INCLUDE",
                    "NonKeyAttributes": [
                        "UserId"
```

```
                    ]
                },
                "IndexStatus": "ACTIVE",
                "ProvisionedThroughput": {
                    "NumberOfDecreasesToday": 0,
                    "ReadCapacityUnits": 0,
                    "WriteCapacityUnits": 0
                },
                "IndexSizeBytes": 0,
                "ItemCount": 0,
                "IndexArn": "arn:aws:dynamodb:us-east-1:XXXXXXXXXXXX:table/
GameScores/index/GameTitleIndex",
                "WarmThroughput": {
                    "ReadUnitsPerSecond": 12000,
                    "WriteUnitsPerSecond": 4000,
                    "Status": "ACTIVE"
                }
            }
        ],
        "DeletionProtectionEnabled": false,
        "WarmThroughput": {
            "ReadUnitsPerSecond": 12000,
            "WriteUnitsPerSecond": 4000,
            "Status": "ACTIVE"
        }
    }
}
```

Erhöhen des Warmdurchsatzes für die vorhandene DynamoDB-Tabelle

Nachdem Sie den aktuellen Warmdurchsatzwert Ihrer DynamoDB-Tabelle überprüft haben, können Sie ihn mit den folgenden Schritten aktualisieren:

AWS-Managementkonsole

So überprüfen Sie den Warmdurchsatzwert Ihrer DynamoDB-Tabelle mithilfe der DynamoDB-Konsole:

1. Melden Sie sich bei der an AWS-Managementkonsole und öffnen Sie die DynamoDB-Konsole unter. https://console.aws.amazon.com/dynamodb/
2. Wählen Sie im linken Navigationsbereich Tables (Tabellen) aus.

3. Wählen Sie auf der Seite Tabellen die gewünschte Tabelle aus.
4. Wählen Sie im Feld Warmdurchsatz die Option Bearbeiten aus.
5. Wählen Sie auf der Seite Warmdurchsatz bearbeiten die Option Warmdurchsatz erhöhen aus.
6. Passen Sie die Leseeinheiten pro Sekunde und die Schreibeinheiten pro Sekunde an. Diese beiden Einstellungen definieren den Durchsatz, den die Tabelle sofort verarbeiten kann.
7. Wählen Sie Speichern aus.
8. Ihre Leseeinheiten pro Sekunde und Schreibeinheiten pro Sekunde werden im Feld Warmdurchsatz aktualisiert, wenn die Bearbeitung der Anforderung abgeschlossen ist.

> **Note**
>
> Die Aktualisierung des Werts für den Warmdurchsatz ist eine asynchrone Aufgabe. `Status` ändert sich von `UPDATING` in `ACTIVE`, wenn die Aktualisierung abgeschlossen ist.

AWS CLI

Das folgende AWS CLI Beispiel zeigt Ihnen, wie Sie den Warmdurchsatzwert Ihrer DynamoDB-Tabelle aktualisieren.

1. Führen Sie die `update-table`-Operation für die DynamoDB-Tabelle aus.

```
aws dynamodb update-table \
    --table-name GameScores \
    --warm-throughput ReadUnitsPerSecond=12345,WriteUnitsPerSecond=4567 \
    --global-secondary-index-updates \
        "[
            {
                \"Update\": {
                    \"IndexName\": \"GameTitleIndex\",
                    \"WarmThroughput\": {
                        \"ReadUnitsPerSecond\": 88,
                        \"WriteUnitsPerSecond\": 77
                    }
                }
            }
        ]" \
    --region us-east-1
```

2. Sie erhalten eine ähnliche Antwort wie die folgende. Ihre `WarmThroughput`-Einstellungen werden als `ReadUnitsPerSecond` und `WriteUnitsPerSecond` angezeigt. `Status` lautet `UPDATING`, wenn der Wert für den Warmdurchsatz aktualisiert wird, und `ACTIVE`, wenn der neue Warmdurchsatzwert festgelegt wird.


```
{
    "TableDescription": {
        "AttributeDefinitions": [
            {
                "AttributeName": "GameTitle",
                "AttributeType": "S"
            },
            {
                "AttributeName": "TopScore",
                "AttributeType": "N"
            },
            {
                "AttributeName": "UserId",
                "AttributeType": "S"
            }
        ],
        "TableName": "GameScores",
        "KeySchema": [
            {
                "AttributeName": "UserId",
                "KeyType": "HASH"
            },
            {
                "AttributeName": "GameTitle",
                "KeyType": "RANGE"
            }
        ],
        "TableStatus": "ACTIVE",
        "CreationDateTime": 1730242189.965,
        "ProvisionedThroughput": {
            "NumberOfDecreasesToday": 0,
            "ReadCapacityUnits": 20,
            "WriteCapacityUnits": 10
        },
        "TableSizeBytes": 0,
        "ItemCount": 0,
        "TableArn": "arn:aws:dynamodb:us-east-1:XXXXXXXXXXXX:table/GameScores",
        "TableId": "XXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXXX",
```

```
        "GlobalSecondaryIndexes": [
            {
                "IndexName": "GameTitleIndex",
                "KeySchema": [
                    {
                        "AttributeName": "GameTitle",
                        "KeyType": "HASH"
                    },
                    {
                        "AttributeName": "TopScore",
                        "KeyType": "RANGE"
                    }
                ],
                "Projection": {
                    "ProjectionType": "INCLUDE",
                    "NonKeyAttributes": [
                        "UserId"
                    ]
                },
                "IndexStatus": "ACTIVE",
                "ProvisionedThroughput": {
                    "NumberOfDecreasesToday": 0,
                    "ReadCapacityUnits": 50,
                    "WriteCapacityUnits": 25
                },
                "IndexSizeBytes": 0,
                "ItemCount": 0,
                "IndexArn": "arn:aws:dynamodb:us-east-1:XXXXXXXXXXXX:table/
GameScores/index/GameTitleIndex",
                "WarmThroughput": {
                    "ReadUnitsPerSecond": 50,
                    "WriteUnitsPerSecond": 25,
                    "Status": "UPDATING"
                }
            }
        ],
        "DeletionProtectionEnabled": false,
        "WarmThroughput": {
            "ReadUnitsPerSecond": 12300,
            "WriteUnitsPerSecond": 4500,
            "Status": "UPDATING"
        }
    }
```

```
}
```

AWS SDK

Die folgenden SDK-Beispiele zeigen Ihnen, wie Sie den Wert für den Warmdurchsatz Ihrer DynamoDB-Tabelle aktualisieren können.

Java

```
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.model.GlobalSecondaryIndexUpdate;
import
 software.amazon.awssdk.services.dynamodb.model.UpdateGlobalSecondaryIndexAction;
import software.amazon.awssdk.services.dynamodb.model.UpdateTableRequest;
import software.amazon.awssdk.services.dynamodb.model.WarmThroughput;

...
public static WarmThroughput buildWarmThroughput(final Long readUnitsPerSecond,
                                                 final Long writeUnitsPerSecond) {
    return WarmThroughput.builder()
            .readUnitsPerSecond(readUnitsPerSecond)
            .writeUnitsPerSecond(writeUnitsPerSecond)
            .build();
}

public static void updateDynamoDBTable(DynamoDbClient ddb,
                                       String tableName,
                                       Long tableReadUnitsPerSecond,
                                       Long tableWriteUnitsPerSecond,
                                       String globalSecondaryIndexName,
                                       Long globalSecondaryIndexReadUnitsPerSecond,
                                       Long globalSecondaryIndexWriteUnitsPerSecond)
 {

    final WarmThroughput tableWarmThroughput =
 buildWarmThroughput(tableReadUnitsPerSecond, tableWriteUnitsPerSecond);
    final WarmThroughput gsiWarmThroughput =
 buildWarmThroughput(globalSecondaryIndexReadUnitsPerSecond,
 globalSecondaryIndexWriteUnitsPerSecond);
```

```
    final GlobalSecondaryIndexUpdate globalSecondaryIndexUpdate =
 GlobalSecondaryIndexUpdate.builder()
            .update(UpdateGlobalSecondaryIndexAction.builder()
                    .indexName(globalSecondaryIndexName)
                    .warmThroughput(gsiWarmThroughput)
                    .build()
            ).build();

    final UpdateTableRequest request = UpdateTableRequest.builder()
            .tableName(tableName)
            .globalSecondaryIndexUpdates(globalSecondaryIndexUpdate)
            .warmThroughput(tableWarmThroughput)
            .build();

    try {
        ddb.updateTable(request);
    } catch (DynamoDbException e) {
        System.err.println(e.getMessage());
        System.exit(1);
    }

    System.out.println("Done!");
}
```

Python

```
from boto3 import resource
from botocore.exceptions import ClientError

def update_dynamodb_table_warm_throughput(table_name, table_read_units,
 table_write_units, gsi_name, gsi_read_units, gsi_write_units, region_name="us-
east-1"):
    """
    Updates the warm throughput of a DynamoDB table and a global secondary index.

    :param table_name: The name of the table to update.
    :param table_read_units: The new read units per second for the table's warm
 throughput.
    :param table_write_units: The new write units per second for the table's warm
 throughput.
    :param gsi_name: The name of the global secondary index to update.
    :param gsi_read_units: The new read units per second for the GSI's warm
 throughput.
```

```
    :param gsi_write_units: The new write units per second for the GSI's warm
throughput.
    :param region_name: The AWS Region name to target. defaults to us-east-1
    """
    try:
        ddb = resource('dynamodb', region_name)

        # Update the table's warm throughput
        table_warm_throughput = {
            "ReadUnitsPerSecond": table_read_units,
            "WriteUnitsPerSecond": table_write_units
        }

        # Update the global secondary index's warm throughput
        gsi_warm_throughput = {
            "ReadUnitsPerSecond": gsi_read_units,
            "WriteUnitsPerSecond": gsi_write_units
        }

        # Construct the global secondary index update
        global_secondary_index_update = [
            {
                "Update": {
                    "IndexName": gsi_name,
                    "WarmThroughput": gsi_warm_throughput
                }
            }
        ]

        # Construct the update table request
        update_table_request = {
            "TableName": table_name,
            "GlobalSecondaryIndexUpdates": global_secondary_index_update,
            "WarmThroughput": table_warm_throughput
        }

        # Update the table
        ddb.update_table(**update_table_request)
        print("Table updated successfully!")
    except ClientError as e:
        print(f"Error updating table: {e}")
        raise e
```

Javascript

```
import { DynamoDBClient, UpdateTableCommand } from "@aws-sdk/client-dynamodb";

async function updateDynamoDBTableWarmThroughput(
  tableName,
  tableReadUnits,
  tableWriteUnits,
  gsiName,
  gsiReadUnits,
  gsiWriteUnits,
  region = "us-east-1"
) {
  try {
    const ddbClient = new DynamoDBClient({ region: region });

    // Construct the update table request
    const updateTableRequest = {
      TableName: tableName,
      GlobalSecondaryIndexUpdates: [
        {
            Update: {
                IndexName: gsiName,
                WarmThroughput: {
                    ReadUnitsPerSecond: gsiReadUnits,
                    WriteUnitsPerSecond: gsiWriteUnits,
                },
            },
        },
      ],
      WarmThroughput: {
          ReadUnitsPerSecond: tableReadUnits,
          WriteUnitsPerSecond: tableWriteUnits,
      },
    };

    const command = new UpdateTableCommand(updateTableRequest);
    const response = await ddbClient.send(command);
    console.log(`Table updated successfully! Response: ${response}`);
  } catch (error) {
    console.error(`Error updating table: ${error}`);
    throw error;
  }
```

```
}
```

Erstellen einer neuen DynamoDB-Tabelle mit einem höheren Warmdurchsatz

Sie können die Werte für den Warmdurchsatz beim Erstellen Ihrer DynamoDB-Tabelle anpassen, indem Sie die folgenden Schritte ausführen. Diese Schritte gelten auch für die Erstellung einer globalen Tabelle oder eines sekundären Index.

AWS-Managementkonsole

So erstellen Sie eine DynamoDB-Tabelle und passen die Werte für den Warmdurchsatz über die Konsole an:

1. Melden Sie sich bei der an AWS-Managementkonsole und öffnen Sie die DynamoDB-Konsole unter. https://console.aws.amazon.com/dynamodb/
2. Wählen Sie Tabelle erstellen aus.
3. Wählen Sie einen Tabellennamen, einen Partitionsschlüssel und einen Sortierschlüssel (optional) aus.
4. Wählen Sie unter Tabelleneinstellungen die Option Einstellungen anpassen aus.
5. Wählen Sie im Feld Warmdurchsatz die Option Warmdurchsatz erhöhen aus.
6. Passen Sie die Leseeinheiten pro Sekunde und die Schreibeinheiten pro Sekunde an. Diese beiden Einstellungen definieren den maximalen Durchsatz, den die Tabelle sofort verarbeiten kann.
7. Fügen Sie weitere Tabellendetails hinzu und wählen Sie dann Tabelle erstellen aus.

AWS CLI

Das folgende AWS CLI Beispiel zeigt Ihnen, wie Sie eine DynamoDB-Tabelle mit benutzerdefinierten Warmdurchsatzwerten erstellen.

1. Führen Sie die `create-table`-Operation aus, um die folgende DynamoDB-Tabelle zu erstellen.

```
aws dynamodb create-table \
    --table-name GameScores \
    --attribute-definitions AttributeName=UserId,AttributeType=S \
                            AttributeName=GameTitle,AttributeType=S \
```

```
                              AttributeName=TopScore,AttributeType=N  \
    --key-schema AttributeName=UserId,KeyType=HASH \
                 AttributeName=GameTitle,KeyType=RANGE \
    --provisioned-throughput ReadCapacityUnits=20,WriteCapacityUnits=10 \
    --global-secondary-indexes \
        "[
            {
                \"IndexName\": \"GameTitleIndex\",
                \"KeySchema\": [{\"AttributeName\":\"GameTitle\",\"KeyType\":\"HASH
\"},
                                {\"AttributeName\":\"TopScore\",\"KeyType\":\"RANGE
\"}],
                \"Projection\":{
                    \"ProjectionType\":\"INCLUDE\",
                    \"NonKeyAttributes\":[\"UserId\"]
                },
                \"ProvisionedThroughput\": {
                    \"ReadCapacityUnits\": 50,
                    \"WriteCapacityUnits\": 25
                },\"WarmThroughput\": {
                    \"ReadUnitsPerSecond\": 1987,
                    \"WriteUnitsPerSecond\": 543
                }
            }
        ]" \
    --warm-throughput ReadUnitsPerSecond=12345,WriteUnitsPerSecond=4567 \
    --region us-east-1
```

2. Sie erhalten eine ähnliche Antwort wie die folgende. Ihre `WarmThroughput`-Einstellungen werden als `ReadUnitsPerSecond` und `WriteUnitsPerSecond` angezeigt. `Status` lautet `UPDATING`, wenn der Wert für den Warmdurchsatz aktualisiert wird, und `ACTIVE`, wenn der neue Warmdurchsatzwert festgelegt wird.

```
{
    "TableDescription": {
        "AttributeDefinitions": [
            {
                "AttributeName": "GameTitle",
                "AttributeType": "S"
            },
            {
                "AttributeName": "TopScore",
                "AttributeType": "N"
```

```
            },
            {
                "AttributeName": "UserId",
                "AttributeType": "S"
            }
        ],
        "TableName": "GameScores",
        "KeySchema": [
            {
                "AttributeName": "UserId",
                "KeyType": "HASH"
            },
            {
                "AttributeName": "GameTitle",
                "KeyType": "RANGE"
            }
        ],
        "TableStatus": "CREATING",
        "CreationDateTime": 1730241788.779,
        "ProvisionedThroughput": {
            "NumberOfDecreasesToday": 0,
            "ReadCapacityUnits": 20,
            "WriteCapacityUnits": 10
        },
        "TableSizeBytes": 0,
        "ItemCount": 0,
        "TableArn": "arn:aws:dynamodb:us-east-1:XXXXXXXXXXXX:table/GameScores",
        "TableId": "XXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXXX",
        "GlobalSecondaryIndexes": [
            {
                "IndexName": "GameTitleIndex",
                "KeySchema": [
                    {
                        "AttributeName": "GameTitle",
                        "KeyType": "HASH"
                    },
                    {
                        "AttributeName": "TopScore",
                        "KeyType": "RANGE"
                    }
                ],
                "Projection": {
                    "ProjectionType": "INCLUDE",
                    "NonKeyAttributes": [
```

```
                        "UserId"
                    ]
                },
                "IndexStatus": "CREATING",
                "ProvisionedThroughput": {
                    "NumberOfDecreasesToday": 0,
                    "ReadCapacityUnits": 50,
                    "WriteCapacityUnits": 25
                },
                "IndexSizeBytes": 0,
                "ItemCount": 0,
                "IndexArn": "arn:aws:dynamodb:us-east-1:XXXXXXXXXXXX:table/
GameScores/index/GameTitleIndex",
                "WarmThroughput": {
                    "ReadUnitsPerSecond": 1987,
                    "WriteUnitsPerSecond": 543,
                    "Status": "UPDATING"
                }
            }
        ],
        "DeletionProtectionEnabled": false,
        "WarmThroughput": {
            "ReadUnitsPerSecond": 12345,
            "WriteUnitsPerSecond": 4567,
            "Status": "UPDATING"
        }
    }
}
```

AWS SDK

Die folgenden SDK-Beispiele zeigen Ihnen, wie Sie eine DynamoDB-Tabelle mit benutzerdefinierten Werden für den Warmdurchsatz erstellen.

Java

```
import software.amazon.awscdk.services.dynamodb.ProjectionType;
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.CreateTableResponse;
import software.amazon.awssdk.services.dynamodb.model.CreateTableRequest;
import software.amazon.awssdk.services.dynamodb.model.KeySchemaElement;
import software.amazon.awssdk.services.dynamodb.model.KeyType;
```

```
import software.amazon.awssdk.services.dynamodb.model.ProvisionedThroughput;
import software.amazon.awssdk.services.dynamodb.model.Projection;
import software.amazon.awssdk.services.dynamodb.model.GlobalSecondaryIndex;
import software.amazon.awssdk.services.dynamodb.model.AttributeDefinition;
import software.amazon.awssdk.services.dynamodb.model.ScalarAttributeType;
import software.amazon.awssdk.services.dynamodb.model.WarmThroughput;
...

public static WarmThroughput buildWarmThroughput(final Long readUnitsPerSecond,
                                                 final Long writeUnitsPerSecond) {
    return WarmThroughput.builder()
            .readUnitsPerSecond(readUnitsPerSecond)
            .writeUnitsPerSecond(writeUnitsPerSecond)
            .build();
}
private static AttributeDefinition buildAttributeDefinition(final String
 attributeName,
                                                             final
 ScalarAttributeType scalarAttributeType) {
    return AttributeDefinition.builder()
            .attributeName(attributeName)
            .attributeType(scalarAttributeType)
            .build();
}
private static KeySchemaElement buildKeySchemaElement(final String attributeName,
                                                      final KeyType keyType) {
    return KeySchemaElement.builder()
            .attributeName(attributeName)
            .keyType(keyType)
            .build();
}
public static void createDynamoDBTable(DynamoDbClient ddb,
                                       String tableName,
                                       String partitionKey,
                                       String sortKey,
                                       String miscellaneousKeyAttribute,
                                       String nonKeyAttribute,
                                       Long tableReadCapacityUnits,
                                       Long tableWriteCapacityUnits,
                                       Long tableWarmReadUnitsPerSecond,
                                       Long tableWarmWriteUnitsPerSecond,
                                       String globalSecondaryIndexName,
                                       Long globalSecondaryIndexReadCapacityUnits,
                                       Long globalSecondaryIndexWriteCapacityUnits,
```

```
                                        Long
globalSecondaryIndexWarmReadUnitsPerSecond,
                                        Long
globalSecondaryIndexWarmWriteUnitsPerSecond) {

   // Define the table attributes
   final AttributeDefinition partitionKeyAttribute =
buildAttributeDefinition(partitionKey, ScalarAttributeType.S);
   final AttributeDefinition sortKeyAttribute = buildAttributeDefinition(sortKey,
ScalarAttributeType.S);
   final AttributeDefinition miscellaneousKeyAttributeDefinition =
buildAttributeDefinition(miscellaneousKeyAttribute, ScalarAttributeType.N);
   final AttributeDefinition[] attributeDefinitions = {partitionKeyAttribute,
sortKeyAttribute, miscellaneousKeyAttributeDefinition};

   // Define the table key schema
   final KeySchemaElement partitionKeyElement = buildKeySchemaElement(partitionKey,
KeyType.HASH);
   final KeySchemaElement sortKeyElement = buildKeySchemaElement(sortKey,
KeyType.RANGE);
   final KeySchemaElement[] keySchema = {partitionKeyElement, sortKeyElement};

   // Define the provisioned throughput for the table
   final ProvisionedThroughput provisionedThroughput =
ProvisionedThroughput.builder()
           .readCapacityUnits(tableReadCapacityUnits)
           .writeCapacityUnits(tableWriteCapacityUnits)
           .build();

   // Define the Global Secondary Index (GSI)
   final KeySchemaElement globalSecondaryIndexPartitionKeyElement =
buildKeySchemaElement(sortKey, KeyType.HASH);
   final KeySchemaElement globalSecondaryIndexSortKeyElement =
buildKeySchemaElement(miscellaneousKeyAttribute, KeyType.RANGE);
   final KeySchemaElement[] gsiKeySchema =
{globalSecondaryIndexPartitionKeyElement, globalSecondaryIndexSortKeyElement};

   final Projection gsiProjection = Projection.builder()
           .projectionType(String.valueOf(ProjectionType.INCLUDE))
           .nonKeyAttributes(nonKeyAttribute)
           .build();
   final ProvisionedThroughput gsiProvisionedThroughput =
ProvisionedThroughput.builder()
           .readCapacityUnits(globalSecondaryIndexReadCapacityUnits)
```

```
            .writeCapacityUnits(globalSecondaryIndexWriteCapacityUnits)
            .build();
    // Define the warm throughput for the Global Secondary Index (GSI)
    final WarmThroughput gsiWarmThroughput =
 buildWarmThroughput(globalSecondaryIndexWarmReadUnitsPerSecond,
 globalSecondaryIndexWarmWriteUnitsPerSecond);
    final GlobalSecondaryIndex globalSecondaryIndex = GlobalSecondaryIndex.builder()
            .indexName(globalSecondaryIndexName)
            .keySchema(gsiKeySchema)
            .projection(gsiProjection)
            .provisionedThroughput(gsiProvisionedThroughput)
            .warmThroughput(gsiWarmThroughput)
            .build();

    // Define the warm throughput for the table
    final WarmThroughput tableWarmThroughput =
 buildWarmThroughput(tableWarmReadUnitsPerSecond, tableWarmWriteUnitsPerSecond);

    final CreateTableRequest request = CreateTableRequest.builder()
            .tableName(tableName)
            .attributeDefinitions(attributeDefinitions)
            .keySchema(keySchema)
            .provisionedThroughput(provisionedThroughput)
            .globalSecondaryIndexes(globalSecondaryIndex)
            .warmThroughput(tableWarmThroughput)
            .build();

    CreateTableResponse response = ddb.createTable(request);
    System.out.println(response);
}
```

Python

```
from boto3 import resource
from botocore.exceptions import ClientError

def create_dynamodb_table_warm_throughput(table_name, partition_key,
 sort_key, misc_key_attr, non_key_attr, table_provisioned_read_units,
 table_provisioned_write_units, table_warm_reads, table_warm_writes, gsi_name,
 gsi_provisioned_read_units, gsi_provisioned_write_units, gsi_warm_reads,
 gsi_warm_writes, region_name="us-east-1"):
    """
    Creates a DynamoDB table with a warm throughput setting configured.
```

```
   :param table_name: The name of the table to be created.
   :param partition_key: The partition key for the table being created.
   :param sort_key: The sort key for the table being created.
   :param misc_key_attr: A miscellaneous key attribute for the table being created.
   :param non_key_attr: A non-key attribute for the table being created.
   :param table_provisioned_read_units: The newly created table's provisioned read
capacity units.
   :param table_provisioned_write_units: The newly created table's provisioned
write capacity units.
   :param table_warm_reads: The read units per second setting for the table's warm
throughput.
   :param table_warm_writes: The write units per second setting for the table's
warm throughput.
   :param gsi_name: The name of the Global Secondary Index (GSI) to be created on
the table.
   :param gsi_provisioned_read_units: The configured Global Secondary Index (GSI)
provisioned read capacity units.
   :param gsi_provisioned_write_units: The configured Global Secondary Index (GSI)
provisioned write capacity units.
   :param gsi_warm_reads: The read units per second setting for the Global
Secondary Index (GSI)'s warm throughput.
   :param gsi_warm_writes: The write units per second setting for the Global
Secondary Index (GSI)'s warm throughput.
   :param region_name: The AWS Region name to target. defaults to us-east-1
   """
   try:
       ddb = resource('dynamodb', region_name)

       # Define the table attributes
       attribute_definitions = [
           { "AttributeName": partition_key, "AttributeType": "S" },
           { "AttributeName": sort_key, "AttributeType": "S" },
           { "AttributeName": misc_key_attr, "AttributeType": "N" }
       ]

       # Define the table key schema
       key_schema = [
           { "AttributeName": partition_key, "KeyType": "HASH" },
           { "AttributeName": sort_key, "KeyType": "RANGE" }
       ]

       # Define the provisioned throughput for the table
       provisioned_throughput = {
```

```
            "ReadCapacityUnits": table_provisioned_read_units,
            "WriteCapacityUnits": table_provisioned_write_units
        }

        # Define the global secondary index
        gsi_key_schema = [
            { "AttributeName": sort_key, "KeyType": "HASH" },
            { "AttributeName": misc_key_attr, "KeyType": "RANGE" }
        ]
        gsi_projection = {
            "ProjectionType": "INCLUDE",
            "NonKeyAttributes": [non_key_attr]
        }
        gsi_provisioned_throughput = {
            "ReadCapacityUnits": gsi_provisioned_read_units,
            "WriteCapacityUnits": gsi_provisioned_write_units
        }
        gsi_warm_throughput = {
            "ReadUnitsPerSecond": gsi_warm_reads,
            "WriteUnitsPerSecond": gsi_warm_writes
        }
        global_secondary_indexes = [
            {
                "IndexName": gsi_name,
                "KeySchema": gsi_key_schema,
                "Projection": gsi_projection,
                "ProvisionedThroughput": gsi_provisioned_throughput,
                "WarmThroughput": gsi_warm_throughput
            }
        ]

        # Define the warm throughput for the table
        warm_throughput = {
            "ReadUnitsPerSecond": table_warm_reads,
            "WriteUnitsPerSecond": table_warm_writes
        }

        # Create the DynamoDB client and create the table
        response = ddb.create_table(
            TableName=table_name,
            AttributeDefinitions=attribute_definitions,
            KeySchema=key_schema,
            ProvisionedThroughput=provisioned_throughput,
            GlobalSecondaryIndexes=global_secondary_indexes,
```

```
            WarmThroughput=warm_throughput
        )

        print(response)
    except ClientError as e:
        print(f"Error creating table: {e}")
        raise e
```

Javascript

```
import { DynamoDBClient, CreateTableCommand } from "@aws-sdk/client-dynamodb";

async function createDynamoDBTableWithWarmThroughput(
  tableName,
  partitionKey,
  sortKey,
  miscKeyAttr,
  nonKeyAttr,
  tableProvisionedReadUnits,
  tableProvisionedWriteUnits,
  tableWarmReads,
  tableWarmWrites,
  indexName,
  indexProvisionedReadUnits,
  indexProvisionedWriteUnits,
  indexWarmReads,
  indexWarmWrites,
  region = "us-east-1"
) {
  try {
    const ddbClient = new DynamoDBClient({ region: region });
    const command = new CreateTableCommand({
      TableName: tableName,
      AttributeDefinitions: [
          { AttributeName: partitionKey, AttributeType: "S" },
          { AttributeName: sortKey, AttributeType: "S" },
          { AttributeName: miscKeyAttr, AttributeType: "N" },
      ],
      KeySchema: [
          { AttributeName: partitionKey, KeyType: "HASH" },
          { AttributeName: sortKey, KeyType: "RANGE" },
      ],
      ProvisionedThroughput: {
```

```
            ReadCapacityUnits: tableProvisionedReadUnits,
            WriteCapacityUnits: tableProvisionedWriteUnits,
        },
        WarmThroughput: {
            ReadUnitsPerSecond: tableWarmReads,
            WriteUnitsPerSecond: tableWarmWrites,
        },
        GlobalSecondaryIndexes: [
            {
              IndexName: indexName,
              KeySchema: [
                  { AttributeName: sortKey, KeyType: "HASH" },
                  { AttributeName: miscKeyAttr, KeyType: "RANGE" },
              ],
              Projection: {
                  ProjectionType: "INCLUDE",
                  NonKeyAttributes: [nonKeyAttr],
              },
              ProvisionedThroughput: {
                  ReadCapacityUnits: indexProvisionedReadUnits,
                  WriteCapacityUnits: indexProvisionedWriteUnits,
              },
              WarmThroughput: {
                  ReadUnitsPerSecond: indexWarmReads,
                  WriteUnitsPerSecond: indexWarmWrites,
              },
            },
        ],
      });
      const response = await ddbClient.send(command);
      console.log(response);
    } catch (error) {
      console.error(`Error creating table: ${error}`);
      throw error;
    }
}
```

Grundlegendes zum DynamoDB-Warmdurchsatz in verschiedenen Szenarien

Im Folgenden finden Sie verschiedene Szenarien, die Ihnen bei der Arbeit mit DynamoDB-Warmdurchsatz begegnen können.

Themen

- Warmdurchsatz und ungleiche Zugriffsmuster
- Warmdurchsatz für eine bereitgestellte Tabelle
- Warmdurchsatz für eine On-Demand-Tabelle
- Warmdurchsatz für eine On-Demand-Tabelle mit konfiguriertem maximalem Durchsatz

Warmdurchsatz und ungleiche Zugriffsmuster

Auch wenn eine Tabelle einen Warmdurchsatz von 30 000 Leseeinheiten pro Sekunde und 10 000 Schreibeinheiten pro Sekunde hat, kann es dennoch zu Drosselungen bei Lese- oder Schreibvorgängen kommen, bevor diese Werte erreicht werden. Das liegt wahrscheinlich an einer Hot-Partition. DynamoDB kann zwar weiter skaliert werden, um einen praktisch unbegrenzten Durchsatz zu unterstützen, aber jede einzelne Partition ist auf 1 000 Schreibeinheiten pro Sekunde und 3 000 Leseeinheiten pro Sekunde begrenzt. Wenn Ihre Anwendung zu viel Datenverkehr auf einen kleinen Teil der Tabellenpartitionen lenkt, kann es zu einer Drosselung kommen, noch bevor Sie die Werte für den Warmdurchsatz der Tabelle erreichen. Wir empfehlen, die bewährten Methoden für DynamoDB zu befolgen, um eine nahtlose Skalierbarkeit zu gewährleisten und Hot-Partitionen zu vermeiden.

Warmdurchsatz für eine bereitgestellte Tabelle

Stellen Sie sich eine bereitgestellte Tabelle mit einem Warmdurchsatz von 30 000 Leseeinheiten pro Sekunde und 10 000 Schreibeinheiten pro Sekunde vor, die derzeit jedoch über einen bereitgestellten Durchsatz von 4 000 RCU und 8 000 WCU verfügt. Sie können den bereitgestellten Durchsatz der Tabelle sofort auf bis zu 30 000 RCU oder 10 000 WCU skalieren, indem Sie Ihre Einstellungen für den bereitgestellten Durchsatz aktualisieren. Wenn Sie den bereitgestellten Durchsatz über diese Werte hinaus erhöhen, passt sich der Warmdurchsatz automatisch an die neuen höheren Werte an, da Sie einen neuen Spitzendurchsatz festgelegt haben. Wenn Sie beispielsweise den bereitgestellten Durchsatz auf 50 000 RCU festlegen, erhöht sich der Warmdurchsatz auf 50 000 Leseeinheiten pro Sekunde.

```
"ProvisionedThroughput":
    {
        "ReadCapacityUnits": 4000,
        "WriteCapacityUnits": 8000
    }
"WarmThroughput":
    {
        "ReadUnitsPerSecond": 30000,
        "WriteUnitsPerSecond": 10000
    }
```

Warmdurchsatz für eine On-Demand-Tabelle

Eine neue On-Demand-Tabelle beginnt mit einem Warmdurchsatz von 12 000 Leseeinheiten pro Sekunde und 4 000 Schreibeinheiten pro Sekunde. Ihre Tabelle kann anhaltenden Datenverkehr bis zu diesen Werten sofort verarbeiten. Wenn Ihre Anforderungen 12 000 Leseeinheiten pro Sekunde oder 4 000 Schreibeinheiten pro Sekunde überschreiten, passt sich der Warmdurchsatz automatisch an höhere Werte an.

```
"WarmThroughput":
    {
        "ReadUnitsPerSecond": 12000,
        "WriteUnitsPerSecond": 4000
    }
```

Warmdurchsatz für eine On-Demand-Tabelle mit konfiguriertem maximalem Durchsatz

Stellen Sie sich eine On-Demand-Tabelle mit einem Warmdurchsatz von 30 000 Leseeinheiten pro Sekunde vor, für die der maximale Durchsatz jedoch auf 5 000 Leseanforderungseinheiten (RRU) konfiguriert ist. In diesem Szenario wird der Durchsatz der Tabelle auf das von Ihnen festgelegte Maximum von 5 000 RRU begrenzt. Alle Durchsatzanforderungen, die diesen Höchstwert überschreiten, werden gedrosselt. Sie können den tabellenspezifischen maximalen Durchsatz jedoch jederzeit an die Anforderungen Ihrer Anwendung anpassen.

```
"OnDemandThroughput":
    {
        "MaxReadRequestUnits": 5000,
        "MaxWriteRequestUnits": 4000
    }
"WarmThroughput":
```

```
    {
        "ReadUnitsPerSecond": 30000,
        "WriteUnitsPerSecond": 10000
    }
```

DynamoDB – Burst- und adaptive Kapazität

Um die Drosselung aufgrund von Durchsatzausnahmen zu minimieren, verwendet DynamoDB die Burst-Kapazität, um Nutzungsspitzen zu bewältigen. DynamoDB nutzt die adaptive Kapazität, um ungleichen Zugriffsmustern gerecht zu werden.

Burst-Kapazität

DynamoDB bietet eine gewisse Flexibilität bei der Bereitstellung des Durchsatzes pro Partition durch Bereitstellung von Burst-Kapazität. Wenn Sie den verfügbaren Durchsatz nicht vollständig nutzen, reserviert DynamoDB einen Teil dieser ungenutzten Kapazität für spätere Durchsatz-Bursts, um Nutzungsspitzen zu bewältigen. Mit Burst-Kapazität können unerwartete Lese- oder Schreibanforderungen erfolgreich sein, wo sie andernfalls gedrosselt werden würden.

DynamoDB behält derzeit bis zu fünf Minuten (300 Sekunden) ungenutzte Lese- und Schreibkapazität bei. Während eines gelegentlichen Bursts von Lese- oder Schreibaktivitäten können diese zusätzlichen Kapazitätseinheiten schnell verbraucht werden – sogar schneller als die pro Sekunde bereitgestellte Durchsatzkapazität, die Sie für Ihre Tabelle definiert haben.

Zudem kann DynamoDB die Burst-Kapazität für Wartungsaktivitäten im Hintergrund sowie für andere Aufgaben verbrauchen, ohne Sie vorher zu informieren.

Beachten Sie, dass die Burst-Kapazität in Zukunft geändert werden kann.

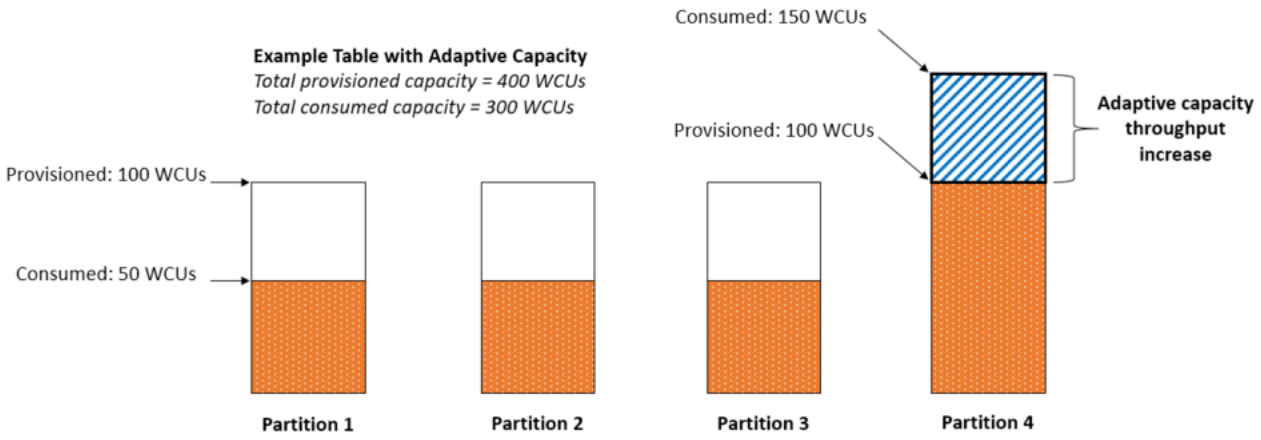
Adaptive Kapazität

DynamoDB verteilt Ihre Daten automatisch auf alle Partitionen, die auf mehreren Servern in der AWS Cloud gespeichert werden. Es ist nicht immer möglich, die Lese- und Schreibaktivität jederzeit gleichmäßig zu verteilen. Wenn der Datenzugriff nicht ausgeglichen ist, kann es sein, dass eine „Hot Partition” mehr Lese- und Schreibdatenvolumen erhält als andere Partitionen. Da Lese- und Schreibvorgänge auf einer Partition unabhängig verwaltet werden, tritt eine Drosselung auf, wenn eine einzelne Partition mehr als 3 000 Lesevorgänge oder mehr als 1 000 Schreibvorgänge empfängt. Adaptive Kapazität funktioniert durch automatische Erhöhung der Durchsatzkapazität für Partitionen, die mehr Datenverkehr erhalten.

Um ungleichmäßigen Zugriffsmustern besser gerecht zu werden, ermöglicht die adaptive Kapazität von DynamoDB Ihrer Anwendung, weiterhin zu lesen und auf Hot-Partitionen zu schreiben, ohne gedrosselt zu werden, vorausgesetzt, dass der Datenverkehr die gesamte bereitgestellte Kapazität Ihrer Tabelle oder die maximale Partitionskapazität nicht überschreitet. Die adaptive Kapazität funktioniert durch automatische und sofortige Erhöhung der Durchsatzkapazität für Partitionen, die mehr Datenverkehr erhalten.

Das folgende Diagramm veranschaulicht, wie adaptive Kapazität funktioniert. Die Beispieltabelle enthält 400 Partitionen, die WCUs gleichmäßig auf vier Partitionen verteilt werden, sodass jede Partition bis zu 100 Partitionen WCUs pro Sekunde verwalten kann. Die Partitionen 1, 2 und 3 erhalten jeweils einen Schreibverkehr von 50. WCU/sec. Partition 4 receives 150 WCU/sec. This hot partition can accept write traffic while it still has unused burst capacity, but eventually it throttles traffic that exceeds 100 WCU/sec

DynamoDB Adaptive Capacity reagiert darauf, indem die Kapazität von Partition 4 erhöht wird, sodass sie die höhere Arbeitslast von 150 bewältigen kann, WCU/sec ohne gedrosselt zu werden.



Die adaptive Kapazität wird für jede DynamoDB-Tabelle ohne zusätzliche Kosten automatisch aktiviert. Die adaptive Kapazität muss nicht ausdrücklich aktiviert oder deaktiviert werden.

Isolieren von Elementen mit häufigen Zugriffen

Wenn eine Anwendung unverhältnismäßig viel Datenverkehr an einzelne oder mehrere Elemente leitet, gleicht die Funktion für adaptive Kapazität die Partitionen so aus, dass häufig aufgerufene Elemente nicht in derselben Partition abgelegt werden. Durch das Isolieren von Elementen, auf die häufig zugegriffen wird, wird die Wahrscheinlichkeit einer Anforderungsdrosselung aufgrund eines Workloads, der das Durchsatzkontingent für eine einzelne Partition überschreitet, reduziert. Sie können eine Sammlung von Elementen auch nach Sortierschlüssel in Segmente aufteilen, sofern

es sich bei der Sammlung nicht um Datenverkehr handelt, der durch eine monotone Erhöhung oder Verringerung des Sortierschlüssels verfolgt wird.

Wenn die Anwendung dauernd Datenverkehr in großem Umfang an ein einzelnes Element leitet, gleicht die Funktion für die adaptive Kapazität die Daten so aus, dass eine Partition nur ein einzelnes Element enthält, auf das häufig zugegriffen wird. In diesem Fall kann DynamoDB einen Durchsatz bis zum Partitionsmaximum von 3.000 RCUs und 1.000 für WCUs den Primärschlüssel dieses einzelnen Elements bereitstellen. Die adaptive Kapazität teilt Elementsammlungen nicht auf mehrere Partitionen der Tabelle auf, wenn ein lokaler sekundärer Index in der Tabelle enthalten ist.

Überlegungen beim Umstellen der Kapazitätsmodi in DynamoDB

Beim Erstellen einer DynamoDB-Tabelle müssen Sie entweder den On-Demand-Kapazitätsmodus oder den Modus bereitgestellter Kapazität auswählen.

Sie können Tabellen innerhalb eines fortlaufenden 24-stündigen Zeitfensters bis zu viermal vom Modus mit bereitgestellter Kapazität auf den On-Demand-Modus umstellen. Sie können Tabellen jederzeit vom On-Demand-Modus auf den Modus mit bereitgestellter Kapazität umstellen.

Themen

- Umstellen vom Modus bereitgestellter Kapazität auf den On-Demand-Kapazitätsmodus
- Umstellen vom On-Demand-Kapazitätsmodus auf den Modus bereitgestellter Kapazität

Umstellen vom Modus bereitgestellter Kapazität auf den On-Demand-Kapazitätsmodus

Im Modus bereitgestellter Kapazität legen Sie die Lese- und Schreibkapazität basierend auf den erwarteten Anwendungsanforderungen fest. Wenn Sie eine Tabelle vom Modus bereitgestellter Kapazität auf den On-Demand-Modus aktualisieren, brauchen Sie nicht anzugeben, wie viel Lese- und Schreibdurchsatz Ihre Anwendung erwartungsgemäß durchführen wird. DynamoDB On-Demand bietet eine einfache pay-per-request Preisgestaltung für Lese- und Schreibanforderungen, sodass Sie nur für das bezahlen, was Sie tatsächlich nutzen, sodass Kosten und Leistung leicht in Einklang gebracht werden können. Optional können Sie auch den maximalen Lese- oder Schreibdurchsatz (oder beides) für einzelne On-Demand-Tabellen und zugehörige globale sekundäre Indizes konfigurieren, um Kosten und Nutzung zu begrenzen. Weitere Informationen zum Einstellen des maximalen Durchsatzes für eine bestimmte Tabelle oder einen bestimmten Index finden Sie unter Maximaler DynamoDB-Durchsatz für On-Demand-Tabellen.

Wenn Sie vom Modus bereitgestellter Kapazität auf On-Demand-Kapazitätsmodus umstellen, nimmt DynamoDB mehrere Änderungen an der Struktur der Tabelle und den Partitionen vor. Dieser Vorgang kann einige Minuten dauern. Während der Wechsel vollzogen wird, ist der von der Tabelle gelieferte Durchsatz mit der zuvor bereitgestellten Menge an Schreibkapazitätseinheiten und Lesekapazitätseinheiten konsistent.

Anfänglicher Durchsatz für den On-Demand-Kapazitätsmodus

Wenn Sie vor Kurzem bei einer vorhandenen Tabelle erstmals zum On-Demand-Kapazitätsmodus gewechselt sind, weist die Tabelle den folgenden vorherigen eingestellten Höchststand auf, obwohl für die Tabelle bisher noch keinen Datenverkehr im On-Demand-Kapazitätsmodus bedient hat.

Im Folgenden finden Sie Beispiele für mögliche Szenarien:

- Jede bereitgestellte Tabelle mit einer Konfiguration unter 4 000 WCU und 12 000 RCU, für die noch nie zuvor mehr bereitgestellt wurde. Wenn Sie diese Tabelle zum ersten Mal auf On-Demand-Modus umstellen, stellt DynamoDB sicher, dass sie so skaliert wird, dass sie sofort mindestens 4.000 Schreib units/sec - und 12.000 Leseeinheiten pro Sekunde unterstützt.
- Eine bereitgestellte Tabelle, die als 8.000 WCU und 24.000 RCU konfiguriert ist. Wenn Sie diese Tabelle auf On-Demand-Modus umstellen, kann sie weiterhin jederzeit mindestens 8.000 Schreibvorgänge units/sec und 24.000 Lesevorgänge verarbeiten. units/sec
- Eine bereitgestellte Tabelle, konfiguriert mit 8.000 WCU und 24.000 RCU, die über einen längeren Zeitraum 6.000 Schreibvorgänge units/sec und 18.000 Lesevorgänge beanspruchte. units/sec Wenn Sie diese Tabelle auf On-Demand-Modus umstellen, kann sie weiterhin mindestens 8.000 Schreib- und 24.000 Leseeinheiten pro Sekunde verarbeiten. units/sec Der vorherige Datenverkehr kann es der Tabelle außerdem ermöglichen, ein viel höheres Datenverkehrsaufkommen ohne Drosselung zu unterstützen.
- Eine Tabelle, die zuvor mit 10 000 WCU und 10 000 RCU bereitgestellt wurde, derzeit jedoch mit 10 RCU und 10 WCU bereitgestellt wird. Wenn Sie diese Tabelle auf On-Demand-Tabelle umstellen, kann sie mindestens 10.000 Schreib units/sec - und 10.000 Leseeinheiten pro Sekunde verarbeiten.

Auto-Scaling-Einstellungen

Wenn Sie eine Tabelle vom Modus bereitgestellter Kapazität auf den On-Demand-Modus aktualisieren:

- Wenn Sie die Konsole verwenden, werden alle Ihre Auto Scaling-Einstellungen (sofern vorhanden) gelöscht.
- Wenn Sie das AWS SDK AWS CLI oder verwenden, werden alle Ihre Auto Scaling-Einstellungen beibehalten. Diese Einstellungen können übernommen werden, wenn Sie Ihre Tabelle wieder auf den Fakturierungsmodus bereitgestellter Kapazität aktualisieren.

Massenbearbeitung des Kapazitätsmodus in der DynamoDB-Konsole

Mithilfe der DynamoDB-Konsole können Sie mehrere Tabellen per Massenbearbeitung vom Modus bereitgestellter Kapazität auf den On-Demand-Kapazitätsmodus umstellen. So stellen Sie den Kapazitätsmodus per Massenbearbeitung um:

1. Öffnen Sie die Seite Tabellen in der DynamoDB-Konsole.
2. Aktivieren Sie die Kontrollkästchen für die Tabellen, die Sie in den On-Demand-Kapazitätsmodus aktualisieren möchten.
3. Wählen Sie Aktionen und dann Auf On-Demand-Kapazitätsmodus aktualisieren aus.

Mit diesem Massenvorgang können Sie mehrere Tabellen effizient in den On-Demand-Kapazitätsmodus umstellen, ohne jede Tabelle einzeln aktualisieren zu müssen.

Umstellen vom On-Demand-Kapazitätsmodus auf den Modus bereitgestellter Kapazität

Wenn vom On-Demand-Kapazitätsmodus zum Modus bereitgestellter Kapazität zurückgewechselt wird, ist der von Ihrer Tabelle gebotene Durchsatz mit dem zuvor erreichten Höchststand konsistent, als für die Tabelle der On-Demand-Modus eingestellt war.

Verwalten der Kapazität

Berücksichtigen Sie beim Aktualisieren einer Tabelle vom On-Demand-Modus auf den Modus bereitgestellter Kapazität Folgendes:

- Wenn Sie das AWS SDK AWS CLI oder verwenden, wählen Sie die richtigen Einstellungen für die bereitgestellte Kapazität Ihrer Tabelle und der globalen Sekundärindizes aus, indem Sie Amazon verwenden, CloudWatch um Ihren historischen Verbrauch (`ConsumedWriteCapacityUnits`und Ihre `ConsumedReadCapacityUnits` Kennzahlen) zu überprüfen, um die neuen Durchsatzeinstellungen zu ermitteln.

> **Note**
>
> Wenn Sie eine globale Tabelle in den Modus mit bereitgestellter Kapazität versetzen, zeigen Sie den maximalen Verbrauch für Basistabellen und globale sekundäre Indizes über alle regionalen Replikate hinweg an, wenn Sie die neuen Durchsatzeinstellungen bestimmen.

- Wenn Sie vom On-Demand-Modus zurück auf den Modus bereitgestellter Kapazität umstellen, achten Sie darauf, dass die anfänglich bereitgestellten Einheiten hoch genug sind, um Ihre Tabellen- oder Indexkapazität während des Übergangs bewältigen zu können.

Verwalten von Auto Scaling

Wenn Sie eine Tabelle vom On-Demand-Modus auf den Modus bereitgestellter Kapazität aktualisieren:

- Wenn Sie die Konsole verwenden, empfehlen wir, Auto Scaling mit den folgenden Standardeinstellungen zu aktivieren:
  - Zielauslastung: 70 %
  - Minimal bereitgestellte Kapazität: 5 Einheiten
  - Maximal bereitgestellte Kapazität: Der Höchstwert der Region
- Wenn Sie das SDK AWS CLI oder verwenden, werden Ihre vorherigen Auto Scaling-Einstellungen (falls vorhanden) beibehalten.

Programmieren mit DynamoDB und AWS SDKs

In diesem Abschnitt werden entwicklerbezogene Themen behandelt. Wenn Sie stattdessen Codebeispiele ausführen möchten, lesen Sie Ausführen der Codebeispiele in diesem Entwicklerhandbuch.

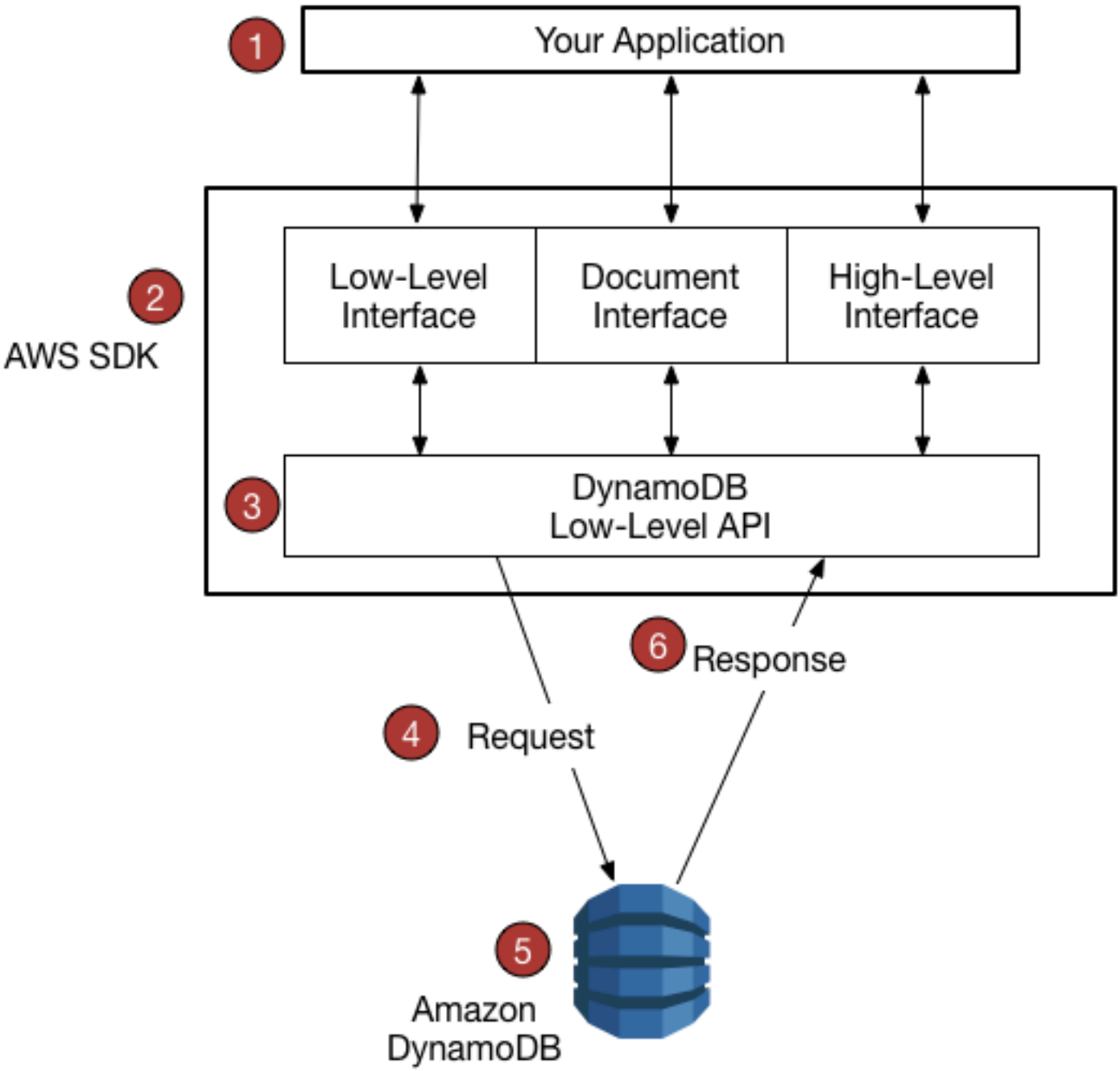
> **Note**
>
> Im Dezember 2017 AWS begann der Prozess der Migration aller Amazon DynamoDB DynamoDB-Endpunkte zur Verwendung von sicheren Zertifikaten, die von Amazon Trust Services (ATS) ausgestellt wurden. Weitere Informationen finden Sie unter Behebung von Problemen SSL/TLS beim Verbindungsaufbau mit DynamoDB.

Themen

- Überblick über die AWS SDK-Unterstützung für DynamoDB
- Programmieren von Amazon DynamoDB mit Python und Boto3
- Programmieren von Amazon DynamoDB mit JavaScript
- Programmieren von DynamoDB mit dem AWS SDK for Java 2.x
- Fehlerbehandlung mit DynamoDB
- DynamoDB mit einem SDK verwenden AWS

Überblick über die AWS SDK-Unterstützung für DynamoDB

Das folgende Diagramm bietet einen allgemeinen Überblick über die Amazon DynamoDB DynamoDB-Anwendungsprogrammierung mit dem. AWS SDKs



1. Sie schreiben eine Anwendung mit einem AWS SDK für Ihre Programmiersprache.
2. Jedes AWS SDK bietet eine oder mehrere programmatische Schnittstellen für die Arbeit mit DynamoDB. Welche spezifischen Schnittstellen verfügbar sind, hängt davon ab, welche Programmiersprache und welches AWS SDK Sie verwenden. Zu den Optionen gehören:
   - Low-Level-Schnittstellen, die mit DynamoDB funktionieren
   - Dokumentschnittstellen, die mit DynamoDB funktionieren
   - Schnittstellen für Objektpersistenz, die mit DynamoDB funktionieren

- High-Level-Schnittstellen

3. Das AWS SDK erstellt HTTP (S) -Anfragen zur Verwendung mit der Low-Level-DynamoDB-API.
4. Das AWS SDK sendet die Anfrage an den DynamoDB-Endpunkt.
5. DynamoDB führt die Anforderung aus. Wenn die Anforderung erfolgreich ist, gibt DynamoDB einen HTTP-200-Antwortcode zurück (OK). Wenn die Anforderung fehlschlägt, gibt DynamoDB einen HTTP-Fehlercode und eine Fehlermeldung zurück.
6. Das AWS SDK verarbeitet die Antwort und leitet sie zurück an Ihre Anwendung.

Jedes von ihnen AWS SDKs stellt wichtige Dienste für Ihre Anwendung bereit, darunter die folgenden:

- Formatieren von HTTP(S)-Anforderungen und Serialisieren von Anforderungsparametern
- Erstellen einer kryptografischen Signatur für jede Anforderung
- Weiterleiten von Anforderungen an einen DynamoDB-Endpunkt und Empfangen von Antworten von DynamoDB.
- Extrahieren der Ergebnisse dieser Antworten
- Implementieren einer grundlegenden Retry-Logik im Falle von Fehlern

Sie müssen für keine dieser Aufgaben einen Code schreiben.

> **Note**
>
> Weitere Informationen AWS SDKs, einschließlich Installationsanweisungen und Dokumentation, finden Sie unter Tools for Amazon Web Services.

SDK-Unterstützung für AWS kontobasierte Endgeräte

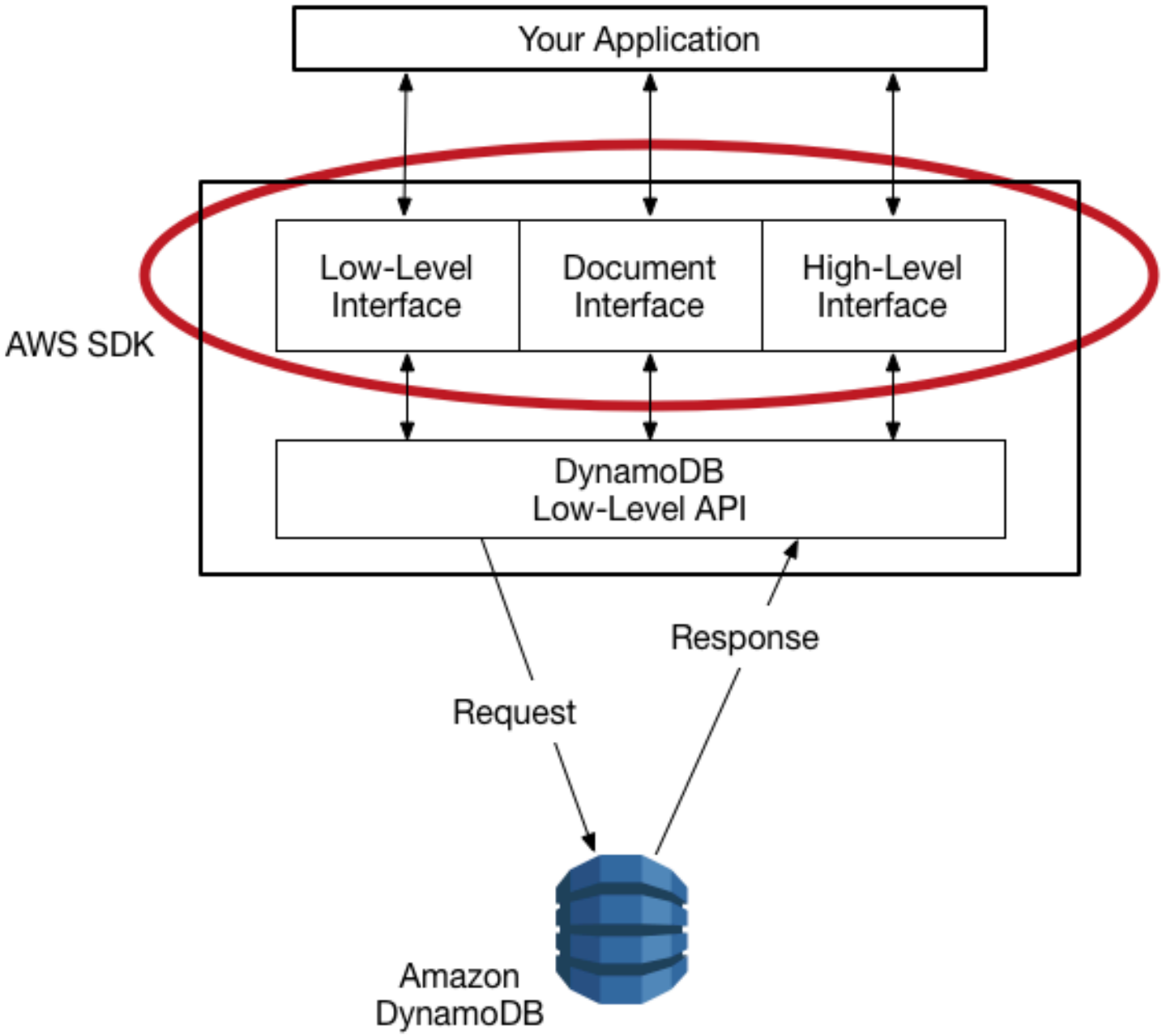
AWS führt die SDK-Unterstützung für AWS-account-basierte Endpunkte für DynamoDB ein, beginnend mit dem AWS SDK for Java V1 am 4. September 2024. Diese neuen Endpunkte tragen dazu bei AWS , eine hohe Leistung und Skalierbarkeit sicherzustellen. Die aktualisierte SDKs Version verwendet automatisch die neuen Endpunkte, die das Format haben. `https://(account-id).ddb.(region).amazonaws.com`

Wenn Sie eine einzelne Instance eines SDK-Clients verwenden, um Anfragen an mehrere Konten zu stellen, hat Ihre Anwendung weniger Möglichkeiten, Verbindungen wiederzuverwenden.

AWS empfiehlt, Ihre Anwendung so anzupassen, dass pro SDK-Client-Instance weniger Konten angesprochen werden. Eine Alternative besteht darin, Ihren SDK-Client mithilfe dieser `ACCOUNT_ID_ENDPOINT_MODE` Einstellung so einzurichten, dass er weiterhin regionale Endpunkte verwendet, wie im Referenzhandbuch AWS SDKsund im Tools-Referenzhandbuch dokumentiert.

Programmierschnittstellen, die mit DynamoDB funktionieren

Jedes AWS SDK bietet eine oder mehrere programmgesteuerte Schnittstellen für die Arbeit mit Amazon DynamoDB. Diese Schnittstellen reichen von einfachen Low-Level-DynamoDB-Wrapper zu objektorientierten Persistenzschichten. Die verfügbaren Schnittstellen variieren je nach AWS SDK und verwendeter Programmiersprache.



Der folgende Abschnitt hebt einige der verfügbaren Schnittstellen hervor und verwendet AWS SDK für Java als Beispiel. (Nicht alle Schnittstellen sind in allen verfügbar.) AWS SDKs

Themen

- Low-Level-Schnittstellen, die mit DynamoDB funktionieren
- Dokumentschnittstellen, die mit DynamoDB funktionieren
- Schnittstellen für Objektpersistenz, die mit DynamoDB funktionieren

Low-Level-Schnittstellen, die mit DynamoDB funktionieren

Jedes sprachspezifische AWS SDK bietet eine Low-Level-Schnittstelle für Amazon DynamoDB mit Methoden, die DynamoDB-API-Anfragen auf niedriger Ebene sehr ähnlich sind.

In einigen Fällen müssen Sie die Datentypen der Attribute mithilfe von Datentypbeschreibungen identifizieren, so wie `S` für Zeichenfolge und `N` für Zahl.

> **Note**
>
> Ein Low-Level-Schnittstelle ist in jeder sprachspezifischen AWS -SDK verfügbar.

Das folgende Java-Programm verwendet die Low-Level-Schnittstelle des AWS SDK für Java.

Beispiel einer Low-Level-Schnittstelle

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.AttributeValue;
import software.amazon.awssdk.services.dynamodb.model.GetItemRequest;
import java.util.HashMap;
import java.util.Map;
import java.util.Set;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-started.html
 *
 * To get an item from an Amazon DynamoDB table using the AWS SDK for Java V2,
 * its better practice to use the
 * Enhanced Client, see the EnhancedGetItem example.
 */
public class GetItem {
    public static void main(String[] args) {
        final String usage = """

                Usage:
```

```
                    <tableName> <key> <keyVal>

                Where:
                    tableName - The Amazon DynamoDB table from which an item is
retrieved (for example, Music3).\s
                    key - The key used in the Amazon DynamoDB table (for example,
Artist).\s
                    keyval - The key value that represents the item to get (for
example, Famous Band).
                """;

        if (args.length != 3) {
            System.out.println(usage);
            System.exit(1);
        }

        String tableName = args[0];
        String key = args[1];
        String keyVal = args[2];
        System.out.format("Retrieving item \"%s\" from \"%s\"\n", keyVal, tableName);
        Region region = Region.US_EAST_1;
        DynamoDbClient ddb = DynamoDbClient.builder()
                .region(region)
                .build();

        getDynamoDBItem(ddb, tableName, key, keyVal);
        ddb.close();
    }

    public static void getDynamoDBItem(DynamoDbClient ddb, String tableName, String
key, String keyVal) {
        HashMap<String, AttributeValue> keyToGet = new HashMap<>();
        keyToGet.put(key, AttributeValue.builder()
                .s(keyVal)
                .build());

        GetItemRequest request = GetItemRequest.builder()
                .key(keyToGet)
                .tableName(tableName)
                .build();

        try {
            // If there is no matching item, GetItem does not return any data.
            Map<String, AttributeValue> returnedItem = ddb.getItem(request).item();
```

```
            if (returnedItem.isEmpty())
                System.out.format("No item found with the key %s!\n", key);
            else {
                Set<String> keys = returnedItem.keySet();
                System.out.println("Amazon DynamoDB table attributes: \n");
                for (String key1 : keys) {
                    System.out.format("%s: %s\n", key1,
 returnedItem.get(key1).toString());
                }
            }

        } catch (DynamoDbException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
    }
}
```

Dokumentschnittstellen, die mit DynamoDB funktionieren

Viele AWS SDKs bieten eine Dokumentschnittstelle, über die Sie Operationen auf Datenebene (Erstellen, Lesen, Aktualisieren, Löschen) für Tabellen und Indizes ausführen können. Mit einer Dokumentschnittstelle müssen Sie Datentypbeschreibungen nicht angeben. Die Datentypen werden durch die Semantiken der Daten selber bereitgestellt. Diese bieten AWS SDKs auch Methoden zur einfachen Konvertierung von JSON-Dokumenten in und aus nativen Amazon DynamoDB DynamoDB-Datentypen.

> **Note**
>
> Dokumentschnittstellen sind im AWS SDKs für Java, .NET, Node.js und JavaScript SDK verfügbar.

Das folgende Java-Programm verwendet die Dokumentschnittstelle des AWS SDK für Java. Das Programm erstellt ein `Table`-Objekt, das die `Music`-Tabelle repräsentiert und fordert dieses Objekt auf, `GetItem` zu verwenden, um einen Song abzurufen. Das Programm gibt dann das Jahr aus, in dem der Song veröffentlicht wurde.

Die `software.amazon.dynamodb.document.DynamoDB`-Klasse implementiert die DynamoDB-Dokumentschnittstelle. Beachten Sie, wie `DynamoDB` als Wrapper um den Low-Level-Client agiert (`AmazonDynamoDB`).

Beispiel für eine Dokumentschnittstelle

```
package com.amazonaws.codesamples.gsg;

import software.amazon.dynamodb.AmazonDynamoDB;
import software.amazon.dynamodb.AmazonDynamoDBClientBuilder;
import software.amazon.dynamodb.document.DynamoDB;
import software.amazon.dynamodb.document.GetItemOutcome;
import software.amazon.dynamodb.document.Table;

public class MusicDocumentDemo {

    public static void main(String[] args) {

        AmazonDynamoDB client = AmazonDynamoDBClientBuilder.standard().build();
        DynamoDB docClient = new DynamoDB(client);

        Table table = docClient.getTable("Music");
        GetItemOutcome outcome = table.getItemOutcome(
                "Artist", "No One You Know",
                "SongTitle", "Call Me Today");

        int year = outcome.getItem().getInt("Year");
        System.out.println("The song was released in " + year);

    }
}
```

Schnittstellen für Objektpersistenz, die mit DynamoDB funktionieren

Einige AWS SDKs bieten eine Schnittstelle zur Objektpersistenz, über die Sie keine direkten Operationen auf der Datenebene ausführen können. Stattdessen erstellen Sie Objekte, die Elemente in Amazon-DynamoDB-Tabellen und Indexe repräsentieren, und interagieren ausschließlich mit diesen Objekten. Auf diese Weise können Sie objektorientierte Codes statt datenbankorientierter Codes schreiben.

> **Note**
>
> Schnittstellen AWS SDKs für Objektpersistenz sind in Java und .NET verfügbar. Weitere Informationen finden Sie unter Higher-Level-Programmierschnittstellen für DynamoDB für DynamoDB.

Beispiel einer Objektpersistenzschnittstelle

```
import com.example.dynamodb.Customer;
import software.amazon.awssdk.enhanced.dynamodb.DynamoDbEnhancedClient;
import software.amazon.awssdk.enhanced.dynamodb.DynamoDbTable;
import software.amazon.awssdk.enhanced.dynamodb.Key;
import software.amazon.awssdk.enhanced.dynamodb.TableSchema;
import software.amazon.awssdk.enhanced.dynamodb.model.GetItemEnhancedRequest;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
```

```
import com.example.dynamodb.Customer;
import software.amazon.awssdk.enhanced.dynamodb.DynamoDbEnhancedClient;
import software.amazon.awssdk.enhanced.dynamodb.DynamoDbTable;
import software.amazon.awssdk.enhanced.dynamodb.Key;
import software.amazon.awssdk.enhanced.dynamodb.TableSchema;
import software.amazon.awssdk.enhanced.dynamodb.model.GetItemEnhancedRequest;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;

/*
 * Before running this code example, create an Amazon DynamoDB table named Customer
 with these columns:
 *   - id - the id of the record that is the key. Be sure one of the id values is
 `id101`
 *   - custName - the customer name
 *   - email - the email value
 *   - registrationDate - an instant value when the item was added to the table. These
 values
 *                        need to be in the form of `YYYY-MM-DDTHH:mm:ssZ`, such as
 2022-07-11T00:00:00Z
 *
 * Also, ensure that you have set up your development environment, including your
 credentials.
 *
 * For information, see this documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-started.html
 */

public class EnhancedGetItem {
```

```
    public static void main(String[] args) {
        Region region = Region.US_EAST_1;
        DynamoDbClient ddb = DynamoDbClient.builder()
                .region(region)
                .build();

        DynamoDbEnhancedClient enhancedClient = DynamoDbEnhancedClient.builder()
                .dynamoDbClient(ddb)
                .build();

        getItem(enhancedClient);
        ddb.close();
    }

    public static String getItem(DynamoDbEnhancedClient enhancedClient) {
        Customer result = null;
        try {
            DynamoDbTable<Customer> table = enhancedClient.table("Customer",
 TableSchema.fromBean(Customer.class));
            Key key = Key.builder()
                    .partitionValue("id101").sortValue("tred@noserver.com")
                    .build();

            // Get the item by using the key.
            result = table.getItem(
                    (GetItemEnhancedRequest.Builder requestBuilder) ->
 requestBuilder.key(key));
            System.out.println("******* The description value is " +
 result.getCustName());

        } catch (DynamoDbException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
        return result.getCustName();
    }
}
```

Higher-Level-Programmierschnittstellen für DynamoDB

AWS SDKs Sie bieten Anwendungen mit Low-Level-Schnittstellen für die Arbeit mit Amazon DynamoDB. Diese clientseitigen Klassen und Methoden entsprechen direkt der Low-Level-API von DynamoDB API. Doch viele Entwickler stellen Impedanzfehlanpassungen fest, wenn sie

Elementen in einer Datenbanktabelle komplexe Datentypen zuordnen müssen. Mit einer Low-Level-Datenbankschnittstelle müssen Entwickler Methoden zum Lesen oder Schreiben von Objektdaten in Datenbanktabellen und umgekehrt entwickeln. Die Menge des zusätzlich erforderlichen Codes für jede Kombination aus Objekttyp und Datenbanktabelle kann überwältigend erscheinen.

Um die Entwicklung zu vereinfachen, bieten die AWS SDKs für Java und .NET zusätzliche Schnittstellen mit höheren Abstraktionsebenen. Mit den High-Level-Schnittstellen für DynamoDB-Objekte können Sie die Beziehungen zwischen Objekten in Ihrem Programm und den Datenbanktabellen definieren, die die Daten dieser Objekte speichern. Nachdem Sie diese Mappings definiert haben, rufen Sie einfache Objektmethoden wie `save`, `load` oder `delete` auf. Die zugrunde liegenden DynamoDB-Low-Level-Operationen werden dann in Ihrem Namen automatisch aufgerufen. Auf diese Weise können Sie objektorientierte Codes statt datenbankorientierter Codes schreiben.

Die übergeordneten Programmierschnittstellen für DynamoDB sind in den Versionen AWS SDKs für Java und .NET verfügbar.

Java

- Java 1.x: Dynamo DBMapper
- Java 2.x: Erweiterter DynamoDB-Client

.NET

- Arbeiten dem .NET-Dokumentmodell in DynamoDB
- Arbeiten mit dem Object Persistence-Modell von .NET und DynamoDB

Java 1.x: Dynamo DBMapper

> **Note**
>
> Das SDK für Java ist in zwei Versionen verfügbar: 1.x und 2.x. Die Version end-of-support für 1.x wurde am 12. Januar 2024 angekündigt. Es wird und end-of-support ist am 31. Dezember 2025 fällig. Für Neuentwicklungen wird die Verwendung von Version 2.x empfohlen.

Die AWS SDK für Java stellt eine `DynamoDBMapper` Klasse bereit, mit der Sie Ihre clientseitigen Klassen Amazon DynamoDB-Tabellen zuordnen können. Um `DynamoDBMapper` zu verwenden, definieren Sie die Beziehung zwischen Elementen in einer DynamoDB-Tabelle und ihren

entsprechenden Objekt-Instances im Code. Die `DynamoDBMapper`-Klasse ermöglicht Ihnen auch die Ausführung verschiedener Create-, Read-, Update-, und Delete-Operationen (CRUD) für Elemente sowie das Ausführen von Abfragen und Scans für Tabellen.

Themen

- Dynamo-Klasse DBMapper
- Unterstützte Datentypen für Dynamo DBMapper für Java
- Java-Anmerkungen für DynamoDB
- Optionale Konfigurationseinstellungen für Dynamo DBMapper
- DynamoDB und optimistische Sperren mit Versionsnummer
- Zuordnen willkürlicher Daten in DynamoDB
- Beispiele für DynamoDBMapper

> **Note**
>
> Die `DynamoDBMapper`-Klasse erlaubt kein Erstellen, Aktualisieren oder Löschen von Tabellen. Zum Ausführen dieser Aufgaben verwenden Sie stattdessen die Low-Level-SDK-für-Java-Schnittstelle.

Das SDK für Java stellt eine Reihe von Anmerkungstypen bereit, damit Sie Ihre Klassen zu Tabellen zuweisen können. Betrachten Sie beispielsweise eine `ProductCatalog`-Tabelle, die `Id` als Partitionsschlüssel verwendet.

```
ProductCatalog(Id, ...)
```

Sie können der `ProductCatalog`-Tabelle eine Klasse in Ihrer Client-Anwendung zuweisen wie im folgenden Java-Code gezeigt. Diese Code definiert ein Plain Old Java Object (POJO) mit dem Namen `CatalogItem`, das Anmerkungen verwendet, um Objektfelder zu DynamoDB-Attributnamen zuzuweisen:

Example

```
package com.amazonaws.codesamples;

import java.util.Set;
```

```
import software.amazon.dynamodb.datamodeling.DynamoDBAttribute;
import software.amazon.dynamodb.datamodeling.DynamoDBHashKey;
import software.amazon.dynamodb.datamodeling.DynamoDBIgnore;
import software.amazon.dynamodb.datamodeling.DynamoDBTable;

@DynamoDBTable(tableName="ProductCatalog")
public class CatalogItem {

    private Integer id;
    private String title;
    private String ISBN;
    private Set<String> bookAuthors;
    private String someProp;

    @DynamoDBHashKey(attributeName="Id")
    public Integer getId() { return id; }
    public void setId(Integer id) {this.id = id; }

    @DynamoDBAttribute(attributeName="Title")
    public String getTitle() {return title; }
    public void setTitle(String title) { this.title = title; }

    @DynamoDBAttribute(attributeName="ISBN")
    public String getISBN() { return ISBN; }
    public void setISBN(String ISBN) { this.ISBN = ISBN; }

    @DynamoDBAttribute(attributeName="Authors")
    public Set<String> getBookAuthors() { return bookAuthors; }
    public void setBookAuthors(Set<String> bookAuthors) { this.bookAuthors =
 bookAuthors; }

    @DynamoDBIgnore
    public String getSomeProp() { return someProp; }
    public void setSomeProp(String someProp) { this.someProp = someProp; }
}
```

Im vorherigen Code weist die Anmerkung `@DynamoDBTable` die Klasse `CatalogItem` der Tabelle `ProductCatalog` zu. Sie können einzelne Klassen-Instances als Elemente in der Tabelle speichern. Die Anmerkung `@DynamoDBHashKey` weist die Eigenschaft `Id` dem Primärschlüssel zu.

Standardmäßig sind die Klasseneigenschaften denselben Attributnamen in der Tabelle zugeordnet. Die Eigenschaften `Title` und `ISBN` sind den gleichen Attributnamen in der Tabelle zugeordnet.

Die `@DynamoDBAttribute`-Anmerkung ist optional, wenn der Name des DynamoDB-Attributs dem Namen der in der Klasse angegebenen Eigenschaft entspricht. Wenn sich die Namen unterscheiden, verwenden Sie diese Anmerkung mit dem Parameter `attributeName`, um anzugeben, mit welchem DynamoDB-Attribut diese Eigenschaft übereinstimmt.

Im vorangegangenen Beispiel wird jeder Eigenschaft die Annotation `@DynamoDBAttribute` hinzugefügt, um sicherzustellen, dass Eigenschaftsnamen genau den Tabellen entsprechen, die in einem vorherigen Schritt erstellt wurden und um die Attributnamen, die in anderen Codebeispielen in diesem Handbuch verwendet werden, einzuhalten.

Ihre Klassendefinition kann Eigenschaften besitzen, die keinen Attributen in der Tabelle zugeordnet sind. Sie erkennen diese Eigenschaften, indem Sie die `@DynamoDBIgnore`-Anmerkung hinzufügen. Im vorangegangenen Beispiel ist die `SomeProp`-Eigenschaft mit der `@DynamoDBIgnore`-Anmerkung gekennzeichnet. Wenn Sie eine `CatalogItem`-Instance in die Tabelle hochladen, enthält Ihre `DynamoDBMapper`-Instance die Eigenschaft `SomeProp` nicht. Darüber hinaus gibt der Mapper dieses Attribut nicht zurück, wenn Sie ein Element aus der Tabelle abrufen.

Nachdem Sie die Mappingklasse definierthaben, können Sie `DynamoDBMapper`-Methoden verwenden, um eine Instance dieser Klasse zu einem entsprechenden Element in der Tabelle `Catalog` zu schreiben. Das folgende Codebeispiel zeigt diese Technik.

```
AmazonDynamoDB client = AmazonDynamoDBClientBuilder.standard().build();

DynamoDBMapper mapper = new DynamoDBMapper(client);

CatalogItem item = new CatalogItem();
item.setId(102);
item.setTitle("Book 102 Title");
item.setISBN("222-2222222222");
item.setBookAuthors(new HashSet<String>(Arrays.asList("Author 1", "Author 2")));
item.setSomeProp("Test");

mapper.save(item);
```

Im folgenden Codebeispiel wird gezeigt, wie Sie das Element abrufen und auf einige seiner Attribute zugreifen.

```
CatalogItem partitionKey = new CatalogItem();

partitionKey.setId(102);
```

```
DynamoDBQueryExpression<CatalogItem> queryExpression = new
 DynamoDBQueryExpression<CatalogItem>()
    .withHashKeyValues(partitionKey);

List<CatalogItem> itemList = mapper.query(CatalogItem.class, queryExpression);

for (int i = 0; i < itemList.size(); i++) {
    System.out.println(itemList.get(i).getTitle());
    System.out.println(itemList.get(i).getBookAuthors());
}
```

`DynamoDBMapper` ermöglicht ein intuitives und natürliches Arbeiten mit DynamoDB-Daten in Java. Er bietet auch verschiedene integrierte Funktionen wie optimistische Sperre, ACID-Transaktionen, automatisch generierte Partitions- und Sortierschlüsselwerte sowie Objektversionierung.

Dynamo-Klasse DBMapper

Die Klasse A`DynamoDBMapper` ist der Eintrittspunkt für Amazon DynamoDB. Sie stellt Zugriff auf einen DynamoDB-Endpunkt bereit und ermöglicht Ihnen, auf Ihre Daten in verschiedenen Tabellen zuzugreifen. Sie ermöglicht Ihnen auch die Ausführung verschiedener Create-, Read-, Update-, und Delete-Operationen (CRUD) für Elemente sowie das Ausführen von Abfragen und Scans auf Tabellen. Diese Klasse bietet die folgenden Methoden für die Arbeit mit DynamoDB.

Die entsprechende Javadoc-Dokumentation finden Sie unter Dynamo DBMapper in der API-Referenz.AWS SDK für Java

Themen

- save
- load
- delete
- query
- queryPage
- scan
- scanPage
- parallelScan
- batchSave
- batchLoad

- batchDelete
- batchWrite
- transactionWrite
- transactionLoad
- count
- generateCreateTableAnfrage
- createS3Link
- GetS3 ClientCache

save

Speichert das angegebene Objekt in der Tabelle. Das Objekt, das Sie speichern möchten, ist der einzige erforderliche Parameter für diese Methode. Sie können optionale Konfigurationsparameter mithilfe des `DynamoDBMapperConfig`-Objekts bereitstellen.

Wenn ein Element mit demselben Primärschlüssel nicht vorhanden ist, erstellt diese Methode ein neues Element in der Tabelle. Wenn ein Element mit demselben Primärschlüssel vorhanden ist, aktualisiert sie das vorhandene Element. Wenn der Partitions- und Sortierschlüssel den Typ String haben und mit `@DynamoDBAutoGeneratedKey` angemerkt sind, erhalten sie einen Random Universally Unique Identifier (UUID), wenn sie nicht initialisiert werden. Versionsfelder, die mit `@DynamoDBVersionAttribute` angemerkt sind, werden um eins erhöht. Wenn ein Versionsfeld aktualisiert oder ein Schlüssel generiert wird, wird das übergebene Objekt als Folge der Operation aktualisiert.

Standardmäßig werden nur Attribute aktualisiert, die Eigenschaften der zugewiesenen Klasse entsprechen. Zusätzlich für ein Element vorhandene Attribute sind nicht betroffen. Wenn Sie jedoch `SaveBehavior.CLOBBER` angeben, können Sie die vollständige Überschreibung des Elements erzwingen.

```
DynamoDBMapperConfig config = DynamoDBMapperConfig.builder()
    .withSaveBehavior(DynamoDBMapperConfig.SaveBehavior.CLOBBER).build();

mapper.save(item, config);
```

Wenn Versioning aktiviert ist, müssen die client- und serverseitigen Elementversionen übereinstimmen. Jedoch muss die Version nicht übereinstimmen, wenn die Option

`SaveBehavior.CLOBBER` verwendet wird. Weitere Informationen über das Versioning finden Sie unter DynamoDB und optimistische Sperren mit Versionsnummer.

load

Ruft ein Element aus einer Tabelle ab. Sie müssen den Primärschlüssel des Elements bereitstellen, das Sie abrufen möchten. Sie können optionale Konfigurationsparameter mithilfe des `DynamoDBMapperConfig`-Objekts bereitstellen. Beispielsweise können Sie optional Strongly Consistent-Lesevorgänge anfordern, um sicherzustellen, dass diese Methode ausschließlich die neuesten Elementwerte, wie in der folgenden Java-Anweisung dargestellt, bereitstellt.

```
DynamoDBMapperConfig config = DynamoDBMapperConfig.builder()
    .withConsistentReads(DynamoDBMapperConfig.ConsistentReads.CONSISTENT).build();

CatalogItem item = mapper.load(CatalogItem.class, item.getId(), config);
```

Standardmäßig gibt DynamoDB das Element zurück, das über Eventually-Consistent-Werte verfügt. Weitere Informationen über das Eventual-Consistency-Modell von DynamoDB finden Sie unter DynamoDB-Lesekonsistenz.

delete

Löscht ein Element aus der Tabelle. Sie müssen eine Objekt-Instance aus der zugeordneten Klasse übergeben.

Wenn Versioning aktiviert ist, müssen die client- und serverseitigen Elementversionen übereinstimmen. Jedoch muss die Version nicht übereinstimmen, wenn die Option `SaveBehavior.CLOBBER` verwendet wird. Weitere Informationen zum Versioning finden Sie unter DynamoDB und optimistische Sperren mit Versionsnummer.

query

Fragt eine Tabelle oder einen sekundären Index ab.

Angenommen, Sie verfügen über die Tabelle `Reply`, die Antworten für Forum-Threads speichert. Jedes Thread-Thema kann null oder mehr Antworten enthalten. Der Primärschlüssel der Tabelle `Reply` besteht aus den Feldern `Id` und `ReplyDateTime`, wobei `Id` der Partitionsschlüssel und `ReplyDateTime` der Sortierschlüssel des Primärschlüssels ist.

```
Reply ( Id, ReplyDateTime, ... )
```

Angenommen, Sie haben eine Mapping zwischen der Klasse `Reply` und der entsprechenden Tabelle `Reply` in DynamoDB erstellt. Der folgende Java-Code verwendet `DynamoDBMapper`, um alle Antworten der letzten zwei Wochen zu einem bestimmten Thread-Thema zu finden.

Example

```
String forumName = "&DDB;";
String forumSubject = "&DDB; Thread 1";
String partitionKey = forumName + "#" + forumSubject;

long twoWeeksAgoMilli = (new Date()).getTime() - (14L*24L*60L*60L*1000L);
Date twoWeeksAgo = new Date();
twoWeeksAgo.setTime(twoWeeksAgoMilli);
SimpleDateFormat df = new SimpleDateFormat("yyyy-MM-dd'T'HH:mm:ss.SSS'Z'");
String twoWeeksAgoStr = df.format(twoWeeksAgo);

Map<String, AttributeValue> eav = new HashMap<String, AttributeValue>();
eav.put(":v1", new AttributeValue().withS(partitionKey));
eav.put(":v2",new AttributeValue().withS(twoWeeksAgoStr.toString()));

DynamoDBQueryExpression<Reply> queryExpression = new DynamoDBQueryExpression<Reply>()
    .withKeyConditionExpression("Id = :v1 and ReplyDateTime > :v2")
    .withExpressionAttributeValues(eav);

List<Reply> latestReplies = mapper.query(Reply.class, queryExpression);
```

Die Abfrage gibt eine Sammlung von `Reply`-Objekten zurück.

Die `query`-Methode gibt standardmäßig eine "lazy-loaded"-Sammlung zurück. Sie gibt anfänglich ausschließlich eine Ergebnisseite zurück und führt dann bei Bedarf einen Dienstaufruf für die nächste Seite durch. Um alle übereinstimmenden Elemente zu erhalten, müssen Sie die Sammlung `latestReplies` durchlaufen.

Beachten Sie, dass beim Aufrufen der `size()`-Methode für die Sammlung jedes Ergebnis geladen wird, um eine genaue Zählung zu liefern. Dies kann dazu führen, dass viel bereitgestellter Durchsatz verbraucht wird, und bei einer sehr großen Tabelle könnte sogar der gesamte Speicher in Ihrer JVM ausgeschöpft werden.

Für eine Indexabfrage müssen Sie zuerst den Index als eine Mapper-Klasse modellieren. Angenommen, die *`Reply`* Tabelle hat einen globalen sekundären Index namens -Message-Index. PostedBy Der Partitionsschlüssel für diesen Index ist `PostedBy` und der Sortierschlüssel ist `Message`. Die Klassendefinition für ein Element im Index würde wie folgt aussehen.

```
@DynamoDBTable(tableName="Reply")
public class PostedByMessage {
    private String postedBy;
    private String message;

    @DynamoDBIndexHashKey(globalSecondaryIndexName = "PostedBy-Message-Index",
 attributeName = "PostedBy")
    public String getPostedBy() { return postedBy; }
    public void setPostedBy(String postedBy) { this.postedBy = postedBy; }

    @DynamoDBIndexRangeKey(globalSecondaryIndexName = "PostedBy-Message-Index",
 attributeName = "Message")
    public String getMessage() { return message; }
    public void setMessage(String message) { this.message = message; }

   // Additional properties go here.
}
```

Die Anmerkung `@DynamoDBTable` gibt an, dass dieser Index mit der Tabelle `Reply` verknüpft ist. Die `@DynamoDBIndexHashKey` Anmerkung bezeichnet den Partitionsschlüssel (PostedBy) des Indexes und `@DynamoDBIndexRangeKey` den Sortierschlüssel (Message) des Indexes.

Sie können jetzt `DynamoDBMapper` nutzen, um den Index abzufragen, der eine Teilmenge der Nachrichten, die von einem bestimmten Benutzer gepostet wurden, abruft. Sie müssen den Indexnamen nicht angeben, wenn Sie keine widersprüchlichen Zuordnungen zwischen Tabellen und Indizes haben und die Zuordnungen bereits im Mapper vorgenommen wurden. Der Mapper leitet anhand des Primärschlüssels und des Sortierschlüssels ab. Der folgende Code fragt den globalen sekundären Index ab. Da globale sekundäre Indizes Eventually Consistent-Lesevorgänge, jedoch keine Strongly Consistent-Lesevorgänge unterstützen, müssen Sie `withConsistentRead(false)` angeben.

```
HashMap<String, AttributeValue> eav = new HashMap<String, AttributeValue>();
eav.put(":v1",  new AttributeValue().withS("User A"));
eav.put(":v2",  new AttributeValue().withS("DynamoDB"));

DynamoDBQueryExpression<PostedByMessage> queryExpression = new
 DynamoDBQueryExpression<PostedByMessage>()
    .withIndexName("PostedBy-Message-Index")
    .withConsistentRead(false)
    .withKeyConditionExpression("PostedBy = :v1 and begins_with(Message, :v2)")
    .withExpressionAttributeValues(eav);
```

```
List<PostedByMessage> iList =  mapper.query(PostedByMessage.class, queryExpression);
```

Die Abfrage gibt eine Sammlung von `PostedByMessage`-Objekten zurück.

queryPage

Fragt eine Tabelle oder einen sekundären Index ab und gibt eine einzelne Seite der übereinstimmenden Ergebnisse zurück. Ebenso wie bei der `query`-Methode, müssen Sie einen Partitions-Schlüsselwert und einen Abfragefilter angeben, der auf das Sortierschlüsselattribut angewendet wird. `queryPage` gibt jedoch ausschließlich die erste „Seite“ der Daten zurück, d. h. Menge von Daten, die in 1 MB passt.

scan

Scannt eine gesamte Tabelle oder einen sekundären Index. Sie können optional einen `FilterExpression` angeben, um den Ergebnissatz zu filtern.

Angenommen, Sie verfügen über die Tabelle `Reply`, die Antworten für Forum-Threads speichert. Jedes Thread-Thema kann null oder mehr Antworten enthalten. Der Primärschlüssel der Tabelle `Reply` besteht aus den Feldern `Id` und `ReplyDateTime`, wobei `Id` der Partitionsschlüssel und `ReplyDateTime` der Sortierschlüssel des Primärschlüssels ist.

```
Reply ( Id, ReplyDateTime, ... )
```

Wenn Sie der Tabelle `Reply` eine Java-Klasse zugewiesen haben, können Sie `DynamoDBMapper` verwenden, um die Tabelle zu scannen. Der folgende Java-Code scannt beispielsweise die gesamte Tabelle `Reply` und gibt nur die Antworten für ein bestimmtes Jahr zurück.

Example

```
HashMap<String, AttributeValue> eav = new HashMap<String, AttributeValue>();
eav.put(":v1", new AttributeValue().withS("2015"));

DynamoDBScanExpression scanExpression = new DynamoDBScanExpression()
    .withFilterExpression("begins_with(ReplyDateTime,:v1)")
    .withExpressionAttributeValues(eav);

List<Reply> replies =  mapper.scan(Reply.class, scanExpression);
```

Die `scan`-Methode gibt standardmäßig eine "lazy-loaded"-Sammlung zurück. Sie gibt anfänglich ausschließlich eine Ergebnisseite zurück und führt dann bei Bedarf einen Dienstaufruf für die nächste Seite durch. Um alle übereinstimmenden Elemente zu erhalten, müssen Sie die Sammlung `replies` durchlaufen.

Beachten Sie, dass beim Aufrufen der `size()`-Methode für die Sammlung jedes Ergebnis geladen wird, um eine genaue Zählung zu liefern. Dies kann dazu führen, dass viel bereitgestellter Durchsatz verbraucht wird, und bei einer sehr großen Tabelle könnte sogar der gesamte Speicher in Ihrer JVM ausgeschöpft werden.

Um einen Index zu scannen, müssen Sie zuerst den Index als eine Mapper-Klasse modellieren. Angenommen, die `Reply`-Tabelle hat einen globalen sekundären Index namens `PostedBy-Message-Index`. Der Partitionsschlüssel für diesen Index ist `PostedBy` und der Sortierschlüssel ist `Message`. Im Abschnitt query wird eine Mapper-Klasse für diesen Index gezeigt. Sie verwendet die Anmerkungen `@DynamoDBIndexHashKey` und `@DynamoDBIndexRangeKey`, um den Partitions- und Sortierschlüssel des Index anzugeben.

Im folgenden Codebeispiel wird `PostedBy-Message-Index` gescannt. Er verwendet keinen Scan-Filter, sodass alle Elemente im Index an Sie zurückgegeben werden.

```
DynamoDBScanExpression scanExpression = new DynamoDBScanExpression()
    .withIndexName("PostedBy-Message-Index")
    .withConsistentRead(false);

    List<PostedByMessage> iList =  mapper.scan(PostedByMessage.class, scanExpression);
    Iterator<PostedByMessage> indexItems = iList.iterator();
```

scanPage

Scannt eine Tabelle oder einen sekundären Index und gibt eine einzelne Seite der übereinstimmenden Ergebnisse zurück. Ebenso wie bei der `scan`-Methode, können Sie optional einen `FilterExpression` festlegen, um den Ergebnissatz zu filtern. `scanPage` gibt jedoch nur die erste „Seite“ der Daten zurück, d. h. Menge von Daten, die in 1 MB passt.

parallelScan

Führt einen parallelen Scan einer gesamten Tabelle oder einem sekundären Index durch. Sie geben eine Reihe von logischen Segmente für die Tabelle zusammen mit einem Scan-Ausdruck an, um Ergebnisse zu filtern. Der `parallelScan` teilt die Scan-Aufgabe unter mehreren Benutzern

auf, einer für jedes logische Segment; die Benutzer verarbeiten die Daten parallel und geben die Ergebnisse zurück.

Im folgenden Java-Codebeispiel wird ein paralleler Scan der Tabelle `Product` ausgeführt.

```
int numberOfThreads = 4;

Map<String, AttributeValue> eav = new HashMap<String, AttributeValue>();
eav.put(":n", new AttributeValue().withN("100"));

DynamoDBScanExpression scanExpression = new DynamoDBScanExpression()
    .withFilterExpression("Price <= :n")
    .withExpressionAttributeValues(eav);

List<Product> scanResult = mapper.parallelScan(Product.class, scanExpression,
 numberOfThreads);
```

batchSave

Speichert Objekte in eine oder mehrere Tabellen mithilfe eines Aufrufs oder mehrerer Aufrufe der Methode `AmazonDynamoDB.batchWriteItem`. Diese Methode bietet keine Transaktionsgarantien.

Der folgende Java-Code speichert zwei Elemente (Bücher) in der Tabelle `ProductCatalog`.

```
Book book1 = new Book();
book1.setId(901);
book1.setProductCategory("Book");
book1.setTitle("Book 901 Title");

Book book2 = new Book();
book2.setId(902);
book2.setProductCategory("Book");
book2.setTitle("Book 902 Title");

mapper.batchSave(Arrays.asList(book1, book2));
```

batchLoad

Ruft mehrere Elemente aus einer oder mehreren Tabellen anhand ihrer Primärschlüssel ab.

Der folgende Java-Code ruft zwei Elemente aus zwei verschiedenen Tabellen ab.

```
ArrayList<Object> itemsToGet = new ArrayList<Object>();

ForumItem forumItem = new ForumItem();
forumItem.setForumName("Amazon DynamoDB");
itemsToGet.add(forumItem);

ThreadItem threadItem = new ThreadItem();
threadItem.setForumName("Amazon DynamoDB");
threadItem.setSubject("Amazon DynamoDB thread 1 message text");
itemsToGet.add(threadItem);

Map<String, List<Object>> items = mapper.batchLoad(itemsToGet);
```

batchDelete

Löscht Objekte aus einer oder mehreren Tabellen mithilfe einer oder mehrerer Aufrufe der `AmazonDynamoDB.batchWriteItem`-Methode. Diese Methode bietet keine Transaktionsgarantien.

Der folgende Java-Code löscht zwei Elemente (Bücher) aus der Tabelle `ProductCatalog`.

```
Book book1 = mapper.load(Book.class, 901);
Book book2 = mapper.load(Book.class, 902);
mapper.batchDelete(Arrays.asList(book1, book2));
```

batchWrite

Speichert Objekte in und löscht Objekte aus einer oder mehreren Tabellen mithilfe einer oder mehrerer Aufrufe der `AmazonDynamoDB.batchWriteItem`-Methode. Diese Methode bietet keine Transaktionsgarantien oder Versioning-Support (bedingte Ablege- oder Löschvorgänge).

Der folgende Java-Code schreibt ein neues Element in die Tabelle `Forum`, schreibt ein neues Element in die Tabelle `Thread` und löscht ein Element aus der Tabelle `ProductCatalog`.

```
// Create a Forum item to save
Forum forumItem = new Forum();
forumItem.setName("Test BatchWrite Forum");

// Create a Thread item to save
Thread threadItem = new Thread();
threadItem.setForumName("AmazonDynamoDB");
threadItem.setSubject("My sample question");
```

```
// Load a ProductCatalog item to delete
Book book3 = mapper.load(Book.class, 903);

List<Object> objectsToWrite = Arrays.asList(forumItem, threadItem);
List<Book> objectsToDelete = Arrays.asList(book3);

mapper.batchWrite(objectsToWrite, objectsToDelete);
```

transactionWrite

Speichert Objekte in und löscht Objekte aus mindestens einer Tabelle mithilfe eines Aufrufs der `AmazonDynamoDB.transactWriteItems`-Methode.

Eine Liste der transaktionsspezifischen Ausnahmen finden Sie unter Fehler. TransactWriteItems

Weitere Informationen zu DynamoDB-Transaktionen und den bereitgestellten Garantien für Atomizität, Konsistenz, Isolation und Beständigkeit (Atomicity, Consistency, Isolation, Durability – ACID) finden Sie unter Amazon DynamoDB Transactions.

> **Note**
>
> Diese Methode unterstützt Folgendes nicht:
>
> - DBMapperDynamo-Konfiguration. SaveBehavior.

Der folgende Java-Code schreibt transaktional jeweils ein neues Element in die Tabellen `Forum` und `Thread`.

```
Thread s3ForumThread = new Thread();
s3ForumThread.setForumName("S3 Forum");
s3ForumThread.setSubject("Sample Subject 1");
s3ForumThread.setMessage("Sample Question 1");

Forum s3Forum = new Forum();
s3Forum.setName("S3 Forum");
s3Forum.setCategory("Amazon Web Services");
s3Forum.setThreads(1);

TransactionWriteRequest transactionWriteRequest = new TransactionWriteRequest();
```

```
transactionWriteRequest.addPut(s3Forum);
transactionWriteRequest.addPut(s3ForumThread);
mapper.transactionWrite(transactionWriteRequest);
```

transactionLoad

Lädt Objekte aus mindestens einer Tabelle mithilfe eines Aufrufs der `AmazonDynamoDB.transactGetItems`-Methode.

Eine Liste der transaktionsspezifischen Ausnahmen finden Sie unter Fehler. TransactGetItems

Weitere Informationen zu DynamoDB-Transaktionen und den bereitgestellten Garantien für Atomizität, Konsistenz, Isolation und Beständigkeit (Atomicity, Consistency, Isolation, Durability – ACID) finden Sie unter Amazon DynamoDB Transactions.

Der folgende Java-Code lädt transaktional jeweils ein Element aus den Tabellen `Forum` und `Thread`.

```
Forum dynamodbForum = new Forum();
dynamodbForum.setName("DynamoDB Forum");
Thread dynamodbForumThread = new Thread();
dynamodbForumThread.setForumName("DynamoDB Forum");

TransactionLoadRequest transactionLoadRequest = new TransactionLoadRequest();
transactionLoadRequest.addLoad(dynamodbForum);
transactionLoadRequest.addLoad(dynamodbForumThread);
mapper.transactionLoad(transactionLoadRequest);
```

count

Bewertet den angegebenen Scanausdruck und gibt die Anzahl der übereinstimmenden Elemente zurück. Es werden keine Elementdaten zurückgegeben.

generateCreateTableAnfrage

Analysiert eine POJO-Klasse, die eine DynamoDB-Tabelle repräsentiert und gibt eine `CreateTableRequest` für diese Tabelle zurück.

createS3Link

Erstellt einen Link zu einem Objekt in Amazon S3. Sie müssen einen Bucket-Namen und einen Schlüsselnamen angeben, welches das Objekt in dem Bucket eindeutig identifiziert.

Um `createS3Link` zu verwenden, muss die Mapper-Klasse die Methoden "Getter" und "Setter" definieren. Das folgende Codebeispiel veranschaulicht dies, indem der `CatalogItem` Klasse ein neues Attribut und neue getter/setter Methoden hinzugefügt werden.

```
@DynamoDBTable(tableName="ProductCatalog")
public class CatalogItem {

    ...

    public S3Link productImage;

    ....

    @DynamoDBAttribute(attributeName = "ProductImage")
    public S3Link getProductImage() {
            return productImage;
    }

    public void setProductImage(S3Link productImage) {
        this.productImage = productImage;
    }

...
}
```

Der folgende Java-Code definiert ein neues Element, das in die Tabelle `Product` geschrieben werden soll. Das Element enthält einen Link zu einemr Produktimage. Die Imagedaten werden auf Amazon S3 hochgeladen.

```
CatalogItem item = new CatalogItem();

item.setId(150);
item.setTitle("Book 150 Title");

String amzn-s3-demo-bucket = "amzn-s3-demo-bucket";
String myS3Key = "productImages/book_150_cover.jpg";
item.setProductImage(mapper.createS3Link(amzn-s3-demo-bucket, myS3Key));

item.getProductImage().uploadFrom(new File("/file/path/book_150_cover.jpg"));

mapper.save(item);
```

Die `S3Link`-Klasse bietet viele andere Methoden für die Bearbeitung von Objekten in Amazon S3. Weitere Informationen finden Sie unter Javadocs for `S3Link`.

GetS3 ClientCache

Gibt den zugrunde liegenden `S3ClientCache` für den Zugriff auf Amazon S3 zurück. Ein `S3ClientCache` ist ein intelligentes Mapping für `AmazonS3Client`-Objekte. Wenn Sie mehrere Kunden haben, `S3ClientCache` kann Ihnen an dabei helfen, die Kunden nach AWS Regionen zu organisieren und bei Bedarf neue Amazon S3 S3-Clients zu erstellen.

Unterstützte Datentypen für Dynamo DBMapper für Java

Dieser Abschnitt beschreibt die unterstützten primitiven Java-Datentypen, Sammlungen und beliebige Datentypen in Amazon DynamoDB.

Amazon DynamoDB unterstützt folgenden primitiven Java-Datentypen und primitiven Wrapperklassen.

- `String`
- `Boolean`, `boolean`
- `Byte`, `byte`
- `Date` (als ISO_8601 Millisekunden-Präzisionszeichenfolge, verschoben nach UTC)
- `Calendar` (als ISO_8601 Millisekunden-Präzisionszeichenfolge, verschoben nach UTC)
- `Long`, `long`
- `Integer`, `int`
- `Double`, `double`
- `Float`, `float`
- `BigDecimal`
- `BigInteger`

> **Note**
>
> - Weitere Informationen zu DynamoDB-Benennungsregeln und den verschiedenen unterstützten Datentypen finden sie unter Unterstützte Datentypen und Benennungsregeln in Amazon DynamoDB.

- Leere Binärwerte werden von Dynamo DBMapper unterstützt.
- Leere Zeichenfolgenwerte werden unterstützt von AWS SDK for Java 2.x.

  In AWS SDK for Java 1.x DBMapper unterstützt Dynamo das Lesen leerer String-Attributwerte, schreibt jedoch keine leeren String-Attributwerte, da diese Attribute aus der Anforderung gelöscht werden.

DynamoDB unterstützt die Java-Sammlungstypen Set, List und Map. Die folgende Tabelle fasst zusammen, wie diese Java-Typen den DynamoDB-Typen zugewiesen werden.

Java-Typ	DynamoDB-Typ
Alle Zahlentypen	`N` (Zahlentyp)
Zeichenfolgen	`S` (Zeichenfolgetyp)
Boolesch	`BOOL` (Boolescher Typ), 0 oder 1.
ByteBuffer	`B` (Binärtyp)
Date	`S` (Zeichenfolgetyp). Die Datumswerte werden als ISO-8601-formatierte Zeichenfolge gespeichert.
Set-Sammlungstypen	`SS` (Zeichenfolgesatz)-Typ, `NS` (Zahlensatz)-Typ oder `BS` (Binärsatz)-Typ.

Die `DynamoDBTypeConverter`-Schnittstelle ermöglicht das Mapping eigener beliebiger Datentypen zu einem Datentyp, der von DynamoDB nativ unterstützt wird. Weitere Informationen finden Sie unter Zuordnen willkürlicher Daten in DynamoDB.

Java-Anmerkungen für DynamoDB

Dieser Abschnitt beschreibt die Anmerkungen, die für das Mapping Ihrer Klassen und Eigenschaften zu Tabellen und Attributen in Amazon DynamoDB verfügbar sind.

Die entsprechende Javadoc-Dokumentation finden Sie unter Annotation Types Summary in der AWS SDK für Java API-Referenz.

> **Note**
>
> In den folgenden Anmerkungen sind ausschließlich `DynamoDBTable` und der `DynamoDBHashKey` erforderlich.

Themen

- Dynamo DBAttribute
- Dynamo DBAuto GeneratedKey
- Dynamo DBAuto GeneratedTimestamp
- Dynamo DBDocument
- Dynamo-Schlüssel DBHash
- Dynamo DBIgnore
- Dynamo DBIndex HashKey
- Dynamo DBIndex RangeKey
- Dynamo-Schlüssel DBRange
- Dynamo DBTable
- Dynamo wurde konvertiert DBType
- Dynamo DBTyped
- Dynamo-Attribut DBVersion

Dynamo DBAttribute

Ordnet eine Eigenschaft einem Tabellenattribut zu. Standardmäßig wird jede Klasseneigenschaft einem Elementattribut mit demselben Namen zugeordnet. Wenn die Namen jedoch nicht identisch sind, können Sie diese Anmerkung verwenden, um dem Attribut eine Eigenschaft zuzuordnen. Im folgenden Java-Ausschnitt ordnet `DynamoDBAttribute` die `BookAuthors`-Eigenschaft dem `Authors`-Attributnamen in der Tabelle zu.

```
@DynamoDBAttribute(attributeName = "Authors")
public List<String> getBookAuthors() { return BookAuthors; }
public void setBookAuthors(List<String> BookAuthors) { this.BookAuthors =
 BookAuthors; }
```

Der `DynamoDBMapper` verwendet `Authors` als Attributnamen bei der Speicherung des Objekts in die Tabelle.

Dynamo DBAuto GeneratedKey

Kennzeichnet eine Partitions- oder Sortierschlüsseleigenschaft als automatisch generiert. `DynamoDBMapper` generiert beim Speichern dieser Attribute nach dem Zufallsprinzip eine UUID. Ausschließlich Zeichenfolgeeigenschaften können als automatisch generierte Schlüssel markiert werden.

Das folgende Beispiel zeigt die Verwendung von automatisch generierten Schlüsseln.

```
@DynamoDBTable(tableName="AutoGeneratedKeysExample")
public class AutoGeneratedKeys {
    private String id;
    private String payload;

    @DynamoDBHashKey(attributeName = "Id")
    @DynamoDBAutoGeneratedKey
    public String getId() { return id; }
    public void setId(String id) { this.id = id; }

    @DynamoDBAttribute(attributeName="payload")
    public String getPayload() { return this.payload; }
    public void setPayload(String payload) { this.payload = payload; }

    public static void saveItem() {
        AutoGeneratedKeys obj = new AutoGeneratedKeys();
        obj.setPayload("abc123");

        // id field is null at this point
        DynamoDBMapper mapper = new DynamoDBMapper(dynamoDBClient);
        mapper.save(obj);

        System.out.println("Object was saved with id " + obj.getId());
    }
}
```

Dynamo DBAuto GeneratedTimestamp

Generiert automatisch einen Zeitstempel.

```
@DynamoDBAutoGeneratedTimestamp(strategy=DynamoDBAutoGenerateStrategy.ALWAYS)
```

```
public Date getLastUpdatedDate() { return lastUpdatedDate; }
public void setLastUpdatedDate(Date lastUpdatedDate) { this.lastUpdatedDate =
 lastUpdatedDate; }
```

Optional kann die Strategie für die automatische Generierung durch Angabe eines Strategieattributs definiert werden. Der Standardwert ist `ALWAYS`.

Dynamo DBDocument

Gibt an, dass eine Klasse als Amazon-DynamoDB-Dokument serialisiert werden kann.

Angenommen, Sie möchten ein JSON-Dokument einem DynamoDB-Attribut des Typs „Map“ zuweisen (`M`). Im folgenden Codebeispiel wird ein Element definiert, das ein verschachteltes Attribut (Bilder) des Typs „Map“ enthält.

```
public class ProductCatalogItem {

    private Integer id;  //partition key
    private Pictures pictures;
    /* ...other attributes omitted... */

    @DynamoDBHashKey(attributeName="Id")
    public Integer getId() { return id;}
    public void setId(Integer id) {this.id = id;}

    @DynamoDBAttribute(attributeName="Pictures")
    public Pictures getPictures() { return pictures;}
    public void setPictures(Pictures pictures) {this.pictures = pictures;}

    // Additional properties go here.

    @DynamoDBDocument
    public static class Pictures {
        private String frontView;
        private String rearView;
        private String sideView;

        @DynamoDBAttribute(attributeName = "FrontView")
        public String getFrontView() { return frontView; }
        public void setFrontView(String frontView) { this.frontView = frontView; }

        @DynamoDBAttribute(attributeName = "RearView")
        public String getRearView() { return rearView; }
```

```
        public void setRearView(String rearView) { this.rearView = rearView; }

        @DynamoDBAttribute(attributeName = "SideView")
        public String getSideView() { return sideView; }
        public void setSideView(String sideView) { this.sideView = sideView; }

    }
}
```

Anschließend könnten Sie ein neues `ProductCatalog`-Element mit `Pictures` speichern wie im folgenden Beispiel gezeigt.

```
ProductCatalogItem item = new ProductCatalogItem();

Pictures pix = new Pictures();
pix.setFrontView("http://example.com/products/123_front.jpg");
pix.setRearView("http://example.com/products/123_rear.jpg");
pix.setSideView("http://example.com/products/123_left_side.jpg");
item.setPictures(pix);

item.setId(123);

mapper.save(item);
```

Das resultierende `ProductCatalog`-Element würde wie folgt aussehen (im JSON-Format).

```
{
  "Id" : 123
  "Pictures" : {
    "SideView" : "http://example.com/products/123_left_side.jpg",
    "RearView" : "http://example.com/products/123_rear.jpg",
    "FrontView" : "http://example.com/products/123_front.jpg"
  }
}
```

Dynamo-Schlüssel DBHash

Ordnet eine Klasseneigenschaft dem Partitionsschlüssel der Tabelle zu. Die Eigenschaft muss ein skalarer Zeichenfolge-, Zahlen- oder Binärtyp sein. Die Eigenschaft darf kein Sammlungstyp sein.

Angenommen, Sie besitzen eine Tabelle `ProductCatalog`, die `Id` als Primärschlüssel verwendet. Der folgende Java-Code definiert eine `CatalogItem`-Klasse und ordnet ihre Eigenschaft `Id` dem Primärschlüssel der Tabelle `ProductCatalog` mithilfe des Tags `@DynamoDBHashKey` zu.

```
@DynamoDBTable(tableName="ProductCatalog")
public class CatalogItem {
    private Integer Id;
   @DynamoDBHashKey(attributeName="Id")
   public Integer getId() {
        return Id;
   }
   public void setId(Integer Id) {
        this.Id = Id;
   }
   // Additional properties go here.
}
```

Dynamo DBIgnore

Zeigt der `DynamoDBMapper`-Instance an, dass die zugeordnete Eigenschaft ignoriert werden sollte. Beim Speichern der Daten in die Tabelle, speichert der `DynamoDBMapper` diese Eigenschaft nicht in die Tabelle.

Wird auf die Getter-Methode oder das Klassenfeld für eine nicht modellierte Eigenschaft angewendet. Wenn die Anmerkung direkt auf das Klassenfeld angewendet wird, müssen die entsprechenden Getter- und Setter-Methoden in derselben Klasse deklariert werden.

Dynamo DBIndex HashKey

Ordnet eine Klasseneigenschaft dem Partitionsschlüssel eines globalen sekundären Index zu. Die Eigenschaft muss ein skalarer Zeichenfolge-, Zahlen- oder Binärtyp sein. Die Eigenschaft darf kein Sammlungstyp sein.

Verwenden Sie diese Anmerkung, wenn Sie `Query` einen globalen sekundären Index brauchen. Sie müssen den Indexnamen angeben (`globalSecondaryIndexName`). Wenn sich der Name der Klasseneigenschaft vom Indexpartitionsschlüssel unterscheidet, müssen Sie auch den Namen dieses Indexattributs (`attributeName`).

Dynamo DBIndex RangeKey

Ordnet eine Klasseneigenschaft dem Sortierschlüssel eines globalen sekundären Index oder eines lokalen sekundären Index zu. Die Eigenschaft muss ein skalarer Zeichenfolge-, Zahlen- oder Binärtyp sein. Die Eigenschaft darf kein Sammlungstyp sein.

Verwenden Sie diese Anmerkung, wenn Sie `Query` für einen lokalen sekundären Index oder einen globalen sekundären Index ausführen müssen und die Ergebnisse mithilfe des Index-Sortierschlüssels verfeinern möchten. Sie müssen den Indexnamen angeben (entweder `globalSecondaryIndexName` oder `localSecondaryIndexName`). Wenn der Name der Klasseneigenschaft sich von dem Indexsortierschlüssel unterscheidet, müssen Sie auch den Namen dieses Indexattributs (`attributeName`).

Dynamo-Schlüssel DBRange

Ordnet eine Klasseneigenschaft dem Sortierschlüssel der Tabelle zu. Die Eigenschaft muss ein skalarer Zeichenfolge-, Zahlen- oder Binärtyp sein. Sie darf kein Sammlungstyp sein.

Wenn der Primärschlüssel zusammengesetzt ist (Partitionsschlüssel und Sortierschlüssel), können Sie diesen Tag verwenden, um Ihr Klassenfeld dem Sortierschlüssel zuzuordnen. Angenommen, Sie verfügen über eine Tabelle `Reply`, die Antworten für Forum-Threads speichert. Jeder Thread kann zahlreiche Antworten enthalten. Daher ist der Primärschlüssel dieser Tabelle sowohl die `ThreadId` als auch `ReplyDateTime`. Die `ThreadId` ist der Partitionsschlüssel und `ReplyDateTime` ist der Sortierschlüssel.

Der folgende Java-Code definiert die Klasse `Reply` und ordnet sie der Tabelle `Reply` zu. Es werden die `@DynamoDBHashKey` und `@DynamoDBRangeKey`-Tags für die Identifizierung der Klasseneigenschaften, die dem Primärschlüssel zugeordnet werden, verwendet.

```
@DynamoDBTable(tableName="Reply")
public class Reply {
    private Integer id;
    private String replyDateTime;

    @DynamoDBHashKey(attributeName="Id")
    public Integer getId() { return id; }
    public void setId(Integer id) { this.id = id; }

    @DynamoDBRangeKey(attributeName="ReplyDateTime")
    public String getReplyDateTime() { return replyDateTime; }
```

```
    public void setReplyDateTime(String replyDateTime) { this.replyDateTime =
 replyDateTime; }

   // Additional properties go here.
}
```

Dynamo DBTable

Identifiziert die Zieltabelle in DynamoDB. Der folgende Java-Code definiert beispielsweise die Klasse `Developer` und ordnet sie der `People`-Tabelle in DynamoDB zu.

```
@DynamoDBTable(tableName="People")
public class Developer { ...}
```

Die `@DynamoDBTable`-Anmerkung kann geerbt werden. Jede neue Klasse, die von der Klasse `Developer` erbt, wird ebenfalls der Tabelle `People` zugeordnet. Angenommen, Sie erstellen beispielsweise eine `Lead` -Klasse, die von der `Developer`-Klasse erbt. Da Sie die Klasse `Developer` der Tabelle `People` zugeordnet haben, werden die Objekte der Klasse `Lead` ebenfalls in dieser Tabelle gespeichert.

Die `@DynamoDBTable` kann auch außer Kraft gesetzt werden. Jede neue Klasse, die standardmäßig von der Klasse `Developer` erbt, wird dieser Tabelle `People` zugeordnet. Sie können jedoch dieses Standardmapping außer Kraft setzen. Wenn Sie beispielsweise eine Klasse erstellen, die von der Klasse `Developer` erbt, können Sie sie ausdrücklich einer anderen Tabelle zuweisen, indem Sie die Anmerkung `@DynamoDBTable` hinzufügen wie im folgenden Java-Codebeispiel gezeigt.

```
@DynamoDBTable(tableName="Managers")
public class Manager extends Developer { ...}
```

Dynamo wurde konvertiert DBType

Eine Anmerkung, die angibt, dass eine Eigenschaft einen benutzerdefinierten Typkonverter verwendet. Kann in einer benutzerdefinierten Anmerkung mit Anmerkungen versehen werden, um dem `DynamoDBTypeConverter` zusätzliche Eigenschaften zu übergeben.

Die `DynamoDBTypeConverter`-Schnittstelle ermöglicht das Mapping eigener beliebiger Datentypen zu einem Datentyp, der von DynamoDB nativ unterstützt wird. Weitere Informationen finden Sie unter Zuordnen willkürlicher Daten in DynamoDB.

Dynamo DBTyped

Eine Anmerkung, mit der die standardmäßige Attributtypbindung überschrieben wird. Standard-Typen erfordern die Anmerkung nicht, wenn Sie die Standard-Attributbindung für diesen Typ anwenden.

Dynamo-Attribut DBVersion

Identifiziert eine Klasseneigenschaft für das Speichern einer Versionsnummer der optimistischen Sperre. `DynamoDBMapper` weist dieser Eigenschaft eine Versionsnummer zu, wenn ein neues Element gespeichert wird, und erhöht sie bei jeder Aktualisierung des Elements. Ausschließlich Zahlentypen und skalare Typen werden unterstützt. Weitere Informationen zu Datentypen finden Sie unter Datentypen. Weitere Informationen über das Versioning finden Sie unter DynamoDB und optimistische Sperren mit Versionsnummer.

Optionale Konfigurationseinstellungen für Dynamo DBMapper

Wenn Sie eine Instance von `DynamoDBMapper` erstellen, verfügt sie über bestimmte Standardverhaltensweisen. Sie können diese Standards überschreiben, indem Sie die `DynamoDBMapperConfig`-Klasse verwenden.

Der folgende Codeausschnitt erstellt einen `DynamoDBMapper` mit benutzerdefinierten Einstellungen:

```
AmazonDynamoDB client = AmazonDynamoDBClientBuilder.standard().build();

DynamoDBMapperConfig mapperConfig = DynamoDBMapperConfig.builder()
        .withSaveBehavior(DynamoDBMapperConfig.SaveBehavior.CLOBBER)
        .withConsistentReads(DynamoDBMapperConfig.ConsistentReads.CONSISTENT)
        .withTableNameOverride(null)

 .withPaginationLoadingStrategy(DynamoDBMapperConfig.PaginationLoadingStrategy.EAGER_LOADING)
    .build();

DynamoDBMapper mapper = new DynamoDBMapper(client, mapperConfig);
```

Weitere Informationen finden Sie unter Dynamo DBMapper Config in der AWS SDK für Java API-Referenz.

Sie können folgende Argumente für die Instance von `DynamoDBMapperConfig` verwenden:

- Einen `DynamoDBMapperConfig.ConsistentReads`-Aufzählungswert:
  - `EVENTUAL` – die Mapper-Instance verwendet eine Eventually Consistent-Anforderung.

- `CONSISTENT` – die Mapper-Instance verwendet eine Strongly-Consistent-Leseanforderung. Sie können diese optionale Einstellung mit der `load`-, `query`- oder `scan`-Operation verwenden. Strongly-Consistent-Lesevorgänge haben Auswirkungen auf Leistung und Fakturierung. Weitere Informationen finden Sie auf der DynamoDB-Produktdetailseite.

  Wenn Sie keine Lesekonsistenzeinstellung für die Mapper-Instance festgelegt haben, ist der Standard `EVENTUAL`.

  > **Note**
  >
  > Dieser Wert wird bei den `batch load` Operationen`query`, `querypageload`, und von Dynamo DBMapper angewendet.

- Ein `DynamoDBMapperConfig.PaginationLoadingStrategy`-Aufzählungswert – kontrolliert wie die Mapper-Instance eine paginierte Datenliste, z. B. ein `query`- oder `scan`-Ergebnis, verarbeitet:
  - `LAZY_LOADING` – Die Mapper-Instance lädt wenn möglich Daten und belässt alle geladenen Ergebnisse im Speicher.
  - `EAGER_LOADING` – die Mapper-Instance lädt die Daten, sobald die Liste initialisiert wird.
  - `ITERATION_ONLY` – Sie können nur einen Iterator nutzen, um aus der Liste zu lesen. Während der Iteration wird die Liste alle vorherigen Ergebnisse löschen, bevor die nächste Seite geladen wird, sodass die Liste höchstens eine Seite der geladenen Ergebnisse im Speicher hält. Dies bedeutet auch, dass die Liste nur einmal iteriert werden kann. Diese Strategie wird bei der Bearbeitung von großen Elementen empfohlen, um den Speichermehraufwand zu reduzieren.

  Wenn Sie keine Paginierungsladestrategie für die Mapper-Instance angeben, ist der Standard `LAZY_LOADING`.
- Ein `DynamoDBMapperConfig.SaveBehavior`-Aufzählungswert – gibt an wie die Mapper-Instance während Speichervorgängen mit den Attributen umgehen sollte:
  - `UPDATE` – während eines Speichervorgangs werden alle modellierten Attribute aktualisiert, nicht modellierte Attribute bleiben davon unberührt. Primitive Zahlentypen (Byte, int, long) werden auf 0 gesetzt. Objekttypen werden auf Null gesetzt.
  - `CLOBBER` – löscht und ersetzt alle Attribute während eines Speichervorgangs, nicht modellierte Attribute eingeschlossen. Dies erfolgt durch das Löschen des Elements und seiner Neuerstellung. Versionierte Feldeinschränkungen werden ebenfalls ignoriert.

Wenn Sie die Speicherverhaltensweise der Mapper-Instance nicht festlegen, ist der Standard `UPDATE`.

> **Note**
>
> DBMapper Dynamo-Transaktionsoperationen unterstützen keine Aufzählung. `DynamoDBMapperConfig.SaveBehavior`

- Ein `DynamoDBMapperConfig.TableNameOverride`-Objekt – weist die Mapper-Instance an, den Tabellennamen, der durch die `DynamoDBTable`-Anmerkung einer Klasse festgelegt wurde, zu ignorieren und stattdessen einen anderen Tabellennamen zu verwenden, den Sie festlegen. Dies ist beim Partitionieren der Daten in mehrere Tabellen zur Laufzeit nützlich.

Sie können das Standardkonfigurationsobjekt für `DynamoDBMapper` pro Operation falls nötig überschreiben.

DynamoDB und optimistische Sperren mit Versionsnummer

Optimistische Sperren stellen eine Strategie dar, die sicherstellt, dass das clientseitige Element, das Sie aktualisieren (oder löschen), das gleiche Element wie in Amazon DynamoDB ist. Wenn Sie diese Strategie verwenden, werden Ihre Datenbankschreibvorgänge vor dem Überschreiben durch Schreibvorgänge anderer Benutzer geschützt und umgekehrt.

Mit der optimistischen Sperre verfügt jedes Element über ein Attribut, das als Versionsnummer fungiert. Wenn Sie ein Element in einer Tabelle abrufen, zeichnet die Anwendung die Versionsnummer dieses Elements auf. Sie können das Element aktualisieren, jedoch nur, wenn die serverseitige Versionsnummer nicht geändert wurde. Wenn Versionen nicht übereinstimmen, bedeutet das, dass ein anderer Benutzer das Element vor Ihnen geändert hat. Die Aktualisierung schlägt fehlt, da Sie eine veraltete Version des Elements besitzen. Wenn dies der Fall ist, wiederholen Sie den Vorgang, indem Sie das Element abrufen und anschließend versuchen, es zu aktualisieren. Die optimistische Sperre hindert Sie daran, versehentlich Änderungen anderer Benutzer zu überschreiben. Sie hindert auch andere Benutzer daran, versehentlich Ihre Änderungen zu überschreiben.

Sie können zwar Ihre eigene optimistische Sperrstrategie implementieren, stellt jedoch die AWS SDK für Java Anmerkung bereit. `@DynamoDBVersionAttribute` In der Mappingklasse für Ihre Tabelle legen Sie eine Eigenschaft zum Speichern der Versionsnummer fest und kennzeichnen diese mithilfe dieser Anmerkung. Wenn Sie ein Objekt speichern, wird das entsprechende Element

in der DynamoDB-Tabelle über ein Attribut verfügen, das die Versionsnummer speichert. Der `DynamoDBMapper` weist eine Versionsnummer zu, wenn Sie das Element zum ersten Mal speichern und er erhöht automatisch die Versionsnummer bei jeder Aktualisierung des Elements. Die Aktualisierungs- oder Löschanforderungen sind nur erfolgreich, wenn die clientseitige Objektversion der entsprechenden Versionsnummer des Elements in der DynamoDB-Tabelle entspricht.

`ConditionalCheckFailedException` wird aufgeworfen, wenn:

- Sie eine optimistische Sperre mit `@DynamoDBVersionAttribute` verwenden und sich der Versionswert auf dem Server von dem Wert auf der Client-Seite unterscheidet.
- Sie Ihre eigenen bedingten Einschränkungen beim Speichern der Daten angeben, indem Sie `DynamoDBMapper` mit `DynamoDBSaveExpression` verwenden, und diese Einschränkungen fehlgeschlagen sind.

> **Note**
>
> - Globale Tabellen in DynamoDB verwenden bei gleichzeitigen Updates einen Mechanismus, bei dem der letzte Schreibvorgang gültig ist. Bei Verwendung von globalen Tabellen ist immer der letzte Schreibvorgang gültig. In diesem Fall funktioniert das Sperren daher nicht wie erwartet.
> - Transaktionale Schreibvorgänge von `DynamoDBMapper` unterstützen keine `@DynamoDBVersionAttribute`-Anmerkung und -Bedingungsausdrücke auf demselben Objekt. Wenn ein Objekt innerhalb eines transaktionalen Schreibvorgangs mit einer Anmerkung versehen ist `@DynamoDBVersionAttribute` und auch über einen Bedingungsausdruck verfügt, SdkClientException wird ein ausgelöst.

Der folgende Java-Code definiert beispielsweise die Klasse `CatalogItem`, die über mehrere Eigenschaften verfügt. Die `Version`-Eigenschaft wird mit der `@DynamoDBVersionAttribute`-Anmerkung gekennzeichnet.

Example

```
@DynamoDBTable(tableName="ProductCatalog")
public class CatalogItem {

    private Integer id;
    private String title;
```

```
    private String ISBN;
    private Set<String> bookAuthors;
    private String someProp;
    private Long version;

    @DynamoDBHashKey(attributeName="Id")
    public Integer getId() { return id; }
    public void setId(Integer Id) { this.id = Id; }

    @DynamoDBAttribute(attributeName="Title")
    public String getTitle() { return title; }
    public void setTitle(String title) { this.title = title; }

    @DynamoDBAttribute(attributeName="ISBN")
    public String getISBN() { return ISBN; }
    public void setISBN(String ISBN) { this.ISBN = ISBN;}

    @DynamoDBAttribute(attributeName = "Authors")
    public Set<String> getBookAuthors() { return bookAuthors; }
    public void setBookAuthors(Set<String> bookAuthors) { this.bookAuthors =
 bookAuthors; }

    @DynamoDBIgnore
    public String getSomeProp() { return someProp;}
    public void setSomeProp(String someProp) {this.someProp = someProp;}

    @DynamoDBVersionAttribute
    public Long getVersion() { return version; }
    public void setVersion(Long version) { this.version = version;}
}
```

Sie können die `@DynamoDBVersionAttribute`-Anmerkung auf löschbare Typen anwenden, die durch primitive Wrapper-Klassen bereitgestellt wurden, die wiederum einen löschbaren Typ wie `Long` und `Integer` bereitstellen.

Die optimistische Sperre hat folgende Auswirkungen auf diese `DynamoDBMapper`-Methoden:

- `save` – für ein neues Element weist der `DynamoDBMapper` eine erste Versionsnummer 1 zu. Wenn Sie ein Element abrufen, eine oder mehrere seiner Eigenschaften aktualisieren und versuchen, die Änderungen zu speichern, ist die Speicheroperation nur dann erfolgreich, wenn die clientseitige und die serverseitige Versionsnummer übereinstimmen. Der `DynamoDBMapper` erhöht die Versionsnummer automatisch.

- `delete` – die `delete`-Methode übernimmt ein Objekt als einen Parameter und der `DynamoDBMapper` führt einen Versionscheck durch, bevor er das Element löscht. Der Versionscheck kann deaktiviert werden, wenn `DynamoDBMapperConfig.SaveBehavior.CLOBBER` in der Anforderung angegeben wird.

  Die interne Implementierung der optimistischen Sperre im `DynamoDBMapper` nutzt die von DynamoDB bereitgestellte Unterstützung für bedingtes Aktualisieren und bedingtes Löschen.
- `transactionWrite` —
  - `Put` – für ein neues Element weist der `DynamoDBMapper` eine erste Versionsnummer 1 zu. Wenn Sie ein Element abrufen, eine oder mehrere seiner Eigenschaften aktualisieren und versuchen, die Änderungen zu speichern, ist die Put-Operation nur dann erfolgreich, wenn die clientseitige und die serverseitige Versionsnummer übereinstimmen. Der `DynamoDBMapper` erhöht die Versionsnummer automatisch.
  - `Update` – für ein neues Element weist der `DynamoDBMapper` eine erste Versionsnummer 1 zu. Wenn Sie ein Element abrufen, eine oder mehrere seiner Eigenschaften aktualisieren und versuchen, die Änderungen zu speichern, ist die Update-Operation nur dann erfolgreich, wenn die clientseitige und die serverseitige Versionsnummer übereinstimmen. Der `DynamoDBMapper` erhöht die Versionsnummer automatisch.
  - `Delete` – `DynamoDBMapper` führt eine Versionsprüfung durch, bevor er das Element löscht. Die Lösch-Operation ist nur dann erfolgreich, wenn die Versionsnummern auf Clientseite und auf Serverseite übereinstimmen.
  - `ConditionCheck` – die `@DynamoDBVersionAttribute`-Anmerkung wird für `ConditionCheck`-Operationen nicht unterstützt. Ein SdkClientException wird ausgelöst, wenn ein `ConditionCheck` Element mit einer Anmerkung versehen ist. `@DynamoDBVersionAttribute`

Deaktivieren der optimistischen Sperre

Um die optimistische Sperre zu deaktivieren, können Sie den `DynamoDBMapperConfig.SaveBehavior`-Aufzählungswert von `UPDATE` auf `CLOBBER` ändern. Sie können dies tun, indem Sie eine `DynamoDBMapperConfig`-Instance erstellen, die eine Versionsüberprüfung überspringt und diese Instance für alle Anforderungen verwendet. Weitere Informationen zu `DynamoDBMapperConfig.SaveBehavior` und anderen optionalen `DynamoDBMapper`-Parametern finden Sie unter Optionale Konfigurationseinstellungen für Dynamo DBMapper.

Sie können auch ein Sperrverhalten ausschließlich für eine bestimmte Operation festlegen. Der folgende Java-Ausschnitt verwendet beispielsweise den `DynamoDBMapper`, um ein Katalogelement zu speichern. Er legt ein `DynamoDBMapperConfig.SaveBehavior` fest, indem er optionale `DynamoDBMapperConfig`-Parameter der `save`-Methode hinzufügt.

> **Note**
>
> Die TransactionWrite-Methode unterstützt Dynamo DBMapper Config nicht. SaveBehaviorKonfiguration. Das Deaktivieren der optimistischen Sperre für transactionWrite wird nicht unterstützt.

Example

```
DynamoDBMapper mapper = new DynamoDBMapper(client);

// Load a catalog item.
CatalogItem item = mapper.load(CatalogItem.class, 101);
item.setTitle("This is a new title for the item");
...
// Save the item.
mapper.save(item,
    new DynamoDBMapperConfig(
        DynamoDBMapperConfig.SaveBehavior.CLOBBER));
```

Zuordnen willkürlicher Daten in DynamoDB

Zusätzlich zu den unterstützten Java-Typen (siehe Unterstützte Datentypen für Dynamo DBMapper für Java) können Sie Typen in der Anwendungen verwenden, für die es keine direkte Mapping zu Amazon-DynamoDB-Typen gibt. Um diese Typen zuzuweisen, müssen Sie eine Implementierung bereitstellen, die Ihren komplexen Typ in einen von DynamoDB unterstützten Typ und umgekehrt konvertiert, und die komplexe Typzugriffsmethode unter Verwendung der Anmerkung `@DynamoDBTypeConverted` kommentieren. Der Konverter-Code wandelt Daten um, wenn Objekte gespeichert oder geladen werden. Er wird außerdem für alle Operationen verwendet, die komplexe Typen nutzen. Beachten Sie, dass beim Vergleichen von Daten während der Abfrage- und Scan-Operationen, die Vergleiche für die in DynamoDB gespeicherten Daten gemacht werden.

Betrachten Sie zum Beispiel die folgende `CatalogItem`-Klasse, die die Eigenschaft `Dimension` mit dem Typ `DimensionType` definiert. Diese Eigenschaft speichert die Elementabmessungen wie Höhe, Breite und Dicke. Gehen Sie davon aus, dass Sie sich dazu entscheiden, diese Elementmaße

als Zeichenfolge (z. B. 8,5 x 11 x ,05) in DynamoDB zu speichern. Das folgende Beispiel stellt einen Konverter-Code bereit, der das `DimensionType`-Objekt in eine Zeichenfolge und eine Zeichenfolge in einen `DimensionType` konvertiert.

> **Note**
>
> In diesem Codebeispiel wird davon ausgegangen, dass Sie bereits Daten für Ihr Konto in DynamoDB geladen haben, indem Sie die Anweisungen im Abschnitt Erstellen von Tabellen und Laden von Daten für Codebeispiele in DynamoDB befolgen.
> step-by-stepAnweisungen zur Ausführung des folgenden Beispiels finden Sie unter. Java-Codebeispiele

Example

```
public class DynamoDBMapperExample {

    static AmazonDynamoDB client;

    public static void main(String[] args) throws IOException {

        // Set the AWS region you want to access.
        Regions usWest2 = Regions.US_WEST_2;
        client = AmazonDynamoDBClientBuilder.standard().withRegion(usWest2).build();

        DimensionType dimType = new DimensionType();
        dimType.setHeight("8.00");
        dimType.setLength("11.0");
        dimType.setThickness("1.0");

        Book book = new Book();
        book.setId(502);
        book.setTitle("Book 502");
        book.setISBN("555-5555555555");
        book.setBookAuthors(new HashSet<String>(Arrays.asList("Author1", "Author2")));
        book.setDimensions(dimType);

        DynamoDBMapper mapper = new DynamoDBMapper(client);
        mapper.save(book);

        Book bookRetrieved = mapper.load(Book.class, 502);
```

```
        System.out.println("Book info: " + "\n" + bookRetrieved);

        bookRetrieved.getDimensions().setHeight("9.0");
        bookRetrieved.getDimensions().setLength("12.0");
        bookRetrieved.getDimensions().setThickness("2.0");

        mapper.save(bookRetrieved);

        bookRetrieved = mapper.load(Book.class, 502);
        System.out.println("Updated book info: " + "\n" + bookRetrieved);
    }

    @DynamoDBTable(tableName = "ProductCatalog")
    public static class Book {
        private int id;
        private String title;
        private String ISBN;
        private Set<String> bookAuthors;
        private DimensionType dimensionType;

        // Partition key
        @DynamoDBHashKey(attributeName = "Id")
        public int getId() {
            return id;
        }

        public void setId(int id) {
            this.id = id;
        }

        @DynamoDBAttribute(attributeName = "Title")
        public String getTitle() {
            return title;
        }

        public void setTitle(String title) {
            this.title = title;
        }

        @DynamoDBAttribute(attributeName = "ISBN")
        public String getISBN() {
            return ISBN;
        }
```

```
        public void setISBN(String ISBN) {
            this.ISBN = ISBN;
        }

        @DynamoDBAttribute(attributeName = "Authors")
        public Set<String> getBookAuthors() {
            return bookAuthors;
        }

        public void setBookAuthors(Set<String> bookAuthors) {
            this.bookAuthors = bookAuthors;
        }

        @DynamoDBTypeConverted(converter = DimensionTypeConverter.class)
        @DynamoDBAttribute(attributeName = "Dimensions")
        public DimensionType getDimensions() {
            return dimensionType;
        }

        @DynamoDBAttribute(attributeName = "Dimensions")
        public void setDimensions(DimensionType dimensionType) {
            this.dimensionType = dimensionType;
        }

        @Override
        public String toString() {
            return "Book [ISBN=" + ISBN + ", bookAuthors=" + bookAuthors + ",
dimensionType= "
                    + dimensionType.getHeight() + " X " + dimensionType.getLength() + "
X "
                    + dimensionType.getThickness()
                    + ", Id=" + id + ", Title=" + title + "]";
        }
    }

    static public class DimensionType {

        private String length;
        private String height;
        private String thickness;

        public String getLength() {
            return length;
        }
```

```
        public void setLength(String length) {
            this.length = length;
        }

        public String getHeight() {
            return height;
        }

        public void setHeight(String height) {
            this.height = height;
        }

        public String getThickness() {
            return thickness;
        }

        public void setThickness(String thickness) {
            this.thickness = thickness;
        }
    }

    // Converts the complex type DimensionType to a string and vice-versa.
    static public class DimensionTypeConverter implements DynamoDBTypeConverter<String,
DimensionType> {

        @Override
        public String convert(DimensionType object) {
            DimensionType itemDimensions = (DimensionType) object;
            String dimension = null;
            try {
                if (itemDimensions != null) {
                    dimension = String.format("%s x %s x %s",
itemDimensions.getLength(), itemDimensions.getHeight(),
                            itemDimensions.getThickness());
                }
            } catch (Exception e) {
                e.printStackTrace();
            }
            return dimension;
        }

        @Override
        public DimensionType unconvert(String s) {
```

```
            DimensionType itemDimension = new DimensionType();
            try {
                if (s != null && s.length() != 0) {
                    String[] data = s.split("x");
                    itemDimension.setLength(data[0].trim());
                    itemDimension.setHeight(data[1].trim());
                    itemDimension.setThickness(data[2].trim());
                }
            } catch (Exception e) {
                e.printStackTrace();
            }

            return itemDimension;
        }
    }
}
```

Beispiele für DynamoDBMapper

Das AWS-SDK für Java stellt eine `DynamoDBMapper`-Klasse bereit, mit der Sie Ihre clientseitigen Klassen zu Amazon-DynamoDB-Tabellen zuweisen können. Um `DynamoDBMapper` zu verwenden, definieren Sie die Beziehung zwischen Elementen in einer DynamoDB-Tabelle und ihren entsprechenden Objekt-Instances im Code. Die `DynamoDBMapper`-Klasse ermöglicht Ihnen auch die Ausführung verschiedener Create-, Read-, Update-, und Delete-Operationen (CRUD) für Elemente sowie das Ausführen von Abfragen und Scans für Tabellen.

Weitere Informationen für die Verwendung von `DynamoDBMapper` finden Sie unter DynamoDB-Beispiele für die Verwendung des AWS-SDK für Java im AWS-SDK für Java 1.x-Entwicklerhandbuch.

Java 2.x: Erweiterter DynamoDB-Client

Der erweiterte DynamoDB-Client ist eine Bibliothek auf hoher Ebene, die Teil von AWS SDK für Java Version 2 (v2) ist. Er bietet eine einfache Möglichkeit, clientseitige Klassen zu DynamoDB-Tabellen zuzuordnen. Sie definieren die Beziehungen zwischen Tabellen und ihren jeweiligen Modellklassen im Code. Nach der Definition dieser Beziehungen können Sie verschiedene Operationen zum Erstellen, Lesen, Aktualisieren oder Löschen (Create, Read, Update, Delete, CRUD) für Tabellen oder Elemente in DynamoDB auf intuitive Weise ausführen.

Weitere Informationen darüber, wie Sie den erweiterten Client mit DynamoDB verwenden können, finden Sie unter Verwenden des erweiterten DynamoDB-Clients im AWS SDK für Java 2.x.

Arbeiten dem .NET-Dokumentmodell in DynamoDB

Das AWS SDK für .NET bietet Dokumentmodellklassen, die einige der Low-Level-Amazon-DynamoDB-Operationen zur weiteren Vereinfachung der Codierung umschließen. Die primären Klassen im Dokumentmodell sind `Table` und `Document`. Die `Table`-Klasse bietet Datenoperationsmethoden wie `PutItem`, `GetItem` und `DeleteItem`. Außerdem stellt sie auch die `Query`- und die `Scan`-Methode bereit. Die `Document`-Klasse steht für ein einzelnes Element in einer Tabelle.

Die zuvor genannten Dokumentmodellklassen sind in dem `Amazon.DynamoDBv2.DocumentModel`-Namespace verfügbar.

> **Note**
>
> Sie können die Dokumentmodellklassen nicht zum Erstellen, Aktualisieren und Löschen von Tabellen nutzen. Das Dokumentmodell unterstützt jedoch die meisten allgemeinen Datenoperationen.

Themen

- Unterstützte Datentypen

Unterstützte Datentypen

Das Dokumentmodell unterstützt eine Reihe von primitiven .NET-Datentypen und Sammlungsdatentypen. Das Modell unterstützt die folgenden primitiven Datentypen.

- `bool`
- `byte`
- `char`
- `DateTime`
- `decimal`
- `double`
- `float`
- `Guid`
- `Int16`

- `Int32`
- `Int64`
- `SByte`
- `string`
- `UInt16`
- `UInt32`
- `UInt64`

In der folgenden Tabelle ist das Mapping der vorhergehenden .NET-Typen zu den DynamoDB-Typen zusammengefasst.

.NET-primitiver Typ	DynamoDB-Typ
Alle Zahlentypen	`N` (Zahlentyp)
Alle Zeichenfolgetypen	`S` (Zeichenfolgetyp)
MemoryStream, Byte []	`B` (Binärtyp)
bool	`N` (Zahlentyp) 0 repräsentiert False und 1 steht für True.
DateTime	`S` (Zeichenfolgetyp). Die `DateTime`-Werte werden als ISO-8601-formatierte Zeichenfolgen gespeichert.
Guid	`S` (Zeichenfolgetyp).
Sammlungstypen (Liste, HashSet und Array)	`BS`-(Binärsatz)-Typ, `SS`-(Zeichenfolgesatz)-Typ und `NS`-(Zahlensatz)-Typ.

AWS SDK für .NET definiert Typen für das Mapping der booleschen, Null-, Listen- und Map-Typen von DynamoDB zur .NET-Dokumentmodell-API:

- Verwenden Sie als booleschen Typ `DynamoDBBool`.
- Verwenden Sie als den Null-Typ `DynamoDBNull`.

- Verwenden Sie als Listentyp `DynamoDBList`.
- Verwenden Sie als Map-Typ `Document`.

> **Note**
>
> - Leere Binärwerte werden unterstützt.
> - Das Lesen von leeren Zeichenfolgenwerten wird unterstützt. Leere Zeichenfolgen-Attributwerte werden beim Schreiben zu DynamoDB in Attributwerten vom Typ Zeichenfolgensatz unterstützt. Leere Zeichenfolgen-Attributwerte des Zeichenfolgentyps und leere Zeichenfolgenwerte innerhalb des Listen- oder Map-Typs werden aus Schreibanforderungen gelöscht.

Arbeiten mit dem Object Persistence-Modell von .NET und DynamoDB

Das AWS SDK für .NET bietet ein Objektpersistenzmodell, mit dem Sie Ihre clientseitigen Klassen Amazon DynamoDB-Tabellen zuordnen können. Die einzelnen Objekt-Instances werden anschließend einem Element in den entsprechenden Tabellen zugeordnet. Zum Speichern der clientseitigen Objekte in den Tabellen stellt das Object-Persistence-Modell die Klasse `DynamoDBContext` bereit, einen Eintrittspunkt für DynamoDB. Mit dieser Klasse verfügen Sie über eine Verbindung zu DynamoDB, können auf Tabellen zugreifen und verschiedene CRUD-Operationen sowie Abfragen ausführen.

Das Objektpersistenzmodell bietet eine Reihe von Attributen, um clientseitige Klassen Tabellen und Tabellenattributen zuzuordnen. properties/fields

> **Note**
>
> Das Object Persistence-Modell stellt keine API zum Erstellen, Aktualisieren oder Löschen von Tabellen bereit. Es stellt ausschließlich Datenoperationen bereit. Sie können nur die AWS SDK für .NET Low-Level-API verwenden, um Tabellen zu erstellen, zu aktualisieren und zu löschen.

Das folgende Beispiel zeigt, wie das Object Persistence-Modell funktioniert. Es wird mit der Tabelle `ProductCatalog` gestartet. Es besitzt `Id` als Primärschlüssel.

```
ProductCatalog(Id, ...)
```

Angenommen, Sie besitzen die Klasse Book mit den Eigenschaften Title, ISBN und Authors. Sie können die Klasse Book der Tabelle ProductCatalog zuordnen, indem Sie die durch das Object Persistence-Modell definierten Attribute hinzufügen wie im folgenden C#-Codeausschnitt gezeigt.

Example

```
[DynamoDBTable("ProductCatalog")]
  public class Book
  {
    [DynamoDBHashKey]
    public int Id { get; set; }

    public string Title { get; set; }
    public int ISBN { get; set; }

    [DynamoDBProperty("Authors")]
    public List<string> BookAuthors { get; set; }

    [DynamoDBIgnore]
    public string CoverPage { get; set; }
  }
```

Im vorherigen Beispiel ordnet das Attribut DynamoDBTable die Klasse Book der Tabelle ProductCatalog zu.

Das Object Persistence-Modell unterstützt die explizite und das Standardmapping zwischen Klasseneigenschaften und Tabellenattributen.

- Explizites Mapping – Um eine Eigenschaft einem Primärschlüssel zuzuordnen, müssen Sie die DynamoDBHashKey- und DynamoDBRangeKey-Attribute des „Object-Persistence“-Modells verwenden. Außerdem gilt für andere als Primärschlüsselattribute, dass Sie das Mapping durch explizites Hinzufügen des Attributs DynamoDBProperty definieren müssen, wenn der Name einer Eigenschaft in Ihrer Klasse und des Tabellenattributs, dem Sie diese zuordnen möchten, nicht identisch sind.

  Im vorherigen Beispiel werden die Eigenschaft Id dem Primärschlüssel mit dem gleichen Namen und die Eigenschaft BookAuthors dem Attribut Authors in der Tabelle ProductCatalog zugeordnet.

- Standard-Mapping – Standardmäßig ordnet das „Object Persistence“-Modell die Klasseneigenschaften den Attributen mit identischem Namen in der Tabelle zu.

  Im vorherigen Beispiel werden die Eigenschaften `Title` und `ISBN` den Attributen mit den gleichen Namen in der Tabelle `ProductCatalog` zugeordnet.

Sie müssen nicht jede einzelne Klasseneigenschaft zuordnen. Sie erkennen diese Eigenschaften, indem Sie das `DynamoDBIgnore`-Attribut hinzufügen. Wenn Sie eine `Book`-Instance in die Tabelle hochladen, enthält der `DynamoDBContext` nicht die `CoverPage`-Eigenschaft. Diese Eigenschaft wird auch nicht zurückgegeben, wenn Sie die Book-Instance abrufen.

Sie können Eigenschaften von primitiven .NET-Typen wie „int“ und „string“ zuweisen. Sie können auch beliebige Datentypen zuordnen, solange Sie einen geeigneten Konverter bereitstellen, um diese Datentypen einem DynamoDB-Datentyp zuzuordnen. Weitere Informationen zum Mapping beliebiger Datentypen finden Sie unter Zuordnen beliebiger Daten mit DynamoDB mithilfe des AWS SDK für .NET Objektpersistenzmodells.

Das Object Persistence-Modell unterstützt die optimistische Sperre. Während einer Aktualisierung stellt diese Funktion sicher, dass Sie über die neueste Kopie des zu aktualisierenden Elements verfügen. Weitere Informationen finden Sie unter Optimistisches Sperren mit DynamoDB und dem AWS SDK für .NET Objektpersistenzmodell.

Weitere Informationen finden Sie in den folgenden Themen.

Themen

- Unterstützte Datentypen
- DynamoDB-Attribute aus dem Object Persistence-Modell von .NET
- DBContext Dynamo-Klasse aus dem .NET-Objektpersistenzmodell
- Optimistisches Sperren mit DynamoDB und dem AWS SDK für .NET Objektpersistenzmodell
- Zuordnen beliebiger Daten mit DynamoDB mithilfe des AWS SDK für .NET Objektpersistenzmodells

Unterstützte Datentypen

Das Object Persistence-Modell unterstützt eine Reihe primitiver .NET-Datentypen, Sammlungen sowie beliebiger Datentypen. Das Modell unterstützt die folgenden primitiven Datentypen.

- `bool`

- `byte`
- `char`
- `DateTime`
- `decimal`
- `double`
- `float`
- `Int16`
- `Int32`
- `Int64`
- `SByte`
- `string`
- `UInt16`
- `UInt32`
- `UInt64`

Das Objektpersistenzmodell unterstützt auch die .NET-Sammlungstypen. `DynamoDBContext`ist in der Lage, konkrete Sammlungstypen und einfache Plain Old CLR-Objekte () POCOs zu konvertieren.

In der folgenden Tabelle ist das Mapping der vorhergehenden .NET-Typen zu den DynamoDB-Typen zusammengefasst.

.NET-primitiver Typ	DynamoDB-Typ
Alle Zahlentypen	`N` (Zahlentyp)
Alle Zeichenfolgetypen	`S` (Zeichenfolgetyp)
MemoryStream, Byte []	`B` (Binärtyp)
bool	`N` (Zahlentyp) 0 repräsentiert False und 1 steht für True.
Sammlungstypen	`BS`-(Binärsatz)-Typ, `SS`-(Zeichenfolgesatz)-Typ und `NS`-(Zahlensatz)-Typ.

.NET-primitiver Typ	DynamoDB-Typ
DateTime	`S` (Zeichenfolgetyp). Die `DateTime`-Werte werden als ISO-8601-formatierte Zeichenfolgen gespeichert.

Das Object Persistence-Modell unterstützt außerdem beliebige Datentypen. Allerdings müssen Sie einen Konverter-Code bereitstellen, um die komplexen Typen den DynamoDB-Typen zuzuordnen.

> **Note**
>
> - Leere Binärwerte werden unterstützt.
> - Das Lesen von leeren Zeichenfolgenwerten wird unterstützt. Leere Zeichenfolgen-Attributwerte werden beim Schreiben zu DynamoDB in Attributwerten vom Typ Zeichenfolgensatz unterstützt. Leere Zeichenfolgen-Attributwerte des Zeichenfolgentyps und leere Zeichenfolgenwerte innerhalb des Listen- oder Map-Typs werden aus Schreibanforderungen gelöscht.

DynamoDB-Attribute aus dem Object Persistence-Modell von .NET

Dieser Abschnitt beschreibt die Attribute, die das Object-Persistence-Modell bereitstellt, damit Sie Ihre Klassen und Eigenschaften den DynamoDB-Tabellen und -Attributen zuordnen können.

> **Note**
>
> In den folgenden Attributen werden ausschließlich `DynamoDBTable` und `DynamoDBHashKey` benötigt.

Dynamo DBGlobal SecondaryIndexHashKey

Ordnet eine Klasseneigenschaft dem Partitionsschlüssel eines globalen sekundären Indexes zu. Verwenden Sie dieses Attribut, wenn Sie einen globalen sekundären Index `Query` möchten.

Dynamo DBGlobal SecondaryIndexRangeKey

Ordnet eine Klasseneigenschaft dem Sortierungsschlüssel eines globalen sekundären Indexes zu. Verwenden Sie dieses Attribut, wenn Sie eine `Query` für einen globalen sekundären Schlüssel ausführen müssen und die Ergebnisse mithilfe des Indexsortierschlüssels verfeinern möchten.

Dynamo-Schlüssel DBHash

Ordnet eine Klasseneigenschaft dem Partitionsschlüssel des Primärschlüssels der Tabelle zu. Die Primärschlüsselattribute können nicht ein Sammlungstyp sein.

Im folgenden C#-Codebeispiel werden die Klasse `Book` der Tabelle `ProductCatalog` und die Eigenschaft `Id` dem Partitionsschlüssel des Primärschlüssels der Tabelle zugeordnet.

```
[DynamoDBTable("ProductCatalog")]
public class Book
{
    [DynamoDBHashKey]
    public int Id { get; set; }

    // Additional properties go here.
}
```

Dynamo DBIgnore

Zeigt an, dass die zugeordnete Eigenschaft ignoriert werden sollte. Wenn Sie keine Klasseneigenschaften speichern möchten, können Sie dieses Attribut hinzufügen, um `DynamoDBContext` anzuweisen, diese Eigenschaft beim Speichern von Objekten zur Tabelle nicht einzufügen.

Dynamo DBLocal SecondaryIndexRangeKey

Ordnet eine Klasseneigenschaft dem Sortierungsschlüssel eines lokalen sekundären Indexes zu. Verwenden Sie dieses Attribut, wenn Sie eine `Query` für einen lokalen sekundären Index ausführen müssen und die Ergebnisse mithilfe des Indexsortierschlüssels verfeinern möchten.

Dynamo DBProperty

Ordnet eine Klasseneigenschaft einem Tabellenattribut zu. Wenn die Klasseneigenschaft einem Tabellenattribut mit dem gleichen Namen zugeordnet ist, müssen Sie dieses Attribut nicht angeben. Wenn die Namen jedoch nicht identisch sind, können Sie diesen Tag verwenden, um das Mapping

bereitzustellen. In der folgenden C#-Anweisung ordnet `DynamoDBProperty` die Eigenschaft `BookAuthors` dem Attribut `Authors` in der Tabelle zu.

```
[DynamoDBProperty("Authors")]
public List<string> BookAuthors { get; set; }
```

`DynamoDBContext` verwendet diese Mappinginformationen, um beim Speichern von Objektdaten zur entsprechenden Tabelle das Attribut `Authors` zu erstellen.

Dynamo DBRenamable

Gibt einen alternativen Namen für eine Klasseneigenschaft an. Dies eignet sich für das Schreiben eines benutzerdefinierten Konverters, um beliebige Daten einer DynamoDB-Tabelle, in der sich der Name der Klasseneigenschaft von dem des Tabellenattributs unterscheidet, zuzuordnen.

Dynamo-Schlüssel DBRange

Ordnet eine Klasseneigenschaft dem Sortierschlüssel des Primärschlüssels der Tabelle zu. Wenn die Tabelle über einen zusammengesetzten Primärschlüssel (Partitionsschlüssel und Sortierschlüssel) verfügt, müssen Sie sowohl das Attribut `DynamoDBHashKey` als auch das Attribut `DynamoDBRangeKey` im Klassenmapping angeben.

Die Beispieltabelle `Reply` verfügt über einen Primärschlüssel, der aus dem Partitionsschlüssel `Id` und dem Sortierschlüssel `Replenishment` besteht. Im folgenden C#-Codebeispiel wird die Klasse `Reply` der Tabelle `Reply` zugeordnet. Die Klassendefinition gibt ebenfalls an, dass zwei ihrer Eigenschaften den Primärschlüsseln zugeordnet werden.

```
[DynamoDBTable("Reply")]
public class Reply
{
   [DynamoDBHashKey]
   public int ThreadId { get; set; }
   [DynamoDBRangeKey]
   public string Replenishment { get; set; }

   // Additional properties go here.
}
```

Dynamo DBTable

Identifiziert die Zieltabelle in DynamoDB, zu der die Klasse zugeordnet wird. Im folgenden C#-Codebeispiel wird beispielsweise die Klasse `Developer` der Tabelle `People` in DynamoDB zugeordnet.

```
[DynamoDBTable("People")]
public class Developer { ...}
```

Dieses Attribut kann geerbt oder überschrieben werden.

- Das `DynamoDBTable`-Attribut kann geerbt werden. Wenn Sie im vorherigen Beispiel die neue Klasse `Lead` hinzufügen, die von der Klasse `Developer` erbt, wird diese ebenfalls der Tabelle `People` zugeordnet. Sowohl das Objekt `Developer` als auch das Objekt `Lead` werden in der Tabelle `People` gespeichert.
- Das `DynamoDBTable`-Attribut kann auch überschrieben werden. Im folgenden C#-Codebeispiel erbt die Klasse `Manager` von der Klasse `Developer`. Durch das explizite Hinzufügen des Attributs `DynamoDBTable` wird die Klasse jedoch einer anderen Tabelle (`Managers`) zugeordnet.

  ```
  [DynamoDBTable("Managers")]
  public class Manager : Developer { ...}
  ```

Sie können den optionalen Parameter `LowerCamelCaseProperties` hinzufügen, um DynamoDB aufzufordern, den ersten Buchstaben des Eigenschaftsnamens beim Speichern von Objekten in einer Tabelle klein zu schreiben wie im folgenden C#-Codebeispiel gezeigt.

```
[DynamoDBTable("People", LowerCamelCaseProperties=true)]
public class Developer
{
    string DeveloperName;
    ...
}
```

Beim Speichern von Instances der Klasse `Developer` speichert `DynamoDBContext` die Eigenschaft `DeveloperName` als `developerName`.

Dynamo DBVersion

Identifiziert eine Klasseneigenschaft für das Speichern der Versionsnummer des Elements. Weitere Informationen über das Versioning finden Sie unter Optimistisches Sperren mit DynamoDB und dem AWS SDK für .NET Objektpersistenzmodell.

DBContext Dynamo-Klasse aus dem .NET-Objektpersistenzmodell

Die `DynamoDBContext`-Klasse ist der Eintrittspunkt zu der Amazon-DynamoDB-Datenbank. Mit dieser Klasse verfügen Sie über eine Verbindung zu DynamoDB und können auf die Daten in unterschiedlichen Tabellen zugreifen, verschiedene CRUD-Operationen durchführen sowie Abfragen ausführen. Die Klasse `DynamoDBContext`-stellt folgende Methoden bereit.

Themen

- Erstellen MultiTable BatchGet
- Erschaffen MultiTable BatchWrite
- CreateBatchGet
- CreateBatchWrite
- Delete
- Dispose
- ExecuteBatchGet
- ExecuteBatchWrite
- FromDocument
- FromQuery
- FromScan
- GetTargetTable
- Load
- Query
- Save
- Scan
- ToDocument
- Optionale Parameter für Dynamo angeben DBContext

Erstellen MultiTable BatchGet

Erstellt ein `MultiTableBatchGet`-Objekt, das aus verschiedenen einzelnen `BatchGet`-Objekten besteht. Jedes dieser `BatchGet`-Objekte kann verwendet werden, um Elemente aus einer einzelnen DynamoDB-Tabelle abzurufen.

Um die Elemente aus Tabellen abzurufen, verwenden Sie die Methode `ExecuteBatchGet`, indem Sie das `MultiTableBatchGet`-Objekt als Parameter übergeben.

Erschaffen MultiTable BatchWrite

Erstellt ein `MultiTableBatchWrite`-Objekt, das aus verschiedenen einzelnen `BatchWrite`-Objekten besteht. Jedes dieser `BatchWrite`-Objekte kann für das Schreiben und das Löschen von Elementen in einer einzelnen DynamoDB-Tabelle verwendet werden.

Um zu Tabellen zu schreiben, verwenden Sie die Methode `ExecuteBatchWrite`, indem Sie das `MultiTableBatchWrite`-Objekt als Parameter übergeben.

CreateBatchGet

Erstellt ein `BatchGet`-Objekt, welches für das Abrufen mehrerer Elemente aus einer Tabelle verwenden werden kann.

CreateBatchWrite

Erstellt ein `BatchWrite`-Objekt, das Sie zum Ablegen mehrerer Elemente in einer Tabelle oder zum Löschen mehrerer Elemente aus einer Tabelle verwenden können.

Delete

Löscht ein Element aus der Tabelle. Die Methode erfordert den Primärschlüssel des Elements, das Sie löschen möchten. Sie können entweder den Primärschlüsselwert oder ein clientseitiges Objekt bereitstellen, das einen Primärschlüsselwert als Parameter für diese Methode enthält.

- Wenn Sie ein clientseitiges Objekt als Parameter angeben und die optimistische Sperre aktiviert haben, ist der Löschvorgang nur dann erfolgreich, wenn die clientseitigen und die serverseitigen Versionen des Objekts übereinstimmen.
- Wenn Sie lediglich den Primärschlüsselwert als Parameter angeben, ist der Löschvorgang erfolgreich, unabhängig davon, ob die optimistische Sperre aktiv ist oder nicht.

> **Note**
>
> Um diese Operation im Hintergrund auszuführen, nutzen Sie stattdessen die `DeleteAsync`-Methode.

Dispose

Entsorgt alle verwalteten und nicht verwalteten Ressourcen.

ExecuteBatchGet

Liest Daten aus einer oder mehreren Tabellen und verarbeitet alle `BatchGet`-Objekte in einem `MultiTableBatchGet`.

> **Note**
>
> Um diese Operation im Hintergrund auszuführen, nutzen Sie stattdessen die `ExecuteBatchGetAsync`-Methode.

ExecuteBatchWrite

Liest oder löscht Daten aus einer oder mehreren Tabellen und verarbeitet alle `BatchWrite`-Objekte in einem `MultiTableBatchWrite`.

> **Note**
>
> Um diese Operation im Hintergrund auszuführen, nutzen Sie stattdessen die `ExecuteBatchWriteAsync`-Methode.

FromDocument

Angesichts einer `Document`-Instance, gibt die `FromDocument`-Methode eine Instance der clientseitigen Klasse zurück.

Dies ist hilfreich, wenn Sie die Dokumentenmodell-Klassen zusammen mit dem Object Persistence-Modell verwenden möchten, um beliebige Datenoperationen durchzuführen. Weitere Hinweise zu den von der bereitgestellten Dokumentmodellklassen finden Sie unterArbeiten dem .NET-Dokumentmodell in DynamoDB. AWS SDK für .NET

Angenommen, Sie besitzen ein `Document`-Objekt mit dem Namen `doc`, das eine Darstellung eines `Forum`-Elements enthält. (Informationen zum Konstruieren dieses Objekts finden Sie in der Beschreibung der Methode `ToDocument` weiter unten in diesem Thema.) Sie können `FromDocument` verwenden, um das `Forum`-Element aus dem `Document` abzurufen, wie im folgendem C#-Codebeispiel gezeigt.

Example

```
forum101 = context.FromDocument<Forum>(101);
```

> **Note**
>
> Wenn das `Document`-Objekt die `IEnumerable`-Schnittstelle implementiert, können Sie stattdessen die `FromDocuments`-Methode nutzen. Auf diese Weise können Sie alle Klassen-Instances im `Document` durchlaufen.

FromQuery

Führt eine `Query`-Operation mit den Abfrageparametern aus, die in einem `QueryOperationConfig`-Objekt definiert sind.

> **Note**
>
> Um diese Operation im Hintergrund auszuführen, nutzen Sie stattdessen die `FromQueryAsync`-Methode.

FromScan

Führt eine `Scan`-Operation mit den Scan-Parametern aus, die in einem `ScanOperationConfig`-Objekt definiert sind.

> **Note**
>
> Um diese Operation im Hintergrund auszuführen, nutzen Sie stattdessen die `FromScanAsync`-Methode.

GetTargetTable

Ruft die Zieltabelle für den angegebenen Typ ab. Dies ist nützlich, wenn Sie einen benutzerdefinierten Konverter schreiben, der einer DynamoDB-Tabelle beliebige Daten zuordnet, und ermitteln müssen, welche Tabelle einem benutzerdefinierten Datentyp zugeordnet ist.

Load

Ruft ein Element aus einer Tabelle ab. Die Methode erfordert lediglich den Primärschlüssel des Elements, das Sie abrufen möchten.

Standardmäßig gibt DynamoDB das Element mit Werten die Eventually Consistent sind zurück. Weitere Informationen zum Eventual Consistency-Modell finden Sie unter DynamoDB-Lesekonsistenz.

`Load`oder `LoadAsync` Methode ruft den GetItemVorgang auf, bei dem Sie den Primärschlüssel für die Tabelle angeben müssen. Da der `IndexName`-Parameter von `GetItem` ignoriert wird, können Sie ein Element nicht mithilfe der Partition oder des Sortierschlüssels eines Indexes laden. Daher müssen Sie den Primärschlüssel der Tabelle verwenden, um ein Element zu laden.

> **Note**
>
> Um diese Operation im Hintergrund auszuführen, nutzen Sie stattdessen die `LoadAsync`-Methode. Ein Beispiel für die Verwendung der `LoadAsync`-Methode zur Ausführung von CRUD-Vorgängen auf hoher Ebene in einer DynamoDB-Tabelle finden Sie im Folgenden.

```
/// <summary>
/// Shows how to perform high-level CRUD operations on an Amazon DynamoDB
/// table.
/// </summary>
public class HighLevelItemCrud
{
    public static async Task Main()
    {
        var client = new AmazonDynamoDBClient();
        DynamoDBContext context = new DynamoDBContext(client);
        await PerformCRUDOperations(context);
    }
```

```
        public static async Task PerformCRUDOperations(IDynamoDBContext context)
        {
            int bookId = 1001; // Some unique value.
            Book myBook = new Book
            {
                Id = bookId,
                Title = "object persistence-AWS SDK for.NET SDK-Book 1001",
                Isbn = "111-1111111001",
                BookAuthors = new List<string> { "Author 1", "Author 2" },
            };

            // Save the book to the ProductCatalog table.
            await context.SaveAsync(myBook);

            // Retrieve the book from the ProductCatalog table.
            Book bookRetrieved = await context.LoadAsync<Book>(bookId);

            // Update some properties.
            bookRetrieved.Isbn = "222-2222221001";

            // Update existing authors list with the following values.
            bookRetrieved.BookAuthors = new List<string> { " Author 1", "Author x" };
            await context.SaveAsync(bookRetrieved);

            // Retrieve the updated book. This time, add the optional
            // ConsistentRead parameter using DynamoDBContextConfig object.
            await context.LoadAsync<Book>(bookId, new DynamoDBContextConfig
            {
                ConsistentRead = true,
            });

            // Delete the book.
            await context.DeleteAsync<Book>(bookId);

            // Try to retrieve deleted book. It should return null.
            Book deletedBook = await context.LoadAsync<Book>(bookId, new
DynamoDBContextConfig
            {
                ConsistentRead = true,
            });

            if (deletedBook == null)
            {
                Console.WriteLine("Book is deleted");
```

```
            }
        }
    }
```

Query

Abfragen einer Tabelle basierend auf den Abfrageparametern, die Sie bereitstellen.

Sie können eine Tabelle nur dann abfragen, wenn sie über einen zusammengesetzten Primärschlüssel verfügt (Partitionsschlüssel und Sortierschlüssel). Bei der Abfrage müssen Sie einen Partitionsschlüssel und eine Bedingung angeben, die für den Sortierschlüssel gilt.

Angenommen, Sie verfügen über die clientseitige Klasse `Reply`, die der Tabelle `Reply` in DynamoDB zugeordnet ist. Das folgende C#-Codebeispiel fragt die Tabelle `Reply` ab, um alle Forum-Thread-Antworten zu finden, die in den letzten 15 Tagen veröffentlicht wurden. Die Tabelle `Reply` verfügt über einen Primärschlüssel, der aus einem Partitionsschlüssel (`Id`) und Sortierschlüssel (`ReplyDateTime`) besteht.

Example

```
DynamoDBContext context = new DynamoDBContext(client);

string replyId = "DynamoDB#DynamoDB Thread 1"; //Partition key
DateTime twoWeeksAgoDate = DateTime.UtcNow.Subtract(new TimeSpan(14, 0, 0, 0)); // Date
 to compare.
IEnumerable<Reply> latestReplies = context.Query<Reply>(replyId,
 QueryOperator.GreaterThan, twoWeeksAgoDate);
```

Dies gibt eine Sammlung von `Reply`-Objekten zurück.

Die Methode `Query` gibt standardmäßig eine „lazy-loaded“ `IEnumerable`-Sammlung zurück. Sie gibt anfänglich ausschließlich eine Ergebnisseite zurück und führt dann bei Bedarf einen Dienstaufruf für die nächste Seite durch. Um alle übereinstimmenden Elemente zu erhalten, müssen Sie lediglich die `IEnumerable`-Sammlung durchlaufen.

Wenn die Tabelle über einen einfachen Primärschlüssel (Partitionsschlüssel) verfügt, können Sie die Methode `Query` nicht verwenden. Stattdessen können Sie die Methode `Load` verwenden und den Partitionsschlüssel bereitstellen, um das Element abzurufen.

> **Note**
>
> Um diese Operation im Hintergrund auszuführen, nutzen Sie stattdessen die `QueryAsync`-Methode.

Save

Speichert das angegebene Objekt in der Tabelle. Wenn der im Eingabeobjekt angegebene Primärschlüssel in der Tabelle nicht vorhanden ist, fügt die Methode der Tabelle ein neues Element hinzu. Wenn der Primärschlüssel vorhanden ist, aktualisiert die Methode das vorhandene Element.

Wenn Sie eine optimistische Sperre konfiguriert haben, ist die Aktualisierung nur dann erfolgreich, wenn die client- und serverseitigen Versionen des Elements übereinstimmen. Weitere Informationen finden Sie unter Optimistisches Sperren mit DynamoDB und dem AWS SDK für .NET Objektpersistenzmodell.

> **Note**
>
> Um diese Operation im Hintergrund auszuführen, nutzen Sie stattdessen die `SaveAsync`-Methode.

Scan

Führt einen gesamten Tabellen-Scan durch.

Sie können Scan-Ergebnisse filtern, indem Sie eine Scan-Bedingung angeben. Die Bedingung kann auf jedem Attribut in der Tabelle ausgewertet werden. Angenommen, Sie verfügen über die clientseitige Klasse `Book`, die der Tabelle `ProductCatalog` in DynamoDB zugeordnet ist. Im folgenden C#-Beispiel werden die Tabelle gescannt und ausschließlich die Book-Elemente zurückgegeben, deren Preis kleiner als 0 ist.

Example

```
IEnumerable<Book> itemsWithWrongPrice = context.Scan<Book>(
                    new ScanCondition("Price", ScanOperator.LessThan, price),
                    new ScanCondition("ProductCategory", ScanOperator.Equal, "Book")
      );
```

Die Methode `Scan` gibt standardmäßig eine „lazy-loaded“ `IEnumerable`-Sammlung zurück. Sie gibt anfänglich ausschließlich eine Ergebnisseite zurück und führt dann bei Bedarf einen Dienstaufruf für die nächste Seite durch. Um alle übereinstimmenden Elemente zu erhalten, müssen Sie lediglich die `IEnumerable`-Sammlung durchlaufen.

Aus Leistungsgründen sollten Sie Ihre Tabellen abfragen und einen Tabellen-Scan vermeiden.

> **Note**
>
> Um diese Operation im Hintergrund auszuführen, nutzen Sie stattdessen die `ScanAsync`-Methode.

ToDocument

Gibt eine Instance der `Document`-Dokumentenmodell-Klasse aus Ihrer Klassen-Instance zurück.

Dies ist hilfreich, wenn Sie die Dokumentenmodell-Klassen zusammen mit dem Object Persistence-Modell verwenden möchten, um beliebige Datenoperationen durchzuführen. Weitere Hinweise zu den von der bereitgestellten Dokumentmodellklassen finden Sie unterArbeiten dem .NET-Dokumentmodell in DynamoDB. AWS SDK für .NET

Angenommen, Sie verfügen über eine clientseitige Klasse, die der Tabelle `Forum` zugeordnet ist. Sie können anschließend einen `DynamoDBContext` verwenden, um ein Element als `Document`-Objekt aus der Tabelle `Forum` abzurufen, wie im folgenden C#-Codebeispiel gezeigt.

Example

```
DynamoDBContext context = new DynamoDBContext(client);

Forum forum101 = context.Load<Forum>(101); // Retrieve a forum by primary key.
Document doc = context.ToDocument<Forum>(forum101);
```

Optionale Parameter für Dynamo angeben DBContext

Wenn Sie das Object Persistence-Modell verwenden, können Sie für den `DynamoDBContext` folgende optionale Parameter angeben.

- **`ConsistentRead`** – Beim Abrufen von Daten mithilfe der `Load`, `Query` oder `Scan`-Operation können Sie diesen Parameter optional hinzufügen, um die neuesten Werte der Daten anzufordern.

- **`IgnoreNullValues`** – Dieser Parameter teilt `DynamoDBContext` mit, Null-Werte von Attributen während einer `Save`-Operation zu ignorieren. Wenn dieser Parameter „false“ oder nicht festgelegt ist, wird ein Nullwert als Anweisung interpretiert, das betreffende Attribut zu löschen.
- **`SkipVersionCheck`**— Dieser Parameter teilt `DynamoDBContext` mit, beim Speichern oder Löschen von Elementen keine Versionen zu vergleichen. Weitere Informationen über das Versioning finden Sie unter Optimistisches Sperren mit DynamoDB und dem AWS SDK für .NET Objektpersistenzmodell.
- **`TableNamePrefix`** – Stellt allen Tabellennamen eine bestimmte Zeichenfolge voran. Wenn dieser Parameter Null ist (oder nicht festgelegt ist), dann wird kein Präfix verwendet.
- `DynamoDBEntryConversion` – Gibt das Konvertierungsschema an, das vom Client verwendet wird. Sie können diesen Parameter auf Version V1 oder V2 festlegen. V1 ist die Standardversion.

  Je nach der von Ihnen festgelegten Version ändert sich das Verhalten dieses Parameters. Beispiel:

  - In V1 wird der Datentyp `bool` in den Zahlentyp `N` konvertiert, wobei 0 für falsch und 1 für wahr steht. In V2 wird `bool` in `BOOL` konvertiert.
  - In V2 werden Listen und Arrays nicht zusammen mit gruppiert. HashSets Listen und Arrays mit numerischen, auf Zeichenketten basierenden Typen und binären Typen werden in den Typ `L` (List) konvertiert, der leer gesendet werden kann, um eine Liste zu aktualisieren. Dies ist anders als bei V1, wo das Senden einer leeren Liste nicht möglich ist.

    In V1 werden Sammlungstypen wie List und Arrays gleich behandelt. HashSet Liste und HashSet Array von Zahlen werden in den Typ `NS` (Zahlensatz) konvertiert.

  Im folgenden Beispiel wird die Version des Konvertierungsschemas auf V2 festgelegt, wodurch das Konvertierungsverhalten zwischen .NET-Typen und DynamoDB-Datentypen geändert wird.

```
var config = new DynamoDBContextConfig
{
    Conversion = DynamoDBEntryConversion.V2
};
var contextV2 = new DynamoDBContext(client, config);
```

Im folgenden C#-Codebeispiel wird durch Angabe von zwei der vorherigen optionalen Parameter, `ConsistentRead` und `SkipVersionCheck`, ein neuer Parameter `DynamoDBContext` erstellt.

Example

```
AmazonDynamoDBClient client = new AmazonDynamoDBClient();
...
DynamoDBContext context =
       new DynamoDBContext(client, new DynamoDBContextConfig { ConsistentRead = true,
 SkipVersionCheck = true});
```

`DynamoDBContext` berücksichtigt diese optionalen Parameter bei jeder Anforderung, die unter Verwendung dieses Kontextes gesendet werden.

Anstatt diese Parameter auf der `DynamoDBContext`-Ebene festzulegen, können Sie diese mithilfe von `DynamoDBContext` für einzelne von Ihnen ausgeführte Operationen festlegen, wie im folgenden C#-Codebeispiel gezeigt. Das Beispiel lädt ein bestimmtes Book-Element. Die `Load`-Methode von `DynamoDBContext` gibt die optionalen Parameter `ConsistentRead` und `SkipVersionCheck` an.

Example

```
AmazonDynamoDBClient client = new AmazonDynamoDBClient();
...
DynamoDBContext context = new DynamoDBContext(client);
Book bookItem = context.Load<Book>(productId,new DynamoDBContextConfig{ ConsistentRead
 = true, SkipVersionCheck = true });
```

In diesem Fall berücksichtigt `DynamoDBContext` diese Parameter ausschließlich beim Senden der `Get`-Anforderung.

Optimistisches Sperren mit DynamoDB und dem AWS SDK für .NET Objektpersistenzmodell

Die Unterstützung der optimistischen Sperre im Object Persistence-Modell stellt sicher, dass die Elementversion für Ihre Anwendung dieselbe ist wie die serverseitige Elementversion, bevor das Element aktualisiert oder gelöscht wird. Angenommen, Sie rufen ein Element für eine Aktualisierung ab. Bevor Sie jedoch Ihre Aktualisierungen zurücksenden, aktualisiert eine andere Anwendung das gleiche Element. Jetzt verfügt Ihre Anwendung über eine veraltete Kopie des Elements. Ohne optimistische Sperre wird jede von Ihnen durchgeführte Aktualisierung die Aktualisierung von anderen Anwendungen überschreiben.

Die Funktion "optimistische Sperre" des Object Persistence-Modells stellt den `DynamoDBVersion`-Tag bereit, den Sie für die Aktivierung der optimistische Sperre nutzen können. Um diese Funktion zu verwenden, fügen Sie Ihrer Klasse eine Eigenschaft hinzu, mit der die Versionsnummer gespeichert

wird. Sie fügen der Eigenschaft das Attribut `DynamoDBVersion` hinzu. Wenn Sie das Element zum ersten Mal speichern, weist `DynamoDBContext` diesem eine Versionsnummer hinzu. Dieser Wert wird bei jeder Aktualisierung des Elements inkrementell erhöht.

Ihre Aktualisierungs- oder Löschanforderungen werden nur erfolgreich ausgeführt, wenn die clientseitige Objektversion mit der entsprechenden Versionsnummer des Elements auf der Serverseite übereinstimmt. Wenn Ihre Anwendung über eine veraltete Kopie verfügt, muss sie die aktuelle Version vom Server erhalten, bevor sie das Element aktualisieren oder löschen kann.

Im folgenden C#-Codebeispiel wird die Klasse `Book` mit Object Persistence-Attributen definiert, die diese der Tabelle `ProductCatalog` zuordnen. Die Eigenschaft `VersionNumber` in der Klasse, die das `DynamoDBVersion`-Attribut aufweist, speichert den Wert der Versionsnummer.

Example

```
[DynamoDBTable("ProductCatalog")]
  public class Book
  {
    [DynamoDBHashKey]   //Partition key
    public int Id { get; set; }
    [DynamoDBProperty]
    public string Title { get; set; }
    [DynamoDBProperty]
    public string ISBN { get; set; }
    [DynamoDBProperty("Authors")]
    public List<string> BookAuthors { get; set; }
    [DynamoDBVersion]
    public int? VersionNumber { get; set; }
  }
```

> **Note**
>
> Sie können das `DynamoDBVersion`-Attribut ausschließlich auf einen löschbaren numerischen primitiven Typen anwenden (z. B. `int?`).

Die optimistische Sperre hat folgende Auswirkungen auf diese `DynamoDBContext`-Operationen:

- Für ein neues Element weist `DynamoDBContext` die Erstversionsnummer 0 zu. Wenn Sie ein vorhandenes Element abrufen, eine oder mehrere von dessen Eigenschaften

aktualisieren und versuchen, die Änderungen zu speichern, wird die Speicheroperation nur dann erfolgreich ausgeführt, wenn die client- und serverseitige Versionsnummer übereinstimmen. `DynamoDBContext` erhöht die Versionsnummer inkrementell. Sie müssen die Versionsnummer nicht festlegen.

- Die Methode `Delete` stellt Überladungen bereit, die entweder einen Primärschlüsselwert oder ein Objekt als Parameter aufnehmen können, wie im folgenden C#-Codebeispiel gezeigt.

  Example

  ```
  DynamoDBContext context = new DynamoDBContext(client);
  ...
  // Load a book.
  Book book = context.Load<ProductCatalog>(111);
  // Do other operations.
  // Delete 1 - Pass in the book object.
  context.Delete<ProductCatalog>(book);

  // Delete 2 - Pass in the Id (primary key)
  context.Delete<ProductCatalog>(222);
  ```

  Wenn Sie ein Objekt als Parameter bereitstellen, ist der Löschvorgang nur dann erfolgreich, wenn die Objektversion mit der entsprechenden serverseitigen Elementversion übereinstimmt. Wenn Sie jedoch einen Primärschlüsselwert als Parameter bereitstellen, erkennt `DynamoDBContext` keine Versionsnummern und löscht das Element, ohne die Version zu prüfen.

  Beachten Sie, dass die interne Implementierung von optimistischen Sperren in dem Code des Object-Persistence-Modells bedingte Aktualisierungen und bedingte Löschungs-API-Aktionen in DynamoDB verwendet.

Deaktivieren der optimistischen Sperre

Um die optimistische Sperre zu deaktivieren, verwenden Sie die Konfigurationseigenschaft `SkipVersionCheck`. Sie können diese Eigenschaft bei der Erstellung von `DynamoDBContext` festlegen. In diesem Fall wird die optimistische Sperre für alle Anforderungen deaktiviert, die Sie unter Verwendung des Kontextes ausführen. Weitere Informationen finden Sie unter Optionale Parameter für Dynamo angeben DBContext.

Anstatt die Eigenschaft auf der Kontextebene festzulegen, können Sie die optimistische Sperre für eine bestimmte Operation deaktivieren wie im folgenden C#-Codebeispiel gezeigt. Im Beispiel wird

der Kontext verwendet, um ein Book-Element zu löschen. Die Methode `Delete` legt die optionale Eigenschaft `SkipVersionCheck` auf „true“ fest und deaktiviert damit die Versionsprüfung.

Example

```
DynamoDBContext context = new DynamoDBContext(client);
// Load a book.
Book book = context.Load<ProductCatalog>(111);
...
// Delete the book.
context.Delete<Book>(book, new DynamoDBContextConfig { SkipVersionCheck = true });
```

Zuordnen beliebiger Daten mit DynamoDB mithilfe des AWS SDK für .NET Objektpersistenzmodells

Zusätzlich zu den unterstützten .NET-Typen (siehe Unterstützte Datentypen) können Sie Typen in Ihrer Anwendung verwenden, für die es kein direktes Mapping zu Amazon-DynamoDB-Typen gibt. Das Object-Persistence-Modell unterstützt das Speichern von Daten mit beliebigen Typen, solange Sie einen Konverter bereitstellen, um Daten des beliebigen Typs in den DynamoDB-Typ und umgekehrt zu konvertieren. Der Konverter-Code wandelt Daten während des Speicherns und Ladens der Objekte um.

Sie können clientseitig alle Typen erstellen. Die in den Tabellen gespeicherten Daten haben jedoch einen DynamoDB-Datentyp. Während Abfragen und Scans werden alle Datenvergleiche anhand der in DynamoDB gespeicherten Daten ausgeführt.

Das folgende C#-Codebeispiel definiert eine `Book`-Klasse mit den Eigenschaften `Id`, `Title`, `ISBN` und `Dimension`. Die Eigenschaft `Dimension` gehört zum `DimensionType`, der die Eigenschaften `Height`, `Width` und `Thickness` beschreibt. Der Beispielcode stellt die Konverter-Methoden `ToEntry` und `FromEntry` bereit, um Daten zwischen den Zeichenfolgetypen `DimensionType` und DynamoDB zu konvertieren. Beispielsweise erstellt der Konverter beim Speichern einer `Book`-Instance eine `Dimension`-Zeichenfolge wie „8.5x11x.05“. Wenn Sie ein Book abrufen, wird die Zeichenfolge in eine `DimensionType`-Instance konvertiert.

Im Beispiel wird der `Book`-Typ der Tabelle `ProductCatalog` zugewiesen. Es speichert eine `Book`-Beispiel-Instance, ruft sie ab, aktualisiert ihre Abmessungen und speichert die aktualisierte `Book`-Instance erneut.

step-by-stepAnweisungen zum Testen des folgenden Beispiels finden Sie unter. .NET-Codebeispiele

Example

```
using System;
using System.Collections.Generic;
using Amazon.DynamoDBv2;
using Amazon.DynamoDBv2.DataModel;
using Amazon.DynamoDBv2.DocumentModel;
using Amazon.Runtime;
using Amazon.SecurityToken;

namespace com.amazonaws.codesamples
{
    class HighLevelMappingArbitraryData
    {
        private static AmazonDynamoDBClient client = new AmazonDynamoDBClient();

        static void Main(string[] args)
        {
            try
            {
                DynamoDBContext context = new DynamoDBContext(client);

                // 1. Create a book.
                DimensionType myBookDimensions = new DimensionType()
                {
                    Length = 8M,
                    Height = 11M,
                    Thickness = 0.5M
                };

                Book myBook = new Book
                {
                    Id = 501,
                    Title = "AWS SDK for .NET Object Persistence Model Handling
 Arbitrary Data",
                    ISBN = "999-9999999999",
                    BookAuthors = new List<string> { "Author 1", "Author 2" },
                    Dimensions = myBookDimensions
                };

                context.Save(myBook);

                // 2. Retrieve the book.
                Book bookRetrieved = context.Load<Book>(501);
```

```
            // 3. Update property (book dimensions).
            bookRetrieved.Dimensions.Height += 1;
            bookRetrieved.Dimensions.Length += 1;
            bookRetrieved.Dimensions.Thickness += 0.2M;
            // Update the book.
            context.Save(bookRetrieved);

            Console.WriteLine("To continue, press Enter");
            Console.ReadLine();
        }
        catch (AmazonDynamoDBException e) { Console.WriteLine(e.Message); }
        catch (AmazonServiceException e) { Console.WriteLine(e.Message); }
        catch (Exception e) { Console.WriteLine(e.Message); }
    }
}
[DynamoDBTable("ProductCatalog")]
public class Book
{
    [DynamoDBHashKey] //Partition key
    public int Id
    {
        get; set;
    }
    [DynamoDBProperty]
    public string Title
    {
        get; set;
    }
    [DynamoDBProperty]
    public string ISBN
    {
        get; set;
    }
    // Multi-valued (set type) attribute.
    [DynamoDBProperty("Authors")]
    public List<string> BookAuthors
    {
        get; set;
    }
    // Arbitrary type, with a converter to map it to DynamoDB type.
    [DynamoDBProperty(typeof(DimensionTypeConverter))]
    public DimensionType Dimensions
    {
```

```
            get; set;
        }
    }

    public class DimensionType
    {
        public decimal Length
        {
            get; set;
        }
        public decimal Height
        {
            get; set;
        }
        public decimal Thickness
        {
            get; set;
        }
    }

    // Converts the complex type DimensionType to string and vice-versa.
    public class DimensionTypeConverter : IPropertyConverter
    {
        public DynamoDBEntry ToEntry(object value)
        {
            DimensionType bookDimensions = value as DimensionType;
            if (bookDimensions == null) throw new ArgumentOutOfRangeException();

            string data = string.Format("{1}{0}{2}{0}{3}", " x ",
                              bookDimensions.Length, bookDimensions.Height,
bookDimensions.Thickness);

            DynamoDBEntry entry = new Primitive
            {
                Value = data
            };
            return entry;
        }

        public object FromEntry(DynamoDBEntry entry)
        {
            Primitive primitive = entry as Primitive;
            if (primitive == null || !(primitive.Value is String) ||
string.IsNullOrEmpty((string)primitive.Value))
```

```
                throw new ArgumentOutOfRangeException();

            string[] data = ((string)(primitive.Value)).Split(new string[] { " x " },
 StringSplitOptions.None);
            if (data.Length != 3) throw new ArgumentOutOfRangeException();

            DimensionType complexData = new DimensionType
            {
                Length = Convert.ToDecimal(data[0]),
                Height = Convert.ToDecimal(data[1]),
                Thickness = Convert.ToDecimal(data[2])
            };
            return complexData;
        }
    }
}
```

Ausführen der Codebeispiele in diesem Entwicklerhandbuch

AWS SDKs Sie bieten umfassende Unterstützung für Amazon DynamoDB in den folgenden Sprachen:

- Java
- JavaScript im Browser
- .NET
- Node.js
- PHP
- Python
- Ruby
- C++
- Go
- Android
- iOS

Die Codebeispiele in diesem Entwicklerhandbuch stellen umfassendere Informationen zu DynamoDB-Operationen unter Verwendung der folgenden Programmiersprachen bereit:

- Java-Codebeispiele
- .NET-Codebeispiele

Bevor Sie mit dieser Übung beginnen können, müssen Sie ein AWS Konto erstellen, Ihren Zugriffsschlüssel und Ihren geheimen Schlüssel abrufen und AWS Command Line Interface (AWS CLI) auf Ihrem Computer einrichten. Weitere Informationen finden Sie unter Einrichten von DynamoDB (Webservice).

> **Note**
>
> Wenn Sie die herunterladbare Version von DynamoDB verwenden, müssen Sie die verwenden, AWS CLI um die Tabellen und Beispieldaten zu erstellen. Sie müssen den `--endpoint-url` Parameter auch bei jedem AWS CLI Befehl angeben. Weitere Informationen finden Sie unter Festlegen des lokalen Endpunkts.

Erstellen von Tabellen und Laden von Daten für Codebeispiele in DynamoDB

Im Folgenden finden Sie die Grundlagen zum Erstellen von Tabellen in DynamoDB, zum Laden eines Beispiel-Datensatzes, zum Abfragen der Daten und zum Aktualisieren der Daten.

- Schritt 1: Erstellen einer Tabelle in DynamoDB
- Schritt 2: Schreiben von Daten in eine DynamoDB-Tabelle
- Schritt 3: Lesen von Daten aus einer DynamoDB-Tabelle
- Schritt 4: Aktualisieren von Daten in einer DynamoDB-Tabelle

Java-Codebeispiele

Themen

- Java: Deine AWS Zugangsdaten einrichten
- Java: AWS Region und Endpunkt festlegen

Dieses Entwicklerhandbuch enthält Java-Codefragmente und ready-to-run -Programme. Sie finden diese Codebeispiele in den folgenden Abschnitten:

- Arbeiten mit Elementen und Attributen in DynamoDB

- Arbeiten mit Tabellen und Daten in DynamoDB
- Abfragen von Tabellen in DynamoDB
- Scannen von Tabellen in DynamoDB
- Verbessern des Datenzugriffs mit sekundären Indizes in DynamoDB
- Java 1.x: Dynamo DBMapper
- Ändern Sie die Datenerfassung für DynamoDB Streams

Sie können sofort beginnen, indem Sie Eclipse mit dem AWS Toolkit for Eclipse verwenden. Neben einer IDE mit vollem Funktionsumfang erhalten Sie auch die AWS SDK für Java mit automatischen Updates und vorkonfigurierten Vorlagen für die Erstellung von Anwendungen. AWS

So führen Sie Java-Codebeispiele (mit Eclipse) aus

1. Laden Sie die Eclipse-IDE herunter und installieren Sie sie.
2. Laden Sie das AWS Toolkit for Eclipse herunter und installieren Sie es.
3. Starten Sie Eclipse und wählen Sie im Eclipse-Menü File (Datei), New (Neu) und anschließend Other (Sonstiges) aus.
4. Wählen Sie unter Assistent auswählen AWS aus, wählen Sie AWS Java-Projekt und dann Weiter aus.
5. Gehen Sie unter Create an AWS Java wie folgt vor:

    a. Geben Sie im Feld Projektname einen Namen für Ihr Projekt an.

    b. Wählen Sie in Select Account (Konto wählen) das Anmeldeinformationsprofil in der Liste aus.

    Wenn Sie das zum ersten Mal verwenden AWS Toolkit for Eclipse, wählen Sie AWS Konten konfigurieren, um Ihre AWS Anmeldeinformationen einzurichten.
6. Wählen Sie Finish aus, um das Projekt zu erstellen.
7. Wählen Sie im Eclipse-Menü File, New und anschließend Class aus.
8. Geben Sie in Java Class (Java-Klasse) unter Name (Name) einen Namen für Ihre Klasse ein (verwenden Sie denselben Namen wie das Codebeispiel, das Sie ausführen möchten). Wählen Sie dann Finish (Fertigstellen) aus, um die Klasse zu erstellen.
9. Kopieren Sie das Codebeispiel aus der Dokumentationsseite in den Eclipse-Editor.
10. Wählen Sie im Eclipse-Menü Run (Ausführen) aus, um den Code auszuführen.

Das SDK für Java stellt threadsichere Clients für die Arbeit mit DynamoDB bereit. Als bewährte Methode sollten Ihre Anwendungen einen Client erstellen und diesen zwischen den Threads wiederverwenden.

Weitere Informationen hierzu finden Sie unter AWS SDK für Java.

> **Note**
>
> Die Codebeispiele in diesem Handbuch sind für die Verwendung mit der neuesten Version des AWS SDK für Java vorgesehen.
> Wenn Sie das verwenden AWS Toolkit for Eclipse, können Sie automatische Updates für das SDK for Java konfigurieren. Gehen Sie dazu in Eclipse zu Einstellungen und wählen Sie „AWS Toolkit SDKs Automatisch neu herunterladen“. AWS SDK für Java

Java: Deine AWS Zugangsdaten einrichten

Das SDK for Java erfordert, dass Sie zur Laufzeit AWS Anmeldeinformationen für Ihre Anwendung angeben. Bei den Codebeispielen in diesem Handbuch wird davon ausgegangen, dass Sie eine AWS Anmeldeinformationsdatei verwenden, wie unter Einrichten Ihrer AWS Anmeldeinformationen im AWS SDK für Java Entwicklerhandbuch beschrieben.

Im Folgenden finden Sie ein Beispiel für eine AWS Anmeldeinformationsdatei mit dem Namen`~/.aws/credentials`, wobei die Tilde (~) für Ihr Home-Verzeichnis steht.

```
[default]
aws_access_key_id = AWS access key ID goes here
aws_secret_access_key = Secret key goes here
```

Java: AWS Region und Endpunkt festlegen

Standardmäßig greifen die Codebeispiele auf DynamoDB in der Region USA West (Oregon) auf. Sie können die Region ändern, indem Sie die `AmazonDynamoDB`-Eigenschaften ändern.

Im folgenden Codebeispiel wird ein neuer `AmazonDynamoDB` instanziiert.

```
import software.amazon.dynamodb.AmazonDynamoDBClientBuilder;
import com.amazonaws.regions.Regions;
...
// This client will default to US West (Oregon)
```

```
AmazonDynamoDB client = AmazonDynamoDBClientBuilder.standard()
.withRegion(Regions.US_WEST_2)
.build();
```

Sie können die Methode `withRegion` verwenden, um den Code für DynamoDB in jeder verfügbaren Region auszuführen. Die vollständige Liste finden Sie unter AWS -Regionen und -Endpunkte in der Allgemeine Amazon Web Services-Referenz.

Wenn Sie die Codebeispiele mit DynamoDB lokal auf Ihrem Computer ausführen möchten, müssen Sie den Endpunkt wie folgt festlegen.

AWS SDK V1

```
AmazonDynamoDB client =
 AmazonDynamoDBClientBuilder.standard().withEndpointConfiguration(
new AwsClientBuilder.EndpointConfiguration("http://localhost:8000", "us-west-2"))
.build();
```

AWS SDK V2

```
DynamoDbClient client = DynamoDbClient.builder()
    .endpointOverride(URI.create("http://localhost:8000"))
    // The region is meaningless for local DynamoDb but required for client builder
 validation
    .region(Region.US_EAST_1)
    .credentialsProvider(StaticCredentialsProvider.create(
    AwsBasicCredentials.create("dummy-key", "dummy-secret")))
    .build();
```

.NET-Codebeispiele

Themen

- .NET: Ihre Anmeldeinformationen einrichten AWS
- .NET: AWS Region und Endpunkt festlegen

Dieses Handbuch enthält.NET-Codefragmente und -Programme. ready-to-run Sie finden diese Codebeispiele in den folgenden Abschnitten:

- Arbeiten mit Elementen und Attributen in DynamoDB

- Arbeiten mit Tabellen und Daten in DynamoDB
- Abfragen von Tabellen in DynamoDB
- Scannen von Tabellen in DynamoDB
- Verbessern des Datenzugriffs mit sekundären Indizes in DynamoDB
- Arbeiten dem .NET-Dokumentmodell in DynamoDB
- Arbeiten mit dem Object Persistence-Modell von .NET und DynamoDB
- Ändern Sie die Datenerfassung für DynamoDB Streams

Sie können schnell loslegen, indem Sie das AWS SDK für .NET mit dem Toolkit for Visual Studio verwenden.

So führen Sie die .NET-Codebeispiele (mit Visual Studio) aus

1. Laden Sie Microsoft Visual Studio herunter und installieren Sie es.
2. Downloaden Sie das Toolkit for Visual Studio herunter und installieren Sie es.
3. Starten Sie Visual Studio. Wählen Sie File (Datei), New (Neu) und Project (Projekt) aus.
4. Wählen Sie unter Neues Projekt die Option AWS Leeres Projekt und dann OK aus.
5. Wählen Sie in AWS Zugangsdaten die Option Vorhandenes Profil verwenden und anschließend Ihre Anmeldeinformationen aus der Liste und OK aus.

   Wenn Sie Toolkit for Visual Studio zum ersten Mal verwenden, wählen Sie Neues Profil verwenden, um Ihre AWS Anmeldeinformationen einzurichten.
6. Wählen Sie in dem Visual Studio-Projekt die Registerkarte für den Quellcode des Programms aus (`Program.cs`). Kopieren Sie das Codebeispiel von der Dokumentationsseite in den Visual Studio-Editor unter Ersetzung aller anderen Codes, die im Editor angezeigt werden.
7. Wenn Sie Fehlermeldungen der Form Der Typ- oder Namespace-Name... konnte nicht gefunden werden sehen, müssen Sie die AWS SDK-Assembly für DynamoDB wie folgt installieren:

   a. Öffnen Sie im Projektmappen-Explorer das Kontextmenü (Rechtsklick) für Ihr Projekt und wählen Sie dann Pakete verwalten aus. NuGet
   b. Wählen Sie im NuGet Package Manager die Option Durchsuchen aus.
   c. Geben Sie im Suchfeld **`AWSSDK.DynamoDBv2`** ein und warten Sie, bis die Suche abgeschlossen ist.
   d. Wählen Sie AWSSDK.Dynamo DBv2 und dann Installieren.

e. Wenn die Installation abgeschlossen ist, wählen Sie die Registerkarte Program.cs aus, um zum Programm zurückzukehren.

8. Wählen Sie auf der Visual Studio-Symbolleiste die Schaltfläche Start (Starten) aus, um den Code auszuführen.

Der SDK für .NET bietet Thread-sichere Clients für die Arbeit mit DynamoDB. Als bewährte Methode sollten Ihre Anwendungen einen Client erstellen und diesen zwischen den Threads wiederverwenden.

Weitere Informationen finden Sie unter AWS SDK for .NET.

> **Note**
>
> Die Codebeispiele in diesem Handbuch sind für die Verwendung mit der neuesten Version des AWS SDK für .NET vorgesehen.

.NET: Ihre Anmeldeinformationen einrichten AWS

Das SDK für .NET erfordert, dass Sie zur Laufzeit AWS Anmeldeinformationen für Ihre Anwendung angeben. Bei den Codebeispielen in diesem Handbuch wird davon ausgegangen, dass Sie den SDK Store zur Verwaltung Ihrer AWS Anmeldeinformationsdatei verwenden, wie unter Verwenden des AWS SDK für .NET SDK-Speichers im Entwicklerhandbuch beschrieben.

Toolkit for Visual Studio unterstützt mehrere Gruppen von Anmeldeinformationen aus einer beliebigen Anzahl von Konten. Jede Gruppe wird als profile bezeichnet. Visual Studio fügt der `App.config` Projektdatei Einträge hinzu, sodass Ihre Anwendung die AWS Anmeldeinformationen zur Laufzeit finden kann.

Das folgende Beispiel zeigt die `App.config`-Standarddatei, die bei der Erstellung eines neuen Projekts mit Toolkit for Visual Studio generiert wird.

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
    <appSettings>
    <add key="AWSProfileName" value="default"/>
    <add key="AWSRegion" value="us-west-2" />
 </appSettings>
</configuration>
```

Zur Laufzeit verwendet das `default` Programm die im `AWSProfileName` Eintrag angegebenen AWS Anmeldeinformationen. Die AWS Anmeldeinformationen selbst werden in verschlüsselter Form im SDK-Speicher gespeichert. Das Toolkit for Visual Studio bietet eine grafische Benutzeroberfläche, um Ihre Anmeldeinformationen vollständig über Visual Studio zu verwalten. Weitere Informationen finden Sie unter Anmeldeinformationen angeben im AWS Toolkit for Visual Studio -Benutzerleitfaden.

> **Note**
>
> Standardmäßig greifen die Codebeispiele auf DynamoDB in der Region USA West (Oregon) auf. Sie können die Region ändern, indem Sie den `AWSRegion`-Eintrag in der Datei App.config ändern. Sie können `AWSRegion` auf jede Region festlegen, in der DynamoDB verfügbar ist. Die vollständige Liste finden Sie unter AWS -Regionen und -Endpunkte in der Allgemeine Amazon Web Services-Referenz.

.NET: AWS Region und Endpunkt festlegen

Standardmäßig greifen die Codebeispiele auf DynamoDB in der Region USA West (Oregon) auf. Sie können die Region ändern, indem Sie den `AWSRegion`-Eintrag in der Datei `App.config` ändern. Sie können auch die Region ändern, indem Sie die `AmazonDynamoDBClient`-Eigenschaften ändern.

Im folgenden Codebeispiel wird ein neuer `AmazonDynamoDBClient` instanziiert. Der Client wird so geändert, dass der Code für DynamoDB in einer anderen Region ausgeführt wird.

```
AmazonDynamoDBConfig clientConfig = new AmazonDynamoDBConfig();
// This client will access the US East 1 region.
clientConfig.RegionEndpoint = RegionEndpoint.USEast1;
AmazonDynamoDBClient client = new AmazonDynamoDBClient(clientConfig);
```

Die vollständige Liste der Regionen finden Sie unter AWS -Regionen und -Endpunkte in der Allgemeine Amazon Web Services-Referenz.

Wenn Sie die Codebeispiele mit DynamoDB lokal auf Ihrem Computer ausführen möchten, müssen Sie den Endpunkt wie folgt festlegen.

```
AmazonDynamoDBConfig clientConfig = new AmazonDynamoDBConfig();
// Set the endpoint URL
clientConfig.ServiceURL = "http://localhost:8000";
AmazonDynamoDBClient client = new AmazonDynamoDBClient(clientConfig);
```

DynamoDB Low-Level-API

Die Amazon-DynamoDB-Low-Level-API ist die Schnittstelle auf Protokollebene für DynamoDB. Auf dieser Ebene muss jede HTTP(S)-Anforderung ordnungsgemäß formatiert sein und eine gültige digitale Signatur aufweisen.

Sie AWS SDKs erstellen DynamoDB-API-Anfragen auf niedriger Ebene in Ihrem Namen und verarbeiten die Antworten von DynamoDB. So können Sie sich auf Ihre Anwendungslogik konzentrieren und müssen sich nicht mit Einzelheiten der unteren Ebene aufhalten. Dennoch ist es hilfreich, grundlegende Kenntnisse darüber zu haben, wie die DynamoDB-API auf niedriger Ebene funktioniert.

Weitere Informationen zur DynamoDB-Low-Level-API finden Sie unter Amazon-DynamoDB-API-Referenz.

> **Note**
>
> DynamoDB Streams verfügt über eine eigene Low-Level-API, die von der von DynamoDB getrennt ist und vollständig von der unterstützt wird. AWS SDKs
> Weitere Informationen finden Sie unter Ändern Sie die Datenerfassung für DynamoDB Streams. Informationen zur DynamoDB Streams-API auf niedriger Ebene finden Sie in der Amazon-DynamoDB-Streams-API-Referenz.

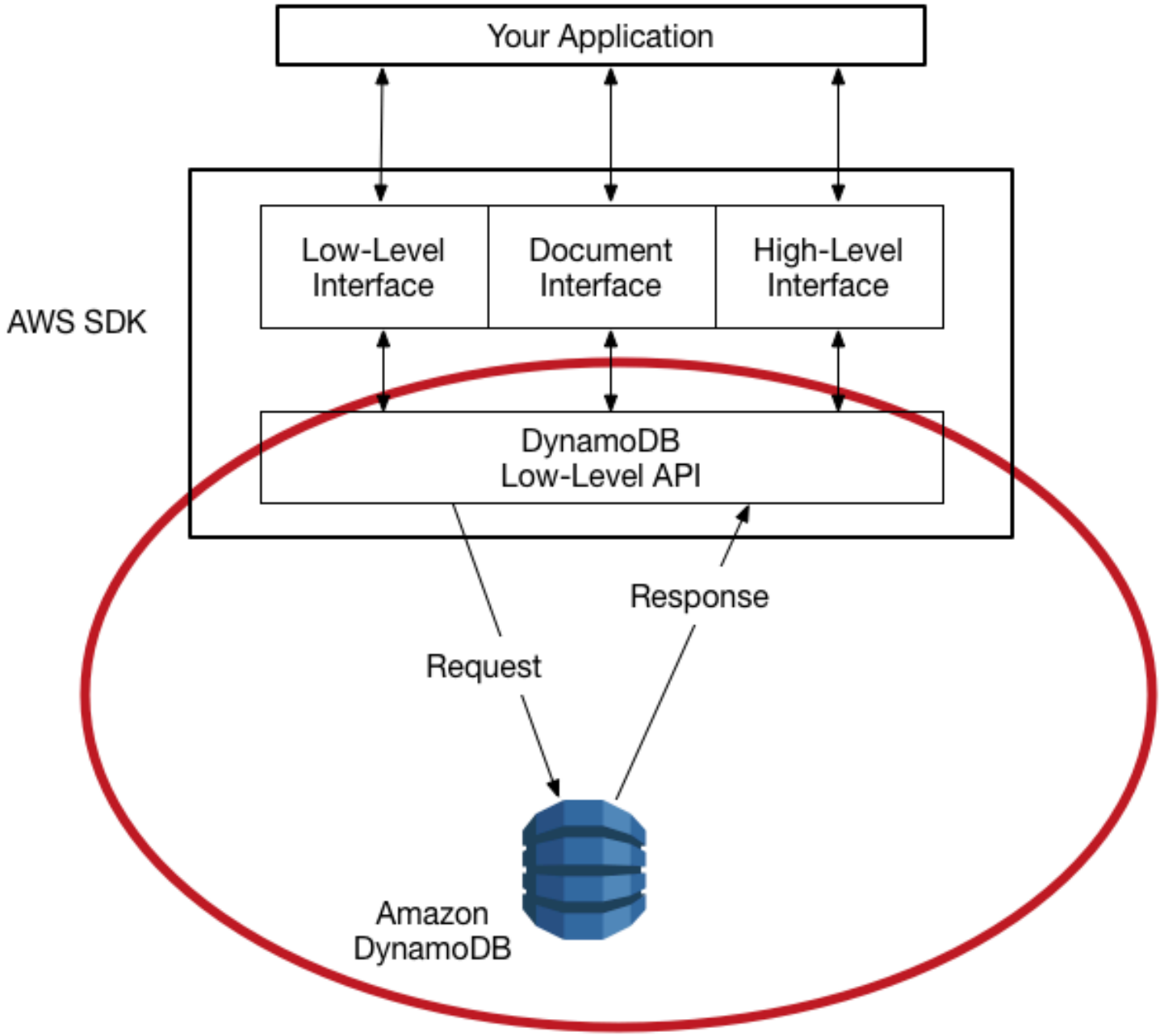
Die DynamoDB-API auf niedriger Ebene verwendet JavaScript Object Notation (JSON) als Wire-Protokollformat. JSON stellt Daten in einer Hierarchie dar, sodass Datenwerte und Datenstruktur gleichzeitig übermittelt werden. Name-Wert-Paare werden im Format `name:value` definiert. Die Datenhierarchie wird durch verschachtelte Klammern von Name-Wert-Paaren definiert.

DynamoDB verwendet JSON nur als Transportprotokoll, nicht als Speicherformat. AWS SDKs
Sie verwenden JSON, um Daten an DynamoDB zu senden, und DynamoDB antwortet mit JSON. DynamoDB speichert Daten nicht dauerhaft im JSON-Format.

> **Note**
>
> Weitere Informationen zu JSON finden Sie in der Einführung zu JSON auf der Website `JSON.org`.

Themen

- Anforderungsformat
- Reaktionsformat
- Datentypbeschreibungen
- Numerische Daten
- Binäre Daten



Anforderungsformat

Die DynamoDB-Low-Level-API akzeptiert HTTP(S)-POST-Anforderungen als Eingabe. Sie AWS SDKs erstellen diese Anfragen für Sie.

Angenommen, Sie verfügen über eine Tabelle namens `Pets` mit einem Schlüsselschema bestehend aus `AnimalType` (Partitionsschlüssel) und `Name` (Sortierschlüssel). Beide Attribute sind vom Typ `string`. Um ein Element von abzurufen`Pets`, erstellt das AWS SDK die folgende Anfrage.

```
POST / HTTP/1.1
Host: dynamodb.<region>.<domain>;
Accept-Encoding: identity
Content-Length: <PayloadSizeBytes>
User-Agent: <UserAgentString>
Content-Type: application/x-amz-json-1.0
Authorization: AWS4-HMAC-SHA256 Credential=<Credential>, SignedHeaders=<Headers>,
 Signature=<Signature>
X-Amz-Date: <Date>
X-Amz-Target: DynamoDB_20120810.GetItem

{
    "TableName": "Pets",
    "Key": {
        "AnimalType": {"S": "Dog"},
        "Name": {"S": "Fido"}
    }
}
```

Beachten Sie bei dieser Anforderung Folgendes:

- Der Header `Authorization` enthält Informationen, die DynamoDB benötigt, um die Anforderung zu authentifizieren. Weitere Informationen finden Sie unter Signieren von AWS API-Anfragen und Signiervorgang für Signature Version 4 in der Allgemeine Amazon Web Services-Referenz.
- Der Header `X-Amz-Target` enthält den Namen einer DynamoDB-Operation: `GetItem`. (Ebenfalls enthalten ist die Version der API auf niedriger Ebene, in diesem Fall `20120810`.)
- Die Nutzlast (der Text) der Anforderung enthält die Parameter für die Operation im JSON-Format. Für die Operation `GetItem` lauten die Parameter `TableName` und `Key`.

Reaktionsformat

Nach Eingang der Anforderung wird diese von DynamoDB verarbeitet und der Service gibt eine Antwort zurück. Bei der zuvor gezeigten Anforderung enthält die Nutzlast der HTTP(S)-Antwort die Ergebnisse aus der Operation wie in dem folgenden Beispiel gezeigt.

```
HTTP/1.1 200 OK
x-amzn-RequestId: <RequestId>
x-amz-crc32: <Checksum>
Content-Type: application/x-amz-json-1.0
Content-Length: <PayloadSizeBytes>
Date: <Date>
{
    "Item": {
        "Age": {"N": "8"},
        "Colors": {
            "L": [
                {"S": "White"},
                {"S": "Brown"},
                {"S": "Black"}
            ]
        },
        "Name": {"S": "Fido"},
        "Vaccinations": {
            "M": {
                "Rabies": {
                    "L": [
                        {"S": "2009-03-17"},
                        {"S": "2011-09-21"},
                        {"S": "2014-07-08"}
                    ]
                },
                "Distemper": {"S": "2015-10-13"}
            }
        },
        "Breed": {"S": "Beagle"},
        "AnimalType": {"S": "Dog"}
    }
}
```

Zu diesem Zeitpunkt sendet das AWS SDK die Antwortdaten zur weiteren Verarbeitung an Ihre Anwendung zurück.

> **Note**
>
> Wenn DynamoDB eine Anforderung nicht verarbeiten kann, gibt der Service einen HTTP-Fehlercode mit einer entsprechenden Meldung zurück. Das AWS -SDK überträgt diese an Ihre Anwendung in Form von Ausnahmen. Weitere Informationen finden Sie unter Fehlerbehandlung mit DynamoDB.

Datentypbeschreibungen

Das Low-Level-DynamoDB-API-Protokoll erfordert, dass jedes Attribut von einem Datentypdeskriptor begleitet wird. Datentypdeskriptoren sind Token, die DynamoDB mitteilen, wie jedes Attribut zu interpretieren ist.

Die Beispiele unter Anforderungsformat und Reaktionsformat zeigen, wie Datentypbeschreibungen verwendet werden. Die `GetItem`-Anforderung gibt `S` für die `Pets`-Schlüsselschemaattribute an (`AnimalType` und `Name`, vom Typ `string`). Die `GetItem`-Antwort enthält ein Pets-Element mit Attributen vom Typ `string` (`S`), `number` (`N`), `map` (`M`) und `list` (`L`).

Im Folgenden sehen Sie eine vollständige Liste der DynamoDB-Datentypbeschreibungen:

- **`S`** – Zeichenfolge
- **`N`** – Zahl
- **`B`** – Binary
- **`BOOL`** – Boolean
- **`NULL`** – Nullwert
- **`M`** – Zuordnung
- **`L`** – Liste
- **`SS`** – Zeichenfolgensatz
- **`NS`** – Zahlensatz
- **`BS`** – Binärzahlensatz

Die folgende Tabelle zeigt das richtige JSON-Format für jeden Datentypdeskriptor. Beachten Sie, dass Zahlen aus Gründen der Genauigkeit als Zeichenketten dargestellt werden, während boolesche Werte und Nullen ihre nativen JSON-Typen verwenden.

Deskriptor	JSON-Format	Hinweise
S	`{"S": "Hello"}`	Der Wert ist eine JSON-Zeic henfolge.
N	`{"N": "123.45"}`	Der Wert ist eine Zeichenfolge, keine JSON-Nummer. Dadurch bleibt die Präzision in allen Sprachen erhalten.
B	`{"B": "dGhpcyBpcyBhIHRlc 3Q="}`	Der Wert ist eine Base64-kodierte Zeichenfolge.
BOOL	`{"BOOL": true}`	Der Wert ist ein boolescher JSON-Wert (`true`oder`false`), keine Zeichenfolge.
NULL	`{"NULL": true}`	Value ist der boolesche JSON-Wert, der Null angibt`true`.
M	`{"M": {"Name": {"S": "Joe"}}}`	Value ist ein JSON-Objekt mit Attributname-Wert-Paaren.
L	`{"L": [{"S": "Red"}, {"N": "5"}]}`	Value ist ein JSON-Array von Attributwerten.
SS	`{"SS": ["Red", "Blue"]}`	Value ist ein JSON-Array von Zeichenketten.
NS	`{"NS": ["1", "2.5"]}`	Value ist ein JSON-Array von Zahlenzeichenfolgen.
BS	`{"BS": ["U3Vubnk=", "UmFpbnk="]}`	Value ist ein JSON-Array von Base64-codierten Zeichenketten.

> **Note**
>
> Detaillierte Beschreibungen von DynamoDB-Datentypen finden Sie unter Datentypen.

Numerische Daten

Die unterschiedlichen Programmiersprachen bieten verschiedene Ebenen der Unterstützung für JSON. In einigen Fällen möchten Sie möglicherweise eine Drittanbieterbibliothek zum Validieren und Parsen von JSON-Dokumenten verwenden.

Einige Drittanbieterbibliotheken bieten basierend auf dem JSON-Nummerntyp eigene Typen wie z. B. `int`, `long` oder `double` an. Der native Zahlendatentyp in DynamoDB ist diesen Datentypen nicht exakt zuordenbar, sodass es bei Unterschieden zu Konflikten kommen kann. Außerdem verarbeiten viele JSON-Bibliotheken keine numerischen Werte mit fester Präzision und leiten automatisch einen Double-Datentyp für Ziffernfolgen ab, die ein Dezimaltrennzeichen enthalten.

Um diese Art von Problemen zu lösen, stellt DynamoDB einen einzigen numerischen Typ ohne Datenverluste bereit. Um unerwünschte implizite Umwandlungen in einen Double-Wert zu verhindern, verwendet DynamoDB Zeichenfolgen für die Datenübertragung von numerischen Werten. Dieser Ansatz bietet Flexibilität bei der Aktualisierung von Attributwerten und sorgt außerdem für eine korrekte Sortiersemantik (z. B. werden die Werte "01", "2" und "03" in die richtige Reihenfolge gebracht).

Wenn für Ihre Anwendung die Zahlengenauigkeit wichtig ist, sollten Sie numerische Werte in Zeichenfolgen konvertieren, bevor Sie sie an DynamoDB übergeben.

Binäre Daten

DynamoDB unterstützt binäre Attribute. JSON bietet für binäre Daten jedoch keine native Unterstützung. Um binäre Daten in einer Anforderung zu senden, müssen Sie sie im base64-Format kodieren. Nach Eingang der Anforderung werden die base64-Daten von DynamoDB zurück in binäre Daten dekodiert.

Das von DynamoDB verwendete base64-Codierungsschema wird unter RFC 4648 auf der Website der Internet Engineering Task Force (IETF) beschrieben.

Programmieren von Amazon DynamoDB mit Python und Boto3

Dieser Leitfaden bietet eine Einführung für Programmierer, die Amazon DynamoDB mit Python verwenden möchten. Erfahren Sie mehr über die Abstraktionsebenen, das Konfigurationsmanagement, den Umgang mit Fehlern, die Steuerung von Wiederholungsrichtlinien, die Verwaltung von Keepalive und mehr.

Themen

- Informationen zu Boto
- Verwenden der Boto-Dokumentation
- Grundlegendes zu den Abstraktionsebenen für Clients und Ressourcen
- Verwenden der Tabellenressource batch_writer
- Zusätzliche Codebeispiele zur Client- und Ressourcenebene
- Verständnis der Interaktion von Client- und Resource-Objekten mit Sitzungen und Threads
- Anpassen des Config-Objekts
- Fehlerbehandlung
- Protokollierung
- Ereignis-Hooks
- Paginierung und der Paginator
- Waiter

Informationen zu Boto

Sie können von Python aus auf DynamoDB zugreifen, indem Sie das offizielle AWS SDK für Python verwenden, das allgemein als Boto3 bezeichnet wird. Der Name Boto (ausgesprochen boh-toh) stammt von einem Süßwasserdelfin, der im Amazon heimisch ist. Die Boto3-Bibliothek ist die dritte Hauptversion der Bibliothek, die erstmals 2015 veröffentlicht wurde. Die Boto3-Bibliothek ist ziemlich groß, da sie alle AWS Dienste unterstützt, nicht nur DynamoDB. Dieser Leitfaden zielt nur auf die Teile von Boto3 ab, die für DynamoDB relevant sind.

Boto wird von einem Open-Source-Projekt verwaltet und veröffentlicht, das auf gehostet wird AWS . GitHub Es ist in zwei Pakete aufgeteilt: Botocore und Boto3.

- Botocore bietet die Low-Level-Funktionalität. In Botocore finden Sie die Klassen Client, Session, Credentials, Config und Exception.
- Boto3 baut auf Botocore auf. Es bietet eine übergeordnete, pythonischere Oberfläche. Insbesondere macht es eine DynamoDB-Tabelle als Ressource verfügbar und bietet eine einfachere, elegantere Oberfläche als die serviceorientierte Client-Schnittstelle auf niedrigerer Ebene.

Da diese Projekte auf gehostet werden GitHub, können Sie den Quellcode einsehen, offene Probleme verfolgen oder Ihre eigenen Probleme einreichen.

Verwenden der Boto-Dokumentation

Beginnen Sie mit der Boto-Dokumentation anhand der folgenden Ressourcen:

- Beginnen Sie mit dem Quickstart-Abschnitt, der einen soliden Ausgangspunkt für die Paketinstallation bietet. Dort finden Sie Anweisungen zur Installation von Boto3, falls dies noch nicht geschehen ist (Boto3 ist häufig automatisch in AWS Diensten wie) verfügbar. AWS Lambda
- Konzentrieren Sie sich danach auf den DynamoDB-Leitfaden der Dokumentation. Darin erfahren Sie, wie Sie die grundlegenden DynamoDB-Aktivitäten ausführen: eine Tabelle erstellen und löschen, Elemente bearbeiten, Batch-Operationen ausführen, eine Abfrage ausführen und einen Scan durchführen. Für die Beispiele wird die Schnittstelle Ressourcen verwendet. Wenn Sie `boto3.resource('dynamodb')` sehen, bedeutet das, dass Sie die übergeordnete Schnittstelle Ressourcen verwenden.
- Nach dem Leitfaden können Sie sich die DynamoDB-Referenz nochmals ansehen. Diese Startseite bietet eine vollständige Liste der Klassen und Methoden, die Ihnen zur Verfügung stehen. Ganz oben sehen Sie die `DynamoDB.Client`-Klasse. Diese bietet Low-Level-Zugriff auf alle Operationen auf der Steuerungsebene und der Datenebene. Schauen Sie sich ganz unten die `DynamoDB.ServiceResource`-Klasse an. Dies ist die Pythonic-Schnittstelle auf höherer Ebene. Damit können Sie eine Tabelle erstellen, tabellenübergreifende Batch-Operationen ausführen oder eine `DynamoDB.ServiceResource.Table`-Instance für tabellenspezifische Aktionen abrufen.

Grundlegendes zu den Abstraktionsebenen für Clients und Ressourcen

Die beiden Schnittstellen, mit denen Sie arbeiten werden, sind die Client- und die Ressourcen-Schnittstelle.

- Die Low-Level-Client-Schnittstelle bietet eine Eins-zu-Eins-Zuordnung zur zugrunde liegenden Service-API. Jede von DynamoDB angebotene API ist über den Client verfügbar. Das bedeutet, dass die Client-Schnittstelle vollständige Funktionalität bieten kann, aber häufig ausführlicher und komplexer in der Anwendung ist.
- Die High-Level-Ressourcen-Schnittstelle bietet keine 1-zu-1-Zuordnung der zugrunde liegenden Service-API. Sie bietet jedoch Methoden, die Ihnen den Zugriff auf den Service erleichtern, z. B `batch_writer`.

Hier ein Beispiel für das Einfügen eines Elements über die Client-Schnittstelle. Beachten Sie, dass alle Werte als Map übergeben werden, wobei der Schlüssel ihren Typ ('S' für Zeichenfolge, 'N' für Zahl) und ihren Wert als Zeichenfolge angibt. Dies wird als DynamoDB-JSON-Format bezeichnet.

```
import boto3

dynamodb = boto3.client('dynamodb')

dynamodb.put_item(
    TableName='YourTableName',
    Item={
        'pk': {'S': 'id#1'},
        'sk': {'S': 'cart#123'},
        'name': {'S': 'SomeName'},
        'inventory': {'N': '500'},
        # ... more attributes ...
    }
)
```

Hier ist dieselbe `PutItem`-Operation, jedoch unter Verwendung der Ressourcenschnittstelle. Die Datentypisierung ist implizit:

```
import boto3

dynamodb = boto3.resource('dynamodb')

table = dynamodb.Table('YourTableName')

table.put_item(
    Item={
        'pk': 'id#1',
        'sk': 'cart#123',
        'name': 'SomeName',
        'inventory': 500,
        # ... more attributes ...
    }
)
```

Bei Bedarf können Sie mithilfe der in boto3 bereitgestellten Klassen `TypeSerializer` und `TypeDeserializer` von dem normalen JSON- zum DynamoDB-JSON-Format konvertieren:

```
def dynamo_to_python(dynamo_object: dict) -> dict:
    deserializer = TypeDeserializer()
    return {
        k: deserializer.deserialize(v)
        for k, v in dynamo_object.items()
    }

def python_to_dynamo(python_object: dict) -> dict:
    serializer = TypeSerializer()
    return {
        k: serializer.serialize(v)
        for k, v in python_object.items()
    }
```

So führen Sie eine Abfrage mithilfe der Client-Schnittstelle durch. Sie drückt die Abfrage als ein JSON-Konstrukt aus. Es wird eine `KeyConditionExpression`-Zeichenfolge verwendet, die eine Variablenersetzung erfordert, um mögliche Schlüsselwortkonflikte zu behandeln:

```
import boto3

client = boto3.client('dynamodb')

# Construct the query
response = client.query(
    TableName='YourTableName',
    KeyConditionExpression='pk = :pk_val AND begins_with(sk, :sk_val)',
    FilterExpression='#name = :name_val',
    ExpressionAttributeValues={
        ':pk_val': {'S': 'id#1'},
        ':sk_val': {'S': 'cart#'},
        ':name_val': {'S': 'SomeName'},
    },
    ExpressionAttributeNames={
        '#name': 'name',
    }
)
```

Dieselbe Abfrageoperation, die die Ressourcenschnittstelle verwendet, kann verkürzt und vereinfacht werden:

```
import boto3
from boto3.dynamodb.conditions import Key, Attr
```

```
dynamodb = boto3.resource('dynamodb')
table = dynamodb.Table('YourTableName')

response = table.query(
    KeyConditionExpression=Key('pk').eq('id#1') & Key('sk').begins_with('cart#'),
    FilterExpression=Attr('name').eq('SomeName')
)
```

Stellen Sie sich als letztes Beispiel vor, Sie möchten die ungefähre Größe einer Tabelle ermitteln (dabei handelt es sich um Metadaten, die in der Tabelle gespeichert sind und etwa alle 6 Stunden aktualisiert werden). Mit der Client-Schnittstelle führen Sie eine `describe_table()`-Operation aus und ziehen die Antwort aus der zurückgegebenen JSON-Struktur:

```
import boto3

dynamodb = boto3.client('dynamodb')

response = dynamodb.describe_table(TableName='YourTableName')
size = response['Table']['TableSizeBytes']
```

Mit der Ressourcenschnittstelle führt die Tabelle die Beschreibungsoperation implizit aus und präsentiert die Daten direkt als Attribut:

```
import boto3

dynamodb = boto3.resource('dynamodb')

table = dynamodb.Table('YourTableName')
size = table.table_size_bytes
```

> **Note**
>
> Beachten Sie bei der Entscheidung, ob Sie mit der Client- oder der Ressourcenschnittstelle entwickeln möchten, dass der Ressourcenschnittstelle gemäß der Ressourcendokumentation keine neuen Funktionen hinzugefügt werden: „Das AWS Python SDK-Team beabsichtigt nicht, der Ressourcenschnittstelle in boto3 neue Funktionen hinzuzufügen. Vorhandene Schnittstellen werden während des Lebenszyklus von boto3 weiterhin funktionieren. Kunden können über die Client-Oberfläche auf neuere Service-Features zugreifen.“

Verwenden der Tabellenressource batch_writer

Ein Vorteil, der nur mit der High-Level-Tabellenressource verfügbar ist, ist der `batch_writer`. DynamoDB unterstützt Batch-Schreibvorgänge und ermöglicht bis zu 25 Put- oder Löschvorgänge in einer Netzwerkanfrage. Eine solche Batchverarbeitung verbessert die Effizienz, da Netzwerk-Roundtrips minimiert werden.

Mit der Low-Level-Client-Bibliothek verwenden Sie die `client.batch_write_item()`-Operation, um Batches auszuführen. Sie müssen Ihre Arbeit manuell in Batches von 25 aufteilen. Nach jedem Vorgang müssen Sie außerdem eine Liste der unverarbeiteten Elemente anfordern (einige Schreibvorgänge können erfolgreich sein, während andere fehlschlagen könnten). Anschließend müssen Sie diese unverarbeiteten Elemente erneut an eine spätere `batch_write_item()`-Operation übergeben. Es gibt eine beträchtliche Menge an Boilerplate-Code.

Die Methode Table.batch_writer erstellt einen Kontextmanager zum Schreiben von Objekten in einem Batch. Dieser stellt eine Schnittstelle bereit, bei der es so aussieht, als würden Sie Elemente einzeln schreiben, aber intern werden die Elemente gepuffert und in Batches gesendet. Außerdem verarbeitet er auch implizite Wiederholungen unverarbeiteter Elemente.

```
dynamodb = boto3.resource('dynamodb')

table = dynamodb.Table('YourTableName')

movies = # long list of movies in {'pk': 'val', 'sk': 'val', etc} format
with table.batch_writer() as writer:
    for movie in movies:
        writer.put_item(Item=movie)
```

Zusätzliche Codebeispiele zur Client- und Ressourcenebene

Sie können auch auf die folgenden Beispiel-Repositorys verweisen, die die Nutzung verschiedener Funktionen sowohl mit dem Client als auch mit der Ressourcenebene untersuchen:

- Offizielle AWS -Single-Action-Codebeispiele
- Offizielle AWS szenarioorientierte Codebeispiele.
- Von der Community verwaltete Beispiele für Single-Action-Codes

Verständnis der Interaktion von Client- und Resource-Objekten mit Sitzungen und Threads

Das Resource-Objekt ist nicht Thread-sicher und sollte nicht über Threads oder Prozesse hinweg gemeinsam genutzt werden. Weitere Informationen finden Sie im Leitfaden zur Resource.

Das Client-Objekt hingegen ist im Allgemeinen Thread-sicher. Ausgenommen sind bestimmte fortgeschrittene Features. Weitere Informationen finden Sie im Leitfaden zu Clients.

Das Sitzungsobjekt ist nicht Thread-sicher. Daher sollten Sie in einer Multi-Threading-Umgebung für jeden neuen Client oder jedes neue Ressourcenobjekt zunächst eine neue Sitzung erstellen und daraus dann den jeweiligen Client oder das Ressourcenobjekt ableiten. Weitere Informationen finden Sie im Leitfaden zu Sitzungen.

Wenn Sie `boto3.resource()` aufrufen, verwenden Sie implizit die Standardsitzung. Das ist praktisch für das Schreiben von Single-Threaded-Code. Für Multi-Threaded-Code sollten Sie hingegen für jeden Thread eine eigene Session erstellen und daraus die jeweilige Resource abrufen:

```
# Explicitly create a new Session for this thread
session = boto3.Session()
dynamodb = session.resource('dynamodb')
```

Anpassen des Config-Objekts

Beim Erstellen eines Client- oder Resource-Objekts können Sie optionale benannte Parameter übergeben, um das Verhalten zu steuern. Der Parameter `config` schaltet dabei eine Vielzahl von Funktionen frei. Es handelt sich um eine Instance von `botocore.client.Config`, und die Referenzdokumentation zu Config zeigt alle verfügbaren Optionen zur Steuerung. Die Anleitung zur Konfiguration bietet einen guten Überblick.

> **Note**
>
> Sie können viele dieser Verhaltenseinstellungen auf Sitzungsebene, in der AWS -Konfigurationsdatei oder als Umgebungsvariablen ändern.

Config für Timeouts

Eine Anwendung einer benutzerdefinierten Konfiguration besteht darin, das Netzwerkverhalten anzupassen:

- connect_timeout (float oder int) – Die Zeit in Sekunden, bis eine Timeout-Ausnahme beim Versuch eines Verbindungsaufbaus ausgelöst wird. Standardmäßig ist ein Zeitraum von 60 Sekunden festgelegt.
- read_timeout (float oder int) – Die Zeit in Sekunden, bis beim Versuch einer Verbindungslesung eine Timeout-Ausnahme ausgelöst wird. Standardmäßig ist ein Zeitraum von 60 Sekunden festgelegt.

Timeouts von 60 Sekunden sind für DynamoDB zu hoch. Das bedeutet, dass eine vorübergehende Netzwerkstörung eine Verzögerung von einer Minute verursacht, bevor der Client einen erneuten Versuch unternehmen kann. Der folgende Code verkürzt die Timeouts auf eine Sekunde:

```
import boto3
from botocore.config import Config

my_config = Config(
   connect_timeout = 1.0,
   read_timeout = 1.0
)
dynamodb = boto3.resource('dynamodb', config=my_config)
```

Weitere Informationen zu Timeouts finden Sie unter Tuning der AWS Java SDK-HTTP-Anforderungseinstellungen für latenzbewusste DynamoDB-Anwendungen. Beachten Sie, dass das Java-SDK mehr Timeout-Konfigurationsmöglichkeiten bietet als das Python-SDK.

Config für Keepalive

Wenn Sie Botocore 1.27.84 oder höher verwenden, können Sie auch TCP Keepalive steuern:

- tcp_keepalive (bool) – Aktiviert die Socket-Option TCP-Keepalive beim Erstellen neuer Verbindungen, wenn diese auf `True` gesetzt wurde (Standard ist `False`). Dies ist nur ab Botocore 1.27.84 verfügbar.

Wenn Sie TCP Keepalive auf `True` setzen, können Sie die durchschnittlichen Latenzen reduzieren. Hier ein Beispielcode, der TCP Keepalive bedingt auf true setzt, wenn Sie die richtige Botocore-Version haben:

```
import botocore
import boto3
from botocore.config import Config
from distutils.version import LooseVersion

required_version = "1.27.84"
current_version = botocore.__version__

my_config = Config(
   connect_timeout = 0.5,
   read_timeout = 0.5
)
if LooseVersion(current_version) > LooseVersion(required_version):
    my_config = my_config.merge(Config(tcp_keepalive = True))

dynamodb = boto3.resource('dynamodb', config=my_config)
```

> **Note**
>
> TCP Keepalive unterscheidet sich von HTTP Keepalive. Bei TCP Keepalive werden kleine Pakete vom zugrunde liegenden Betriebssystem über die Socket-Verbindung gesendet, um die Verbindung aufrechtzuerhalten und etwaige Unterbrechungen sofort zu erkennen. Mit HTTP Keepalive wird die auf dem zugrunde liegenden Socket aufgebaute Webverbindung wiederverwendet. HTTP Keepalive ist bei boto3 immer aktiviert.

Es gibt jedoch eine Begrenzung, wie lange eine inaktive Verbindung aufrechterhalten werden kann. Erwägen Sie, regelmäßige Anfragen zu senden (z. B. jede Minute), wenn Sie eine Leerlaufverbindung haben, aber möchten, dass die nächste Anfrage eine bereits bestehende Verbindung nutzt.

Config für Wiederholungsversuche

Die Konfiguration akzeptiert auch ein Wörterbuch namens Wiederholungen, in dem Sie Ihr gewünschtes Wiederholungsverhalten angeben können. Wiederholungen finden innerhalb des SDK statt, wenn das SDK einen Fehler empfängt und der Fehler vorübergehend ist. Wenn ein Fehler intern wiederholt wird (und der Wiederholungsversuch schließlich erfolgreich ist), wird aus Sicht des aufrufenden Codes kein Fehler sichtbar, sondern nur eine leicht erhöhte Latenz. Hier sind die Werte, die Sie angeben können:

- max_attempts – Eine Ganzzahl, die die maximale Anzahl von Wiederholungsversuchen angibt, die bei einer einzelnen Anfrage durchgeführt werden. Wenn Sie diesen Wert beispielsweise auf 2 setzen, wird die Anfrage nach der ersten Anfrage höchstens zweimal wiederholt. Wenn Sie diesen Wert auf 0 setzen, werden nach der ersten Anfrage keine weiteren Wiederholungsversuche durchgeführt.
- total_max_attempts – Eine Ganzzahl, die die maximale Gesamtzahl der Versuche angibt, die bei einer einzelnen Anfrage unternommen werden. Dies schließt die erste Anfrage ein, sodass ein Wert von 1 bedeutet, dass keine weiteren Anfragen versucht werden. Wenn sowohl `total_max_attempts` als auch `max_attempts` angegeben sind, hat `total_max_attempts` Vorrang. `total_max_attempts` und nicht `max_attempts` wird bevorzugt, da er mit der Umgebungsvariable `AWS_MAX_ATTEMPTS` und dem Konfigurationswert `max_attempts` in der Konfigurationsdatei übereinstimmt.
- mode – Eine Zeichenfolge, die den Typ des Wiederholungsmodus darstellt, den Botocore verwenden soll. Gültige Werte sind:
  - legacy – Der Standardmodus. Wartet 50 ms beim ersten Wiederholungsversuch und verwendet dann ein exponentielles Backoff mit dem Basisfaktor 2. Für DynamoDB sind es insgesamt maximal 10 Versuche (sofern nicht wie oben gezeigt überschrieben wurde).

    > **Note**
    >
    > Bei exponentiellem Backoff wird fast 13 Sekunden mit dem letzten Versuch gewartet.

  - Standard — Wird als Standard bezeichnet, weil er konsistenter mit anderen ist. AWS SDKs Beim ersten Wiederholungsversuch wird eine zufällige Zeitspanne zwischen 0 ms und 1 000 ms abgewartet. Sollte ein weiterer Versuch erforderlich sein, wird erneut ein zufälliger Zeitraum zwischen 0 ms und 1 000 ms gewählt und mit 2 multipliziert. Sollten weitere Wiederholungsversuche erforderlich sein, erfolgt dieselbe zufällige Auswahl, multipliziert mit 4 usw. Jede Wartezeit ist auf 20 Sekunden begrenzt. Dieser Modus führt mehr Wiederholungen für mehr erkannte Fehlerbedingungen aus als der `legacy`-Modus. Für DynamoDB sind es insgesamt maximal 3 Versuche (sofern nicht wie oben gezeigt überschrieben wurde).
  - adaptive - Ein experimenteller Wiederholungsmodus, der alle Funktionalitäten des Standardmodus beinhaltet, aber eine automatische clientseitige Drosselung hinzufügt. Mit adaptiver Ratenbegrenzung SDKs kann die Geschwindigkeit, mit der Anfragen gesendet werden, verlangsamt werden, um der Kapazität der AWS Dienste besser gerecht zu werden. Dies ist ein vorläufiger Modus, dessen Verhalten sich ändern kann.

Eine erweiterte Definition dieser Wiederholungsmodi finden Sie in der Anleitung zu Wiederholungsversuchen sowie im Thema Wiederholungsverhalten in der SDK-Referenz.

Hier ein Beispiel, das explizit die `legacy`-Wiederholungsrichtlinie mit insgesamt maximal 3 Anfragen (2 Wiederholungen) verwendet:

```
import boto3
from botocore.config import Config

my_config = Config(
   connect_timeout = 1.0,
   read_timeout = 1.0,
   retries = {
     'mode': 'legacy',
     'total_max_attempts': 3
   }
)
dynamodb = boto3.resource('dynamodb', config=my_config)
```

Da es sich bei DynamoDB um ein System mit hoher Verfügbarkeit und geringer Latenz handelt, sollten Sie bei der Geschwindigkeit von Wiederholungsversuchen möglicherweise etwas aggressiver vorgehen, als es die integrierten Wiederholungsrichtlinien zulassen. Sie können Ihre eigene Wiederholungsrichtlinie implementieren, indem Sie die maximale Anzahl der Versuche auf 0 setzen, die Ausnahmen selbst abfangen und es gegebenenfalls in Ihrem eigenen Code erneut versuchen, anstatt sich bei impliziten Wiederholungsversuchen auf boto3 zu verlassen.

Wenn Sie Ihre eigene Wiederholungsrichtlinie verwalten, sollten Sie zwischen Drosselungen und Fehlern unterscheiden:

- Eine Drosselung (gekennzeichnet durch `ProvisionedThroughputExceededException` oder `ThrottlingException`) weist auf einen fehlerfreien Service hin, der Sie darüber informiert, dass Sie Ihre Lese- oder Schreibkapazität in einer DynamoDB-Tabelle oder Partition überschritten haben. Mit jeder Millisekunde, die vergeht, wird etwas mehr Lese- oder Schreibkapazität zur Verfügung gestellt, sodass Sie schnell (z. B. alle 50 ms) erneut versuchen können, auf die neu freigegebene Kapazität zuzugreifen. Bei Drosselungen ist kein exponentieller Backoff erforderlich, da Drosselungen für DynamoDB leichtgewichtig sind und keine Gebühren pro Anfrage anfallen. Exponentielles Backoff weist längere Wartezeiten jenen Client-Threads zu, die bereits am längsten gewartet haben, wodurch sich die statistischen Kennzahlen p50 und p99 weiter nach außen verschieben.

- Ein Fehler (unter anderem durch `InternalServerError` oder `ServiceUnavailable` gekennzeichnet) weist auf ein vorübergehendes Problem mit dem Service hin. Dies kann für die gesamte Tabelle oder möglicherweise nur für die Partition gelten, aus der Sie lesen oder in die Sie schreiben. Bei Fehlern sollten Sie längere Pausen zwischen den Wiederholungsversuchen einlegen (etwa 250 ms oder 500 ms) und Jitter verwenden, um die Wiederholungen zeitlich zu staffeln.

Config für maximale Poolverbindungen

Abschließend ermöglicht die Konfiguration die Steuerung der Größe des Verbindungs-Pools:

- max_pool_connections (int) – Die maximale Anzahl von Verbindungen, die in einem Verbindungspool beibehalten werden sollen. Wenn kein Wert festgelegt ist, wird der Standardwert 10 verwendet.

Diese Option steuert die maximale Anzahl von HTTP-Verbindungen, die zur Wiederverwendung gepoolt werden sollen. Für jede Sitzung wird ein anderer Pool verwaltet. Wenn Sie erwarten, dass mehr als 10 Threads gleichzeitig auf Clients oder Ressourcen zugreifen, die aus derselben Sitzung erstellt wurden, sollten Sie diesen Wert erhöhen, damit Threads nicht auf andere Threads warten müssen, die gerade eine gepoolte Verbindung verwenden.

```
import boto3
from botocore.config import Config

my_config = Config(
   max_pool_connections = 20
)

# Setup a single session holding up to 20 pooled connections
session = boto3.Session(my_config)

# Create up to 20 resources against that session for handing to threads
# Notice the single-threaded access to the Session and each Resource
resource1 = session.resource('dynamodb')
resource2 = session.resource('dynamodb')
# etc
```

Fehlerbehandlung

AWS Nicht alle Serviceausnahmen sind in Boto3 statisch definiert. Dies liegt daran, dass Fehler und Ausnahmen bei AWS Diensten sehr unterschiedlich sind und sich ändern können. Boto3 verpackt alle Service-Ausnahmen als `ClientError` und stellt die Details als strukturiertes JSON bereit. Ein Fehlerantwort könnte beispielsweise wie folgt aufgebaut sein:

```
{
    'Error': {
        'Code': 'SomeServiceException',
        'Message': 'Details/context around the exception or error'
    },
    'ResponseMetadata': {
        'RequestId': '1234567890ABCDEF',
        'HostId': 'host ID data will appear here as a hash',
        'HTTPStatusCode': 400,
        'HTTPHeaders': {'header metadata key/values will appear here'},
        'RetryAttempts': 0
    }
}
```

Der folgende Code fängt alle `ClientError`-Ausnahmen ab und prüft den Zeichenfolgenwert des `Code` innerhalb des `Error`, um zu bestimmen, welche Aktion ausgeführt werden soll:

```
import botocore
import boto3

dynamodb = boto3.client('dynamodb')

try:
    response = dynamodb.put_item(...)

except botocore.exceptions.ClientError as err:
    print('Error Code: {}'.format(err.response['Error']['Code']))
    print('Error Message: {}'.format(err.response['Error']['Message']))
    print('Http Code: {}'.format(err.response['ResponseMetadata']['HTTPStatusCode']))
    print('Request ID: {}'.format(err.response['ResponseMetadata']['RequestId']))

    if err.response['Error']['Code'] in ('ProvisionedThroughputExceededException',
 'ThrottlingException'):
        print("Received a throttle")
    elif err.response['Error']['Code'] == 'InternalServerError':
```

```
        print("Received a server error")
    else:
        raise err
```

Einige (aber nicht alle) Ausnahmecodes wurden als Top-Level-Klassen materialisiert. Sie können diese direkt behandeln. Wenn Sie die Client-Schnittstelle verwenden, werden diese Ausnahmen dynamisch in Ihrem Client-Objekt bereitgestellt, und Sie fangen sie über Ihre Client-Instance ab:

```
except ddb_client.exceptions.ProvisionedThroughputExceededException:
```

Wenn Sie die Ressourcenschnittstelle verwenden, müssen Sie `.meta.client` nutzen, um vom Ressourcenobjekt zum zugrunde liegenden Client zu gelangen und auf die Ausnahmen zuzugreifen:

```
except ddb_resource.meta.client.exceptions.ProvisionedThroughputExceededException:
```

Um die Liste der materialisierten Ausnahmetypen zu überprüfen, können Sie die Liste dynamisch generieren:

```
ddb = boto3.client("dynamodb")
print([e for e in dir(ddb.exceptions) if e.endswith('Exception') or
 e.endswith('Error')])
```

Wenn Sie eine Schreiboperation mit einem Bedingungsausdruck ausführen, können Sie vorgeben, dass der Wert des Elements in der Fehlerantwort zurückgegeben wird, falls der Ausdruck fehlschlägt.

```
try:
    response = table.put_item(
        Item=item,
        ConditionExpression='attribute_not_exists(pk)',
        ReturnValuesOnConditionCheckFailure='ALL_OLD'
    )
except table.meta.client.exceptions.ConditionalCheckFailedException as e:
    print('Item already exists:', e.response['Item'])
```

Weitere Informationen zur Fehlerbehandlung und zu Ausnahmen finden Sie unter:

- Der boto3-Leitfaden zur Fehlerbehandlung enthält weitere Informationen über Methoden zur Fehlerbehandlung.
- Im DynamoDB-Entwicklerhandbuch zu Programmierfehlern ist aufgeführt, auf welche Fehler Sie stoßen könnten.

- Der Abschnitt Häufige Fehler in der API-Referenz.
- In der Dokumentation zu den einzelnen API-Vorgängen ist aufgeführt, welche Fehler dieser Aufruf erzeugen könnte (zum Beispiel BatchWriteItem).

Protokollierung

Die Boto3-Bibliothek integriert sich mit dem in Python eingebauten Protokollierungsmodul, um Vorgänge während einer Sitzung zu protokollieren. Sie können das Protokollierungsmodul konfigurieren, um die Protokollierungsebenen zu steuern:

```
import logging

logging.basicConfig(level=logging.INFO)
```

Dadurch wird der Root-Logger so konfiguriert, dass er `INFO` und Nachrichten höherer Ebenen protokolliert. Protokollierungsnachrichten, die weniger schwerwiegend sind als Level 4, werden ignoriert. Zu den Protokollierungsebenen gehören `DEBUG`, `INFO`, `WARNING`, `ERROR` und `CRITICAL`. Der Standardwert ist `WARNING`.

Die Logger in boto3 sind hierarchisch aufgebaut. Die Bibliothek verwendet verschiedene Logger, die jeweils unterschiedlichen Teilen der Bibliothek entsprechen. Sie können das Verhalten der einzelnen Logger separat steuern:

- boto3: Der Hauptlogger für das boto3-Modul.
- botocore: Der Hauptlogger für das Botocore-Paket.
- botocore.auth: Wird zum Protokollieren der AWS -Signaturerstellung für Anfragen verwendet.
- botocore.credentials: Wird zum Protokollieren des Abrufs und der Aktualisierung von Anmeldeinformationen verwendet.
- botocore.endpoint: Wird verwendet, um die Erstellung von Anfragen zu protokollieren, bevor sie über das Netzwerk gesendet werden.
- botocore.hooks: Wird zum Protokollieren von Ereignissen verwendet, die in der Bibliothek ausgelöst wurden.
- botocore.loaders: Wird für die Protokollierung verwendet, wenn Teile von AWS Servicemodellen geladen werden.
- botocore.parsers: Wird zum Protokollieren von AWS -Serviceantworten verwendet, bevor sie geparst werden.

- botocore.retryhandler: Wird zum Protokollieren der Verarbeitung von erneuten Versuchen von Serviceanfragen verwendet (Legacy-Modus). AWS
- botocore.retries.standard: Wird zum Protokollieren der Verarbeitung von erneuten Versuchen von Serviceanfragen verwendet (Standard- oder adaptiver Modus). AWS
- botocore.utils: Wird zum Protokollieren verschiedener Aktivitäten in der Bibliothek verwendet.
- botocore.waiter: Wird verwendet, um die Funktionalität von Kellnern zu protokollieren, die einen Dienst abfragen, bis ein bestimmter Status erreicht ist. AWS

Andere Bibliotheken protokollieren ebenfalls. Intern verwendet boto3 die URLlib3 eines Drittanbieters für die HTTP-Verbindungsverarbeitung. Wenn Ihnen Latenz wichtig ist, können Sie sich die Protokolle ansehen, um sicherzustellen, dass Ihr Pool gut ausgelastet ist, indem Sie nachsehen, wann urllib3 eine neue Verbindung aufbaut oder eine inaktive Verbindung schließt.

- urllib3.connectionpool: Wird für die Protokollierung von Ereignissen bei der Verwaltung des Verbindungspools verwendet.

Der folgende Codeausschnitt setzt die allgemeine Protokollierung auf `INFO`, während für Endpunkt- und Verbindungspoolaktivitäten die `DEBUG`-Protokollierung aktiviert wird:

```
import logging

logging.getLogger('boto3').setLevel(logging.INFO)
logging.getLogger('botocore').setLevel(logging.INFO)
logging.getLogger('botocore.endpoint').setLevel(logging.DEBUG)
logging.getLogger('urllib3.connectionpool').setLevel(logging.DEBUG)
```

Ereignis-Hooks

Botocore gibt während verschiedener Phasen seiner Ausführung Ereignisse aus. Sie können Handler für diese Ereignisse registrieren, sodass Ihr Handler immer dann aufgerufen wird, wenn ein Ereignis ausgelöst wird. Auf diese Weise können Sie das Verhalten von Botocore erweitern, ohne die internen Funktionen ändern zu müssen.

Angenommen, Sie möchten jedes Mal nachverfolgen, wenn in Ihrer Anwendung eine `PutItem`-Operation auf einer beliebigen DynamoDB-Tabelle ausgeführt wird. Dann könnten Sie sich auf das Ereignis `'provide-client-params.dynamodb.PutItem'` registrieren, um jeden Aufruf von `PutItem` innerhalb der zugehörigen Sitzung zu erfassen und zu protokollieren. Hier ein Beispiel:

```
import boto3
import botocore
import logging

def log_put_params(params, **kwargs):
    if 'TableName' in params and 'Item' in params:
        logging.info(f"PutItem on table {params['TableName']}: {params['Item']}")

logging.basicConfig(level=logging.INFO)

session = boto3.Session()
event_system = session.events

# Register our interest in hooking in when the parameters are provided to PutItem
event_system.register('provide-client-params.dynamodb.PutItem', log_put_params)

# Now, every time you use this session to put an item in DynamoDB,
# it will log the table name and item data.
dynamodb = session.resource('dynamodb')
table = dynamodb.Table('YourTableName')
table.put_item(
    Item={
        'pk': '123',
        'sk': 'cart#123',
        'item_data': 'YourItemData',
        # ... more attributes ...
    }
)
```

Innerhalb des Handlers können Sie die Parameter sogar programmatisch verändern, um das Verhalten anzupassen:

```
params['TableName'] = "NewTableName"
```

Weitere Informationen zu Ereignissen finden Sie in der Botocore-Dokumentation zu Ereignissen und in der boto3-Dokumentation zu Ereignissen.

Paginierung und der Paginator

Manche Anfragen wie Query und Scan begrenzen die Datenmenge, die mit einer einzelnen Anfrage zurückgegeben wird, und erfordern wiederholte Anfragen, um nachfolgende Seiten abzurufen.

Mit dem `limit`-Parameter können Sie die maximale Anzahl von Elementen steuern, die pro Seite gelesen werden sollen. Beispielsweise können Sie `limit` verwenden, um nur die letzten 10 Elemente abzurufen. Beachten Sie, dass der Grenzwert angibt, wie viel aus der Tabelle gelesen werden sollte, bevor eine Filterung angewendet wird. Es gibt keine Möglichkeit, genau 10 nach dem Filtern anzugeben. Sie können nur die Anzahl der vorgefilterten Filter kontrollieren und clientseitig überprüfen, ob Sie tatsächlich 10 abgerufen haben. Unabhängig vom Grenzwert hat jede Antwort immer eine maximale Größe von 1 MB.

Wenn die Antwort ein `LastEvaluatedKey` enthält, bedeutet dies, dass die Antwort beendet wurde, weil sie eine Anzahl- oder Größenbeschränkung erreicht hat. Dieser Schlüssel ist der letzte Schlüssel, der für diese Antwort ausgewertet wird. Sie können diesen `LastEvaluatedKey` abrufen und an einen Folgeaufruf als `ExclusiveStartKey` weitergeben, um den nächsten Abschnitt von diesem Startpunkt aus zu lesen. Wenn kein `LastEvaluatedKey` zurückgegeben wird, bedeutet das, dass keine weiteren Elemente zu der Abfrage oder dem Scan passen.

Hier ein einfaches Beispiel (unter Verwendung der Ressourcenschnittstelle, jedoch hat die Client-Schnittstelle das gleiche Muster), bei dem maximal 100 Elemente pro Seite gelesen werden und eine Schleife ausgeführt wird, bis alle Elemente gelesen sind.

```
import boto3

dynamodb = boto3.resource('dynamodb')
table = dynamodb.Table('YourTableName')

query_params = {
    'KeyConditionExpression': Key('pk').eq('123') & Key('sk').gt(1000),
    'Limit': 100
}

while True:
    response = table.query(**query_params)

    # Process the items however you like
    for item in response['Items']:
        print(item)

    # No LastEvaluatedKey means no more items to retrieve
    if 'LastEvaluatedKey' not in response:
        break

    # If there are possibly more items, update the start key for the next page
```

```
    query_params['ExclusiveStartKey'] = response['LastEvaluatedKey']
```

Der Einfachheit halber kann boto3 dies mit Paginatoren für Sie erledigen. Das funktioniert jedoch nur mit der Client-Schnittstelle. Hier ist der Code, der für die Verwendung von Paginatoren neu geschrieben wurde:

```
import boto3

dynamodb = boto3.client('dynamodb')

paginator = dynamodb.get_paginator('query')

query_params = {
    'TableName': 'YourTableName',
    'KeyConditionExpression': 'pk = :pk_val AND sk > :sk_val',
    'ExpressionAttributeValues': {
        ':pk_val': {'S': '123'},
        ':sk_val': {'N': '1000'},
    },
    'Limit': 100
}

page_iterator = paginator.paginate(**query_params)

for page in page_iterator:
    # Process the items however you like
    for item in page['Items']:
        print(item)
```

Weitere Informationen finden Sie im Leitfaden zu Paginatoren und in der API-Referenz für DynamoDB.Paginator.Query.

> **Note**
>
> Paginatoren haben auch ihre eigenen Konfigurationseinstellungen: `MaxItems`, `StartingToken` und `PageSize`. Für die Paginierung mit DynamoDB sollten Sie diese Einstellungen ignorieren.

Waiter

Waiter bieten die Möglichkeit, auf den Abschluss eines Vorgangs zu warten, bevor der nächste Schritt ausgeführt wird. Derzeit unterstützen sie nur das Warten auf die Erstellung oder Löschung einer Tabelle. Im Hintergrund prüft der Waiter alle 20 Sekunden, bis zu 25 Mal, ob der gewünschte Zustand erreicht wurde. Zwar könnten Sie diese Prüfung selbst implementieren, aber die Verwendung eines Waiters ist eine elegante Lösung beim Schreiben von Automatisierungsskripten.

Hier ist ein Beispielcode, der zeigt, wie Sie darauf warten können, dass eine bestimmte Tabelle erstellt wurde:

```
# Create a table, wait until it exists, and print its ARN
response = client.create_table(...)
waiter = client.get_waiter('table_exists')
waiter.wait(TableName='YourTableName')
print('Table created:', response['TableDescription']['TableArn']
```

Weitere Informationen finden Sie im Waiter-Leitfaden und in der Waiter-Referenz.

Programmieren von Amazon DynamoDB mit JavaScript

Dieses Handbuch bietet Programmierern, die Amazon DynamoDB mit verwenden möchten, eine Orientierung. JavaScript Erfahren Sie mehr über die verfügbaren Abstraktionsebenen AWS SDK für JavaScript, die Konfiguration von Verbindungen, den Umgang mit Fehlern, die Definition von Wiederholungsrichtlinien, die Verwaltung von Keep-Alive und mehr.

Themen

- Über AWS SDK für JavaScript
- Verwenden von AWS SDK für JavaScript V3
- Zugriff auf die JavaScript Dokumentation
- Abstraktionsebenen
- Verwenden der Hilfsfunktion Marshall
- Lesen von Elementen
- Bedingte Schreibvorgänge
- Paginierung
- Angeben der Konfiguration

- Waiter
- Fehlerbehandlung
- Protokollierung
- Überlegungen

Über AWS SDK für JavaScript

Das AWS SDK für JavaScript bietet Zugriff auf die AWS-Services Verwendung von Browserskripten oder Node.js. Diese Dokumentation konzentriert sich auf die neueste SDK-Version (V3). Die AWS SDK für JavaScript Version 3 wird von AWS als Open-Source-Projekt verwaltet, das auf GitHub gehostet wird. Probleme und Funktionsanfragen sind öffentlich und Sie können auf der Problemseite des GitHub Repositorys darauf zugreifen.

JavaScript V2 ähnelt V3, enthält jedoch Syntaxunterschiede. V3 ist modularer aufgebaut, was es einfacher macht, kleinere Abhängigkeiten zu versenden, und bietet erstklassigen TypeScript Support. Wir empfehlen die neueste SDK-Version zu verwenden.

Verwenden von AWS SDK für JavaScript V3

Mit dem Node Package Manager können Sie das SDK zu Ihrer Node.js-Anwendung hinzufügen. Die folgenden Beispiele zeigen, wie Sie die gängigsten SDK-Pakete für die Arbeit mit DynamoDB hinzufügen.

- `npm install @aws-sdk/client-dynamodb`
- `npm install @aws-sdk/lib-dynamodb`
- `npm install @aws-sdk/util-dynamodb`

Die Installation von Paketen fügt Verweise im Abschnitt „Abhängigkeiten“ zu Ihrer package.json-Projektdatei hinzu. Sie haben die Möglichkeit, die neuere ECMAScript Modulsyntax zu verwenden. Weitere Informationen zu diesen beiden Ansätzen finden Sie im Abschnitt „Überlegungen“.

Zugriff auf die JavaScript Dokumentation

Beginnen Sie mit der JavaScript Dokumentation mit den folgenden Ressourcen:

- Die JavaScript Kerndokumentation finden Sie im Entwicklerhandbuch. Installationsanweisungen finden Sie im Abschnitt Einrichtung.

- In der API-Referenzdokumentation können Sie sich mit allen verfügbaren Klassen und Methoden vertraut machen.
- Das SDK für JavaScript unterstützt viele AWS-Services andere als DynamoDB. Gehen Sie wie folgt vor, um eine spezifische API-Abdeckung für DynamoDB zu finden:

  1. Wählen Sie unter Services die Option DynamoDB und Bibliotheken aus. Hier wird der Low-Level-Client dokumentiert.
  2. Wählen Sie lib-dynamodb aus. Hier wird der High-Level-Client dokumentiert. Die beiden Clients stellen zwei verschiedene Abstraktionsebenen dar, die Sie verwenden können. Weitere Informationen zu Abstraktionsebenen finden Sie im folgenden Abschnitt.

Abstraktionsebenen

Das SDK für JavaScript V3 hat einen Low-Level-Client (`DynamoDBClient`) und einen High-Level-Client (). `DynamoDBDocumentClient`

Themen

- Low-Level-Client (DynamoDBClient)
- High-Level-Client (DynamoDBDocumentClient)

Low-Level-Client (**`DynamoDBClient`**)

Der Low-Level-Client bietet keine zusätzlichen Abstraktionen gegenüber dem zugrunde liegenden Wire-Protokoll. Es bietet Ihnen volle Kontrolle über alle Aspekte der Kommunikation, aber da keine Abstraktionen vorhanden sind, müssen Sie beispielsweise Elementdefinitionen im DynamoDB-JSON-Format bereitstellen.

Wie das folgende Beispiel zeigt, müssen bei diesem Format die Datentypen explizit angegeben werden. Ein S steht für einen Zeichenkettenwert und ein N steht für einen Zahlenwert. Zahlen werden beim Übertragen stets als Zeichenketten gesendet, die als Zahlentypen gekennzeichnet sind, um einen Präzisionsverlust zu vermeiden. Low-Level-API-Aufrufe haben ein Benennungsmuster wie `PutItemCommand` und `GetItemCommand`.

Im folgenden Beispiel wird der Low-Level-Client mit einem `Item` verwendet, das im DynamoDB-JSON-Format definiert ist:

```
const { DynamoDBClient, PutItemCommand } = require("@aws-sdk/client-dynamodb");
```

```
const client = new DynamoDBClient({});

async function addProduct() {
  const params = {
    TableName: "products",
    Item: {
      "id": { S: "Product01" },
      "description": { S: "Hiking Boots" },
      "category": { S: "footwear" },
      "sku": { S: "hiking-sku-01" },
      "size": { N: "9" }
    }
  };

  try {
    const data = await client.send(new PutItemCommand(params));
    console.log('result : ' + JSON.stringify(data));
  } catch (error) {
    console.error("Error:", error);
  }
}
addProduct();
```

High-Level-Client (**`DynamoDBDocumentClient`**)

Der DynamoDB-Dokumentenclient auf hoher Ebene bietet integrierte Komfortfunktionen, z. B. macht das manuelle Marshalling von Daten überflüssig und ermöglicht direkte Lese- und Schreibvorgänge mithilfe von Standardobjekten. JavaScript Die Dokumentation zu `lib-dynamodb` enthält eine Liste der Vorteile.

Um den `DynamoDBDocumentClient` zu instanziieren, erstellen Sie zunächst einen Low-Level-`DynamoDBClient` und umschließen ihn anschließend mit einem `DynamoDBDocumentClient`. Die Namenskonventionen der Funktionen unterscheiden sich leicht zwischen den beiden Paketen. Der Low-Level-Client verwendet beispielsweise `PutItemCommand`, während der High-Level-Client `PutCommand` verwendet. Durch die unterschiedlichen Namen können beide Funktionssets im selben Kontext koexistieren, sodass Sie sie innerhalb desselben Skripts kombinieren können.

```
const { DynamoDBClient } = require("@aws-sdk/client-dynamodb");
const { DynamoDBDocumentClient, PutCommand } = require("@aws-sdk/lib-dynamodb");
```

```
const client = new DynamoDBClient({});

const docClient = DynamoDBDocumentClient.from(client);

async function addProduct() {
  const params = {
    TableName: "products",
    Item: {
      id: "Product01",
      description: "Hiking Boots",
      category: "footwear",
      sku: "hiking-sku-01",
      size: 9,
    },
  };

  try {
    const data = await docClient.send(new PutCommand(params));
    console.log('result : ' + JSON.stringify(data));
  } catch (error) {
    console.error("Error:", error);
  }
}

addProduct();
```

Das Verwendungsmuster ist konsistent, wenn Sie Elemente mithilfe von API-Operationen wie `GetItem`, `Query` oder `Scan` lesen.

Verwenden der Hilfsfunktion Marshall

Sie können den Low-Level-Client verwenden und die Datentypen selbst mit marshall oder unmarshall konvertieren. Das Hilfspaket util-dynamodb enthält eine `marshall()`-Hilfsfunktion, die JSON entgegennimmt und DynamoDB-JSON erzeugt, sowie eine `unmarshall()`-Funktion, die den umgekehrten Vorgang ausführt. Im folgenden Beispiel wird der Low-Level-Client verwendet, wobei die Datenkonvertierung durch den Aufruf von `marshall()` erfolgt.

```
const { DynamoDBClient, PutItemCommand } = require("@aws-sdk/client-dynamodb");
const { marshall } = require("@aws-sdk/util-dynamodb");

const client = new DynamoDBClient({});
```

```
async function addProduct() {
  const params = {
    TableName: "products",
    Item: marshall({
      id: "Product01",
      description: "Hiking Boots",
      category: "footwear",
      sku: "hiking-sku-01",
      size: 9,
    }),
  };

  try {
    const data = await client.send(new PutItemCommand(params));
  } catch (error) {
    console.error("Error:", error);
  }
}
addProduct();
```

Lesen von Elementen

Um ein einzelnes Element aus DynamoDB zu lesen, verwenden Sie die API-Operation `GetItem`. Ähnlich wie beim `PutItem`-Befehl haben Sie die Wahl zwischen dem Low-Level-Client und dem High-Level-Dokument-Client. Das folgende Beispiel zeigt, wie Sie mit dem High-Level-Dokument-Client ein Element abrufen.

```
const { DynamoDBClient } = require("@aws-sdk/client-dynamodb");
const { DynamoDBDocumentClient, GetCommand } = require("@aws-sdk/lib-dynamodb");

const client = new DynamoDBClient({});

const docClient = DynamoDBDocumentClient.from(client);

async function getProduct() {
  const params = {
    TableName: "products",
    Key: {
      id: "Product01",
    },
  };
```

```
  try {
    const data = await docClient.send(new GetCommand(params));
    console.log('result : ' + JSON.stringify(data));
  } catch (error) {
    console.error("Error:", error);
  }
}

getProduct();
```

Verwenden Sie die `Query`-API-Operation, um mehrere Elemente zu lesen. Sie können den Low-Level-Client oder den Dokument-Client verwenden. Im folgenden Beispiel wird der High-Level-Dokument-Client verwendet.

```
const { DynamoDBClient } = require("@aws-sdk/client-dynamodb");
const {
  DynamoDBDocumentClient,
  QueryCommand,
} = require("@aws-sdk/lib-dynamodb");

const client = new DynamoDBClient({});

const docClient = DynamoDBDocumentClient.from(client);

async function productSearch() {
  const params = {
    TableName: "products",
    IndexName: "GSI1",
    KeyConditionExpression: "#category = :category and begins_with(#sku, :sku)",
    ExpressionAttributeNames: {
      "#category": "category",
      "#sku": "sku",
    },
    ExpressionAttributeValues: {
      ":category": "footwear",
      ":sku": "hiking",
    },
  };

  try {
    const data = await docClient.send(new QueryCommand(params));
    console.log('result : ' + JSON.stringify(data));
  } catch (error) {
```

```
    console.error("Error:", error);
  }
}

productSearch();
```

Bedingte Schreibvorgänge

DynamoDB-Schreibvorgänge können eine logische Bedingungsausdruck angeben, der als true ausgewertet werden muss, damit der Schreibvorgang ausgeführt wird. Wenn die Bedingung nicht als wahr ausgewertet wird, erzeugt die Schreiboperation eine Ausnahme. Der Bedingungsausdruck kann überprüfen, ob das Element bereits existiert oder ob seine Attribute bestimmten Beschränkungen entsprechen.

`ConditionExpression = "version = :ver AND size(VideoClip) < :maxsize"`

Wenn der Bedingungsausdruck fehlschlägt, können Sie `ReturnValuesOnConditionCheckFailure` verwenden, um anzufordern, dass die Fehlermeldung das Element enthält, das die Bedingungen nicht erfüllt hat. So können Sie leichter nachvollziehen, worin das Problem bestand. Weitere Informationen finden Sie unter Behandeln von bedingten Schreibfehlern in Szenarien mit hoher Parallelität mit Amazon DynamoDB.

```
try {
      const response = await client.send(new PutCommand({
          TableName: "YourTableName",
          Item: item,
          ConditionExpression: "attribute_not_exists(pk)",
          ReturnValuesOnConditionCheckFailure: "ALL_OLD"
      }));
  } catch (e) {
      if (e.name === 'ConditionalCheckFailedException') {
          console.log('Item already exists:', e.Item);
      } else {
          throw e;
      }
  }
```

Zusätzliche Codebeispiele, die andere Aspekte der Verwendung von JavsScript SDK V3 zeigen, sind in der JavaScript SDK V3-Dokumentation und im Repository verfügbar. DynamoDB-SDK-Examples GitHub

Paginierung

Themen

- Verwenden der paginateScan-Komfortmethode

Leseanfragen wie `Scan` oder `Query` geben wahrscheinlich mehrere Elemente in einem Datensatz zurück. Wenn Sie einen `Scan` oder eine `Query` mit einem `Limit`-Parameter durchführen, sendet das System nach dem Lesen dieser Anzahl von Elementen eine Teilantwort, und Sie müssen die Ergebnisse paginieren, um weitere Elemente abzurufen.

Das System liest pro Anfrage maximal 1 Megabyte an Daten. Wenn Sie einen `Filter`-Ausdruck verwenden, liest das System weiterhin maximal ein Megabyte von der Festplatte, gibt jedoch nur die Elemente dieses Megabytes zurück, die dem Filter entsprechen. Die Filteroperation kann für eine Seite null Elemente zurückgeben, erfordert aber dennoch eine weitere Paginierung, bevor die Suche abgeschlossen ist.

Sie sollten im Antwortobjekt nach dem `LastEvaluatedKey` suchen und diesen als `ExclusiveStartKey`-Parameter in einer nachfolgenden Anfrage verwenden, um die Datenabfrage fortzusetzen. Dies dient als eine Art Lesezeichen, wie im folgenden Beispiel gezeigt.

> **Note**
>
> Im Beispiel wird beim ersten Durchlauf ein null-Wert für `lastEvaluatedKey` als `ExclusiveStartKey` übergeben, und das ist zulässig.

Beispiel - Verwenden des `LastEvaluatedKey`:

```
const { DynamoDBClient, ScanCommand } = require("@aws-sdk/client-dynamodb");

const client = new DynamoDBClient({});

async function paginatedScan() {
  let lastEvaluatedKey;
  let pageCount = 0;

  do {
    const params = {
      TableName: "products",
```

```
      ExclusiveStartKey: lastEvaluatedKey,
    };

    const response = await client.send(new ScanCommand(params));
    pageCount++;
    console.log(`Page ${pageCount}, Items:`, response.Items);
    lastEvaluatedKey = response.LastEvaluatedKey;
  } while (lastEvaluatedKey);
}

paginatedScan().catch((err) => {
  console.error(err);
});
```

Verwenden der **`paginateScan`**-Komfortmethode

Das SDK stellt Komfortmethoden namens `paginateScan` und `paginateQuery` bereit, die diese Arbeit für Sie übernehmen und die wiederholten Anfragen im Hintergrund ausführen. Die maximale Anzahl der pro Anfrage zu lesenden Elemente geben Sie über den Standardparameter `Limit` an.

```
const { DynamoDBClient, paginateScan } = require("@aws-sdk/client-dynamodb");

const client = new DynamoDBClient({});

async function paginatedScanUsingPaginator() {
  const params = {
    TableName: "products",
    Limit: 100
  };

  const paginator = paginateScan({client}, params);

  let pageCount = 0;

  for await (const page of paginator) {
    pageCount++;
    console.log(`Page ${pageCount}, Items:`, page.Items);
  }
}

paginatedScanUsingPaginator().catch((err) => {
  console.error(err);
```

```
});
```

> **Note**
>
> Regelmäßige vollständige Tabellenscans sind kein empfohlenes Zugriffsmuster, es sei denn, die Tabelle ist klein.

Angeben der Konfiguration

Themen

- Config für Timeouts
- Config für Keepalive
- Config für Wiederholungsversuche

Beim Einrichten des `DynamoDBClient` können Sie verschiedene Konfigurationsüberschreibungen festlegen, indem Sie ein Konfigurationsobjekt an den Konstruktor übergeben. Beispielsweise können Sie die Region angeben, mit der eine Verbindung hergestellt werden soll, falls sie dem aufrufenden Kontext nicht bereits bekannt ist, oder die zu verwendende Endpunkt-URL. Dies ist besonders nützlich, wenn Sie eine lokale DynamoDB-Instance für Entwicklungszwecke ansprechen möchten.

```
const client = new DynamoDBClient({
  region: "eu-west-1",
  endpoint: "http://localhost:8000",
});
```

Config für Timeouts

DynamoDB verwendet HTTPS für die Client-Server-Kommunikation. Sie können einige Aspekte der HTTP-Ebene steuern, indem Sie ein `NodeHttpHandler`-Objekt bereitstellen. Beispielsweise können Sie die wichtigsten Timeout-Werte `connectionTimeout` und `requestTimeout` anpassen. `connectionTimeout` ist die maximale Dauer in Millisekunden, die der Client beim Versuch, eine Verbindung herzustellen, wartet, bevor er aufgibt.

`requestTimeout` legt fest, wie lange der Client nach dem Senden einer Anfrage auf eine Antwort wartet, ebenfalls in Millisekunden. Die Standardwerte für beide Parameter sind null, was bedeutet, dass kein Timeout aktiviert ist und der Client unbegrenzt auf eine Antwort wartet, falls diese nicht

eintrifft. Es ist empfehlenswert, sinnvolle Timeout-Werte zu setzen, damit bei Netzwerkproblemen ein Fehler ausgelöst wird und eine neue Anfrage gestartet werden kann. Beispiel:

```
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";
import { NodeHttpHandler } from "@smithy/node-http-handler";

const requestHandler = new NodeHttpHandler({
  connectionTimeout: 2000,
  requestTimeout: 2000,
});

const client = new DynamoDBClient({
  requestHandler
});
```

> **Note**
>
> Im bereitgestellten Beispiel wird der Smithy-Import verwendet. Smithy ist eine Sprache zur Definition von Diensten und ist eine Open-Source-Sprache SDKs, die von verwaltet wird. AWS

Zusätzlich zur Konfiguration von Timeout-Werten können Sie die maximale Anzahl von Sockets festlegen, was eine höhere Anzahl gleichzeitiger Verbindungen pro Ursprung ermöglicht. Der Entwicklerleitfaden enthält Details zur Konfiguration des Parameters `maxSockets`.

Config für Keepalive

Bei der Verwendung von HTTPS erfordert die erste Anfrage immer eine gewisse back-and-forth Kommunikation, um eine sichere Verbindung herzustellen. HTTP Keepalive ermöglicht es, dass nachfolgende Anfragen die bereits aufgebaute Verbindung wiederverwenden, wodurch die Effizienz steigt und die Latenz sinkt. HTTP Keep-Alive ist standardmäßig mit JavaScript V3 aktiviert.

Es gibt jedoch eine Begrenzung, wie lange eine inaktive Verbindung aufrechterhalten werden kann. Wenn Sie eine Verbindung im Leerlauf haben, die für die nächste Anfrage wiederverwendet werden soll, sollten Sie in Erwägung ziehen, regelmäßig Anfragen zu senden, z. B. einmal pro Minute.

> **Note**
>
> Beachten Sie, dass in der älteren V2-Version des SDK Keepalive standardmäßig deaktiviert war, was bedeutet, dass jede Verbindung unmittelbar nach der Nutzung geschlossen wurde. Wenn Sie V2 verwenden, können Sie diese Einstellung überschreiben.

Config für Wiederholungsversuche

Wenn das SDK eine Fehlermeldung erhält und der Fehler laut SDK als wiederholbar gilt, z. B. bei einer Drosselungsausnahme oder einem temporären Service-Ausnahmefehler, wird automatisch ein erneuter Versuch gestartet. Dies geschieht für den Aufrufer unsichtbar, außer dass die Anfrage möglicherweise etwas länger dauert, bis sie erfolgreich ist.

Das SDK für JavaScript V3 stellt standardmäßig insgesamt 3 Anfragen, bevor es aufgibt und den Fehler an den aufrufenden Kontext weitergibt. Sie können sowohl die Anzahl als auch die Frequenz dieser Wiederholungen anpassen.

Der Konstruktor des `DynamoDBClient` akzeptiert die Einstellung `maxAttempts`, mit der Sie die maximale Anzahl von Versuchen festlegen können. Im folgenden Beispiel wird der Wert von standardmäßig 3 auf insgesamt 5 erhöht. Wenn Sie ihn auf 0 oder 1 setzen, bedeutet das, dass Sie keine automatischen Wiederholungen wünschen und wiederholbare Fehler selbst im catch-Block behandeln möchten.

```
const client = new DynamoDBClient({
  maxAttempts: 5,
});
```

Sie können das Timing der Wiederholungen mit einer benutzerdefinierten Wiederholungsstrategie steuern. Dazu importieren Sie das Hilfspaket `util-retry` und erstellen eine benutzerdefinierte Backoff-Funktion, die die Wartezeit zwischen den Wiederholungen basierend auf der aktuellen Anzahl der Versuche berechnet.

Im folgenden Beispiel sind maximal 5 Versuche vorgesehen, mit Verzögerungen von 15, 30, 90 und 360 Millisekunden, falls der erste Versuch fehlschlägt. Die benutzerdefinierte Backoff-Funktion `calculateRetryBackoff` nimmt die Nummer des Wiederholungsversuchs entgegen (beginnend mit 1 für den ersten Wiederholungsversuch) und gibt die Anzahl der Millisekunden zurück, die vor dem nächsten Versuch gewartet werden sollen.

```
const { ConfiguredRetryStrategy } = require("@aws-sdk/util-retry");

const calculateRetryBackoff = (attempt) => {
  const backoffTimes = [15, 30, 90, 360];
  return backoffTimes[attempt - 1] || 0;
};

const client = new DynamoDBClient({
  retryStrategy: new ConfiguredRetryStrategy(
    5, // max attempts.
    calculateRetryBackoff // backoff function.
  ),
});
```

Waiter

Der DynamoDB-Client enthält außerdem zwei nützliche Waiter-Funktionen, die beim Erstellen, Ändern oder Löschen von Tabellen verwendet werden können, wenn der Code erst fortfahren soll, nachdem die Tabellenoperation abgeschlossen ist. Beispielsweise können Sie eine Tabelle bereitstellen, die Funktion `waitUntilTableExists` aufrufen, und der Code blockiert dann, bis die Tabelle in den Status ACTIVE wechselt. Der Waiter fragt den DynamoDB-Service alle 20 Sekunden intern über `describe-table` ab.

```
import {waitUntilTableExists, waitUntilTableNotExists} from "@aws-sdk/client-dynamodb";

… <create table details>

const results = await waitUntilTableExists({client: client, maxWaitTime: 180},
 {TableName: "products"});
if (results.state == 'SUCCESS') {
  return results.reason.Table
}
console.error(`${results.state} ${results.reason}`);
```

Die Funktion `waitUntilTableExists` gibt die Kontrolle erst dann zurück, wenn ein `describe-table`-Befehl erfolgreich ausgeführt werden kann, der den Tabellenstatus als ACTIVE bestätigt. Dadurch können Sie `waitUntilTableExists` nicht nur verwenden, um auf die vollständige Erstellung einer Tabelle zu warten, sondern auch auf Änderungen wie das Hinzufügen eines GSI-Index, dessen Anwendung einige Zeit in Anspruch nehmen kann, bevor die Tabelle wieder den Status ACTIVE erreicht.

Fehlerbehandlung

In den bisherigen Beispielen wurden alle Fehler pauschal abgefangen. In praktischen Anwendungen ist es jedoch wichtig, zwischen verschiedenen Fehlertypen zu unterscheiden und eine präzisere Fehlerbehandlung zu implementieren.

Fehlermeldungen von DynamoDB enthalten Metadaten, darunter den Namen des Fehlers. Sie können Fehler abfangen und anhand der möglichen Fehlerbezeichnungen entscheiden, wie weiter vorzugehen ist. Bei serverseitigen Fehlern können Sie den Operator `instanceof` zusammen mit den im Paket `@aws-sdk/client-dynamodb` exportierten Fehlertypen verwenden, um die Fehlerbehandlung effizient zu gestalten.

Beachten Sie, dass diese Fehler erst auftreten, nachdem alle Wiederholungsversuche ausgeschöpft wurden. Wenn ein Fehler erneut versucht und schließlich erfolgreich abgeschlossen wird, erscheint aus Sicht des Codes kein Fehler, sondern nur eine leicht erhöhte Latenz. Wiederholungen werden in den CloudWatch Amazon-Diagrammen als erfolglose Anfragen angezeigt, z. B. Drosselungs- oder Fehleranfragen. Wenn der Client die maximale Anzahl an Wiederholungen erreicht, gibt er auf und löst eine Ausnahme aus. Dies signalisiert, dass keine weiteren Wiederholungen erfolgen werden.

Im Folgenden sehen Sie einen Codeausschnitt, der zeigt, wie Sie einen Fehler abfangen und je nach Fehlertyp entsprechend reagieren können.

```
import {
  ResourceNotFoundException
  ProvisionedThroughputExceededException,
  DynamoDBServiceException,
} from "@aws-sdk/client-dynamodb";

try {
  await client.send(someCommand);
} catch (e) {
    if (e instanceof ResourceNotFoundException) {
      // Handle ResourceNotFoundException
    } else if (e instanceof ProvisionedThroughputExceededException) {
      // Handle ProvisionedThroughputExceededException
    } else if (e instanceof DynamoDBServiceException) {
      // Handle DynamoDBServiceException
    } else {
      // Other errors such as those from the SDK
      if (e.name === "TimeoutError") {
        // Handle SDK TimeoutError.
```

```
      } else {
        // Handle other errors.
      }
    }
}
```

Allgemeine Fehlerzeichenfolgen finden Sie unter the section called “Fehlerbehandlung” im DynamoDB-Entwicklerhandbuch. Die genauen Fehler, die bei einem bestimmten API-Aufruf auftreten können, finden Sie in der Dokumentation zu diesem API-Aufruf, z. B. in den Abfrage-API-Dokumenten.

Die Metadaten von Fehlern enthalten je nach Fehler zusätzliche Eigenschaften. Bei einem `TimeoutError` enthält die Fehlermeldung Metadaten wie die Anzahl der durchgeführten Versuche und die `totalRetryDelay`, wie im Folgenden gezeigt.

```
{
  "name": "TimeoutError",
  "$metadata": {
    "attempts": 3,
    "totalRetryDelay": 199
  }
}
```

Wenn Sie Ihre eigene Wiederholungsrichtlinie verwalten, sollten Sie zwischen Drosselungen und Fehlern unterscheiden:

- Eine Drosselung (gekennzeichnet durch `ProvisionedThroughputExceededException` oder `ThrottlingException`) weist auf einen fehlerfreien Service hin, der Sie darüber informiert, dass Sie Ihre Lese- oder Schreibkapazität in einer DynamoDB-Tabelle oder Partition überschritten haben. Mit jeder Millisekunde, die vergeht, wird etwas mehr Lese- oder Schreibkapazität zur Verfügung gestellt, sodass Sie schnell, z. B. alle 50 ms, erneut versuchen können, auf die neu freigegebene Kapazität zuzugreifen.

  Bei Drosselungen ist kein exponentieller Backoff erforderlich, da Drosselungen für DynamoDB leichtgewichtig sind und keine Gebühren pro Anfrage anfallen. Exponentielles Backoff weist längere Wartezeiten jenen Client-Threads zu, die bereits am längsten gewartet haben, wodurch sich die statistischen Kennzahlen p50 und p99 weiter nach außen verschieben.
- Ein Fehler (z. B. `InternalServerError` oder `ServiceUnavailable`) weist auf ein vorübergehendes Problem mit dem Service, möglicherweise der gesamten Tabelle oder nur mit der

Partition hin, aus der Sie lesen oder in die Sie schreiben. Bei Fehlern sollten Sie längere Pausen zwischen den Wiederholungsversuchen einlegen (etwa 250 ms oder 500 ms) und Jitter verwenden, um die Wiederholungen zeitlich zu staffeln.

Protokollierung

Aktivieren Sie die Protokollierung, um mehr Details darüber zu erhalten, was das SDK im Hintergrund tut. Sie können dazu einen Parameter im `DynamoDBClient` setzen, wie im folgenden Beispiel gezeigt. Zusätzliche Protokollinformationen erscheinen in der Konsole und enthalten Metadaten wie den Statuscode und die verbrauchte Kapazität. Wenn Sie den Code lokal in einem Terminalfenster ausführen, erscheinen die Protokolle dort. Wenn Sie den Code ausführen und CloudWatch Amazon-Logs eingerichtet haben, wird die Konsolenausgabe dort geschrieben. AWS Lambda

```
const client = new DynamoDBClient({
  logger: console
});
```

Sie können auch in die internen Aktivitäten des SDK eingreifen und benutzerdefinierte Protokollierung durchführen, sobald bestimmte Ereignisse eintreten. Das folgende Beispiel verwendet den `middlewareStack` des Clients, um jede Anfrage abzufangen, während sie vom SDK gesendet wird, und sie in Echtzeit zu protokollieren.

```
const client = new DynamoDBClient({});

client.middlewareStack.add(
  (next) => async (args) => {
    console.log("Sending request from AWS SDK", { request: args.request });
    return next(args);
  },
  {
    step: "build",
    name: "log-ddb-calls",
  }
);
```

Der `MiddlewareStack` bietet eine leistungsstarke Schnittstelle, um das Verhalten des SDK zu beobachten und zu steuern. Weitere Informationen finden Sie im Blog Introducing Middleware Stack in Modular AWS SDK für JavaScript.

Überlegungen

Bei der Implementierung AWS SDK für JavaScript in Ihrem Projekt sollten Sie folgende weitere Faktoren berücksichtigen.

Modulsysteme

Das SDK unterstützt zwei Modulsysteme, CommonJS und ES (ECMAScript). CommonJS verwendet die `require`-Funktion, während ES das `import`-Schlüsselwort verwendet.

1. Common JS – `const { DynamoDBClient, PutItemCommand } = require("@aws-sdk/client-dynamodb");`
2. ES (ECMAScript— `import { DynamoDBClient, PutItemCommand } from "@aws-sdk/client-dynamodb";`

Der Projekttyp bestimmt, welches Modulsystem verwendet wird, und wird im Abschnitt Typ Ihrer package.json-Datei angegeben. Standardmäßig ist dies CommonJS. Verwenden Sie `"type": "module"`, um ein ES-Projekt zu kennzeichnen. Wenn Sie ein bestehendes Node.js-Projekt haben, das das CommonJS-Paketformat verwendet, können Sie dennoch Funktionen mit der moderneren Import-Syntax von SDK V3 hinzufügen, indem Sie Ihre Funktionsdateien mit der Endung .mjs benennen. Dadurch kann die Codedatei als ES (ECMAScript) behandelt werden.

Asynchrone Operationen

Viele Codebeispiele verwenden Callbacks und Promises, um die Ergebnisse von DynamoDB-Operationen zu verarbeiten. In der heutigen Zeit wird JavaScript diese Komplexität nicht mehr benötigt und Entwickler können die Vorteile der prägnanteren und lesbareren async/await Syntax für asynchrone Operationen nutzen.

Laufzeit des Webbrowsers

Web- und Mobilentwickler, die mit React oder React Native bauen, können das SDK für ihre JavaScript Projekte verwenden. Mit der früheren Version V2 des SDK mussten Webentwickler das vollständige SDK in den Browser laden und dabei auf ein SDK-Image verweisen, das unter https://sdk.amazonaws.com/js/ gehostet wird.

Mit V3 ist es möglich, mit Webpack nur die erforderlichen V3-Client-Module und alle erforderlichen JavaScript Funktionen in einer einzigen JavaScript Datei zu bündeln und sie in einem Skript-Tag auf Ihren HTML-Seiten hinzuzufügen, wie im Abschnitt Erste Schritte in einem Browserskript der SDK-Dokumentation beschrieben. `<head>`

DAX-Operationen auf Datenebene

Die Datenebenenoperationen von Amazon DynamoDB Streams Accelerator (DAX) werden vom SDK für JavaScript V3 unterstützt.

Programmieren von DynamoDB mit dem AWS SDK for Java 2.x

Dieser Programmierleitfaden bietet eine Einführung für Entwickler, die Amazon DynamoDB mit Java verwenden möchten. Es werden verschiedene Konzepte wie Abstraktionsschichten, Konfigurationsmanagement, Fehlerbehandlung, Steuerung von Wiederholungsrichtlinien und Verwaltung von Keepalive-Verbindungen behandelt.

Themen

- Über die AWS SDK for Java 2.x
- Erste Schritte mit dem AWS SDK for Java 2.x
- Überprüfung der AWS SDK for Java 2.x Dokumentation
- Unterstützte Schnittstellen
- Weitere Codebeispiele
- Synchrone und asynchrone Programmierung
- HTTP-Clients
- Konfigurieren eines HTTP-Clients
- Fehlerbehandlung
- AWS Anforderungs-ID
- Protokollierung
- Paginierung
- Annotationen für Datenklassen

Über die AWS SDK for Java 2.x

Sie können von Java aus mit dem offiziellen auf DynamoDB zugreifen. AWS SDK für Java Das SDK für Java ist in zwei Versionen verfügbar: 1.x und 2.x. Das end-of-support für 1.x wurde am 12. Januar 2024 angekündigt. Es wird am 31. Juli 2024 in den Wartungsmodus wechseln und end-of-support ist am 31. Dezember 2025 fällig. Für neue Entwicklungen wird dringend empfohlen, Version 2.x zu

verwenden, die erstmals 2018 veröffentlicht wurde. Dieses Handbuch richtet sich ausschließlich an Version 2.x und konzentriert sich auf die für DynamoDB relevanten Teile des SDK.

Informationen zu Wartung und Support für finden Sie in den AWS SDKs Wartungsrichtlinien für AWS SDK und Tools sowie in der Matrix zur Unterstützung der Tools-Versionen im Referenzhandbuch AWS SDKs und im Tools-Referenzhandbuch.AWS SDKs

Das AWS SDK for Java 2.x ist eine grundlegende Neufassung der 1.x-Codebasis. Das SDK for Java 2.x unterstützt moderne Java-Funktionen, wie z. B. das in Java 8 I/O eingeführte Non-Blocking. Das SDK for Java 2.x bietet auch Unterstützung für steckbare HTTP-Client-Implementierungen, um mehr Flexibilität bei Netzwerkverbindungen und Konfigurationsoptionen zu bieten.

Eine bemerkenswerte Änderung zwischen dem SDK for Java 1.x und dem SDK for Java 2.x ist die Verwendung eines neuen Paketnamens. Das Java 1.x SDK verwendet den `com.amazonaws`-Paketnamen, während das Java 2.x SDK `software.amazon.awssdk` verwendet. In ähnlicher Weise verwenden Maven-Artefakte für das Java 1.x-SDK die `com.amazonaws-groupId`, während Java 2.x-SDK-Artefakte die `software.amazon.awssdk-groupId` verwenden.

> **Important**
>
> Die Version AWS SDK für Java 1.x hat ein DynamoDB-Paket namens. `com.amazonaws.dynamodbv2` Die „v2“ im Paketnamen bedeutet nicht, dass es für Java 2 (J2SE) ist. Vielmehr gibt „v2“ an, dass das Paket die zweite Version der DynamoDB-Low-Level-API anstelle der ursprünglichen Version der Low-Level-API unterstützt.

Support für Java-Versionen

Das AWS SDK for Java 2.x bietet volle Unterstützung für Java-Releases mit langfristigem Support (LTS).

Erste Schritte mit dem AWS SDK for Java 2.x

Das folgende Tutorial zeigt Ihnen, wie Sie Apache Maven verwenden, um Abhängigkeiten für das SDK für Java 2.x zu definieren. Dieses Tutorial zeigt Ihnen auch, wie Sie den Code schreiben, der eine Verbindung zu DynamoDB herstellt, um die verfügbaren DynamoDB-Tabellen aufzulisten. Das Tutorial in diesem Handbuch basiert auf dem Tutorial Erste Schritte mit dem AWS SDK for Java 2.x im AWS SDK for Java 2.x -Entwicklerhandbuch. Wir haben dieses Tutorial so überbearbeitet, dass DynamoDB statt Amazon S3 aufgerufen wird.

Zum Durcharbeiten dieses Tutorials ist Folgendes erforderlich:

- Schritt 1: Einrichtung für dieses Tutorial
- Schritt 2: Erstellen des Projekts
- Schritt 3: Schreiben des Codes
- Schritt 4: Anwendung erstellen und ausführen

Schritt 1: Einrichtung für dieses Tutorial

Bevor Sie mit diesem Tutorial beginnen, brauchen Sie Folgendes:

- Berechtigung zum Zugriff auf DynamoDB.
- Eine Java-Entwicklungsumgebung, die mit Single Sign-On-Zugriff für die AWS-Services AWS-Zugangsportal Verwendung von konfiguriert ist.

Folgen Sie zur Einrichtung dieses Tutorials den Anleitungen in der Übersicht über die Einrichtung im AWS SDK for Java 2.x -Entwicklerhandbuch. Nachdem Sie Ihre Entwicklungsumgebung mit Single Sign-On-Zugriff für das Java SDK konfiguriert haben und eine aktive AWS -Access-Portal-Sitzung eingerichtet haben, fahren Sie mit Schritt 2 dieses Tutorials fort.

Schritt 2: Erstellen des Projekts

Um das Projekt für dieses Tutorial zu erstellen, führen Sie einen Maven-Befehl aus, der Sie zur Eingabe der Konfiguration des Projekts auffordert. Nachdem alle Daten eingegeben und bestätigt wurden, beendet Maven den Aufbau des Projekts, indem es eine `pom.xml`-Datei und Stub-Java-Dateien erstellt.

1. Öffnen Sie ein Terminal oder ein Befehlszeilenfenster und navigieren Sie zu einem Verzeichnis Ihrer Wahl, z. B. zu Ihrem Ordner `Desktop` oder `Home`.
2. Geben Sie im Terminal folgenden Befehl ein und drücken Sie anschließend die Eingabetaste.

```
mvn archetype:generate \
   -DarchetypeGroupId=software.amazon.awssdk \
   -DarchetypeArtifactId=archetype-app-quickstart \
   -DarchetypeVersion=2.22.0
```

3. Geben Sie für jede Prompt den in der zweiten Spalte aufgeführten Wert ein.

Prompt	Einzugebender Wert
`Define value for property 'service':`	`dynamodb`
`Define value for property 'httpClient' :`	`apache-client`
`Define value for property 'nativeImage' :`	`false`
`Define value for property 'credentialProvider'`	`identity-center`
`Define value for property 'groupId':`	`org.example`
`Define value for property 'artifactId':`	`getstarted`
`Define value for property 'version' 1.0-SNAPSHOT:`	*<Enter>*
`Define value for property 'package' org.example:`	*<Enter>*

4. Nachdem Sie den letzten Wert eingegeben haben, listet Maven die von Ihnen getroffenen Entscheidungen auf. Geben Sie zur Bestätigung Y ein. Oder geben Sie N ein und geben Sie dann Ihre Auswahl erneut ein.

Maven erstellt einen Projektordner namens `getstarted`, der auf dem von Ihnen eingegebenen `artifactId`-Wert basiert. Suchen Sie im `getstarted`-Ordners nach einer Datei namens `README.md`, die Sie überprüfen können, nach einer `pom.xml`-Datei und einem `src`-Verzeichnis.

Maven erstellt den folgenden Verzeichnisbaum.

```
getstarted
 ### README.md
 ### pom.xml
```

```
### src
    ### main
    #   ### java
    #   #   ### org
    #   #       ### example
    #   #           ### App.java
    #   #           ### DependencyFactory.java
    #   #           ### Handler.java
    #   ### resources
    #       ### simplelogger.properties
    ### test
        ### java
            ### org
                ### example
                    ### HandlerTest.java

10 directories, 7 files
```

Unten sehen Sie die Inhalte der `pom.xml`-Projektdatei.

`pom.xml`

Der `dependencyManagement`-Abschnitt enthält eine Abhängigkeit von AWS SDK for Java 2.x, und der `dependencies`-Abschnitt enthält eine Abhängigkeit für DynamoDB. Die Angabe dieser Abhängigkeiten zwingt Maven, die entsprechenden `.jar`-Dateien in Ihren Java-Klassenpfad aufzunehmen. Standardmäßig enthält das AWS SDK nicht alle Klassen für alle AWS-Services. Wenn Sie die Low-Level-Schnittstelle für DynamoDB verwenden, sollte eine Abhängigkeit vom `dynamodb`-Artefakt vorhanden sein. Wenn Sie die High-Level-Schnittstelle verwenden, eine Abhängigkeit für den `dynamodb-enhanced`-Artefakt. Sollten Sie die relevanten Abhängigkeiten nicht angeben, kann Ihr Code nicht kompiliert werden. Das Projekt verwendet Java 1.8 aufgrund des `1.8`-Werts in den `maven.compiler.source`- und `maven.compiler.target`-Eigenschaften.

```
<?xml version="1.0" encoding="UTF-8"?>
 <project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/
XMLSchema-instance" xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://
maven.apache.org/xsd/maven-4.0.0.xsd">
     <modelVersion>4.0.0</modelVersion>

     <groupId>org.example</groupId>
     <artifactId>getstarted</artifactId>
     <version>1.0-SNAPSHOT</version>
     <packaging>jar</packaging>
```

```
    <properties>
        <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
        <maven.compiler.source>1.8</maven.compiler.source>
        <maven.compiler.target>1.8</maven.compiler.target>
        <maven.shade.plugin.version>3.2.1</maven.shade.plugin.version>
        <maven.compiler.plugin.version>3.6.1</maven.compiler.plugin.version>
        <exec-maven-plugin.version>1.6.0</exec-maven-plugin.version>
        <aws.java.sdk.version>2.22.0</aws.java.sdk.version> <-------- SDK version
picked up from archetype version.
        <slf4j.version>1.7.28</slf4j.version>
        <junit5.version>5.8.1</junit5.version>
    </properties>

    <dependencyManagement>
        <dependencies>
            <dependency>
                <groupId>software.amazon.awssdk</groupId>
                <artifactId>bom</artifactId>
                <version>${aws.java.sdk.version}</version>
                <type>pom</type>
                <scope>import</scope>
            </dependency>
        </dependencies>
    </dependencyManagement>

    <dependencies>
        <dependency>
            <groupId>software.amazon.awssdk</groupId>
            <artifactId>dynamodb</artifactId>  <-------- DynamoDB dependency
            <exclusions>
                <exclusion>
                    <groupId>software.amazon.awssdk</groupId>
                    <artifactId>netty-nio-client</artifactId>
                </exclusion>
                <exclusion>
                    <groupId>software.amazon.awssdk</groupId>
                    <artifactId>apache-client</artifactId>
                </exclusion>
            </exclusions>
        </dependency>

        <dependency>
            <groupId>software.amazon.awssdk</groupId>
```

```
            <artifactId>sso</artifactId> <-------- Required for identity center
authentication.
        </dependency>

        <dependency>
            <groupId>software.amazon.awssdk</groupId>
            <artifactId>ssooidc</artifactId> <-------- Required for identity center
authentication.
        </dependency>

        <dependency>
            <groupId>software.amazon.awssdk</groupId>
            <artifactId>apache-client</artifactId> <-------- HTTP client specified.
            <exclusions>
                <exclusion>
                    <groupId>commons-logging</groupId>
                    <artifactId>commons-logging</artifactId>
                </exclusion>
            </exclusions>
        </dependency>

        <dependency>
            <groupId>org.slf4j</groupId>
            <artifactId>slf4j-api</artifactId>
            <version>${slf4j.version}</version>
        </dependency>

        <dependency>
            <groupId>org.slf4j</groupId>
            <artifactId>slf4j-simple</artifactId>
            <version>${slf4j.version}</version>
        </dependency>

        <!-- Needed to adapt Apache Commons Logging used by Apache HTTP Client to
Slf4j to avoid
        ClassNotFoundException: org.apache.commons.logging.impl.LogFactoryImpl during
runtime -->
        <dependency>
            <groupId>org.slf4j</groupId>
            <artifactId>jcl-over-slf4j</artifactId>
            <version>${slf4j.version}</version>
        </dependency>

        <!-- Test Dependencies -->
```

```
        <dependency>
            <groupId>org.junit.jupiter</groupId>
            <artifactId>junit-jupiter</artifactId>
            <version>${junit5.version}</version>
            <scope>test</scope>
        </dependency>
    </dependencies>

    <build>
        <plugins>
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-compiler-plugin</artifactId>
                <version>${maven.compiler.plugin.version}</version>
            </plugin>
        </plugins>
    </build>

</project>
```

Schritt 3: Schreiben des Codes

Der folgende Code zeigt die App-Klasse, die von Maven erstellt wird. Die `main`-Methode ist der Einstiegspunkt in die Anwendung, die eine Instance der `Handler`-Klasse erstellt und dann ihre `sendRequest`-Methode aufruft.

`App`-Klasse

```
package org.example;
 import org.slf4j.Logger;
 import org.slf4j.LoggerFactory;

 public class App {
     private static final Logger logger = LoggerFactory.getLogger(App.class);

     public static void main(String... args) {
         logger.info("Application starts");

         Handler handler = new Handler();
         handler.sendRequest();

         logger.info("Application ends");
     }
```

```
}
```

Die `DependencyFactory`-Klasse, die Maven erstellt, enthält die `dynamoDbClient`-Factory-Methode, die eine `DynamoDbClient`-Instance erstellt und zurückgibt. Die `DynamoDbClient`-Instance verwendet eine Instance des Apache-basierten HTTP-Clients. Dies liegt daran, dass Sie `apache-client` angegeben haben, als Maven Sie nach dem zu verwendenden HTTP-Client gefragt hat.

Der folgende Code zeigt die Klasse `DependencyFactory`.

DependencyFactory Klasse

```
package org.example;

 import software.amazon.awssdk.http.apache.ApacheHttpClient;
 import software.amazon.awssdk.services.dynamodb.DynamoDbClient;

 /**
  * The module containing all dependencies required by the {@link Handler}.
  */
 public class DependencyFactory {

     private DependencyFactory() {}

     /**
      * @return an instance of DynamoDbClient
      */
     public static DynamoDbClient dynamoDbClient() {
         return DynamoDbClient.builder()
                         .httpClientBuilder(ApacheHttpClient.builder())
                         .build();
     }
 }
```

Die `Handler`-Klasse enthält die Hauptlogik Ihres Programms. Wenn eine Instance von `Handler` in der Klasse `App` erstellt wird, stellt `DependencyFactory` den Service-Client für `DynamoDbClient` bereit. Ihr Code verwendet die `DynamoDbClient`-Instance, um DynamoDB aufzurufen.

Maven generiert die folgende `Handler`-Klasse mit einem *`TODO`*-Kommentar. Der nächste Schritt im Tutorial ersetzt den *`TODO`*-Kommentar durch Code.

`Handler`-Klasse, von Maven generiert

```
package org.example;

 import software.amazon.awssdk.services.dynamodb.DynamoDbClient;


 public class Handler {
     private final DynamoDbClient dynamoDbClient;

     public Handler() {
         dynamoDbClient = DependencyFactory.dynamoDbClient();
     }

     public void sendRequest() {
         // TODO: invoking the API calls using dynamoDbClient.
     }
 }
```

Um die Logik zu ergänzen, ersetzen Sie den gesamten Inhalt der Klasse `Handler` durch den folgenden Code: Die Methode `sendRequest` ist vollständig implementiert, und die erforderlichen Importe werden hinzugefügt.

`Handler`-Klasse, implementiert

Der folgende Code verwendet die `DynamoDbClient`-Instance, um eine Liste vorhandener Tabellen abzurufen. Wenn Tabellen für ein bestimmtes Konto und eine bestimmte AWS-Region vorhanden sind, verwendet der Code die `Logger`-Instance, um die Namen dieser Tabellen zu protokollieren.

```
package org.example;

 import org.slf4j.Logger;
 import org.slf4j.LoggerFactory;
 import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
 import software.amazon.awssdk.services.dynamodb.model.ListTablesResponse;


 public class Handler {
     private final DynamoDbClient dynamoDbClient;

     public Handler() {
         dynamoDbClient = DependencyFactory.dynamoDbClient();
```

```
    }

    public void sendRequest() {
        Logger logger = LoggerFactory.getLogger(Handler.class);

        logger.info("calling the DynamoDB API to get a list of existing tables");
        ListTablesResponse response = dynamoDbClient.listTables();

        if (!response.hasTableNames()) {
            logger.info("No existing tables found for the configured account &
region");
        } else {
            response.tableNames().forEach(tableName -> logger.info("Table: " +
tableName));
        }
    }
}
```

Schritt 4: Anwendung erstellen und ausführen

Nachdem Sie das Projekt erstellt haben und es die komplette `Handler`-Klasse enthält, erstellen Sie die Anwendung und führen Sie sie aus.

1. Stellen Sie sicher, dass Sie eine aktive AWS IAM Identity Center Sitzung haben. Zur Bestätigung führen Sie den AWS Command Line Interface (AWS CLI)-Befehl `aws sts get-caller-identity` aus und prüfen das Ergebnis: Wenn Sie keine aktive Sitzung haben, finden Sie weitere Anleitungen unter Anmelden mit dem AWS CLI.
2. Öffnen Sie ein Terminal- oder ein Befehlszeilenfenster und navigieren Sie zu Ihrem Projektverzeichnis `getstarted`.
3. Mit dem folgenden Befehl erstellen Sie nun Ihr Projekt:

   ```
   mvn clean package
   ```

4. Um die Anwendung auszuführen, führen Sie den folgenden Befehl aus:

   ```
   mvn exec:java -Dexec.mainClass="org.example.App"
   ```

Nachdem Sie die Datei angezeigt haben, löschen Sie das Objekt und anschließend den Bucket.

Herzlichen Glückwunsch

Wenn Ihr Maven-Projekt ohne Fehler erstellt und ausgeführt wurde, dann herzlichen Glückwunsch! Sie haben erfolgreich Ihre erste Java-Anwendung mit dem SDK for Java 2.x erstellt.

Bereinigen

Um die während dieses Tutorials erstellten Ressourcen zu bereinigen, löschen Sie den Projektordner `getstarted`.

Überprüfung der AWS SDK for Java 2.x Dokumentation

Das AWS SDK for Java 2.x -Entwicklerhandbuch deckt alle Aspekte des SDK in allen AWS-Services ab. Wir empfehlen, dass Sie sich mit den folgenden Themen vertraut machen:

- Migration von Version 1.x auf 2.x – Enthält eine ausführliche Erläuterung der Unterschiede zwischen 1.x und 2.x. Dieses Thema enthält auch Anweisungen zur Verwendung der beiden Hauptversionen side-by-side.
- DynamoDB-Handbuch für Java 2.x SDK – Zeigt Ihnen, wie Sie grundlegende DynamoDB-Operationen ausführen: Erstellen einer Tabelle, Bearbeiten von Elementen und Abrufen von Elementen. In diesen Beispielen hier verwenden wir die Low-Level-Schnittstelle. Java hat mehrere Schnittstellen, wie im folgenden Abschnitt erläutert werden: Unterstützte Schnittstellen.

> **Tip**
>
> Nachdem Sie sich mit diesen Themen befasst haben, setzen Sie ein Lesezeichen für die AWS SDK for Java 2.x API-Referenz. Es deckt alles ab AWS-Services, und wir empfehlen, dass Sie es als Haupt-API-Referenz verwenden.

Unterstützte Schnittstellen

Die AWS SDK for Java 2.x unterstützt die folgenden Schnittstellen, abhängig von der gewünschten Abstraktionsebene.

Themen in diesem Abschnitt

- Low-Level-Schnittstelle
- High-Level-Schnittstelle

- Dokumentschnittstelle
- Vergleichen von Schnittstellen anhand eines Query-Beispiels

Low-Level-Schnittstelle

Die Low-Level-Schnittstelle bietet eine one-to-one Zuordnung zur zugrunde liegenden Service-API. Jede DynamoDB-API ist über diese Schnittstelle verfügbar. Das bedeutet, dass die Low-Level-Schnittstelle vollständige Funktionalität bieten kann, aber häufig ausführlicher und komplexer in der Anwendung ist. Beispielsweise müssen Sie die `.s()`-Funktionen zum Speichern von Zeichenketten, und die `.n()`-Funktionen zum Speichern von Zahlen verwenden. Im folgenden Beispiel wird ein Element mithilfe der Low-Level-Schnittstelle PutItemeingefügt.

```
import org.slf4j.*;
import software.amazon.awssdk.http.crt.AwsCrtHttpClient;
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.*;

import java.util.Map;

public class PutItem {

    // Create a DynamoDB client with the default settings connected to the DynamoDB
    // endpoint in the default region based on the default credentials provider chain.
    private static final DynamoDbClient DYNAMODB_CLIENT = DynamoDbClient.create();
    private static final Logger LOGGER = LoggerFactory.getLogger(PutItem.class);

    private void putItem() {
        PutItemResponse response = DYNAMODB_CLIENT.putItem(PutItemRequest.builder()
                .item(Map.of(
                        "pk", AttributeValue.builder().s("123").build(),
                        "sk", AttributeValue.builder().s("cart#123").build(),
                        "item_data",
 AttributeValue.builder().s("YourItemData").build(),
                        "inventory", AttributeValue.builder().n("500").build()
                        // ... more attributes ...
                ))
                .returnConsumedCapacity(ReturnConsumedCapacity.TOTAL)
                .tableName("YourTableName")
                .build());
        LOGGER.info("PutItem call consumed [" +
 response.consumedCapacity().capacityUnits() + "] Write Capacity Unites (WCU)");
```

```
    }
}
```

High-Level-Schnittstelle

Die High-Level-Schnittstelle im AWS SDK for Java 2.x wird als DynamoDB Enhanced Client bezeichnet. Diese Schnittstelle ermöglicht eine idiomatischere und natürlichere Programmerfahrung.

Der erweiterte Client bietet eine Möglichkeit, clientseitige Datenklassen mit den dafür vorgesehenen DynamoDB-Tabellen zu verknüpfen. Sie definieren die Beziehungen zwischen Tabellen und ihren jeweiligen Modellklassen im Code. Anschließend können Sie sich darauf verlassen, dass das SDK die Umwandlung der Datentypen übernimmt. Weitere Informationen zum erweiterten Client finden Sie unter API für den erweiterten DynamoDB-Client im AWS SDK for Java 2.x -Entwicklerhandbuch.

Im folgenden Beispiel wird die High-Level-Schnittstelle PutItemverwendet. In diesem Beispiel erstellt die `DynamoDbBean` mit dem Namen `YourItem` ein `TableSchema`, das ihre direkte Verwendung als Eingabe für den `putItem()`-Aufruf ermöglicht.

```
import org.slf4j.*;
import software.amazon.awssdk.enhanced.dynamodb.*;
import software.amazon.awssdk.enhanced.dynamodb.mapper.annotations.*;
import software.amazon.awssdk.enhanced.dynamodb.model.*;
import software.amazon.awssdk.services.dynamodb.model.ReturnConsumedCapacity;

public class DynamoDbEnhancedClientPutItem {
    private static final DynamoDbEnhancedClient ENHANCED_DYNAMODB_CLIENT =
 DynamoDbEnhancedClient.builder().build();
    private static final DynamoDbTable<YourItem> DYNAMODB_TABLE =
 ENHANCED_DYNAMODB_CLIENT.table("YourTableName", TableSchema.fromBean(YourItem.class));
    private static final Logger LOGGER = LoggerFactory.getLogger(PutItem.class);

    private void putItem() {
        PutItemEnhancedResponse<YourItem> response =
 DYNAMODB_TABLE.putItemWithResponse(PutItemEnhancedRequest.builder(YourItem.class)
                .item(new YourItem("123", "cart#123", "YourItemData", 500))
                .returnConsumedCapacity(ReturnConsumedCapacity.TOTAL)
                .build());
        LOGGER.info("PutItem call consumed [" +
 response.consumedCapacity().capacityUnits() + "] Write Capacity Unites (WCU)");
    }

    @DynamoDbBean
```

```
    public static class YourItem {

        public YourItem() {}

        public YourItem(String pk, String sk, String itemData, int inventory) {
            this.pk = pk;
            this.sk = sk;
            this.itemData = itemData;
            this.inventory = inventory;
        }

        private String pk;
        private String sk;
        private String itemData;

        private int inventory;

        @DynamoDbPartitionKey
        public void setPk(String pk) {
            this.pk = pk;
        }

        public String getPk() {
            return pk;
        }

        @DynamoDbSortKey
        public void setSk(String sk) {
            this.sk = sk;
        }

        public String getSk() {
            return sk;
        }

        public void setItemData(String itemData) {
            this.itemData = itemData;
        }

        public String getItemData() {
            return itemData;
        }

        public void setInventory(int inventory) {
```

```
            this.inventory = inventory;
        }

        public int getInventory() {
            return inventory;
        }
    }
}
```

Die AWS SDK für Java Version 1.x hat ihre eigene High-Level-Schnittstelle, die oft mit ihrer Hauptklasse `DynamoDBMapper` bezeichnet wird. Das AWS SDK for Java 2.x wird in einem separaten Paket (und einem Maven-Artefakt) mit dem Namen veröffentlicht. `software.amazon.awssdk.enhanced.dynamodb` Das Java SDK 2.x wird oft über seine Hauptklasse `DynamoDbEnhancedClient` referenziert.

High-Level-Schnittstelle mit unveränderlichen Datenklassen

Die Mapping-Funktion der erweiterten DynamoDB-Client-API funktioniert auch mit unveränderlichen Datenklassen. Eine unveränderliche Klasse hat nur Getter und erfordert eine Builder-Klasse, die das SDK verwendet, um Instances der Klasse zu erstellen. Unveränderlichkeit ist ein in Java häufig verwendeter Programmierstil, mit dem Entwickler Klassen erstellen können, die keine Nebeneffekte verursachen. Solche Klassen verhalten sich in komplexen, nebenläufigen Anwendungen vorhersehbarer. Anstelle der Annotation `@DynamoDbBean`, wie im Beispiel zur High-level interface example gezeigt, verwenden unveränderliche Klassen die `@DynamoDbImmutable`-Annotation, die die zugehörige Builder-Klasse als Eingabe erhält.

Im folgenden Beispiel wird die Builder-Klasse `DynamoDbEnhancedClientImmutablePutItem` als Eingabe verwendet, um ein Tabellenschema zu erstellen. Das Beispiel stellt dann das Schema als Eingabe für den PutItemAPI-Aufruf bereit.

```
import org.slf4j.*;
import software.amazon.awssdk.enhanced.dynamodb.*;
import software.amazon.awssdk.enhanced.dynamodb.model.*;
import software.amazon.awssdk.services.dynamodb.model.ReturnConsumedCapacity;

public class DynamoDbEnhancedClientImmutablePutItem {
    private static final DynamoDbEnhancedClient ENHANCED_DYNAMODB_CLIENT =
 DynamoDbEnhancedClient.builder().build();
    private static final DynamoDbTable<YourImmutableItem>
 DYNAMODB_TABLE = ENHANCED_DYNAMODB_CLIENT.table("YourTableName",
 TableSchema.fromImmutableClass(YourImmutableItem.class));
```

```
    private static final Logger LOGGER =
 LoggerFactory.getLogger(DynamoDbEnhancedClientImmutablePutItem.class);

    private void putItem() {
        PutItemEnhancedResponse<YourImmutableItem> response =
 DYNAMODB_TABLE.putItemWithResponse(PutItemEnhancedRequest.builder(YourImmutableItem.class)
                .item(YourImmutableItem.builder()
                                    .pk("123")
                                    .sk("cart#123")
                                    .itemData("YourItemData")
                                    .inventory(500)
                                    .build())
                .returnConsumedCapacity(ReturnConsumedCapacity.TOTAL)
                .build());
        LOGGER.info("PutItem call consumed [" +
 response.consumedCapacity().capacityUnits() + "] Write Capacity Unites (WCU)");
    }
}
```

Im folgenden Beispiel wird die unveränderliche Datenklasse gezeigt.

```
@DynamoDbImmutable(builder = YourImmutableItem.YourImmutableItemBuilder.class)
class YourImmutableItem {
    private final String pk;
    private final String sk;
    private final String itemData;
    private final int inventory;
    public YourImmutableItem(YourImmutableItemBuilder builder) {
        this.pk = builder.pk;
        this.sk = builder.sk;
        this.itemData = builder.itemData;
        this.inventory = builder.inventory;
    }

    public static YourImmutableItemBuilder builder() { return new
 YourImmutableItemBuilder(); }

    @DynamoDbPartitionKey
    public String getPk() {
        return pk;
    }

    @DynamoDbSortKey
```

```
    public String getSk() {
        return sk;
    }

    public String getItemData() {
        return itemData;
    }

    public int getInventory() {
        return inventory;
    }

    static final class YourImmutableItemBuilder {
        private String pk;
        private String sk;
        private String itemData;
        private int inventory;

        private YourImmutableItemBuilder() {}

        public YourImmutableItemBuilder pk(String pk) { this.pk = pk; return this; }
        public YourImmutableItemBuilder sk(String sk) { this.sk = sk; return this; }
        public YourImmutableItemBuilder itemData(String itemData) { this.itemData =
 itemData; return this; }
        public YourImmutableItemBuilder inventory(int inventory) { this.inventory =
 inventory; return this; }

        public YourImmutableItem build() { return new YourImmutableItem(this); }
    }
}
```

High-Level-Schnittstelle mit unveränderlichen Datenklassen und externen Bibliotheken zur Generierung von Boilerplate-Code

Unveränderliche Datenklassen (wie im vorherigen Beispiel gezeigt) erfordern etwas Boilerplate-Code. Beispielsweise die Getter- und Setter-Logik in den Datenklassen sowie zugehörige `Builder`-Klassen. Externe Bibliotheken wie Project Lombok können dabei helfen, diese Art von Boilerplate-Code zu generieren. Wenn Sie den Großteil des Standardcodes reduzieren, können Sie die Menge an Code einschränken, die für die Arbeit mit unveränderlichen Datenklassen und dem SDK benötigt wird. AWS Dies führt zu höherer Produktivität und verbessert die Lesbarkeit Ihres Codes. Weitere Informationen finden Sie unter Verwenden von externen Bibliotheken wie Lombok im -Entwicklerhandbuch.

Im folgenden Beispiel wird gezeigt, wie Project Lombok den Code vereinfacht, der für die Nutzung der erweiterten DynamoDB-Client-API erforderlich ist.

```
import org.slf4j.*;
import software.amazon.awssdk.enhanced.dynamodb.*;
import software.amazon.awssdk.enhanced.dynamodb.model.*;
import software.amazon.awssdk.services.dynamodb.model.ReturnConsumedCapacity;

public class DynamoDbEnhancedClientImmutableLombokPutItem {

    private static final DynamoDbEnhancedClient ENHANCED_DYNAMODB_CLIENT =
 DynamoDbEnhancedClient.builder().build();
    private static final DynamoDbTable<YourImmutableLombokItem>
 DYNAMODB_TABLE = ENHANCED_DYNAMODB_CLIENT.table("YourTableName",
 TableSchema.fromImmutableClass(YourImmutableLombokItem.class));
    private static final Logger LOGGER =
 LoggerFactory.getLogger(DynamoDbEnhancedClientImmutableLombokPutItem.class);

    private void putItem() {
        PutItemEnhancedResponse<YourImmutableLombokItem> response =
 DYNAMODB_TABLE.putItemWithResponse(PutItemEnhancedRequest.builder(YourImmutableLombokItem.clas
                .item(YourImmutableLombokItem.builder()
                        .pk("123")
                        .sk("cart#123")
                        .itemData("YourItemData")
                        .inventory(500)
                        .build())
                .returnConsumedCapacity(ReturnConsumedCapacity.TOTAL)
                .build());
        LOGGER.info("PutItem call consumed [" +
 response.consumedCapacity().capacityUnits() + "] Write Capacity Unites (WCU)");
    }
}
```

Das folgende Beispiel zeigt das unveränderliche Datenobjekt der unveränderlichen Datenklasse.

```
import lombok.*;
import software.amazon.awssdk.enhanced.dynamodb.mapper.annotations.*;

@Builder
@DynamoDbImmutable(builder =
 YourImmutableLombokItem.YourImmutableLombokItemBuilder.class)
@Value
```

```
public class YourImmutableLombokItem {

    @Getter(onMethod_=@DynamoDbPartitionKey)
    String pk;
    @Getter(onMethod_=@DynamoDbSortKey)
    String sk;
    String itemData;
    int inventory;
}
```

Die `YourImmutableLombokItem` Klasse verwendet die folgenden Anmerkungen, die Project Lombok und das SDK bereitstellen: AWS

- @Builder — Produziert einen komplexen Builder APIs für Datenklassen, den Project Lombok bereitstellt.
- @ DynamoDbImmutable — Identifiziert die `DynamoDbImmutable` Klasse als eine vom SDK DynamoDB DynamoDB-Annotation für zuordenbare Entitäten. AWS
- @Value – Die unveränderliche Variante von `@Data`. Standardmäßig sind alle Felder privat und endgültig, und es werden keine Setter generiert. Project Lombok stellt diese Annotation zur Verfügung.

Dokumentschnittstelle

Die AWS SDK for Java 2.x Document-Schnittstelle macht die Angabe von Datentypdeskriptoren überflüssig. Die Datentypen ergeben sich aus der Semantik der Daten selbst. Diese Dokumentschnittstelle ähnelt der Dokumentschnittstelle von AWS SDK für Java 1.x, verfügt jedoch über eine neu gestaltete Oberfläche.

Im Folgenden Document interface example wird der über die Dokumentschnittstelle ausgedrückte `PutItem`-Aufruf dargestellt. Das Beispiel verwendet auch. EnhancedDocument Um Befehle gegen eine DynamoDB-Tabelle mit der erweiterten Dokument-API auszuführen, müssen Sie die Tabelle zunächst mit Ihrem Dokumenttabellenschema verknüpfen, um ein `DynamoDBTable`-Ressourcenobjekt zu erstellen. Der Tabellenschema-Builder für Dokumente erfordert den Primärschlüssel des Indexes sowie Anbieter für Attributkonverter.

Sie können `AttributeConverterProvider.defaultProvider()` verwenden, um Dokumentattribute mit Standardtypen zu konvertieren. Das allgemeine Standardverhalten lässt sich durch eine benutzerdefinierte `AttributeConverterProvider`-Implementierung ändern. Ebenso

können Sie den Konverter für ein einzelnes Attribut anpassen. Das AWS SDKs Referenzhandbuch zu Tools enthält weitere Details und Beispiele zur Verwendung benutzerdefinierter Konverter. Ihr Hauptzweck besteht darin, Attribute Ihrer Domänenklassen zu unterstützen, für die kein Standardkonverter verfügbar ist. Mit einem benutzerdefinierten Konverter können Sie dem SDK die nötigen Informationen bereitstellen, um Daten in DynamoDB zu schreiben oder daraus zu lesen.

```
import org.slf4j.*;
import software.amazon.awssdk.enhanced.dynamodb.*;
import software.amazon.awssdk.enhanced.dynamodb.document.EnhancedDocument;
import software.amazon.awssdk.enhanced.dynamodb.model.*;
import software.amazon.awssdk.services.dynamodb.model.ReturnConsumedCapacity;

public class DynamoDbEnhancedDocumentClientPutItem {
    private static final DynamoDbEnhancedClient ENHANCED_DYNAMODB_CLIENT =
 DynamoDbEnhancedClient.builder().build();
    private static final DynamoDbTable<EnhancedDocument> DYNAMODB_TABLE =
            ENHANCED_DYNAMODB_CLIENT.table("YourTableName",
 TableSchema.documentSchemaBuilder()

 .addIndexPartitionKey(TableMetadata.primaryIndexName(),"pk", AttributeValueType.S)
                            .addIndexSortKey(TableMetadata.primaryIndexName(), "sk",
 AttributeValueType.S)

 .attributeConverterProviders(AttributeConverterProvider.defaultProvider())
                            .build());

    private static final Logger LOGGER =
 LoggerFactory.getLogger(DynamoDbEnhancedDocumentClientPutItem.class);

    private void putItem() {
        PutItemEnhancedResponse<EnhancedDocument> response =
 DYNAMODB_TABLE.putItemWithResponse(
                        PutItemEnhancedRequest.builder(EnhancedDocument.class)
                                .item(
                                    EnhancedDocument.builder()

 .attributeConverterProviders(AttributeConverterProvider.defaultProvider())
                                            .putString("pk", "123")
                                            .putString("sk", "cart#123")
                                            .putString("item_data", "YourItemData")
                                            .putNumber("inventory", 500)
                                            .build())
                                .returnConsumedCapacity(ReturnConsumedCapacity.TOTAL)
```

```
                                    .build());
            LOGGER.info("PutItem call consumed [" +
     response.consumedCapacity().capacityUnits() + "] Write Capacity Unites (WCU)");
        }

}
```

Um JSON-Dokumente in native Amazon-DynamoDB-Datentypen zu konvertieren und umgekehrt, können Sie die folgenden Hilfsmethoden verwenden:

- `EnhancedDocument.fromJson(String json)`— Erzeugt eine neue EnhancedDocument Instanz aus einer JSON-Zeichenfolge.
- `EnhancedDocument.toJson()` – Erstellt eine JSON-Zeichenfolgendarstellung des Dokuments, die Sie in Ihrer Anwendung wie jedes andere JSON-Objekt verwenden können.

Vergleichen von Schnittstellen anhand eines **`Query`**-Beispiels

Dieser Abschnitt zeigt den gleichen `Query`-Aufruf, der über die verschiedenen Schnittstellen ausgedrückt wird. Beachten Sie Folgendes, um die Ergebnisse dieser Abfragen zu optimieren:

- DynamoDB zielt auf einen bestimmten Partitionsschlüsselwert ab; daher müssen Sie den Partitionsschlüssel vollständig angeben.
- Damit die Abfrage ausschließlich auf Warenkorbelemente abzielt, verwendet der Sortierschlüssel einen Schlüsselbedingungsausdruck mit `begins_with`.
- Wir verwenden `limit()`, um die Abfrage auf maximal 100 zurückgegebene Elemente zu beschränken.
- Wir haben das `scanIndexForward` auf false gesetzt. Die Ergebnisse werden in der Reihenfolge der UTF-8-Bytes zurückgegeben, was normalerweise bedeutet, dass das Warenkorbelement mit der niedrigsten Nummer zuerst zurückgegeben wird. Indem wir nun `scanIndexForward` auf false setzen, kehren wir die Reihenfolge um, sodass das Warenkorbelement mit der höchsten Nummer zuerst zurückgegeben wird.
- Wir wenden einen Filter an, um alle Ergebnisse zu entfernen, die nicht den Kriterien entsprechen. Die gefilterten Daten verbrauchen Lesekapazität, unabhängig davon, ob das Element dem Filter entspricht oder nicht.

Example **`Query`** unter Verwendung der Low-Level-Schnittstelle

Das folgende Beispiel fragt eine Tabelle mit dem Namen `YourTableName` unter Verwendung eines `keyConditionExpression` ab. Dadurch wird die Abfrage auf einen bestimmten Partition-Key-Wert und einen Sortierschlüsselwert beschränkt, die mit einem bestimmten Präfix beginnen. Diese Schlüsselbedingungen begrenzen die Menge der aus DynamoDB gelesenen Daten. Abschließend wird ein Filter auf die aus DynamoDB abgerufenen Daten angewendet, indem eine `filterExpression` verwendet wird.

```
import org.slf4j.*;
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.*;

import java.util.Map;

public class Query {

    // Create a DynamoDB client with the default settings connected to the DynamoDB
    // endpoint in the default region based on the default credentials provider chain.
    private static final DynamoDbClient DYNAMODB_CLIENT =
 DynamoDbClient.builder().build();
    private static final Logger LOGGER = LoggerFactory.getLogger(Query.class);

    private static void query() {
        QueryResponse response = DYNAMODB_CLIENT.query(QueryRequest.builder()
                .expressionAttributeNames(Map.of("#name", "name"))
                .expressionAttributeValues(Map.of(
                    ":pk_val", AttributeValue.fromS("id#1"),
                    ":sk_val", AttributeValue.fromS("cart#"),
                    ":name_val", AttributeValue.fromS("SomeName")))
                .filterExpression("#name = :name_val")
                .keyConditionExpression("pk = :pk_val AND begins_with(sk, :sk_val)")
                .limit(100)
                .scanIndexForward(false)
                .tableName("YourTableName")
                .build());

        LOGGER.info("nr of items: " + response.count());
        LOGGER.info("First item pk: " + response.items().get(0).get("pk"));
        LOGGER.info("First item sk: " + response.items().get(0).get("sk"));
    }
}
```

Example **`Query`** unter Verwendung der Dokumentschnittstelle

Das folgende Beispiel fragt eine Tabelle mit dem Namen `YourTableName` unter Verwendung der Dokumentschnittstelle ab.

```
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import software.amazon.awssdk.enhanced.dynamodb.*;
import software.amazon.awssdk.enhanced.dynamodb.document.EnhancedDocument;
import software.amazon.awssdk.enhanced.dynamodb.model.*;

import java.util.Map;

public class DynamoDbEnhancedDocumentClientQuery {

    // Create a DynamoDB client with the default settings connected to the DynamoDB
    // endpoint in the default region based on the default credentials provider chain.
    private static final DynamoDbEnhancedClient ENHANCED_DYNAMODB_CLIENT =
 DynamoDbEnhancedClient.builder().build();
    private static final DynamoDbTable<EnhancedDocument> DYNAMODB_TABLE =
            ENHANCED_DYNAMODB_CLIENT.table("YourTableName",
 TableSchema.documentSchemaBuilder()
                    .addIndexPartitionKey(TableMetadata.primaryIndexName(),"pk",
 AttributeValueType.S)
                    .addIndexSortKey(TableMetadata.primaryIndexName(), "sk",
 AttributeValueType.S)

 .attributeConverterProviders(AttributeConverterProvider.defaultProvider())
                    .build());
    private static final Logger LOGGER =
 LoggerFactory.getLogger(DynamoDbEnhancedDocumentClientQuery.class);

    private void query() {
        PageIterable<EnhancedDocument> response =
 DYNAMODB_TABLE.query(QueryEnhancedRequest.builder()
                .filterExpression(Expression.builder()
                        .expression("#name = :name_val")
                        .expressionNames(Map.of("#name", "name"))
                        .expressionValues(Map.of(":name_val",
 AttributeValue.fromS("SomeName")))
                        .build())
                .limit(100)
                .queryConditional(QueryConditional.sortBeginsWith(Key.builder()
                        .partitionValue("id#1")
```

```
                        .sortValue("cart#")
                        .build()))
                .scanIndexForward(false)
                .build());

        LOGGER.info("nr of items: " + response.items().stream().count());
        LOGGER.info("First item pk: " +
 response.items().iterator().next().getString("pk"));
        LOGGER.info("First item sk: " +
 response.items().iterator().next().getString("sk"));

    }
}
```

Example **`Query`** unter Verwendung der High-Level-Schnittstelle

Im folgenden Beispiel wird eine Abfrage auf einer Tabelle mit dem Namen `YourTableName` unter Verwendung der erweiterten DynamoDB-Client-API durchgeführt.

```
import org.slf4j.*;
import software.amazon.awssdk.enhanced.dynamodb.*;
import software.amazon.awssdk.enhanced.dynamodb.mapper.annotations.*;
import software.amazon.awssdk.enhanced.dynamodb.model.*;
import software.amazon.awssdk.services.dynamodb.model.AttributeValue;

import java.util.Map;

public class DynamoDbEnhancedClientQuery {

    private static final DynamoDbEnhancedClient ENHANCED_DYNAMODB_CLIENT =
 DynamoDbEnhancedClient.builder().build();
    private static final DynamoDbTable<YourItem> DYNAMODB_TABLE =
 ENHANCED_DYNAMODB_CLIENT.table("YourTableName",
 TableSchema.fromBean(DynamoDbEnhancedClientQuery.YourItem.class));
    private static final Logger LOGGER =
 LoggerFactory.getLogger(DynamoDbEnhancedClientQuery.class);

    private void query() {
        PageIterable<YourItem> response =
 DYNAMODB_TABLE.query(QueryEnhancedRequest.builder()
                .filterExpression(Expression.builder()
                        .expression("#name = :name_val")
                        .expressionNames(Map.of("#name", "name"))
```

```
                        .expressionValues(Map.of(":name_val",
AttributeValue.fromS("SomeName")))
                        .build())
                .limit(100)
                .queryConditional(QueryConditional.sortBeginsWith(Key.builder()
                        .partitionValue("id#1")
                        .sortValue("cart#")
                        .build()))
                .scanIndexForward(false)
                .build());

        LOGGER.info("nr of items: " + response.items().stream().count());
        LOGGER.info("First item pk: " + response.items().iterator().next().getPk());
        LOGGER.info("First item sk: " + response.items().iterator().next().getSk());
    }

    @DynamoDbBean
    public static class YourItem {

        public YourItem() {}

        public YourItem(String pk, String sk, String name) {
            this.pk = pk;
            this.sk = sk;
            this.name = name;
        }

        private String pk;
        private String sk;
        private String name;

        @DynamoDbPartitionKey
        public void setPk(String pk) {
            this.pk = pk;
        }

        public String getPk() {
            return pk;
        }

        @DynamoDbSortKey
        public void setSk(String sk) {
            this.sk = sk;
        }
```

```
        public String getSk() {
            return sk;
        }

        public void setName(String name) {
            this.name = name;
        }

        public String getName() {
            return name;
        }
    }
}
```

High-Level-Schnittstelle mit unveränderlichen Datenklassen

Wenn Sie eine `Query` mit unveränderlichen Datenklassen durchführen, entspricht der Code dem Beispiel für die High-Level-Schnittstelle; die Konstruktion der Entitätsklasse `YourItem` oder `YourImmutableItem` ist jedoch anders. Weitere Informationen finden Sie im PutItemBeispiel.

High-Level-Schnittstelle mit unveränderlichen Datenklassen und externen Bibliotheken zur Generierung von Boilerplate-Code

Wenn Sie eine `Query` mit unveränderlichen Datenklassen durchführen, entspricht der Code dem Beispiel für die High-Level-Schnittstelle; die Konstruktion der Entitätsklasse `YourItem` oder `YourImmutableLombokItem` ist jedoch anders. Weitere Informationen finden Sie im PutItemBeispiel.

Weitere Codebeispiele

Weitere Beispiele zur Verwendung von DynamoDB mit dem AWS SDK für Java 2.x finden Sie in den folgenden Codebeispiel-Repositories:

- Offizielle AWS -Single-Action-Codebeispiele
- Von der Community verwaltete Beispiele für Single-Action-Codes
- Offizielle AWS szenarioorientierte Codebeispiele

Synchrone und asynchrone Programmierung

Der AWS SDK for Java 2.x bietet sowohl synchrone als auch asynchrone Clients für AWS-Services, z. B. DynamoDB.

Die Klassen `DynamoDbClient` und `DynamoDbEnhancedClient` bieten synchrone Methoden, die die Ausführung Ihres Threads blockieren, bis der Client eine Antwort vom Service erhält. Dieser Client ist die einfachste Möglichkeit, mit DynamoDB zu arbeiten, wenn Sie keine asynchronen Operationen benötigen.

Die Klassen `DynamoDbAsyncClient` und `DynamoDbEnhancedAsyncClient` bieten asynchrone Methoden, die sofort reagieren und die Kontrolle an den aufrufenden Thread zurückgeben, ohne auf eine Antwort zu warten. Der nicht-blockierende Client hat den Vorteil, dass er bei hoher Parallelität über wenige Threads effizient mit I/O-Anfragen umgehen kann und dabei nur minimale Rechenressourcen benötigt. Dies verbessert den Durchsatz und die Reaktionsfähigkeit.

Der AWS SDK for Java 2.x verwendet die native Unterstützung für nicht blockierende I/O. Die Version AWS SDK für Java 1.x musste nicht blockierende I/O simulieren.

Die synchronen Methoden reagieren, bevor eine Antwort verfügbar ist. Daher benötigen Sie eine Möglichkeit, die Antwort zu erhalten, sobald sie bereitsteht. Die asynchronen Methoden AWS SDK für Java geben ein `CompletableFuture`Objekt zurück, das die Ergebnisse der asynchronen Operation in der future enthält. Wenn Sie `get()` oder `join()` auf diesen `CompletableFuture`-Objekten aufrufen, blockiert Ihr Code, bis das Ergebnis verfügbar ist. Wenn Sie diese Methoden gleichzeitig mit der Anfrage aufrufen, verhält sich der Code ähnlich wie bei einem normalen synchronen Aufruf.

Weitere Informationen zur asynchronen Programmierung finden Sie unter Verwenden der asynchronen Programmierung im AWS SDK for Java 2.x -Entwicklerhandbuch.

HTTP-Clients

Zur Unterstützung jedes Clients gibt es einen HTTP-Client, der die Kommunikation mit den AWS-Servicesübernimmt. Sie können alternative HTTP-Clients einbinden und dabei denjenigen auswählen, der am besten zu den Anforderungen Ihrer Anwendung passt. Einige sind besonders ressourcenschonend, andere bieten umfangreiche Konfigurationsmöglichkeiten.

Einige HTTP-Clients unterstützen ausschließlich die synchrone Nutzung, während andere nur für die asynchrone Verwendung geeignet sind. Ein Diagramm, das Ihnen bei der Auswahl des optimalen HTTP-Clients für Ihre Workload hilft, finden Sie unter Empfehlungen zu HTTP-Clients im AWS SDK for Java 2.x -Entwicklerhandbuch.

In der folgenden Liste sehen Sie einige der möglichen HTTP-Clients:

Topics

- Apache-basierter HTTP-Client
- URLConnection-basierter HTTP-Client
- Netty-basierter HTTP-Client
- AWS CRT-basierter HTTP-Client

Apache-basierter HTTP-Client

Die `ApacheHttpClient`-Klasse unterstützt synchrone Service-Clients. Dies ist der Standard-HTTP-Client für die synchrone Verwendung. Weitere Informationen zum Konfigurieren der `ApacheHttpClient`-Klasse finden Sie unter Konfigurieren des Apache-basierten HTTP-Clients im AWS SDK for Java 2.x -Entwicklerhandbuch.

`URLConnection`-basierter HTTP-Client

Die `UrlConnectionHttpClient`-Klasse ist eine weitere Option für synchrone Clients. Sie lädt schneller als der Apache-basierte HTTP-Client, bietet jedoch weniger Funktionen. Informationen zur Konfiguration der `UrlConnectionHttpClient` Klasse finden Sie unter Configure the URLConnection -based HTTP-Client im AWS SDK for Java 2.x Developer Guide.

Netty-basierter HTTP-Client

Die `NettyNioAsyncHttpClient`-Klasse unterstützt asynchrone Clients. Dies ist die Standardauswahl für die asynchrone Verwendung. Weitere Informationen zum Konfigurieren der `NettyNioAsyncHttpClient`-Klasse finden Sie unter Konfigurieren des Netty-basierten HTTP-Clients im AWS SDK for Java 2.x -Entwicklerhandbuch.

AWS CRT-basierter HTTP-Client

Die neueren `AwsCrtAsyncHttpClient` Klassen `AwsCrtHttpClient` und Klassen aus den AWS Common Runtime (CRT) -Bibliotheken sind weitere Optionen, die synchrone und asynchrone Clients unterstützen. Im Vergleich zu anderen HTTP-Clients bietet CRT: AWS

- Schnellere SDK-Startup-Zeiten
- Geringeren Speicherbedarf
- Reduzierte Latenzzeit

- Verbindungsgesundheitsverwaltung
- DNS-Load Balancing

Informationen zur Konfiguration der *`AwsCrtAsyncHttpClient`* Klassen *`AwsCrtHttpClient`* und finden Sie unter Konfiguration der AWS CRT-basierten HTTP-Clients im AWS SDK for Java 2.x Entwicklerhandbuch.

Der AWS CRT-basierte HTTP-Client ist nicht der Standard, da dies die Abwärtskompatibilität vorhandener Anwendungen beeinträchtigen würde. Für DynamoDB empfehlen wir jedoch, den AWS CRT-basierten HTTP-Client sowohl für synchrone als auch asynchrone Zwecke zu verwenden.

Eine Einführung in den AWS CRT-basierten HTTP-Client finden Sie unter Ankündigung der Verfügbarkeit des CRT-HTTP-Clients im Developer AWS Tools-Blog. AWS SDK for Java 2.xAWS

Konfigurieren eines HTTP-Clients

Beim Konfigurieren eines Clients können Sie verschiedene Konfigurationsoptionen angeben, darunter:

- Timeouts für verschiedene Aspekte von API-Aufrufen.
- Aktivieren des TCP-Keepalives.
- Steuerung der Wiederholungsrichtlinie bei Fehlern.
- Festlegen von Ausführungsattributen, die von Execution Interceptor-Instances verändert werden können. Execution Interceptors können Code schreiben, der die Ausführung Ihrer API-Anfragen und -Antworten abfängt. Auf diese Weise können Sie Aufgaben wie das Veröffentlichen von Metriken und das Ändern von Anfragen während der Übertragung ausführen.
- Hinzufügen oder Bearbeiten von HTTP-Headern.
- Aktivieren der Nachverfolgung von Leistungskennzahlen auf Client-Seite. Mithilfe dieser Funktion können Sie Metriken über die Service-Clients in Ihrer Anwendung sammeln und die Ergebnisse in Amazon analysieren CloudWatch.
- Festlegen eines alternativen Executor-Service für die Planung von Aufgaben wie asynchrone Wiederholungsversuche und Timeout.

Sie steuern die Konfiguration, indem Sie der Service-Client-Klasse `Builder` ein `ClientOverrideConfiguration`-Objekt bereitstellen. Sie werden dies in einigen Codebeispielen in den folgenden Abschnitten sehen.

`ClientOverrideConfiguration` bietet Standardkonfigurationsoptionen. Die verschiedenen einsteckbaren HTTP-Clients verfügen zusätzlich über implementierungsspezifische Optionen.

Themen in diesem Abschnitt

- Timeout-Konfiguration
- RetryMode
- DefaultsMode
- Keepalive-Konfigurierung

Timeout-Konfiguration

Sie können die Client-Konfiguration anpassen, um verschiedene Timeout-Einstellungen für Serviceaufrufe zu steuern. DynamoDB im Vergleich zu anderen AWS-Services geringere Latenzen. Darum empfiehlt es sich, diese Eigenschaften auf niedrigere Timeout-Werte zu setzen, um bei Netzwerkproblemen schnell zu reagieren.

Das latenzbezogene Verhalten lässt sich entweder über `ClientOverrideConfiguration` am DynamoDB-Client oder durch detaillierte Einstellungen am zugrunde liegenden HTTP-Client vornehmen.

Folgende wichtige Eigenschaften können über `ClientOverrideConfiguration` konfiguriert werden:

- `apiCallAttemptTimeout` – Die maximale Zeitspanne, die für einen einzelnen Versuch eines HTTP-Requests gewartet wird, bevor abgebrochen und ein Timeout ausgelöst wird.
- `apiCallTimeout` – Die Gesamtzeit, die dem Client zur Verfügung steht, um einen vollständigen API-Aufruf auszuführen. Dies umfasst die Ausführung des Request-Handlers, der alle HTTP-Anfragen einschließlich Wiederholungsversuchen beinhaltet.

Das AWS SDK for Java 2.x bietet Standardwerte für einige Timeout-Optionen, wie Verbindungs-Timeout und Socket-Timeouts. Das SDK bietet keine Standardwerte für Timeouts bei API-Aufrufen oder Timeouts für einzelne API-Aufrufversuche. Wenn diese Timeouts nicht explizit in der `ClientOverrideConfiguration` gesetzt werden, verwendet das SDK stattdessen den Socket-Timeout-Wert als Gesamtzeitlimit für den API-Aufruf. Der Standardwert für den Socket-Timeout beträgt 30 Sekunden.

RetryMode

Eine weitere Einstellung im Zusammenhang mit der Timeout-Konfiguration, die Sie berücksichtigen sollten, ist das Konfigurationsobjekt `RetryMode`. Dieses Konfigurationsobjekt enthält eine Sammlung von Wiederholungsverhalten.

Das SDK for Java 2.x unterstützt die folgenden Wiederholungsmodi:

- `legacy` – Der standardmäßige Wiederholungsmodus, wenn Sie ihn nicht explizit ändern. Dieser Wiederholungsmodus ist spezifisch für das Java SDK. Es zeichnet sich durch bis zu drei Wiederholungen oder mehr für Services wie DynamoDB aus, das bis zu acht Wiederholungen hat.
- `standard`— Wird als „Standard“ bezeichnet, weil es konsistenter mit anderen ist. AWS SDKs Beim ersten Wiederholungsversuch wartet dieser Modus eine zufällige Zeit zwischen 0 ms und 1 000 ms. Wird ein weiterer Versuch erforderlich, wird erneut eine zufällige Zeit zwischen 0 ms und 1 000 ms gewählt und mit zwei multipliziert. Sollten weitere Wiederholungsversuche erforderlich sein, erfolgt dieselbe zufällige Auswahl, multipliziert mit vier usw. Jede Wartezeit ist auf 20 Sekunden begrenzt. Dieser Modus erkennt mehr Fehlerbedingungen als der `legacy`-Modus. Für DynamoDB sind bis zu drei Gesamtversuche vorgesehen, sofern numRetries nicht überschrieben wird.
- `adaptive` – Baut auf dem `standard`-Modus auf und begrenzt dynamisch die Rate der AWS -Anfragen, um die Erfolgsrate zu maximieren. Dies kann auf Kosten der Latenz der Anfragen gehen. Der adaptive Modus wird nicht empfohlen, wenn eine vorhersehbare Latenz wichtig ist.

Eine erweiterte Definition dieser Wiederholungsmodi finden Sie im Thema Wiederholungsverhalten im Referenzhandbuch AWS SDKs und im Tools-Referenzhandbuch.

Wiederholungsrichtlinien

Alle `RetryMode`-Konfigurationen haben eine `RetryPolicy`, die auf einer oder mehreren `RetryCondition`-Konfigurationen basiert. Die `TokenBucketRetryCondition` ist besonders wichtig für das Wiederholungsverhalten der Implementierung des DynamoDB-SDK-Clients. Diese Bedingung begrenzt die Anzahl der Wiederholungen, die das SDK mithilfe eines Token-Bucket-Algorithmus durchführt. Je nach gewähltem Retry-Modus können Drosselungsfehler Token aus dem `TokenBucket` entfernen oder auch nicht.

Wenn ein Client auf einen wiederholbaren Fehler stößt, z B. eine Drosselung oder einen temporären Serverfehler, wiederholt das SDK die Anfrage automatisch. Sie können steuern, wie oft und wie schnell diese Wiederholungen erfolgen.

Bei der Konfiguration eines Clients können Sie eine `RetryPolicy` bereitstellen, die unter anderem die folgenden Parameter unterstützt:

- `numRetries` – Die maximale Anzahl von Wiederholungsversuchen, bevor eine Anfrage als fehlgeschlagen gilt. Der Standardwert ist 8, unabhängig vom verwendeten Wiederholungsmodus.

  > **Warning**
  >
  > Sie sollten diesen Standardwert nur nach reiflicher Überlegung ändern.

- `backoffStrategy` – Die anzuwendende `BackoffStrategy` für Wiederholungsversuche ist standardmäßig die `FullJitterBackoffStrategy`. Diese Strategie führt eine exponentielle Verzögerung zwischen den einzelnen Wiederholungen durch, basierend auf der aktuellen Anzahl der Versuche, einer Basisverzögerung und einer maximalen Backoff-Zeit. Dann wird Jitter hinzugefügt, um für ein gewisses Maß an Zufälligkeit zu sorgen. Die bei der exponentiellen Verzögerung verwendete Basisverzögerung beträgt unabhängig vom Wiederholungsmodus 25 ms.
- `retryCondition` – Die `RetryCondition` bestimmt, ob eine Anfrage überhaupt erneut versucht werden soll. Standardmäßig wird ein bestimmter Satz von HTTP-Statuscodes und Ausnahmen wiederholt, von denen angenommen wird, dass sie erneut versucht werden können. In den meisten Situationen sollte die Standardkonfiguration ausreichend sein.

Der folgende Code bietet eine alternative Wiederholungsrichtlinie. Diese gibt insgesamt fünf Wiederholungen an (insgesamt sechs Anfragen). Die erste Wiederholung sollte nach einer Verzögerung von etwa 100 ms erfolgen, wobei jeder weitere Versuch diese Zeit exponentiell verdoppelt, bis zu einer maximalen Verzögerung von einer Sekunde.

```
DynamoDbClient client = DynamoDbClient.builder()
    .overrideConfiguration(ClientOverrideConfiguration.builder()
        .retryPolicy(RetryPolicy.builder()
            .backoffStrategy(FullJitterBackoffStrategy.builder()
                .baseDelay(Duration.ofMillis(100))
                .maxBackoffTime(Duration.ofSeconds(1))
                .build())
            .numRetries(5)
            .build())
        .build())
    .build();
```

DefaultsMode

Die Timeout-Eigenschaften, die nicht durch `ClientOverrideConfiguration` oder `RetryMode` gesteuert werden, werden in der Regel implizit über die Angabe eines `DefaultsMode` konfiguriert.

Mit AWS SDK for Java 2.x (Version 2.17.102 oder höher) wurde die Unterstützung für eingeführt. `DefaultsMode` Diese Funktion bietet eine Reihe von Standardwerten für häufig konfigurierbare Einstellungen, wie HTTP-Kommunikationseinstellungen, Wiederholungsverhalten, regionale Endpunkteinstellungen des Dienstes und potenziell jede SDK-bezogene Konfiguration. Wenn Sie dieses Feature verwenden, können Sie neue Konfigurationsstandardwerte abrufen, die auf gängige Nutzungsszenarien zugeschnitten sind.

Die Standardmodi sind für alle standardisiert. AWS SDKs Das SDK for Java 2.x unterstützt die folgenden Wiederholungsmodi:

- `legacy` – Stellt Standardeinstellungen bereit, die je nach AWS -SDK variieren und die vor der Etablierung von `DefaultsMode` existierten.
- `standard` – Stellt nicht optimierte Standardeinstellungen für die meisten Szenarien bereit.
- `in-region`— Baut auf dem Standardmodus auf und umfasst Einstellungen, die auf Anwendungen zugeschnitten sind, die von dort AWS-Services aus aufrufen AWS-Region.
- `cross-region` – Baut auf dem Standardmodus auf und umfasst Einstellungen mit hohen Timeouts für Anwendungen, die AWS-Services in einer anderen Region anrufen.
- `mobile` – Baut auf dem Standardmodus auf und umfasst Einstellungen mit hohen Timeouts, die auf mobile Anwendungen mit höheren Latenzen zugeschnitten sind.
- `auto` – Baut auf dem Standardmodus auf und beinhaltet experimentelle Features. Das SDK versucht, die Laufzeitumgebung zu ermitteln, um die entsprechenden Einstellungen automatisch zu ermitteln. Die automatische Erkennung basiert auf Heuristik und bietet keine hundertprozentige Genauigkeit. Wenn die Laufzeitumgebung nicht bestimmt werden kann, wird der Standardmodus verwendet. Bei der automatischen Erkennung werden möglicherweise Instance-Metadaten und Benutzerdaten abgefragt, was zu Latenz führen kann. Wenn die Startlatenz für Ihre Anwendung entscheidend ist, empfehlen wir, stattdessen einen expliziten `DefaultsMode` zu wählen.

Sie können den Standardmodus folgendermaßen konfigurieren:

- Direkt auf einem Client, mithilfe von `AwsClientBuilder.Builder#defaultsMode(DefaultsMode)`.
- In einem Konfigurationsprofil über die Profildateieigenschaft `defaults_mode`.

- Global über die `aws.defaultsMode`-Systemeigenschaft.
- Global über die Umgebungsvariable `AWS_DEFAULTS_MODE`.

> **Note**
>
> Bei allen Modi außer `legacy` können sich die bereitgestellten Standardwerte im Laufe der Zeit ändern, da sich bewährte Methoden weiterentwickeln. Daher empfehlen wir, bei Verwendung eines anderen Modus als `legacy` entsprechende Tests durchzuführen, wenn Sie das SDK aktualisieren.

Die Smart-Konfigurationsstandardwerte im Referenzhandbuch AWS SDKs und im Tools-Referenzhandbuch enthalten eine Liste der Konfigurationseigenschaften und ihrer Standardwerte in den verschiedenen Standardmodi.

Sie wählen den Standardmoduswert auf der Grundlage der Eigenschaften Ihrer Anwendung und der Art, mit der AWS-Service die Anwendung interagiert.

Bei der Konfiguration dieser Werte wurde eine breite Palette von Faktoren berücksichtigt AWS-Services . Für eine typische DynamoDB-Bereitstellung, bei der sowohl Ihre DynamoDB-Tabellen als auch Ihre Anwendung in derselben Region bereitgestellt sind, ist der Modus `in-region` unter den `standard`-Standardmodi am relevantesten.

Example Beispielkonfigurierung des DynamoDB SDK-Clients für latenzoptimierte Aufrufe

Im folgenden Beispiel werden die Timeout-Werte für erwartete, latenzarme DynamoDB-Aufrufe nach unten angepasst.

```
DynamoDbAsyncClient asyncClient = DynamoDbAsyncClient.builder()
    .defaultsMode(DefaultsMode.IN_REGION)
    .httpClientBuilder(AwsCrtAsyncHttpClient.builder())
    .overrideConfiguration(ClientOverrideConfiguration.builder()
        .apiCallTimeout(Duration.ofSeconds(3))
        .apiCallAttemptTimeout(Duration.ofMillis(500))
        .build())
    .build();
```

Die jeweilige HTTP-Client-Implementierung kann Ihnen eine noch feinere Kontrolle über das Timeout-Verhalten und die Nutzung von Verbindungen bieten. Für den AWS CRT-basierten Client können Sie

beispielsweise die Option aktivieren`ConnectionHealthConfiguration`, sodass der Client den Zustand der verwendeten Verbindungen aktiv überwachen kann. Weitere Informationen finden Sie unter Erweiterte Konfiguration von AWS CRT-basierten HTTP-Clients im Entwicklerhandbuch.AWS SDK for Java 2.x

Keepalive-Konfigurierung

Durch die Aktivierung von Keepalive können Latenzen reduziert werden, indem Verbindungen wiederverwendet werden. Es gibt zwei verschiedene Arten von Keepalive: HTTP Keepalive und TCP Keepalive.

- - HTTP Keep-Alive versucht, die HTTPS-Verbindung zwischen Client und Server aufrechtzuerhalten, sodass spätere Anfragen dieselbe Verbindung wiederverwenden können. Dadurch entfällt die aufwendige HTTPS-Authentifizierung bei späteren Anfragen. HTTP Keepalive ist standardmäßig auf allen Clients aktiviert.
- TCP Keepalive veranlasst das zugrunde liegende Betriebssystem, kleine Pakete über die Socket-Verbindung zu senden, um sicherzustellen, dass die Verbindung aktiv bleibt und Verbindungsabbrüche sofort erkannt werden. So wird verhindert, dass spätere Anfragen Zeit mit dem Versuch verschwenden, eine bereits getrennte Verbindung zu nutzen. TCP Keepalive ist standardmäßig bei allen Clients deaktiviert. Die folgenden Codebeispiele zeigen, wie Sie es bei den jeweiligen HTTP-Clients aktivieren können. Bei allen nicht-CRT-basierten HTTP-Clients hängt der tatsächliche Keepalive-Mechanismus vom Betriebssystem ab. Daher müssen zusätzliche TCP Keepalive-Werte wie Timeout und Paketanzahl über das Betriebssystem konfiguriert werden. Sie tun dies mithilfe von `sysctl` unter Linux oder macOS oder über Registrierungswerte unter Windows.

Example So aktivieren Sie TCP Keepalive auf einem Apache-basierten HTTP-Client

```
DynamoDbClient client = DynamoDbClient.builder()
    .httpClientBuilder(ApacheHttpClient.builder().tcpKeepAlive(true))
    .build();
```

`URLConnection`-basierter HTTP-Client

Jeder synchrone Client, der den `URLConnection`-basierten HTTP-Client `HttpURLConnection` verwendet, verfügt nicht über einen Mechanismus zur Aktivierung von Keepalive.

Example So aktivieren Sie TCP Keepalive auf einem Netty-basierten HTTP-Client

```
DynamoDbAsyncClient client = DynamoDbAsyncClient.builder()
    .httpClientBuilder(NettyNioAsyncHttpClient.builder().tcpKeepAlive(true))
    .build();
```

Example um TCP Keep-Alive auf einem CRT-basierten HTTP-Client zu aktivieren AWS

Mit dem AWS CRT-basierten HTTP-Client können Sie TCP-Keep-Alive aktivieren und die Dauer steuern.

```
DynamoDbClient client = DynamoDbClient.builder()
    .httpClientBuilder(AwsCrtHttpClient.builder()
    .tcpKeepAliveConfiguration(TcpKeepAliveConfiguration.builder()
        .keepAliveInterval(Duration.ofSeconds(50))
        .keepAliveTimeout(Duration.ofSeconds(5))
        .build()))
    .build();
```

Wenn Sie den asynchronen DynamoDB-Client verwenden, können Sie TCP Keepalive aktivieren, wie im folgenden Code gezeigt.

```
DynamoDbAsyncClient client = DynamoDbAsyncClient.builder()
    .httpClientBuilder(AwsCrtAsyncHttpClient.builder()
    .tcpKeepAliveConfiguration(TcpKeepAliveConfiguration.builder()
        .keepAliveInterval(Duration.ofSeconds(50))
        .keepAliveTimeout(Duration.ofSeconds(5))
        .build()))
    .build();
```

Fehlerbehandlung

Wenn es um die Ausnahmebehandlung geht, werden Laufzeitausnahmen (ungeprüfte) AWS SDK for Java 2.x verwendet.

Die Basisausnahme, die alle SDK-Ausnahmen abdeckt, ist `SdkServiceException`, die von der Java-Laufzeitklasse `RuntimeException` erbt. Wenn Sie diese Ausnahme abfangen, fangen Sie alle vom SDK ausgelösten Ausnahmen ab.

`SdkServiceException` hat eine Unterklasse namens `AwsServiceException`. Diese Unterklasse weist auf Probleme bei der Kommunikation mit dem AWS-Service hin. Eine weitere Unterklasse ist

`DynamoDbException`, die auf ein Problem bei der Kommunikation mit DynamoDB hinweist. Wenn Sie diese Ausnahme abfangen, fangen Sie alle DynamoDB-bezogenen Ausnahmen, jedoch keine anderen SDK-Ausnahmen ab.

Unter `DynamoDbException` gibt es noch spezifischere Ausnahmetypen. Einige betreffen Operationen auf Steuerungsebene, z. B. `TableAlreadyExistsException`. Andere gelten für Operationen auf Datenebene. Ein häufiges Beispiel für eine Ausnahme auf Datenebene ist:

- `ConditionalCheckFailedException` – Sie haben eine Bedingung angegeben, die als false ausgewertet wurde. Beispiel: Sie haben versucht, eine bedingte Aktualisierung für ein Element durchzuführen, aber der tatsächliche Wert des Attributs stimmt nicht mit dem erwarteten Wert in der Bedingung überein. Eine Anfrage, die auf diese Weise fehlschlägt, wird nicht wiederholt.

In anderen Situationen ist keine spezifische Ausnahme definiert. Wenn Ihre Anfrage z. B. gedrosselt wird, kann entweder die spezifische `ProvisionedThroughputExceededException` oder die allgemeinere `DynamoDbException` ausgelöst werden. In beiden Fällen können Sie feststellen, ob die Drosselung die Ausnahme verursacht hat, indem Sie überprüfen, ob `isThrottlingException()` den Wert `true` zurückgibt.

Je nach den Anforderungen Ihrer Anwendung können Sie alle Instances von `AwsServiceException` oder `DynamoDbException` abfangen. Oft ist jedoch ein unterschiedliches Verhalten je nach Situation erforderlich. Die Logik zur Behandlung eines Bedingungsfehlers unterscheidet sich von der zur Behandlung einer Drosselung. Definieren Sie, welche Ausnahmepfade Sie behandeln möchten, und testen Sie die Alternativen. So stellen Sie sicher, dass Ihre Anwendung mit allen relevanten Szenarien umgehen kann.

Eine Übersicht häufiger Fehler, auf die Sie stoßen könnten, finden Sie unter Fehlerbehandlung mit DynamoDB. Siehe auch Häufig auftretende Fehler in der Amazon DynamoDB-API-Referenz. Die API-Referenz enthält auch die genauen Fehler, die für jeden API-Vorgang möglich sind, z. B. für die `Query`-Operation. Informationen zur Behandlung von Ausnahmen finden Sie unter Ausnahmebehandlung für die AWS SDK for Java 2.x im AWS SDK for Java 2.x -Entwicklerhandbuch.

AWS Anforderungs-ID

Jede Anfrage enthält eine Anfrage-ID, die hilfreich sein kann, wenn Sie mit dem AWS Support zusammenarbeiten, um ein Problem zu diagnostizieren. Jede Ausnahme, die von `SdkServiceException` abgeleitet ist, verfügt über die Methode `requestId()`, mit der Sie die Anfrage-ID abrufen können.

Protokollierung

Die vom SDK bereitgestellte Protokollierung kann sowohl zum Erfassen wichtiger Nachrichten aus den Client-Bibliotheken als auch für tiefgehende Debugging-Zwecke nützlich sein. Logger sind hierarchisch aufgebaut und das SDK verwendet `software.amazon.awssdk` als Root-Logger. Sie können die Protokollierungsebene mit einem der folgenden Werte konfigurieren: `TRACE`, `DEBUG`, `INFO`, `WARN`, `ERROR`, `ALL` oder `OFF`. Die konfigurierte Ebene gilt für diesen Logger und alle untergeordneten Logger in der Hierarchie.

Für die Protokollierung AWS SDK for Java 2.x verwendet der die Simple Logging Façade für Java (SLF4J). Diese dient als Abstraktionsschicht über anderen Loggern, sodass Sie den Logger Ihrer Wahl einbinden können. Anweisungen zum Anschließen von Loggern finden Sie im SLF4J-Benutzerhandbuch.

Jeder Logger verhält sich unterschiedlich. Standardmäßig erstellt der Log4j 2.x-Logger einen `ConsoleAppender`, der Protokollereignisse an `System.out` anhängt und die Protokollierungsebene `ERROR` verwendet.

Der in SLF4 J enthaltene SimpleLogger Logger gibt standardmäßig auf Protokollebene aus `System.err` und verwendet diese standardmäßig. `INFO`

Für produktive Umgebungen empfehlen wir, die Ebene für `software.amazon.awssdk` auf `WARN` zu setzen, um wichtige SDK-Meldungen zu erfassen und gleichzeitig die Ausgabemenge zu begrenzen.

Wenn SLF4 J im Klassenpfad keinen unterstützten Logger finden kann (keine SLF4 J-Bindung), wird standardmäßig eine Implementierung ohne Operation verwendet. Diese Implementierung führt dazu, dass Meldungen protokolliert werden, die `System.err` erklären SLF4, dass ich im Klassenpfad keine Logger-Implementierung finden konnte. Um diese Situation zu verhindern, müssen Sie eine Logger-Implementierung hinzufügen. Dazu können Sie in Ihrem Apache Maven `pom.xml` eine Abhängigkeit von Artefakten wie `org.slf4j.slf4j-simple` oder `org.apache.logging.log4j.log4j-slf4j2-imp` hinzufügen.

Weitere Informationen zur Konfiguration der Protokollierung im SDK, einschließlich der Einbindung von Logging-Abhängigkeiten in Ihre Anwendungskonfiguration, finden Sie unter Protokollieren mit dem SDK für Java 2.x im AWS SDK für Java -Entwicklerhandbuch.

Die folgende Konfiguration in der `Log4j2.xml`-Datei zeigt, wie Sie das Protokollierungsverhalten anpassen können, wenn Sie den Apache Log4j 2-Logger verwenden. Diese Konfiguration legt die Root-Logger-Ebene auf `WARN` fest. Alle Logger in der Hierarchie erben diese Protokollebene, einschließlich des `software.amazon.awssdk`-Loggers.

Standardmäßig geht die Ausgabe an `System.out`. Im folgenden Beispiel überschreiben wir noch immer den standardmäßigen Log4j-Appender für die Ausgabe, um ein maßgeschneidertes Log4j-`PatternLayout` anzuwenden.

Beispiel einer **`Log4j2.xml`**-Konfigurationsdatei

Die folgende Konfiguration protokolliert Meldungen auf der Konsole auf den Ebenen `ERROR` und `WARN` für alle Logger-Hierarchien.

```
<Configuration status="WARN">
  <Appenders>
    <Console name="ConsoleAppender" target="SYSTEM_OUT">
      <PatternLayout pattern="%d{YYYY-MM-dd HH:mm:ss} [%t] %-5p %c:%L - %m%n" />
    </Console>
  </Appenders>

  <Loggers>
    <Root level="WARN">
      <AppenderRef ref="ConsoleAppender"/>
    </Root>
  </Loggers>
</Configuration>
```

AWS ID-Protokollierung anfordern

Wenn etwas schief geht, können Sie die Anfrage IDs innerhalb von Ausnahmen finden. Wenn Sie die Anfrage jedoch IDs für Anfragen verwenden möchten, die keine Ausnahmen generieren, können Sie die Protokollierung verwenden.

Der `software.amazon.awssdk.request` Logger gibt die Anfrage IDs auf der `DEBUG` Ebene aus. Das folgende Beispiel erweitert die vorherige configuration example, indem es die Root-Logger-Ebene auf `ERROR`, den `software.amazon.awssdk` auf die Ebene `WARN` und den Logger `software.amazon.awssdk.request` auf `DEBUG` setzt. Das Festlegen dieser Stufen hilft dabei, die Anfrage IDs und andere anforderungsbezogene Details wie den Endpunkt und den Statuscode abzufangen.

```
<Configuration status="WARN">
  <Appenders>
    <Console name="ConsoleAppender" target="SYSTEM_OUT">
      <PatternLayout pattern="%d{YYYY-MM-dd HH:mm:ss} [%t] %-5p %c:%L - %m%n" />
    </Console>
```

```
  </Appenders>

  <Loggers>
    <Root level="ERROR">
      <AppenderRef ref="ConsoleAppender"/>
    </Root>
    <Logger name="software.amazon.awssdk" level="WARN" />
    <Logger name="software.amazon.awssdk.request" level="DEBUG" />
  </Loggers>
</Configuration>
```

Hier finden Sie ein Beispiel für die Protokollausgabe:

```
2022-09-23 16:02:08 [main] DEBUG software.amazon.awssdk.request:85 - Sending Request:
 DefaultSdkHttpFullRequest(httpMethod=POST, protocol=https, host=dynamodb.us-
east-1.amazonaws.com, encodedPath=/, headers=[amz-sdk-invocation-id, Content-Length,
 Content-Type, User-Agent, X-Amz-Target], queryParameters=[])
 2022-09-23 16:02:08 [main] DEBUG software.amazon.awssdk.request:85 - Received
 successful response: 200, Request ID:
 QS9DUMME2NHEDH8TGT9N5V53OJVV4KQNSO5AEMVJF66Q9ASUAAJG, Extended Request ID: not
 available
```

Paginierung

Einige Anfragen, wie `Query` und `Scan` begrenzen die Datenmenge, die mit einer einzelnen Anfrage zurückgegeben wird, und erfordern wiederholte Anfragen, um nachfolgende Seiten abzurufen.

Mit dem `Limit`-Parameter können Sie die maximale Anzahl von Elementen steuern, die für jede Seite gelesen werden sollen. Sie können den `Limit`-Parameter beispielsweise verwenden, um nur die letzten 10 Elemente abzurufen. Dieser Grenzwert gibt an, wie viele Elemente aus der Tabelle gelesen werden sollen, bevor eine Filterung angewendet wird. Wenn Sie nach dem Filtern genau 10 Elemente benötigen, können Sie dies nicht angeben. Sie können nur die Anzahl der vorgefilterten Elemente steuern und clientseitig überprüfen, ob Sie tatsächlich 10 Elemente abgerufen haben. Unabhängig vom Grenzwert haben Antworten immer eine maximale Größe von 1 MB.

In der API-Antwort kann ein `LastEvaluatedKey` enthalten sein. Dies weist darauf hin, dass die Antwort beendet wurde, weil sie eine Anzahl- oder Größenbeschränkung erreicht hat. Dieser Schlüssel ist der letzte Schlüssel, der für diese Antwort ausgewertet wurde. Durch direkte Interaktion mit der API können Sie diesen `LastEvaluatedKey` abrufen und an einen Folgeaufruf als `ExclusiveStartKey` weitergeben, um den nächsten Abschnitt von diesem Startpunkt aus zu lesen.

Wenn kein `LastEvaluatedKey` zurückgegeben wird, bedeutet das, dass keine weiteren Elemente zum `Query`- oder `Scan`-API-Aufruf passen.

Im folgenden Beispiel wird die Low-Level-Schnittstelle verwendet, um die Anzahl der Elemente auf Basis des `keyConditionExpression`-Parameters auf 100 zu beschränken.

```
QueryRequest.Builder queryRequestBuilder = QueryRequest.builder()
        .expressionAttributeValues(Map.of(
                ":pk_val", AttributeValue.fromS("123"),
                ":sk_val", AttributeValue.fromN("1000")))
        .keyConditionExpression("pk = :pk_val AND sk > :sk_val")
        .limit(100)
        .tableName(TABLE_NAME);

while (true) {
    QueryResponse queryResponse = DYNAMODB_CLIENT.query(queryRequestBuilder.build());

    queryResponse.items().forEach(item -> {
        LOGGER.info("item PK: [" + item.get("pk") + "] and SK: [" + item.get("sk") +
 "]");
    });

    if (!queryResponse.hasLastEvaluatedKey()) {
        break;
    }
    queryRequestBuilder.exclusiveStartKey(queryResponse.lastEvaluatedKey());
}
```

AWS SDK for Java 2.x Sie können diese Interaktion mit DynamoDB vereinfachen, indem sie automatische Paginierungsmethoden bereitstellen, die mehrere Serviceaufrufe tätigen, um automatisch die nächsten Ergebnisseiten für Sie abzurufen. Dies vereinfacht Ihren Code, nimmt Ihnen jedoch die Kontrolle über die Ressourcennutzung, die Sie beim manuellen Lesen von Seiten behalten würden.

Mithilfe der im DynamoDB-Client verfügbaren `Iterable`-Methoden wie `QueryPaginator` und `ScanPaginator` kümmert sich das SDK um die Paginierung. Der Rückgabetyp dieser Methoden ist ein benutzerdefiniertes iterierbares Objekt, mit dem Sie durch alle Seiten iterieren können. Das SDK bearbeitet für Sie Serviceanfragen intern. Mithilfe der Java Stream API können Sie das Ergebnis von `QueryPaginator` wie im folgenden Beispiel gezeigt verarbeiten.

```
QueryPublisher queryPublisher =
```

```
    DYNAMODB_CLIENT.queryPaginator(QueryRequest.builder()
        .expressionAttributeValues(Map.of(
            ":pk_val", AttributeValue.fromS("123"),
            ":sk_val", AttributeValue.fromN("1000")))
        .keyConditionExpression("pk = :pk_val AND sk > :sk_val")
        .limit(100)
        .tableName("YourTableName")
        .build());

queryPublisher.items().subscribe(item ->
    System.out.println(item.get("itemData"))).join();
```

Annotationen für Datenklassen

Das Java SDK stellt mehrere Annotationen bereit, die Sie auf die Attribute Ihrer Datenklasse anwenden können. Diese Annotationen beeinflussen, wie das SDK mit den Attributen interagiert. Durch das Hinzufügen einer Annotation kann ein Attribut beispielsweise als impliziter unteilbarer Zähler fungieren, einen automatisch generierten Zeitstempelwert speichern oder eine Versionsnummer eines Elements nachverfolgen. Weitere Informationen finden Sie unter Annotationen für Datenklassen.

Fehlerbehandlung mit DynamoDB

Dieser Abschnitt beschreibt Laufzeitfehler und wie sie gehandhabt werden können. Es werden auch Fehlermeldungen und Codes beschrieben, die Amazon-DynamoDB-spezifisch sind. Eine Liste der häufigsten Fehler, die für alle AWS Dienste gelten, finden Sie unter Access Management

Themen

- Fehlerkomponenten
- Transaktionsfehler
- Fehlermeldungen und Codes
- Fehlerbehandlung in Ihrer Anwendung
- Wiederholversuche bei Fehlern und exponentielles Backoff
- Batchoperationen und Fehlerbehandlung

Fehlerkomponenten

Wenn das Programm eine Anforderung sendet, versucht DynamoDB diese zu verarbeiten. Bei einer erfolgreichen Anforderung gibt DynamoDB einen HTTP-Erfolgscode (`200 OK`) sowie die Ergebnisse der angeforderten Operation zurück.

Wenn die Anforderung nicht erfolgreich ist, gibt DynamoDB einen Fehler zurück. Jeder Fehler hat drei Komponenten:

- Einen HTTP-Statuscode (z. B. `400`).
- Einen Ausnahmenamen (z. B. `ResourceNotFoundException`).
- Eine Fehlermeldung (z. B. `Requested resource not found: Table: tablename not found`).

AWS SDKs Sie kümmern sich darum, dass Fehler an Ihre Anwendung weitergegeben werden, sodass Sie entsprechende Maßnahmen ergreifen können. Sie können beispielsweise in einem Java-Programm eine `try-catch`-Logik schreiben, um eine `ResourceNotFoundException` zu verarbeiten.

Wenn Sie kein AWS SDK verwenden, müssen Sie den Inhalt der Low-Level-Antwort von DynamoDB analysieren. Nachfolgend finden Sie ein Beispiel einer solchen Antwort.

```
HTTP/1.1 400 Bad Request
x-amzn-RequestId: LDM6CJP8RMQ1FHKSC1RBVJFPNVV4KQNSO5AEMF66Q9ASUAAJG
Content-Type: application/x-amz-json-1.0
Content-Length: 240
Date: Thu, 15 Mar 2012 23:56:23 GMT

{"__type":"com.amazonaws.dynamodb.v20120810#ResourceNotFoundException",
"message":"Requested resource not found: Table: tablename not found"}
```

Transaktionsfehler

Informationen zu Transaktionsfehlern finden Sie unter Handhabung von Transaktionskonflikten in DynamoDB

Fehlermeldungen und Codes

Die folgende Liste enthält von DynamoDB zurückgegebene Ausnahmen, gruppiert nach HTTP-Statuscode. Wenn OK, um es erneut zu versuchen? Ja ist, können Sie die gleiche Anforderung erneut senden. Wenn OK, um es erneut zu versuchen? auf Nein eingestellt ist, müssen Sie das Problem clientseitig lösen, bevor Sie eine neue Anforderung absenden.

HTTP-Statuscode 400

Der HTTP-Statuscode `400` weist auf ein Problem mit der Anforderung hin, wie z. B. einen Authentifizierungsfehler, fehlende Parameter oder die Überschreitung des bereitgestellten Durchsatzes der Tabelle. Sie müssen das Problem zuerst in der Anwendung beheben, bevor Sie die Anforderung erneut senden.

AccessDeniedException

Meldung: Access denied. (Zugriff verweigert.)

Der Client hat die Anforderung nicht richtig signiert. Wenn Sie ein AWS -SDK verwenden, werden die Anforderungen automatisch für Sie signiert. Rufen Sie andernfalls Signaturprozess mit Signature Version 4 in der Allgemeine AWS-Referenz auf.

Erneut versuchen? Nein

ConditionalCheckFailedException

Meldung: The conditional request failed. (Die bedingte Anforderung ist fehlgeschlagen.)

Sie haben eine Bedingung angegeben, die als False ausgewertet wurde. Beispiel: Sie haben versucht, eine bedingte Aktualisierung für ein Element durchzuführen, aber der tatsächliche Wert des Attributs stimmt nicht mit dem erwarteten Wert in der Bedingung überein.

Erneut versuchen? Nein

IncompleteSignatureException

Meldung: Die angeforderte Signatur entspricht nicht den AWS Standards.

Die Anforderungssignatur enthielt nicht alle erforderlichen Komponenten. Wenn Sie ein AWS SDK verwenden, werden Anfragen automatisch für Sie signiert. Andernfalls fahren Sie mit dem Signaturprozess für Signature Version 4 in der fort. Allgemeine AWS-Referenz

Erneut versuchen? Nein

ItemCollectionSizeLimitExceededException

Meldung: Collection size exceeded. (Sammlungsgröße überschritten.)

Für eine Tabelle mit einem lokalen sekundären Index hat eine Gruppe von Elementen mit demselben Partitionsschlüsselwert die maximale Größe von 10 GB überschritten. Weitere Informationen zu Elementsammlungen finden Sie unter Elementauflistungen in lokalen sekundären Indizes.

Erneut versuchen? Ja

LimitExceededException

Meldung: Too many operations for a given subscriber. (Zu viele Operationen für einen bestimmten Abonnenten.)

Es gibt zu viele gleichzeitige Operationen auf Steuerebene. Die kumulative Anzahl von Tabellen und Indizes im Status `CREATING`, `DELETING` oder `UPDATING` darf 500 nicht überschreiten.

Erneut versuchen? Ja

MissingAuthenticationTokenException

Meldung: Anforderung muss eine gültige (registrierte) AWS -Zugriffsschlüssel-ID enthalten.

Die Anforderung schloss die erforderliche Autorisierungskopfzeile nicht ein oder sie war falsch formatiert. Siehe DynamoDB Low-Level-API.

Erneut versuchen? Nein

ProvisionedThroughputExceededException

Meldung: Sie haben Ihren maximal erlaubten bereitgestellten Durchsatz für eine Tabelle oder für einen oder mehrere globale sekundäre Indizes überschritten. Um Leistungskennzahlen für den bereitgestellten Durchsatz im Vergleich zum verbrauchten Durchsatz anzuzeigen, öffnen Sie die CloudWatchAmazon-Konsole.

Der Fehler enthält eine Liste von `ThrottlingReason`-Feldern, die spezifischen Kontext darüber liefern, warum eine Drosselung aufgetreten ist. Diese folgen dem Format `ResourceType +OperationType+LimitType` (z. B. `TableReadProvisionedThroughputExceeded`) sowie dem ARN der betroffenen Ressource. Dies hilft Ihnen dabei zu erkennen, welche Ressource von der Drosselung betroffen ist (Tabelle oder Index), welcher Operationstyp die Drosselung ausgelöst hat (Lese- oder Schreibvorgang) und welche spezifische Beschränkung überschritten wurde (in diesem Fall: bereitgestellte Kapazität). Weitere Informationen zur Diagnose und Lösung von Drosselungsproblemen finden Sie unter Drosselungsdiagnose.

Beispiel: Die Anforderungsrate ist zu hoch. Die AWS SDKs für DynamoDB wiederholen automatisch Anfragen, die diese Ausnahme erhalten. Die Anforderung ist letztendlich erfolgreich, sofern die Warteschlange für Wiederholungsversuche für das Beenden nicht zu groß ist. Verringern Sie die Häufigkeit der Anforderungen mit Wiederholversuche bei Fehlern und exponentielles Backoff.

Erneut versuchen? Ja

ReplicatedWriteConflictException

Nachricht: Ein oder mehrere Elemente in dieser Anfrage werden durch eine Anfrage in einer anderen Region geändert.

Beispiel: Eine Schreiboperation wurde für ein Element in einer globalen Tabelle mit Multi-Region Strong Consistency (MRSC) angefordert, das derzeit durch eine Anfrage in einer anderen Region geändert wird.

Erneut versuchen? Ja

RequestLimitExceeded

Meldung: Throughput exceeds the current throughput limit for your account. (Der Durchsatz überschreitet die aktuelle Durchsatzbegrenzung für Ihr Konto.) Um eine Erhöhung des Limits zu beantragen, wenden Sie sich an den AWS Support unter https://aws.amazon.com/support.

Der Fehler enthält eine Liste von `ThrottlingReason`-Feldern, die spezifischen Kontext darüber liefern, warum eine Drosselung aufgetreten ist. Diese folgen dem Format `ResourceType +OperationType+LimitType` (z. B. `TableWriteAccountLimitExceeded` oder `IndexReadAccountLimitExceeded`) sowie dem ARN der betroffenen Ressource. Dies hilft Ihnen zu erkennen, welche Ressource von der Drosselung betroffen ist (Tabelle oder Index),

welcher Operationstyp die Drosselung ausgelöst hat (Lese- oder Schreibvorgang) und dass Sie die servicebezogenen Kontingente auf Kontoebene überschritten haben. Weitere Informationen zur Diagnose und Lösung von Drosselungsproblemen finden Sie unter Drosselungsdiagnose.

Beispiel: Die Rate der On-Demand-Anforderungen überschreitet den zulässigen Kontodurchsatz.

Erneut versuchen? Ja

ResourceInUseException

Meldung: The resource which you are attempting to change is in use. (Die Ressource, die Sie verändern möchten, wird gerade benutzt.)

Beispiel: Sie haben versucht, eine vorhandene Tabelle neu zu erstellen oder eine Tabelle zu löschen, die sich momentan im Status `CREATING` befindet.

Erneut versuchen? Nein

ResourceNotFoundException

Meldung: Requested resource not found. (Angefragte Ressource nicht gefunden.)

Beispiel: Die Tabelle, die angefordert wird, ist nicht vorhanden oder befindet sich zu früh im Status `CREATING`.

Erneut versuchen? Nein

ThrottlingException

Meldung: Rate of requests exceeds the allowed throughput. (Anforderungsrate überschreitet den erlaubten Durchsatz.)

Diese Ausnahme wird als AmazonServiceException Antwort mit dem Statuscode THROTTLING_EXCEPTION zurückgegeben. Diese Ausnahme wird möglicherweise zurückgegeben, wenn Sie API-Operationen der Steuerebene zu schnell ausführen.

Bei Tabellen, die den On-Demand-Modus verwenden, wird diese Ausnahme möglicherweise für alle API-Operationen der Datenebene zurückgegeben, wenn Ihre Anforderungsrate zu hoch ist. Weitere Informationen zu On-Demand-Skalierungen finden Sie unter Anfänglicher Durchsatz und Skalierungseigenschaften.

Der Fehler enthält eine Liste von `ThrottlingReason`-Feldern, die spezifischen Kontext darüber liefern, warum eine Drosselung aufgetreten ist. Diese folgen dem Format `ResourceType+OperationType+LimitType` (z. B. `TableReadKeyRangeThroughputExceeded` oder `IndexWriteMaxOnDemandThroughputExceeded`) sowie dem ARN der betroffenen Ressource. Diese Informationen helfen Ihnen zu erkennen, welche Ressource von der Drosselung betroffen ist (Tabelle oder Index), welcher Operationstyp die Drosselung ausgelöst hat (Lese- oder Schreibvorgang) und welches spezifische Beschränkung überschritten wurde (Partitionsgrenzen oder maximale Durchsatzrate im On-Demand-Modus). Weitere Informationen zur Diagnose und Lösung von Drosselungsproblemen finden Sie unter Drosselungsdiagnose.

Erneut versuchen? Ja

UnrecognizedClientException

Meldung: The Access Key ID or security token is invalid. (Die Zugriffsschlüssel-ID oder der Sicherheitstoken ist ungültig.)

Die Anforderungssignatur ist nicht korrekt. Die wahrscheinlichste Ursache ist eine ungültige AWS Zugriffsschlüssel-ID oder ein ungültiger geheimer Schlüssel.

Erneut versuchen? Ja

ValidationException

Meldung: Variiert je nach auftretendem spezifischen Fehler(n)

Dieser Fehler kann aus verschiedenen Gründen auftreten, z. B. wenn ein erforderlicher Parameter fehlt, ein Wert außerhalb des Bereichs liegt oder Datentypen nicht übereinstimmen. Die Fehlermeldung enthält Details zum spezifischen Teil der Anforderung, die den Fehler verursacht hat.

Erneut versuchen? Nein

HTTP-Statuscode 5xx

Der HTTP-Statuscode `5xx` weist auf ein Problem hin, das von AWS behoben werden muss. Dies kann ein vorübergehender Fehler sein. In diesem Fall können Sie Ihre Anforderung wiederholen, bis sie korrekt ausgeführt wird. Andernfalls rufen Sie das AWS Service Health Dashboard auf, um zu sehen, ob es Betriebsprobleme mit dem Service gibt.

Weitere Informationen finden Sie unter Wie behebe ich HTTP-5xx-Fehler in Amazon DynamoDB?

InternalServerError (HTTP 500)

DynamoDB konnte die Anforderung nicht verarbeiten.

Erneut versuchen? Ja

> **Note**
>
> Während der Arbeit mit Elementen können interne Serverfehler auftreten. Diese sind während der Lebensdauer einer Tabelle zu erwarten. Alle fehlgeschlagenen Anforderungen können sofort abgerufen werden.
> Wenn Sie bei einem Schreibvorgang einen Statuscode 500 erhalten, ist der Vorgang entweder erfolgreich oder ist fehlgeschlagen. Wenn der Schreibvorgang eine `TransactWriteItem`-Anforderung ist, dann ist es in Ordnung, den Vorgang erneut zu versuchen. Handelt es sich bei dem Schreibvorgang um eine Schreibanforderung für ein einzelnes Element wie `PutItem`, `UpdateItem`, oder`DeleteItem`, dann sollte Ihre Anwendung den Status des Elements lesen, bevor Sie den Vorgang erneut versuchen. and/or Verwenden Sie diese Option, CLI-Beispiel für DynamoDB-Bedingungsausdrücke um sicherzustellen, dass das Element nach dem erneuten Versuch in einem korrekten Zustand bleibt, unabhängig davon, ob der vorherige Vorgang erfolgreich war oder nicht. Wenn Idempotenz eine Voraussetzung für die Schreiboperation ist, verwenden Sie bitte `TransactWriteItem`, das idempotente Anfragen durch automatische Angabe eines `ClientRequestToken` unterstützt, um mehrere Versuche zu disambiguieren, dieselbe Aktion auszuführen.

ServiceUnavailable (HTTP 503)

DynamoDB ist derzeit nicht verfügbar. (Dies sollte ein vorübergehender Status sein.)

Erneut versuchen? Ja

Fehlerbehandlung in Ihrer Anwendung

Damit Ihre Anwendung reibungslos ausgeführt wird, müssen Sie Logik zum Erfassen und Behandeln von Fehlern integrieren. Typische Ansätze umfassen die Verwendung von `try-catch`-Blöcken oder `if-then`-Anweisungen.

AWS SDKs Sie führen ihre eigenen Wiederholungen und Fehlerprüfungen durch. Wenn Sie bei der Verwendung eines der Geräte auf einen Fehler stoßen AWS SDKs, können Ihnen der Fehlercode und die Beschreibung bei der Behebung helfen.

Sie sollten auch eine `Request ID` in der Antwort sehen. Das `Request ID` kann hilfreich sein, wenn Sie mit dem AWS Support zusammenarbeiten müssen, um ein Problem zu diagnostizieren.

Wiederholversuche bei Fehlern und exponentielles Backoff

Zahlreiche Komponenten im Netzwerk, wie z. B. DNS-Server, Switches und Load Balancer, können irgendwann im Lebenszyklus einer Anforderung Fehler generieren. Die übliche Methode zum Umgang mit diesen Fehlermeldungen in einer vernetzten Umgebung besteht darin, Wiederholversuche in der Client-Anwendung zu implementieren. Diese Methode erhöht die Zuverlässigkeit der Anwendung.

Jedes AWS SDK implementiert die Wiederholungslogik automatisch. Sie können die Parameter für Wiederholversuche an Ihre Bedürfnisse anpassen. Nehmen wir als Beispiel eine Java-Anwendung, die eine Fail-Fast-Strategie erfordert, in der im Falle eines Fehlers keine Wiederholversuche zulässig sind. Mit dem AWS SDK für Java könnten Sie die `ClientConfiguration` Klasse verwenden und den `maxErrorRetry` Wert von angeben, `0` um die Wiederholungsversuche auszuschalten. Weitere Informationen finden Sie in der AWS SDK-Dokumentation für Ihre Programmiersprache.

Wenn Sie kein AWS SDK verwenden, sollten Sie die ursprünglichen Anfragen, bei denen Serverfehler auftreten, erneut versuchen (5xx). Client-Fehler (4xx, außer `ThrottlingException` oder `ProvisionedThroughputExceededException`) weisen hingegen darauf hin, dass die Anforderung selbst geändert werden muss, um das Problem zu beheben. Erst dann sollte sie wiederholt werden. Detaillierte Empfehlungen zur Behandlung bestimmter Drosselungsszenarien finden Sie im Abschnitt Problembehandlung bei DynamoDB-Drosselung.

Zusätzlich zu einfachen Wiederholungsversuchen implementiert jedes AWS SDK einen exponentiellen Backoff-Algorithmus für eine bessere Flusskontrolle. Das Konzept hinter dem exponentiellen Backoff ist die Verwendung von progressiv längeren Wartezeiten zwischen Wiederholversuchen für aufeinanderfolgende Fehlermeldungen. Zum Beispiel bis zu 50 Millisekunden vor dem ersten Wiederholversuch, bis zu 100 Millisekunden vor dem zweiten, bis zu 200 Millisekunden vor dem dritten und so weiter. Wenn die Anforderung hingegen nach einer Minute nicht erfolgreich war, liegt das Problem möglicherweise an der Anforderungsgröße, die den bereitgestellten Durchsatz übersteigt, und nicht an der Anforderungsrate. Grenzen Sie die maximale Anzahl von Wiederholversuchen bis zu etwa einer Minute ein. Wenn die Anforderung nicht erfolgreich ist, untersuchen Sie Ihre bereitgestellten Durchsatzoptionen.

> **Note**
>
> Sie AWS SDKs implementieren automatische Wiederholungslogik und exponentiellen Backoff.

Die meisten Algorithmen für das exponentielle Backoff verwenden Jitter (zufällige Verzögerung), um nachfolgende Kollisionen zu verhindern. Da Sie hier nicht versuchen, solche Kollisionen zu verhindern, müssen Sie diese Zufallszahl nicht verwenden. Wenn Sie jedoch gleichzeitige Clients verwenden, kann Jitter dazu beitragen, dass Ihre Anforderungen schneller verarbeitet werden. Weitere Informationen finden Sie im Blogbeitrag zu Exponentielles Backoff und Jitter.

Batchoperationen und Fehlerbehandlung

Die Low-Level-API von DynamoDB unterstützt Batch-Operationen für Lese- und Schreibvorgänge. `BatchGetItem` liest Elemente aus einzelnen oder mehreren Tabellen und `BatchWriteItem` schreibt Elemente in einzelne oder mehrere Tabellen oder löscht sie daraus. Diese Batchoperationen werden als Wrapper um andere Nicht-Batch-DynamoDB-Operationen herum implementiert. Mit anderen Worten, `BatchGetItem` ruft `GetItem` einmal für jedes Element im Stapel auf. Ebenso ruft `BatchWriteItem` `DeleteItem` oder `PutItem` für jedes Element im Stapel auf, je nachdem.

Eine Stapeloperation kann den Ausfall einzelner Anforderungen im Stapel tolerieren. Nehmen Sie beispielsweise eine `BatchGetItem`-Anforderung für das Lesen von fünf Elementen. Auch wenn einige der zugrunde liegenden `GetItem`-Anforderungen fehlschlagen, führt dies nicht zum Ausfall der gesamten `BatchGetItem`-Operation. Wenn jedoch alle fünf Leseoperationen fehlschlagen, schlägt der gesamte `BatchGetItem`-Vorgang fehl.

Die Stapeloperationen geben Informationen zu ausgefallenen individuellen Anforderungen zurück, sodass Sie das Problem diagnostizieren und die Operation wiederholen können. Für `BatchGetItem` werden die jeweiligen Tabellen und primären Schlüssel in den `UnprocessedKeys`-Wert der Antwort zurückgegeben. Für `BatchWriteItem` werden die gleichen Informationen in `UnprocessedItems` zurückgegeben.

Vermutlich ist die Ursache für einen fehlgeschlagenen Lese- oder Schreibvorgang eine Drosselung. Für `BatchGetItem` steht einer oder mehreren Tabellen in der Stapelanforderung nicht ausreichend Lesekapazität zur Verfügung, um die Operation zu unterstützen. Für `BatchWriteItem` steht einer oder mehreren Tabellen nicht ausreichend Schreibkapazität zur Verfügung.

Wenn DynamoDB unverarbeitete Elemente zurückgibt, sollten Sie die Batchoperation für diese Elemente wiederholen. Jedoch empfehlen wir dringend, dass Sie einen exponentiellen Backoff-Algorithmus verwenden. Wenn Sie die Stapeloperation sofort wiederholen, können die zugrunde liegenden Lese- und Schreibanforderungen dennoch aufgrund einer Drosselung der einzelnen Tabellen fehlschlagen. Wenn Sie die Stapeloperation mithilfe des exponentiellen Backoff verzögern, werden die einzelnen Anforderungen in dem Stapel eher erfolgreich sein.

> **Note**
>
> Einige AWS SDKs bieten Clients auf höherer Ebene, die Wiederholungsversuche für unverarbeitete Elemente automatisch verarbeiten, sodass Sie diese Wiederholungslogik nicht selbst implementieren müssen. Beispiel:
>
> - Java — Der DynamoDB Enhanced Client in AWS SDK für Java Version 2 und DynamoDBMapper in Version 1 versuchen beide automatisch, unverarbeitete Elemente erneut zu versuchen, wenn Batch-Operationen ausgeführt werden.
> - Python — Die Tabellenressource boto3 verarbeitet implizit `batch_writer` Wiederholungen unverarbeiteter Elemente für Batch-Schreibvorgänge. Weitere Informationen finden Sie unter the section called “Verwenden von batch_writer”.
>
> Wenn Sie einen Low-Level-Client oder ein SDK verwenden, das dieses Verhalten nicht bietet, müssen Sie die Wiederholungslogik wie oben beschrieben selbst implementieren.

DynamoDB mit einem SDK verwenden AWS

AWS Software Development Kits (SDKs) sind für viele beliebte Programmiersprachen verfügbar. Jedes SDK bietet eine API, Codebeispiele und Dokumentation, die es Entwicklern erleichtern, Anwendungen in ihrer bevorzugten Sprache zu erstellen.

SDK-Dokumentation	Codebeispiele
AWS SDK für C++	AWS SDK für C++ Codebeispiele
AWS CLI	AWS CLI Code-Beispiele
AWS SDK für Go	AWS SDK für Go Code-Beispiele

SDK-Dokumentation	Codebeispiele
AWS SDK für Java	AWS SDK für Java Code-Beispiele
AWS SDK für JavaScript	AWS SDK für JavaScript Code-Beispiele
AWS SDK für Kotlin	AWS SDK für Kotlin Code-Beispiele
AWS SDK für .NET	AWS SDK für .NET Code-Beispiele
AWS SDK für PHP	AWS SDK für PHP Code-Beispiele
AWS -Tools für PowerShell	AWS -Tools für PowerShell Code-Beispiele
AWS SDK für Python (Boto3)	AWS SDK für Python (Boto3) Code-Beispiele
AWS SDK für Ruby	AWS SDK für Ruby Code-Beispiele
AWS SDK für Rust	AWS SDK für Rust Code-Beispiele
AWS SDK für SAP ABAP	AWS SDK für SAP ABAP Code-Beispiele
AWS SDK für Swift	AWS SDK für Swift Code-Beispiele

Spezifische Beispiele zu DynamoDB finden Sie unter Codebeispiele für DynamoDB mit AWS SDKs.

> **Beispiel für die Verfügbarkeit**
>
> Sie können nicht finden, was Sie brauchen? Fordern Sie ein Codebeispiel an, indem Sie unten den Link Provide feedback (Feedback geben) auswählen.

Arbeiten mit Tabellen, Elementen, Abfragen, Scans und Indizes

Dieser Abschnitt enthält Details zum Arbeiten mit Tabellen, Elementen, Abfragen und mehr in Amazon DynamoDB.

Themen

- Arbeiten mit Tabellen und Daten in DynamoDB
- Globale Tabellen: multiaktive, regionsübergreifende Replikation
- Arbeiten mit Elementen und Attributen in DynamoDB
- Verbessern des Datenzugriffs mit sekundären Indizes in DynamoDB
- Verwalten komplexer Workflows mit DynamoDB-Transaktionen
- Ändern der Datenerfassung mit Amazon DynamoDB

Arbeiten mit Tabellen und Daten in DynamoDB

In diesem Abschnitt wird beschrieben, wie Sie die AWS Command Line Interface (AWS CLI) und die verwenden, AWS SDKs um Tabellen in Amazon DynamoDB zu erstellen, zu aktualisieren und zu löschen.

> **Note**
>
> Sie können auch die entsprechenden Aufgaben mithilfe der AWS-Managementkonsole durchführen. Weitere Informationen finden Sie unter Verwenden der Konsole.

Außerdem erhalten Sie weitere Informationen zur Durchsatzkapazität mithilfe von DynamoDB-Auto-Scaling oder durch die manuelle Festlegung des bereitgestellten Durchsatzes.

Themen

- Grundlegende Operationen für DynamoDB-Tabellen
- Überlegungen bei der Auswahl einer Tabellenklasse in DynamoDB
- Hinzufügen von Tags und Labels zu Ressourcen in DynamoDB

Grundlegende Operationen für DynamoDB-Tabellen

Ähnlich wie andere Datenbanksysteme speichert Amazon DynamoDB Daten in Tabellen. Sie können Ihre Tabellen mit einigen grundlegenden Operationen verwalten.

Themen

- Erstellen einer Tabelle
- Beschreiben einer Tabelle
- Aktualisieren einer Tabelle
- Löschen einer Tabelle
- Löschschutz verwenden
- Auflisten von Tabellennamen
- Beschreiben der bereitgestellten Durchsatzkontingente

Erstellen einer Tabelle

Verwenden Sie `CreateTable`, um eine Tabelle in Amazon DynamoDB zu erstellen. Zum Erstellen einer Tabelle müssen Sie folgende Informationen angeben:

- Tabellenname. Der Name muss den Benennungsregeln von DynamoDB entsprechen und für das aktuelle AWS Konto und die Region eindeutig sein. So können Sie beispielsweise eine `People`-Tabelle in USA Ost (Nord-Virginia) und eine andere `People`-Tabelle in Europa (Irland) erstellen. Allerdings würden sich diese beiden Tabellen vollständig unterscheiden. Weitere Informationen finden Sie unter Unterstützte Datentypen und Benennungsregeln in Amazon DynamoDB.
- Primärschlüssel. Der Primärschlüssel kann aus einem Attribut (Partitionsschlüssel) oder zwei Attributen (Partitionsschlüssel und Sortierschlüssel) bestehen. Sie müssen die Attributnamen, Datentypen und die Rolle der einzelnen Attribute angeben: `HASH` (für einen Partitionsschlüssel) und `RANGE` (für einen Sortierschlüssel). Weitere Informationen finden Sie unter Primärschlüssel.
- Durchsatzeinstellungen (für bereitgestellte Tabellen). Im Modus bereitgestellter Kapazität müssen Sie die Anfangseinstellungen für den Lese- und Schreibdurchsatz für die Tabelle angeben. Sie können diese Einstellungen später anpassen oder das DynamoDB-Auto-Scaling aktivieren, um die Einstellungen für Sie zu verwalten. Weitere Informationen erhalten Sie unter DynamoDB – Modus mit bereitgestellter Kapazität und Automatische Verwaltung der Durchsatzkapazität mit DynamoDB-Auto-Scaling.

Beispiel 1: Erstellen einer On-Demand-Tabelle

So erstellen Sie dieselbe Tabelle `Music` im On-Demand-Modus

```
aws dynamodb create-table \
    --table-name Music \
    --attribute-definitions \
        AttributeName=Artist,AttributeType=S \
        AttributeName=SongTitle,AttributeType=S \
    --key-schema \
        AttributeName=Artist,KeyType=HASH \
        AttributeName=SongTitle,KeyType=RANGE \
    --billing-mode=PAY_PER_REQUEST
```

Die `CreateTable`-Operation gibt Metadaten für die Tabelle, wie unten gezeigt, zurück.

```
{
    "TableDescription": {
        "TableArn": "arn:aws:dynamodb:us-east-1:123456789012:table/Music",
        "AttributeDefinitions": [
            {
                "AttributeName": "Artist",
                "AttributeType": "S"
            },
            {
                "AttributeName": "SongTitle",
                "AttributeType": "S"
            }
        ],
        "ProvisionedThroughput": {
            "NumberOfDecreasesToday": 0,
            "WriteCapacityUnits": 0,
            "ReadCapacityUnits": 0
        },
        "TableSizeBytes": 0,
        "TableName": "Music",
        "BillingModeSummary": {
            "BillingMode": "PAY_PER_REQUEST"
        },
        "TableStatus": "CREATING",
        "TableId": "12345678-0123-4567-a123-abcdefghijkl",
        "KeySchema": [
            {
```

```
                "KeyType": "HASH",
                "AttributeName": "Artist"
            },
            {
                "KeyType": "RANGE",
                "AttributeName": "SongTitle"
            }
        ],
        "ItemCount": 0,
        "CreationDateTime": 1542397468.348
    }
}
```

> **Important**
>
> Wenn `DescribeTable` für eine On-Demand-Tabelle aufgerufen wird, werden Lesekapazitätseinheiten und Schreibkapazitätseinheiten auf 0 eingestellt.

Beispiel 2: Erstellen einer Tabelle im Modus bereitgestellter Kapazität

Das folgende AWS CLI Beispiel zeigt, wie eine Tabelle () erstellt wird. `Music` Der Primärschlüssel besteht aus `Artist` (Partitionsschlüssel) und `SongTitle` (Sortierschlüssel), die beide vom Datentyp `String` sind. Der maximale Durchsatz für diese Tabelle ist 10 Lesekapazitätseinheiten und 5 Schreibkapazitätseinheiten.

```
aws dynamodb create-table \
    --table-name Music \
    --attribute-definitions \
        AttributeName=Artist,AttributeType=S \
        AttributeName=SongTitle,AttributeType=S \
    --key-schema \
        AttributeName=Artist,KeyType=HASH \
        AttributeName=SongTitle,KeyType=RANGE \
    --provisioned-throughput \
        ReadCapacityUnits=10,WriteCapacityUnits=5
```

Die `CreateTable`-Operation gibt Metadaten für die Tabelle, wie unten gezeigt, zurück.

```
{
    "TableDescription": {
```

```
        "TableArn": "arn:aws:dynamodb:us-east-1:123456789012:table/Music",
        "AttributeDefinitions": [
            {
                "AttributeName": "Artist",
                "AttributeType": "S"
            },
            {
                "AttributeName": "SongTitle",
                "AttributeType": "S"
            }
        ],
        "ProvisionedThroughput": {
            "NumberOfDecreasesToday": 0,
            "WriteCapacityUnits": 5,
            "ReadCapacityUnits": 10
        },
        "TableSizeBytes": 0,
        "TableName": "Music",
        "TableStatus": "CREATING",
        "TableId": "12345678-0123-4567-a123-abcdefghijkl",
        "KeySchema": [
            {
                "KeyType": "HASH",
                "AttributeName": "Artist"
            },
            {
                "KeyType": "RANGE",
                "AttributeName": "SongTitle"
            }
        ],
        "ItemCount": 0,
        "CreationDateTime": 1542397215.37
    }
}
```

Das Element `TableStatus` gibt den aktuellen Status der Tabelle an (`CREATING`). Es kann eine Weile dauern die Tabelle zu erstellen, abhängig von den Werten die Sie für `ReadCapacityUnits` und `WriteCapacityUnits` angeben. Größere Werte für diese erfordern, dass DynamoDB der Tabelle mehr Ressourcen zuweist.

Beispiel 3: Erstellen Sie eine Tabelle mit der Tabellenklasse DynamoDB Standard-Infrequent Access

So erstellen Sie dieselbe `Music`-Tabelle mit der Tabellenklasse DynamoDB Standard-Infrequent Access.

```
aws dynamodb create-table \
    --table-name Music \
    --attribute-definitions \
        AttributeName=Artist,AttributeType=S \
        AttributeName=SongTitle,AttributeType=S \
    --key-schema \
        AttributeName=Artist,KeyType=HASH \
        AttributeName=SongTitle,KeyType=RANGE \
    --provisioned-throughput \
        ReadCapacityUnits=10,WriteCapacityUnits=5 \
    --table-class STANDARD_INFREQUENT_ACCESS
```

Die `CreateTable`-Operation gibt Metadaten für die Tabelle, wie unten gezeigt, zurück.

```
{
    "TableDescription": {
        "TableArn": "arn:aws:dynamodb:us-east-1:123456789012:table/Music",
        "AttributeDefinitions": [
            {
                "AttributeName": "Artist",
                "AttributeType": "S"
            },
            {
                "AttributeName": "SongTitle",
                "AttributeType": "S"
            }
        ],
        "ProvisionedThroughput": {
            "NumberOfDecreasesToday": 0,
            "WriteCapacityUnits": 5,
            "ReadCapacityUnits": 10
        },
        "TableClassSummary": {
            "LastUpdateDateTime": 1542397215.37,
            "TableClass": "STANDARD_INFREQUENT_ACCESS"
        },
        "TableSizeBytes": 0,
        "TableName": "Music",
```

```
        "TableStatus": "CREATING",
        "TableId": "12345678-0123-4567-a123-abcdefghijkl",
        "KeySchema": [
            {
                "KeyType": "HASH",
                "AttributeName": "Artist"
            },
            {
                "KeyType": "RANGE",
                "AttributeName": "SongTitle"
            }
        ],
        "ItemCount": 0,
        "CreationDateTime": 1542397215.37
    }
}
```

Beschreiben einer Tabelle

Um Details über eine Tabelle anzuzeigen, verwenden Sie die `DescribeTable`-Operation. Sie müssen den Tabellennamen angeben. Die Ausgabe von `DescribeTable` erfolgt in demselben Format wie bei `CreateTable`. Sie enthält den Zeitstempel, mit dem die Tabelle erstellt wurde, ihr Schlüsselschema, ihre bereitgestellten Durchsatzeinstellungen, ihre geschätzte Größe und alle vorhandenen sekundären Indexe.

> **Important**
>
> Wenn `DescribeTable` für eine On-Demand-Tabelle aufgerufen wird, werden Lesekapazitätseinheiten und Schreibkapazitätseinheiten auf 0 eingestellt.

Example

```
aws dynamodb describe-table --table-name Music
```

Die Tabelle ist einsatzbereit, wenn der `TableStatus` sich von `CREATING` in `ACTIVE` ändert.

> **Note**
>
> Wenn Sie eine `DescribeTable`-Anforderung sofort nach der `CreateTable`-Anforderung ausgeben, kann DynamoDB möglicherweise einen Fehler (`ResourceNotFoundException`)

zurückgeben. Der Grund hierfür ist, dass `DescribeTable` eine Eventually Consistent-Abfrage verwendet und dass die Metadaten für Ihre Tabelle zu diesem Zeitpunkt möglicherweise nicht verfügbar sein könnten. Warten Sie einige Sekunden und versuchen Sie dann die `DescribeTable`-Anforderung erneut.
Für Abrechnungszwecke beinhalten Ihre DynamoDB-Speicherkosten einen Overhead pro Artikel von 100 Byte. (Weitere Informationen finden Sie unter DynamoDB-Preise.) Diese zusätzlichen 100 Byte pro Element werden nicht in Kapazitätseinheitenberechnungen oder durch die `DescribeTable`-Operation verwendet.

Aktualisieren einer Tabelle

Die `UpdateTable`-Operation ermöglicht es Ihnen einen der folgenden Schritte ausführen:

- Ändern Sie die bereitgestellten Durchsatzeinstellungen (für Tabellen im Modus bereitgestellter Kapazität).
- Ändern Sie den read/write Kapazitätsmodus der Tabelle.
- Bearbeiten Sie globale sekundäre Indizes für die Tabelle (siehe Verwenden globaler sekundärer Indizes in DynamoDB).
- Aktivieren oder Deaktivieren von DynamoDB Streams für die Tabelle (siehe Ändern Sie die Datenerfassung für DynamoDB Streams).

Example

Das folgende AWS CLI Beispiel zeigt, wie die Einstellungen für den bereitgestellten Durchsatz einer Tabelle geändert werden.

```
aws dynamodb update-table --table-name Music \
    --provisioned-throughput ReadCapacityUnits=20,WriteCapacityUnits=10
```

Note

Wenn Sie eine `UpdateTable`-Anforderung ausgeben, ändert sich der Tabellenstatus von `AVAILABLE` in `UPDATING`. Die Tabelle steht vollständig zur Verfügung, während sie `UPDATING`. Wenn dieser Vorgang abgeschlossen ist, ändert sich der Status der Tabelle von `UPDATING` in `AVAILABLE`.

Example

Das folgende AWS CLI Beispiel zeigt, wie der read/write Kapazitätsmodus einer Tabelle in den On-Demand-Modus geändert wird.

```
aws dynamodb update-table --table-name Music \
    --billing-mode PAY_PER_REQUEST
```

Löschen einer Tabelle

Sie können eine ungenutzte Tabelle mit der `DeleteTable`-Operation entfernen. Das Löschen einer Tabelle ist ein unwiederbringlicher Vorgang. Informationen zum Löschen einer Tabelle mit der AWS-Managementkonsole finden Sie unter the section called “Schritt 6: (optional) Aufräumen”.

Example

Das folgende AWS CLI Beispiel zeigt, wie eine Tabelle gelöscht wird.

```
aws dynamodb delete-table --table-name Music
```

Wenn Sie eine `DeleteTable`-Anforderung ausgeben, ändert sich der Tabellenstatus von `ACTIVE` in `DELETING`. Es kann eine Weile dauern die Tabelle zu löschen, abhängig von den verwendeten Ressourcen (z. B. die in der Tabelle gespeicherten Daten und alle Streams oder Indizes der Tabelle).

Wenn die `DeleteTable`-Operation abschließt, ist die Tabelle in DynamoDB nicht mehr vorhanden.

Löschschutz verwenden

Mit der Löschschutz-Eigenschaft können Sie eine Tabelle vor dem versehentlichen Löschen schützen. Wenn Sie diese Eigenschaft für Tabellen aktivieren, können Sie sicherstellen, dass Tabellen nicht versehentlich während regulärer Tabellenverwaltungsvorgängen durch Ihre Administratoren gelöscht werden. Auf diese Weise können Sie Störungen des normalen Geschäftsbetriebs vermeiden.

Der Tabelleneigentümer oder ein autorisierter Administrator steuert die Löschschutz-Eigenschaft für jede Tabelle. Die Löschschutz-Eigenschaft ist bei Tabellen standardmäßig deaktiviert. Dies umfasst globale Replikate und Tabellen, die aus Backups wiederhergestellt wurden. Wenn der Löschschutz für eine Tabelle deaktiviert ist, kann die Tabelle von allen Benutzern gelöscht werden, die durch eine

Identity and Access Management (IAM)-Richtlinie autorisiert wurden. Wenn der Löschschutz für eine Tabelle aktiviert ist, kann die Tabelle von niemandem gelöscht werden.

Wenn Sie diese Einstellung ändern möchten, rufen Sie Zusätzliche Einstellungen für die Tabelle auf, navigieren Sie zum Bereich Löschschutz und wählen Sie Löschschutz aktivieren aus.

Die Löschschutz-Eigenschaft wird von der DynamoDB-Konsole, der API, CLI/SDK und CloudFormation unterstützt. Die API `CreateTable` unterstützt die Löschschutz-Eigenschaft während der Tabellenerstellung und die API `UpdateTable` unterstützt Änderungen an der Löschschutz-Eigenschaft für vorhandene Tabellen.

> **Note**
>
> - Wenn ein AWS Konto gelöscht wird, werden alle Daten dieses Kontos, einschließlich der Tabellen, weiterhin innerhalb von 90 Tagen gelöscht.
> - Wenn DynamoDB keinen Zugriff mehr auf einen kundenverwalteten Schlüssel hat, der zum Verschlüsseln einer Tabelle verwendet wurde, wird die Tabelle trotzdem archiviert. Bei der Archivierung wird ein Backup der Tabelle erstellt und das Original gelöscht.

Auflisten von Tabellennamen

Die `ListTables` Operation gibt die Namen der DynamoDB-Tabellen für das AWS Girokonto und die Region zurück.

Example

Das folgende AWS CLI Beispiel zeigt, wie die DynamoDB-Tabellennamen aufgelistet werden.

```
aws dynamodb list-tables
```

Beschreiben der bereitgestellten Durchsatzkontingente

Der `DescribeLimits` Vorgang gibt die aktuellen Lese- und Schreibkapazitätskontingente für das aktuelle AWS Konto und die Region zurück.

Example

Das folgende AWS CLI Beispiel zeigt, wie die aktuell bereitgestellten Durchsatzquoten beschrieben werden.

```
aws dynamodb describe-limits
```

Die Ausgabe zeigt die oberen Kontingente der Lese- und Schreibkapazitätseinheiten für das aktuelle AWS Konto und die Region.

Weitere Informationen zu diesen Kontingenten und zum Anfordern einer Kontingenterhöhung finden Sie unter Standardkontingente für den Durchsatz.

Überlegungen bei der Auswahl einer Tabellenklasse in DynamoDB

DynamoDB bietet zwei Tabellenklassen an, mit denen Sie Ihre Kosten optimieren können. Die DynamoDB-Standard-Tabellenklasse ist die Standardeinstellung und wird für die große Mehrheit der Workloads empfohlen. Die Tabellenklasse DynamoDB Standard-Infrequent Access (DynamoDB Standard-IA) ist für Tabellen optimiert, in denen Speicher die dominierenden Kosten sind. Zum Beispiel sind Tabellen, die selten aufgerufene Daten speichern, wie Anwendungsprotokolle, alte Social-Media-Posts, E-Commerce-Bestellhistorie und frühere Spielerrungenschaften, gute Kandidaten für die Standard-IA Tabellenklasse.

Jede DynamoDB-Tabelle ist einer Tabellenklasse zugeordnet. Alle der Tabelle zugeordneten sekundären Indizes verwenden dieselbe Tabellenklasse. Sie können Ihre Tabellenklasse beim Erstellen Ihrer Tabelle festlegen (standardmäßig DynamoDB Standard) und die Tabellenklasse einer vorhandenen Tabelle mithilfe der AWS CLI oder AWS des AWS-Managementkonsole SDK aktualisieren. DynamoDB unterstützt auch die Verwaltung Ihrer Tabellenklasse mithilfe von AWS CloudFormation Tabellen mit nur einer Region (Tabellen, die keine globalen Tabellen sind). Jede Tabellenklasse bietet unterschiedliche Preise für die Datenspeicherung sowie für Lese- und Schreibanfragen. Wenn Sie eine Tabellenklasse für Ihre Tabelle auswählen, müssen Sie Folgendes beachten:

- Die DynamoDB Standard-Tabellenklasse bietet niedrigere Durchsatzkosten als DynamoDB Standard-IA und ist die kostengünstigste Option für Tabellen, bei denen der Durchsatz die dominierenden Kosten darstellt.
- Die DynamoDB Standard-IA Tabellenklasse bietet niedrigere Speicherkosten als DynamoDB Standard und ist die kostengünstigste Option für Tabellen, in denen der Speicher die dominierenden Kosten darstellt. Wenn der Speicher 50% der Durchsatzkosten (Lese- und Schreibvorgänge) einer Tabelle unter Verwendung der DynamoDB Standard-Tabellenklasse übersteigt, kann die DynamoDB Standard-IA Tabellenklasse Ihnen dabei helfen, Ihre Gesamttabellenkosten zu senken.

- DynamoDB-Standard-IA-Tabellen bieten die gleiche Leistung, Haltbarkeit und Verfügbarkeit wie DynamoDB-Standardtabellen.
- Der Wechsel zwischen den DynamoDB-Standard- und DynamoDB-Standard-IA Tabellenklassen erfordert keine Änderung Ihres Anwendungscodes. Sie verwenden dieselben DynamoDB APIs - und Service-Endpunkte, unabhängig von der Tabellenklasse, die Ihre Tabellen verwenden.
- DynamoDB-Standard-IA-Tabellen sind mit allen vorhandenen DynamoDB-Funktionen wie Auto Scaling, On-Demand-Modus time-to-live (TTL), On-Demand-Backups, point-in-time Recovery (PITR) und globalen Sekundärindizes kompatibel.

Die kostengünstigste Tabellenklasse für Ihre Tabelle hängt von den erwarteten Speicher- und Durchsatznutzungsmustern Ihrer Tabelle ab. Mit Kosten- und Nutzungsberichten und dem Cost Explorer können Sie sich die historischen Speicher- und AWS Durchsatzkosten und die AWS Nutzung Ihrer Tabelle ansehen. Verwenden Sie diese historischen Daten, um die kostengünstigste Tabellenklasse für Ihre Tabelle zu ermitteln. Weitere Informationen zur Verwendung von AWS Kosten- und Nutzungsberichten und dem AWS Cost Explorer finden Sie in der Dokumentation zu AWS Billing and Cost Management. Siehe Amazon-DynamoDB Preise für weitere Informationen zu Tabellenklassenpreisen.

> **Note**
>
> Eine Tabellenklassenaktualisierung ist ein Hintergrundprozess. Sie können während einer Tabellenklassenktualisierung weiterhin normal auf Ihre Tabelle zugreifen. Die Zeit zum Aktualisieren Ihrer Tabellenklasse hängt von Ihrem Tabellenverkehr, der Speichergröße und anderen verbundenen Variablen ab. In einem zurückliegenden Zeitraum von 30 Tagen sind nicht mehr als zwei Tabellenklassenaktualisierungen in Ihrer Tabelle zulässig.

Hinzufügen von Tags und Labels zu Ressourcen in DynamoDB

Sie können Amazon-DynamoDB-Ressourcen mit Tags markieren. Tags helfen Ihnen, Ihre Ressourcen auf unterschiedliche Weise zu kategorisieren, z. B. nach Zweck, Besitzer, Umgebung oder anderen Kriterien. Tags können Sie bei Folgendem unterstützen:

- Eine Ressource basierend auf den Tags, die Sie ihr zugeordnet haben, schnell zu erkennen.
- Sehen Sie sich AWS Rechnungen an, aufgeschlüsselt nach Stichwörtern.

> **Note**
>
> Alle lokalen sekundären Indizes (LSI) und globalen sekundären Indizes (GSI) im Zusammenhang mit markierten Tabellen werden automatisch mit denselben Tags gekennzeichnet. Derzeit kann die Nutzung des DynamoDB-Streams nicht markiert werden.

Tagging wird von AWS Diensten wie Amazon EC2, Amazon S3, DynamoDB und mehr unterstützt. Effizientes Tagging kann Kosteneinblicke bereitstellen, mit denen Sie Berichte für Services erstellen können, die ein spezifisches Tag aufweisen.

Gehen Sie wie folgt vor, um sich mit dem Taggen vertraut zu machen:

1. Lesen Sie Markierungseinschränkungen in DynamoDB.
2. Tags mit Markieren von Ressourcen in DynamoDB erstellen.
3. Verwenden Sie diese OptionVerwenden von DynamoDB-Tags zum Erstellen von Kostenzuordnungsberichten, um Ihre AWS Kosten pro aktivem Tag nachzuverfolgen.

Schließlich wird empfohlen, optimale Tagging-Strategien zu befolgen. Weitere Informationen finden Sie unter AWS -Markierungsstrategien.

Markierungseinschränkungen in DynamoDB

Jedes Tag besteht aus einem Schlüssel und einem Wert, die Sie beide selbst definieren. Beachten Sie die folgenden Einschränkungen:

- Jede DynamoDB-Tabelle kann nur über ein Tag mit demselben Schlüssel verfügen. Wenn Sie versuchen, ein vorhandenes Tag (derselbe Schlüssel) hinzuzufügen, wird der vorhandene Tag-Wert auf den neuen Wert aktualisiert.
- Bei Tag-Schlüsseln und -Werten muss die Groß- und Kleinschreibung beachtet werden.
- Die maximale Schlüssellänge beträgt 128 Unicode-Zeichen.
- Die maximale Wertlänge beträgt 256 Unicode-Zeichen.
- Namen dürfen Buchstaben, Leerzeichen und Zahlen sowie die folgenden Sonderzeichen enthalten: `+ - = . _ : /`
- Die maximale Anzahl an Tags pro Ressource beträgt 50.

- Die maximale Größe, die für alle Tags in einer Tabelle unterstützt wird, beträgt 10 KB.
- AWS-zugewiesenen Tagnamen und -werten wird automatisch das `aws:` Präfix zugewiesen, das Sie nicht zuweisen können. AWS-zugewiesene Tagnamen werden nicht auf das Tag-Limit von 50 oder das maximale Größenlimit von 10 KB angerechnet. Von Benutzern zugewiesene Tag-Namen haben das Präfix `user:` im Kostenzuordnungsbericht.
- Sie können die Anwendung eines Tags nicht rückdatieren.

Markieren von Ressourcen in DynamoDB

Sie können die Amazon DynamoDB DynamoDB-Konsole oder die AWS Command Line Interface (AWS CLI) verwenden, um Tags hinzuzufügen, aufzulisten, zu bearbeiten oder zu löschen. Anschließend können Sie diese benutzerdefinierten Tags aktivieren, damit sie in der AWS Fakturierung und Kostenmanagement -Konsole zur Nachverfolgung der Kostenzuordnung erscheinen. Weitere Informationen finden Sie unter Verwenden von DynamoDB-Tags zum Erstellen von Kostenzuordnungsberichten.

Für die Massenbearbeitung können Sie auch den Tag-Editor in der AWS-Managementkonsole verwenden. Weitere Informationen finden Sie unter Arbeiten mit dem Tag Editor.

Informationen zum Verwenden der DynamoDB-API stattdessen finden Sie unter den folgenden Operationen in der Amazon-DynamoDB-API-Referenz:

- TagResource
- UntagResource
- ListTagsOfResource

Themen

- Festlegen von Berechtigungen zum Filtern nach Tags
- Hinzufügen von Tags zu neuen oder vorhandenen Tabellen (AWS-Managementkonsole)
- Hinzufügen von Tags zu neuen oder vorhandenen Tabellen (AWS CLI)

Festlegen von Berechtigungen zum Filtern nach Tags

Wenn Sie Tags zum Filtern Ihrer Tabellenliste in der DynamoDB-Konsole verwenden möchten, stellen Sie sicher, dass die Richtlinien der Benutzer Zugriff auf die folgenden Vorgänge enthalten:

- `tag:GetTagKeys`
- `tag:GetTagValues`

Sie können auf diese Vorgänge zugreifen, indem Sie eine neue IAM-Richtlinie an Ihren Benutzer anhängen. Führen Sie dazu die folgenden Schritte aus.

1. Wechseln Sie mit einem Admin-Benutzer zur IAM-Konsole.
2. Wählen Sie im linken Navigationsbereich „Richtlinien“ aus
3. Wählen Sie Richtlinie erstellen.
4. Fügen Sie das folgende Richtliniendokument in den JSON-Editor ein.

   JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "tag:GetTagKeys",
                "tag:GetTagValues"
            ],
            "Resource": "*"
        }
    ]
}
```

5. Schließen Sie den Assistenten ab, und weisen Sie der Richtlinie einen Namen zu (z. B. `TagKeysAndValuesReadAccess`).
6. Wählen Sie im linken Navigationsmenü „Benutzer“ aus.
7. Wählen Sie in der Liste den Benutzer aus, den Sie normalerweise für den Zugriff auf die DynamoDB-Konsole verwenden.
8. Wählen Sie „Berechtigungen hinzufügen“ aus.
9. Wählen Sie die Option Vorhandene Richtlinien direkt anfügen aus.
10. Wählen Sie in der Liste die Richtlinie aus, die Sie zuvor erstellt haben.
11. Schließen Sie den Assistenten ab.

Hinzufügen von Tags zu neuen oder vorhandenen Tabellen (AWS-Managementkonsole)

Sie können die DynamoDB-Konsole verwenden, um neuen Tabellen Tags hinzuzufügen, wenn Sie sie erstellen, oder um Tags für vorhandene Tabellen hinzuzufügen, zu bearbeiten oder zu löschen.

Markieren von Ressourcen bei der Erstellung (Konsole)

1. Melden Sie sich bei der an AWS-Managementkonsole und öffnen Sie die DynamoDB-Konsole unter. https://console.aws.amazon.com/dynamodb/
2. Wählen Sie im Navigationsbereich Tables (Tabellen) und anschließend Create table (Tabelle erstellen) aus.
3. Geben Sie auf der Seite DynamoDB-Tabelle erstellen einen Namen und einen Primärschlüssel ein. Wählen Sie im Abschnitt Tags (Tags) Add new tag (Neuen Tag hinzufügen) und geben Sie die Tags ein, die Sie verwenden möchten.

   Informationen zur Tag-Struktur finden Sie unter Markierungseinschränkungen in DynamoDB.

   Weitere Informationen zum Erstellen von Tabellen finden Sie unter Grundlegende Operationen für DynamoDB-Tabellen.

Markieren vorhandener Ressourcen (Konsole)

Öffnen Sie die DynamoDB-Konsole unter. https://console.aws.amazon.com/dynamodb/

1. Wählen Sie im Navigationsbereich Tables (Tabellen) aus.
2. Wählen Sie eine Tabelle in der Liste aus und wählen Sie dann die Registerkarte Additional settings (Zusätzliche Einstellungen). Sie können Ihre Tags im Abschnitt Tags unten auf der Seite hinzufügen, bearbeiten oder löschen.

Hinzufügen von Tags zu neuen oder vorhandenen Tabellen (AWS CLI)

Die folgenden Beispiele zeigen, wie Sie die verwenden AWS CLI , um Tags anzugeben, wenn Sie Tabellen und Indizes erstellen, und um vorhandene Ressourcen zu taggen.

Markieren von Ressourcen bei der Erstellung (AWS CLI)

- Im folgenden Beispiel wird eine neue `Movies`-Tabelle erstellt und das `Owner`-Tag mit dem Wert `blueTeam` hinzugefügt:

```
aws dynamodb create-table \
    --table-name Movies \
    --attribute-definitions AttributeName=Title,AttributeType=S \
    --key-schema AttributeName=Title,KeyType=HASH \
    --provisioned-throughput ReadCapacityUnits=5,WriteCapacityUnits=5 \
    --tags Key=Owner,Value=blueTeam
```

Markieren vorhandener Ressourcen (AWS CLI)

- Im folgenden Beispiel wird das `Owner`-Tag mit dem Wert `blueTeam` für die Tabelle `Movies` hinzugefügt:

```
aws dynamodb tag-resource \
    --resource-arn arn:aws:dynamodb:us-east-1:123456789012:table/Movies \
    --tags Key=Owner,Value=blueTeam
```

Auflisten aller Tags für eine Tabelle (AWS CLI)

- Im folgenden Beispiel werden alle Tags aufgelistet, die mit der Tabelle `Movies` verknüpft sind:

```
aws dynamodb list-tags-of-resource \
    --resource-arn arn:aws:dynamodb:us-east-1:123456789012:table/Movies
```

Verwenden von DynamoDB-Tags zum Erstellen von Kostenzuordnungsberichten

AWS verwendet Tags, um die Ressourcenkosten in Ihrem Kostenverteilungsbericht zu organisieren. AWS bietet zwei Arten von Kostenzuordnungs-Tags:

- Ein AWS-generiertes Tag. AWS definiert, erstellt und wendet dieses Tag für Sie an.
- Benutzerdefinierte Tags. Sie definieren und erstellen diese Tags und wenden sie an.

Sie müssen beide Arten von Tags separat aktivieren, bevor sie in Cost Explorer oder einem Kostenzuordnungsbericht angezeigt werden können.

Um AWS-generierte Tags zu aktivieren:

1. Melden Sie sich bei der an AWS-Managementkonsole und öffnen Sie die Billing and Cost Management-Konsole von zu https://console.aws.amazon.com/billing/Hause aus#/.
2. Wählen Sie im Navigationsbereich die Option Cost Allocation Tags (Kostenzuordnungs-Tags) aus.
3. Wählen Sie unter AWS-generierte Kostenzuordnungs-Tags die Option Aktivieren.

So aktivieren Sie benutzerdefinierte Tags:

1. Melden Sie sich bei der an AWS-Managementkonsole und öffnen Sie die Billing and Cost Management-Konsole von zu https://console.aws.amazon.com/billing/Hause aus#/.
2. Wählen Sie im Navigationsbereich die Option Cost Allocation Tags (Kostenzuordnungs-Tags) aus.
3. Wählen Sie unter User-Defined Cost Allocation Tags (Benutzerdefinierte Kostenzuordnungs-Tags) die Option Activate (Aktivieren) aus.

Nachdem Sie Stichwörter erstellt und aktiviert haben, AWS wird ein Kostenzuordnungsbericht generiert, in dem Ihre Nutzung und die Kosten nach Ihren aktiven Stichwörtern gruppiert sind. Der Kostenzuordnungsbericht enthält alle Ihre AWS Kosten für jeden Abrechnungszeitraum. Der Bericht enthält getaggte und nicht getaggte Ressourcen, sodass Sie die Gebühren für Ressourcen übersichtlich verwalten können.

> **Note**
>
> Derzeit werden alle Daten, die aus DynamoDB weitergeleitet werden, im Kostenzuordnungsbericht nicht nach Tags aufgeschlüsselt.

Weitere Informationen finden Sie unter Verwendung von Kostenzuordnungs-Tags.

Globale Tabellen: multiaktive, regionsübergreifende Replikation

Globale Tabellen in Amazon DynamoDB sind ein vollständig verwaltetes Feature für multiregionale und multiaktive Datenbanken, das sich durch eine benutzerfreundliche Datenbankreplikation und schnelle und lokale Lese- und Schreibvorgänge in globale Anwendungen auszeichnet.

Globale Tabellen replizieren Ihre DynamoDB-Tabellendaten automatisch kontenübergreifend AWS-Regionen und optional AWS kontenübergreifend, ohne dass Sie Ihre eigene Replikationslösung erstellen und verwalten müssen. Globale Tabellen eignen sich ideal für Anwendungen, die in

mehreren Regionen bereitgestellt werden und Geschäftskontinuität sowie eine hohe Verfügbarkeit erfordern. Jedes Replikat einer globalen Tabelle kann Lese- und Schreibvorgänge ausführen. Anwendungen können eine hohe Ausfallsicherheit mit niedrigen oder ganz ohne RPO (Recovery Point Objective) erreichen, weil der Datenverkehr in eine andere Region verlagert wird, sobald die Verarbeitung durch die Anwendung in einer Region unterbrochen wird. Globale Tabellen sind in allen Regionen verfügbar, in denen auch DynamoDB verfügbar ist.

Konsistenzmodi

Sie können bei der Erstellung einer globalen Tabelle den Konsistenzmodus konfigurieren. Globale Tabellen unterstützen zwei Konsistenzmodi: Multi-Region Eventual Consistency (MREC) und Multi-Region Strong Consistency (MRSC).

Wenn Sie beim Erstellen einer globalen Tabelle keinen Konsistenzmodus angeben, wird für die globale Tabelle standardmäßig der MREC-Modus verwendet. Replikate, die mit unterschiedlichen Konsistenzmodi konfiguriert wurden, sind in einer globalen Tabelle nicht zulässig. Sie können den Konsistenzmodus einer globalen Tabelle nach der Erstellung nicht mehr ändern.

Kontokonfigurationen

DynamoDB unterstützt jetzt zwei globale Tabellenmodelle, die jeweils für unterschiedliche Architekturmuster konzipiert wurden:

- Globale Tabellen mit demselben Konto — Alle Replikate werden in einem einzigen Konto erstellt und verwaltet. AWS
- Globale Tabellen mit mehreren Konten — Replikate werden für mehrere AWS Konten bereitgestellt und sind gleichzeitig Teil einer gemeinsamen Replikationsgruppe.

Sowohl Modelle mit demselben Konto als auch mit mehreren Konten unterstützen Schreibvorgänge in mehreren Regionen, asynchrone Replikation, last-writer-wins Konfliktlösung und dasselbe Abrechnungsmodell. Sie unterscheiden sich jedoch darin, wie Konten, Berechtigungen, Verschlüsselung und Tabellen-Governance verwaltet werden.

Für MRSC konfigurierte globale Tabellen unterstützen nur Konfigurationen mit demselben Konto.

Sie können eine globale Tabelle mit der Management Console konfigurieren. AWS Globale Tabellen verwenden vorhandene DynamoDB APIs zum Lesen und Schreiben von Daten in Ihre Tabellen, sodass keine Anwendungsänderungen erforderlich sind. Sie zahlen ohne Vorauszahlungen oder Verpflichtungen nur für bereitgestellte oder verwendete Ressourcen.

Vergleich von globalen Tabellen mit demselben Konto und mehreren Konten

Eigenschaften	Globale Tabellen für dasselbe Konto	Globale Tabellen mit mehreren Konten
Primärer Anwendungsfall	Resilienz in mehreren Regionen für Anwendung en innerhalb eines einzigen Kontos AWS	Replikation für mehrere Regionen und Konten für Anwendungen, die unterschi edlichen Teams, unterschi edlichen Geschäftseinheiten oder starken Sicherhei tsgrenzen zwischen Konten gehören
Kontomodell	Alle Repliken wurden in einem AWS Konto erstellt und verwaltet	Replikate, die für mehrere AWS Konten innerhalb derselben Bereitstellung erstellt wurden
Besitz von Ressourcen	Ein einzelnes Konto besitzt die Tabelle und alle Replikate	Jedes Konto besitzt sein lokales Replikat; die Replikati onsgruppe umfasst mehrere Konten
Unterstützte Version	Version 2019.11.21 (aktuell) und Version 2017.11.29 (Legacy)	Version der globalen Tabellen 2019.11.21 (Aktuell)
Betrieb der Kontrollebene	Erstellen, ändern und löschen Sie Replikate über das Konto des Tabellenbesitzers	Operationen auf verteilter Steuerungsebene: Konten treten der Replikationsgruppe bei oder verlassen sie
Operationen auf Datenebene	DynamoDB-Standarde ndpunkte pro Region	Zugriff auf Datenebene pro Konto/Region; Routing über Replikationsgruppe

Eigenschaften	Globale Tabellen für dasselbe Konto	Globale Tabellen mit mehreren Konten
Sicherheitsgrenze	Eine einzige IAM- und KMS-Grenze	Separate IAM-, KMS-, Fakturierungs- und CloudTrail Verwaltungsfunktionen für jedes Konto
Beste Passform	Organizations mit zentralisiertem Besitz von Tabellen	Organizations mit Verbundteams, Verwaltungsgrenzen oder Konfigurationen mit mehreren Konten

Themen

- Kernkonzepte globaler Tabellen
- DynamoDB-Tabelle für dasselbe Konto
- DynamoDB-Tabellen für mehrere Konten
- Grundlegendes zur Amazon DynamoDB-Fakturierung für globale Tabellen
- Versionen von globalen DynamoDB-Tabellen
- Bewährte Methoden für globale Tabellen

Kernkonzepte globaler Tabellen

In den folgenden Abschnitten werden die Konzepte und das Verhalten globaler Tabellen in Amazon DynamoDB beschrieben.

Konzepte

Globale Tabellen sind eine DynamoDB-Funktion, die Tabellendaten regionsübergreifend repliziert. AWS

Eine Replikattabelle (oder kurz ein Replikat) ist eine einzelne DynamoDB-Tabelle, die als Teil einer globalen Tabelle fungiert. Eine globale Tabelle besteht aus zwei oder mehr Replikattabellen in verschiedenen Regionen. AWS Jede globale Tabelle kann nur über eine Replikattabelle je AWS -Region verfügen. Alle Replikate in einer globalen Tabelle verwenden denselben Tabellennamen, dasselbe Primärschlüsselschema und dieselben Elementdaten.

Wenn eine Anwendung Daten in eine Replikattabelle in einer Region schreibt, verteilt DynamoDB den Schreibvorgang automatisch auf die anderen Replikattabellen in der globalen Tabelle. Weitere Informationen zu den ersten Schritten mit globalen Tabellen finden Sie unter Tutorials: Erstellen von globalen Tabellen oderTutorials: Globale Tabellen mit mehreren Konten erstellen.

Versionen

Es gibt zwei Versionen von globalen DynamoDB-Tabellen: Version 2019.11.21 (Aktuell) und Version 2017.11.29 (Legacy) der globalen Tabellen. Sie sollten nach Möglichkeit Global Tables Version 2019.11.21 (Aktuell) verwenden. Die Informationen in diesem Abschnitt der Dokumentation gelten für die Version 2019.11.21 (Aktuell). Weitere Informationen finden Sie unter Ermitteln der Version einer globalen Tabelle. Ermitteln der Version einer globalen Tabelle

Verfügbarkeit

Globale Tabellen tragen zur Verbesserung der Geschäftskontinuität bei, da sie die Implementierung einer hoch verfügbaren Architektur für mehrere Regionen erleichtern. Wenn eine Arbeitslast in einer einzelnen AWS Region beeinträchtigt wird, können Sie den Anwendungsdatenverkehr in eine andere Region verlagern und Lese- und Schreibvorgänge in einer anderen Replikattabelle in derselben globalen Tabelle durchführen.

Jede Replikattabelle in einer globalen Tabelle bietet dieselbe Stabilität und Verfügbarkeit wie eine DynamoDB-Tabelle mit einer Region. Globale Tabellen bieten ein Service Level Agreement (SLA) mit einer Verfügbarkeit von 99,999 %, im Gegensatz zur Verfügbarkeit von 99,99 % bei Tabellen mit einer einzelnen Region.

Testen mit Fehlerinjektionen

Sowohl die globalen MREC- als auch die MRSC-Tabellen sind in den AWSAWS Fault Injection Service (FIS) integriert, einen vollständig verwalteten Dienst zur Durchführung kontrollierter Fehlerinjektionsexperimente zur Verbesserung der Ausfallsicherheit einer Anwendung. Mit AWS FIS können Sie:

- Erstellen Sie Versuchsvorlagen, die spezifische Fehlerszenarien definieren.
- Injizieren Sie Fehler, um die Ausfallsicherheit von Anwendungen zu überprüfen, indem Sie die Regionsisolierung simulieren (d. h. die Replikation zu und von einem ausgewählten Replikat unterbrechen), um die Fehlerbehandlung, Wiederherstellungsmechanismen und das Verhalten bei Störungen in einer Region zu testen. AWS

In einer globalen Tabelle mit Replikaten in USA Ost (Nord-Virginia), USA Ost (Ohio) und USA West (Oregon) können Sie beispielsweise ein Experiment in USA Ost (Ohio) durchführen, um die dortige Regionsisolierung zu testen, während USA Ost (Nord-Virginia) und USA West (Oregon) ihren normalen Betrieb fortsetzen. Diese kontrollierten Tests helfen Ihnen dabei, potenzielle Probleme zu identifizieren und zu lösen, bevor sie sich auf den Produktions-Workload auswirken.

Eine vollständige Liste der von AWS FIS unterstützten Aktionen und regionsübergreifender Konnektivität zum Anhalten der DynamoDB-Replikation zwischen Regionen finden Sie unter Aktionsziele im AWS FIS-Benutzerhandbuch.

Informationen zu globalen Tabellenaktionen von Amazon DynamoDB, die in AWS FIS verfügbar sind, finden Sie unter DynamoDB Global Tables Actions Reference im FIS-Benutzerhandbuch.AWS

Informationen zu den ersten Schritten mit der Durchführung von Fault-Injection-Experimenten finden Sie unter Planung Ihrer AWS FIS-Experimente im FIS-Benutzerhandbuch. AWS

> **Note**
>
> Bei AWS FIS Experimenten in MRSC sind eventuell konsistente Lesevorgänge zulässig, Aktualisierungen der Tabelleneinstellungen — wie z. B. das Ändern des Abrechnungsmodus oder die Konfiguration des Tabellendurchsatzes — sind jedoch nicht zulässig, ähnlich wie bei MREC. Weitere Informationen `FaultInjectionServiceInducedErrors`zum Fehlercode finden Sie in der CloudWatch Metrik.

Time to Live (TTL)

In globalen Tabellen mit MREC-Konfiguration kann die Konfiguration von Time To Live (TTL) gelöscht werden. Die TTL-Einstellungen werden automatisch für alle Replikate einer globalen Tabelle synchronisiert. Wenn TTL ein Element aus einem Replikat in einer Region löscht, wird der Löschvorgang auf alle anderen Replikate in der globalen Tabelle repliziert. TTL verbraucht keine Schreibkapazität; das Löschen von TTL wird Ihnen in der Region, in der der Löschvorgang stattgefunden hat, nicht in Rechnung gestellt. In allen anderen Regionen, deren globale Tabelle ein Replikat enthält, wird Ihnen jedoch der replizierte Löschvorgang in Rechnung gestellt.

Die Replikation eines TTL-Löschvorgangs verbraucht Schreibkapazität in den Replikaten, in die der Löschvorgang repliziert wird. Replikate, die für bereitgestellte Kapazität konfiguriert sind, können Anfragen drosseln, wenn die Kombination aus Schreibdurchsatz und TTL-Löschdurchsatz die bereitgestellte Schreibkapazität überschreitet.

Globale Tabellen mit MRSC-Konfiguration unterstützen die Konfiguration des TTL-Löschvorgangs nicht.

Streams

Globale Tabellen mit MREC-Konfiguration replizieren Änderungen, indem sie diese aus einem DynamoDB-Stream in einer Replikattabelle auslesen und auf alle anderen Replikattabellen anwenden. Streams sind daher standardmäßig für alle Replikate in einer globalen MREC-Tabelle aktiviert und können in diesen Replikaten nicht deaktiviert werden. Der MREC-Replikationsprozess kann mehrere Änderungen in einem kurzen Zeitraum zu einem einzigen replizierten Schreibvorgang kombinieren, was dazu führt, dass der Stream jedes Replikats Datensätze enthält, die sich geringfügig unterscheiden. Streams-Datensätze auf MREC-Replikaten behalten die Reihenfolge für alle Änderungen am selben Objekt bei, aber die relative Reihenfolge der Änderungen an verschiedenen Elementen kann je nach Replikat variieren.

Wenn Sie eine Anwendung schreiben möchten, die Streams-Datensätze für Änderungen verarbeitet, die in einer bestimmten Region, aber nicht in anderen Regionen in einer globalen Tabelle vorgenommen wurden, können Sie jedem Element ein Attribut hinzufügen, das definiert, in welcher Region die Änderung für dieses Element stattgefunden hat. Sie können dieses Attribut verwenden, um Streams-Datensätze nach Änderungen zu filtern, die in anderen Regionen aufgetreten sind und hierzu auch Lambda-Ereignisfilter verwenden, um Lambda-Funktionen nur für Änderungen in einer bestimmten Region aufzurufen.

Globale Tabellen mit MRSC-Konfiguration verwenden keine DynamoDB Streams für die Replikation, sodass Streams in MRSC-Replikaten standardmäßig nicht aktiviert sind. Streams können in einem MRSC-Replikat aktiviert werden. Streams-Datensätze in MRSC-Replikaten sind für jedes Replikat identisch, auch in Bezug auf ihre Reihenfolge.

Transaktionen

In einer globalen Tabelle mit MREC-Konfiguration sind DynamoDB-Transaktionsoperationen (`TransactWriteItems` und `TransactGetItems`) nur innerhalb der Region, in der der Vorgang aufgerufen wurde, unteilbar. Schreibtransaktionen werden nicht als Einheit regionsübergreifend repliziert, was bedeutet, dass nur einige Schreibvorgänge in einer Transaktion zu einem bestimmten Zeitpunkt durch Lesevorgänge in anderen Replikaten zurückgegeben werden können.

Wenn Sie beispielsweise eine globale Tabelle mit Replikaten in den Regionen USA Ost (Ohio) und USA West (Oregon) nutzen und eine `TransactWriteItems`-Operation in der Region USA Ost (Ohio) durchführen, sind unter Umständen partiell durchgeführte Transaktionen in der Region USA

West (Oregon) zu beobachten, während die Änderungen repliziert werden. Die Änderungen werden erst in die anderen Regionen repliziert, nachdem sie in der Quellregion in die Datenbank eingetragen wurden.

Globale Tabellen mit MRSC-Konfiguration unterstützen keine Transaktionsvorgänge und geben einen Fehler zurück, wenn solche Operationen in einem MRSC-Replikat aufgerufen werden.

Lese- und Schreibdurchsatz

Modus bereitgestellter Kapazität

Eine Replikation verbraucht Schreibkapazität. Replikate, die für bereitgestellte Kapazität konfiguriert sind, können Anfragen drosseln, wenn die Kombination aus Anwendungsschreibdurchsatz und Replikationsschreibdurchsatz die bereitgestellte Schreibkapazität überschreitet. Bei globalen Tabellen, die den Bereitstellungsmodus verwenden, werden die Auto Scaling-Einstellungen für Lese- und Schreibkapazitäten zwischen den Replikaten synchronisiert.

Sie können die Lesekapazitätseinstellungen für jedes Replikat in einer globalen Tabelle unabhängig konfigurieren, indem Sie den `ProvisionedThroughputOverride`Parameter auf Replikatebene verwenden. Standardmäßig werden Änderungen an der bereitgestellten Lesekapazität auf alle Replikate in der globalen Tabelle angewendet. Beim Hinzufügen eines neuen Replikats zu einer globalen Tabelle wird die Lesekapazität der Quelltabelle oder des Replikats als Anfangswert verwendet, sofern nicht ausdrücklich eine Überschreibung auf Replikatebene angegeben ist.

On-Demand-Modus

Bei globalen Tabellen, die für den On-Demand-Modus konfiguriert sind, wird die Schreibkapazität automatisch über alle Replikate hinweg synchronisiert. DynamoDB passt die Kapazität automatisch an den Datenverkehr an, und es müssen keine replikatspezifischen Lese- oder Schreibkapazitätseinstellungen verwaltet werden.

Überwachen globaler Tabellen

In globalen Tabellen, die für Multi-Region Eventual Consistency (MREC) konfiguriert sind, wird die Metrik veröffentlicht. `ReplicationLatency` CloudWatch Diese Metrik misst die verstrichene Zeit zwischen dem Schreiben eines Elements in eine Replikattabelle und dem Erscheinen dieses Elements in einem anderen Replikat in der globalen Tabelle. `ReplicationLatency` wird in Millisekunden ausgedrückt und für jedes Paar aus Quell- und Zielregion ausgegeben.

Typische `ReplicationLatency` Werte hängen von der Entfernung zwischen den ausgewählten AWS Regionen sowie von anderen Variablen wie Workload-Typ und Durchsatz ab. Beispiel: Ein

Quellreplikat in der Region USA West (Nordkalifornien) (us-west-1) weist im Vergleich zur Region Afrika (Kapstadt) (af-south-1) einen niedrigeren Wert für `ReplicationLatency` auf als die Region USA West (Oregon) (us-west-2).

Ein erhöhter Wert für `ReplicationLatency` könnte darauf hinweisen, dass Updates von einem Replikat nicht in einem angemessenen Zeitraum an andere Replikattabellen verteilt werden. In diesem Fall können Sie die Lese- und Schreibaktivität Ihrer Anwendung vorübergehend in eine andere AWS Region umleiten.

Globale Tabellen mit MRSC-Konfiguration veröffentlichen keine `ReplicationLatency`-Metrik.

Überlegungen zur Verwaltung globaler Tabellen

Sie können eine Tabelle, die zum Hinzufügen des neuen Replikats einer globalen Tabelle verwendet wurde, erst 24 Stunden nach der Erstellung des neuen Replikats löschen.

Wenn Sie eine AWS Region deaktivieren, die globale Tabellenreplikate enthält, werden diese Replikate 20 Stunden nach der Deaktivierung der Region dauerhaft in Tabellen mit nur einer Region konvertiert.

DynamoDB-Tabelle für dasselbe Konto

Globale Tabellen mit demselben Konto replizieren Ihre DynamoDB-Tabellendaten automatisch regionsübergreifend innerhalb AWS eines einzigen Kontos. AWS Globale Tabellen mit demselben Konto bieten das einfachste Modell für die Ausführung von Anwendungen mit mehreren Regionen, da alle Replikate dasselbe Kontogrenzen-, Eigentums- und Berechtigungsmodell verwenden. Wenn Sie die AWS Regionen für Ihre Replikattabellen auswählen, übernehmen globale Tabellen die gesamte Replikation automatisch. Globale Tabellen sind in allen Regionen verfügbar, in denen auch DynamoDB verfügbar ist.

Globale Tabellen mit demselben Konto bieten die folgenden Vorteile:

- Automatisches Replizieren von DynamoDB-Tabellendaten in allen AWS Regionen Ihrer Wahl, um Daten näher an Ihren Benutzern zu finden
- Höhere Anwendungsverfügbarkeit bei regionaler Isolierung oder Auslagerung von Daten
- Verwenden Sie die integrierte Konfliktlösung, damit Sie sich auf die Geschäftslogik Ihrer Anwendung konzentrieren können
- Wenn Sie eine globale Tabelle für dasselbe Konto erstellen, können Sie entweder oder wählen Multi-Region Eventual Consistency (MREC) Multi-Region Strong Consistency (MRSC)

Themen

- So funktionieren globale DynamoDB-Tabellen
- Tutorials: Erstellen von globalen Tabellen
- Sicherheit von globalen DynamoDB-Tabellen

So funktionieren globale DynamoDB-Tabellen

In den folgenden Abschnitten werden die Konzepte und das Verhalten globaler Tabellen in Amazon DynamoDB beschrieben.

Konzepte

Globale Tabellen sind eine DynamoDB-Funktion, die Tabellendaten regionsübergreifend repliziert. AWS

Eine Replikattabelle (oder kurz ein Replikat) ist eine einzelne DynamoDB-Tabelle, die als Teil einer globalen Tabelle fungiert. Eine globale Tabelle besteht aus zwei oder mehr Replikattabellen in verschiedenen Regionen. AWS Jede globale Tabelle kann nur über eine Replikattabelle je AWS -Region verfügen. Alle Replikate in einer globalen Tabelle verwenden denselben Tabellennamen, dasselbe Primärschlüsselschema und dieselben Elementdaten.

Wenn eine Anwendung Daten in eine Replikattabelle in einer Region schreibt, verteilt DynamoDB den Schreibvorgang automatisch auf die anderen Replikattabellen in der globalen Tabelle. Weitere Informationen über die ersten Schritte mit globalen Tabellen finden Sie unter Tutorials: Erstellen von globalen Tabellen.

Versionen

Es gibt zwei Versionen von globalen DynamoDB-Tabellen: Version 2019.11.21 (Aktuell) und Version 2017.11.29 (Legacy). Sie sollten nach Möglichkeit die Version 2019.11.21 (Aktuell) verwenden. Die Informationen in diesem Abschnitt der Dokumentation gelten für die Version 2019.11.21 (Aktuell). Weitere Informationen finden Sie unter Ermitteln der Version einer globalen Tabelle.

Verfügbarkeit

Globale Tabellen tragen zur Verbesserung der Geschäftskontinuität bei, da sie die Implementierung einer hoch verfügbaren Architektur für mehrere Regionen erleichtern. Wenn eine Arbeitslast in einer einzelnen AWS Region beeinträchtigt wird, können Sie den Anwendungsdatenverkehr in eine andere

Region verlagern und Lese- und Schreibvorgänge in einer anderen Replikattabelle in derselben globalen Tabelle durchführen.

Jede Replikattabelle in einer globalen Tabelle bietet dieselbe Stabilität und Verfügbarkeit wie eine DynamoDB-Tabelle mit einer Region. Globale Tabellen bieten ein Service Level Agreement (SLA) mit einer Verfügbarkeit von 99,999 %, im Gegensatz zur Verfügbarkeit von 99,99 % bei Tabellen mit einer einzelnen Region.

Konsistenzmodi

Sie können bei der Erstellung einer globalen Tabelle den Konsistenzmodus konfigurieren. Globale Tabellen unterstützen zwei Konsistenzmodi: Multi-Region Eventual Consistency (MREC) und Multi-Region Strong Consistency (MRSC).

Wenn Sie beim Erstellen einer globalen Tabelle keinen Konsistenzmodus angeben, wird für die globale Tabelle standardmäßig der MREC-Modus verwendet. Replikate, die mit unterschiedlichen Konsistenzmodi konfiguriert wurden, sind in einer globalen Tabelle nicht zulässig. Sie können den Konsistenzmodus einer globalen Tabelle nach der Erstellung nicht mehr ändern.

Multi-Region Eventual Consistency (MREC)

Der Standardmodus für globale Tabellen ist der MREC-Modus (Multi-Region Eventual Consistency). Elementänderungen im Replikat einer globalen MREC-Tabelle werden asynchron für alle anderen Replikate repliziert, normalerweise innerhalb von maximal einer Sekunde. In dem unwahrscheinlichen Fall, dass ein Replikat in einer globalen MREC-Tabelle isoliert ist oder beeinträchtigt wird, werden alle Daten, die noch nicht in andere Regionen repliziert wurden, repliziert, sobald das Replikat fehlerfrei ist.

Wenn dasselbe Element in mehreren Regionen gleichzeitig geändert wird, löst DynamoDB den Konflikt, indem die Änderung mit dem neuesten internen Zeitstempel pro Element verwendet wird; diese Konfliktlösungsmethode wird als „Last Writer Wins“ bezeichnet. Letztendlich stimmt ein Element in allen Replikaten mit der Version überein, die beim letzten Schreibvorgang erstellt wurde.

Strikt konsistente Lesevorgänge geben die jeweils aktuelle Version eines Elements zurück, wenn dieses zuletzt in der Region aktualisiert wurde, in der der Lesevorgang stattgefunden hat. Hierbei können jedoch veraltete Daten zurückgegeben werden, wenn das Element zuletzt in einer anderen Region aktualisiert wurde. Bei bedingten Schreibvorgängen wird der Bedingungsausdruck mit der Version des Elements in der Region verglichen.

Sie erstellen eine globale MREC-Tabelle, indem Sie einer vorhandenen DynamoDB-Tabelle ein Replikat hinzufügen. Das Hinzufügen eines Replikats hat keine Auswirkungen auf die Leistung

vorhandener DynamoDB-Tabellen mit einer einzigen Region oder von Replikaten globaler Tabellen. Sie können einer globalen MREC-Tabelle Replikate hinzufügen, um die Anzahl der Regionen zu erhöhen, in denen Daten repliziert werden, oder Replikate aus einer globalen MREC-Tabelle entfernen, wenn sie nicht mehr benötigt werden. Eine globale MREC-Tabelle kann ein Replikat in jeder Region haben, in der DynamoDB verfügbar ist, und sie kann so viele Replikate enthalten, wie es Regionen in der AWS -Partition gibt.

Multi-Region Strong Consistency (MRSC)

Sie können bei der Erstellung einer globalen Tabelle den MRSC-Modus (Multi-Region Strong Consistency) konfigurieren. Elementänderungen im Replikat einer globalen MRSC-Tabelle werden synchron in mindestens eine andere Region repliziert, bevor der Schreibvorgang eine erfolgreiche Antwort zurückgibt. Strikt konsistente Lesevorgänge für ein beliebiges MRSC-Replikat geben immer die neueste Version eines Elements zurück. Bei bedingten Schreibvorgängen wird immer der Bedingungsausdruck mit der aktuellen Version des Elements verglichen.

Eine globale MRSC-Tabelle muss in genau drei Regionen bereitgestellt werden. Sie können eine globale MRSC-Tabelle mit drei Replikaten oder zwei Replikaten und einem Witness konfigurieren. Ein Zeuge ist eine Komponente einer globalen MRSC-Tabelle, die in globale Tabellenreplikate geschriebene Daten enthält und eine optionale Alternative zu einem vollständigen Replikat darstellt und gleichzeitig die Verfügbarkeitsarchitektur von MRSC unterstützt. Sie können für einen Witness keine Lese- oder Schreibvorgänge ausführen. Ein Witness befindet sich in einer anderen Region als die beiden Replikate. Wenn Sie eine globale MRSC-Tabelle erstellen, wählen Sie bei ihrer Erstellung die Regionen sowohl für die Bereitstellung der Replikate als auch für den Witness aus. Sie können anhand der Ausgabe der `DescribeTable`-API bestimmen, ob und in welcher Region für eine globale MRSC-Tabelle ein Witness konfiguriert ist. Der Zeuge gehört DynamoDB und wird von DynamoDB verwaltet, und der Zeuge wird nicht in Ihrem AWS Konto in der Region angezeigt, in der er konfiguriert ist.

Die globalen MRSC-Tabellen sind in den folgenden Regionsgruppen verfügbar: US-Regionengruppe (USA Ost Nord-Virginia, USA Ost-Ohio, USA West Oregon), EU-Regionengruppe (Europa Irland, Europa London, Europa Paris, Europa Frankfurt) und AP-Regionengruppe (Asien-Pazifik Tokio, Asien-Pazifik Seoul und Asien-Pazifik Osaka). Globale MRSC-Tabellen dürfen sich nicht über verschiedene Region-Sets erstrecken (z. B. darf eine globale MRSC-Tabelle nicht gleichzeitig Replikate aus US- und EU-Regionen enthalten).

Sie erstellen eine globale MRSC-Tabelle, indem Sie einer vorhandenen DynamoDB-Tabelle, die keine Daten enthält, ein Replikat und einen Witness oder zwei Replikate hinzufügen. Wenn Sie eine bestehende Tabelle mit einer einzigen Region in eine globale MRSC-Tabelle konvertieren,

müssen Sie sicherstellen, dass die Tabelle leer ist. Die Konvertierung einer Tabelle mit nur einer Region in eine globale MRSC-Tabelle mit vorhandenen Elementen wird nicht unterstützt. Stellen Sie sicher, dass während des Konvertierungsvorgangs keine Daten in die Tabelle geschrieben werden. Sie können einer vorhandenen globalen MRSC-Tabelle keine zusätzlichen Replikate hinzufügen. Ein einzelnes Replikat oder ein Witness kann nicht aus einer globalen MRSC-Tabelle gelöscht werden. Sie können zwei Replikate oder ein Replikat und einen Witness aus einer globalen MRSC-Tabelle löschen und das verbleibende Replikat in eine DynamoDB-Tabelle mit einer einzigen Region umwandeln.

Bei einem Schreibvorgang tritt der Fehler `ReplicatedWriteConflictException` auf, wenn versucht wird, ein Element zu ändern, das bereits in einer anderen Region geändert wird. Schreibvorgänge, die mit dem fehlschlagen, `ReplicatedWriteConflictException` können wiederholt werden und sind erfolgreich, wenn das Element in einer anderen Region nicht mehr geändert wird.

Für globale MRSC-Tabellen gelten die folgenden Überlegungen:

- Die Funktion „Time to Live“ (TTL) wird von globalen MRSC-Tabellen nicht unterstützt.
- Lokale sekundäre Indizes (LSIs) werden für globale MRSC-Tabellen nicht unterstützt.
- CloudWatch Contributor Insights-Informationen werden nur für die Region gemeldet, in der ein Vorgang stattgefunden hat.

Auswählen eines Konsistenzmodus

Das Hauptkriterium für die Auswahl eines Konsistenzmodus für mehrere Regionen ist, ob die Anwendung Schreibvorgängen mit geringer Latenz und strikt konsistente Lesevorgänge oder der globalen strikten Konsistenz Priorität einräumt.

Globale MREC-Tabellen weisen im Vergleich zu globalen MRSC-Tabellen Schreib- und strikt konsistente Lesevorgänge mit geringen Latenzen auf. Globale MREC-Tabellen haben ein Recovery Point Objective (RPO), das der Replikationsverzögerung zwischen Replikaten entspricht, die je nach Replikatregion in der Regel einige Sekunden beträgt.

In folgenden Fällen sollten Sie den MREC-Modus verwenden:

- Ihre Anwendung kann veraltete Daten tolerieren, die von strikt konsistenten Lesevorgängen zurückgegeben werden, wenn diese Daten in einer anderen Region aktualisiert wurden.
- Schreib- und strikt konsistente Lesevorgänge mit geringer Latenz haben Vorrang vor der Konsistenz von multiregionalen Lesevorgängen.

- Gemäß Ihrer Strategie für Hochverfügbarkeit für mehrere Regionen ist ein RPO über null zulässig.

Globale MRSC-Tabellen weisen im Vergleich zu globalen MREC-Tabellen Schreib- und strikt konsistente Lesevorgänge mit höheren Latenzen auf. Globale MRSC-Tabellen unterstützen einen Recovery Point Objective (RPO) von null.

In folgenden Fällen sollten Sie den MRSC-Modus verwenden:

- Sie benötigen strikt konsistente Lesevorgänge über mehrere Regionen hinweg.
- Die globale Konsistenz von Lesevorgängen hat Vorrang gegenüber Schreibvorgängen mit geringer Latenz.
- Gemäß Ihrer Strategie für Hochverfügbarkeit für mehrere Regionen ist ein RPO von null erforderlich.

Überwachen globaler Tabellen

In globalen Tabellen, die für Multi-Region Eventual Consistency (MREC) konfiguriert sind, wird die Metrik veröffentlicht. `ReplicationLatency` CloudWatch Diese Metrik misst die verstrichene Zeit zwischen dem Schreiben eines Elements in eine Replikattabelle und dem Erscheinen dieses Elements in einem anderen Replikat in der globalen Tabelle. `ReplicationLatency` wird in Millisekunden ausgedrückt und für jedes Paar aus Quell- und Zielregion ausgegeben.

Typische `ReplicationLatency` Werte hängen von der Entfernung zwischen den ausgewählten AWS Regionen sowie von anderen Variablen wie Workload-Typ und Durchsatz ab. Beispiel: Ein Quellreplikat in der Region USA West (Nordkalifornien) (us-west-1) weist im Vergleich zur Region Afrika (Kapstadt) (af-south-1) einen niedrigeren Wert für `ReplicationLatency` auf als die Region USA West (Oregon) (us-west-2).

Ein erhöhter Wert für `ReplicationLatency` könnte darauf hinweisen, dass Updates von einem Replikat nicht in einem angemessenen Zeitraum an andere Replikattabellen verteilt werden. In diesem Fall können Sie die Lese- und Schreibaktivität Ihrer Anwendung vorübergehend in eine andere AWS Region umleiten.

Globale Tabellen mit MRSC-Konfiguration veröffentlichen keine `ReplicationLatency`-Metrik.

Testen mit Fehlerinjektionen

Sowohl die globalen MREC- als auch die MRSC-Tabellen sind in den AWSAWS Fault Injection Service (FIS) integriert, einen vollständig verwalteten Dienst zur Durchführung kontrollierter

Fehlerinjektionsexperimente zur Verbesserung der Ausfallsicherheit einer Anwendung. Mit AWS FIS können Sie:

- Erstellen Sie Versuchsvorlagen, die spezifische Fehlerszenarien definieren.
- Injizieren Sie Fehler, um die Ausfallsicherheit von Anwendungen zu überprüfen, indem Sie die Regionsisolierung simulieren (d. h. die Replikation zu und von einem ausgewählten Replikat unterbrechen), um die Fehlerbehandlung, Wiederherstellungsmechanismen und das Verhalten bei Störungen in einer Region zu testen. AWS

In einer globalen Tabelle mit Replikaten in USA Ost (Nord-Virginia), USA Ost (Ohio) und USA West (Oregon) können Sie beispielsweise ein Experiment in USA Ost (Ohio) durchführen, um die dortige Regionsisolierung zu testen, während USA Ost (Nord-Virginia) und USA West (Oregon) ihren normalen Betrieb fortsetzen. Diese kontrollierten Tests helfen Ihnen dabei, potenzielle Probleme zu identifizieren und zu lösen, bevor sie sich auf den Produktions-Workload auswirken.

Eine vollständige Liste der von AWS FIS unterstützten Aktionen und regionsübergreifender Konnektivität zum Anhalten der DynamoDB-Replikation zwischen Regionen finden Sie unter Aktionsziele im AWS FIS-Benutzerhandbuch.

Informationen zu globalen Tabellenaktionen von Amazon DynamoDB, die in AWS FIS verfügbar sind, finden Sie unter DynamoDB Global Tables Actions Reference im FIS-Benutzerhandbuch.AWS

Informationen zu den ersten Schritten mit der Durchführung von Fault-Injection-Experimenten finden Sie unter Planung Ihrer AWS FIS-Experimente im FIS-Benutzerhandbuch. AWS

> **Note**
>
> Bei AWS FIS Experimenten in MRSC sind eventuell konsistente Lesevorgänge zulässig, Aktualisierungen der Tabelleneinstellungen — wie z. B. das Ändern des Abrechnungsmodus oder die Konfiguration des Tabellendurchsatzes — sind jedoch nicht zulässig, ähnlich wie bei MREC. Weitere Informationen `FaultInjectionServiceInducedErrors`zum Fehlercode finden Sie in der CloudWatch Metrik.

Time to Live (TTL)

In globalen Tabellen mit MREC-Konfiguration kann die Konfiguration von Time To Live (TTL) gelöscht werden. Die TTL-Einstellungen werden automatisch für alle Replikate einer globalen

Tabelle synchronisiert. Wenn TTL ein Element aus einem Replikat in einer Region löscht, wird der Löschvorgang auf alle anderen Replikate in der globalen Tabelle repliziert. TTL verbraucht keine Schreibkapazität; das Löschen von TTL wird Ihnen in der Region, in der der Löschvorgang stattgefunden hat, nicht in Rechnung gestellt. In allen anderen Regionen, deren globale Tabelle ein Replikat enthält, wird Ihnen jedoch der replizierte Löschvorgang in Rechnung gestellt.

Die Replikation eines TTL-Löschvorgangs verbraucht Schreibkapazität in den Replikaten, in die der Löschvorgang repliziert wird. Replikate, die für bereitgestellte Kapazität konfiguriert sind, können Anfragen drosseln, wenn die Kombination aus Schreibdurchsatz und TTL-Löschdurchsatz die bereitgestellte Schreibkapazität überschreitet.

Globale Tabellen mit MRSC-Konfiguration unterstützen die Konfiguration des TTL-Löschvorgangs nicht.

Streams

Globale Tabellen mit MREC-Konfiguration replizieren Änderungen, indem sie diese aus einem DynamoDB-Stream in einer Replikattabelle auslesen und auf alle anderen Replikattabellen anwenden. Streams sind daher standardmäßig für alle Replikate in einer globalen MREC-Tabelle aktiviert und können in diesen Replikaten nicht deaktiviert werden. Der MREC-Replikationsprozess kann mehrere Änderungen in einem kurzen Zeitraum zu einem einzigen replizierten Schreibvorgang kombinieren, was dazu führt, dass der Stream jedes Replikats Datensätze enthält, die sich geringfügig unterscheiden. Streams-Datensätze in MREC-Replikaten werden immer elementweise sortiert, wobei die Reihenfolge zwischen den Elementen sich von Replikat zu Replikat unterscheiden kann.

Globale Tabellen mit MRSC-Konfiguration verwenden keine DynamoDB Streams für die Replikation, sodass Streams in MRSC-Replikaten standardmäßig nicht aktiviert sind. Streams können in einem MRSC-Replikat aktiviert werden. Streams-Datensätze in MRSC-Replikaten sind für jedes Replikat identisch, auch in Bezug auf ihre Reihenfolge.

Wenn Sie eine Anwendung schreiben möchten, die Streams-Datensätze für Änderungen verarbeitet, die in einer bestimmten Region, aber nicht in anderen Regionen in einer globalen Tabelle vorgenommen wurden, können Sie jedem Element ein Attribut hinzufügen, das definiert, in welcher Region die Änderung für dieses Element stattgefunden hat. Sie können dieses Attribut verwenden, um Streams-Datensätze nach Änderungen zu filtern, die in anderen Regionen aufgetreten sind und hierzu auch Lambda-Ereignisfilter verwenden, um Lambda-Funktionen nur für Änderungen in einer bestimmten Region aufzurufen.

Transaktionen

In einer globalen Tabelle mit MREC-Konfiguration sind DynamoDB-Transaktionsoperationen (`TransactWriteItems` und `TransactGetItems`) nur innerhalb der Region, in der der Vorgang aufgerufen wurde, unteilbar. Schreibtransaktionen werden nicht als Einheit regionsübergreifend repliziert, was bedeutet, dass nur einige Schreibvorgänge in einer Transaktion zu einem bestimmten Zeitpunkt durch Lesevorgänge in anderen Replikaten zurückgegeben werden können.

Wenn Sie beispielsweise eine globale Tabelle mit Replikaten in den Regionen USA Ost (Ohio) und USA West (Oregon) nutzen und eine `TransactWriteItems`-Operation in der Region USA Ost (Ohio) durchführen, sind unter Umständen partiell durchgeführte Transaktionen in der Region USA West (Oregon) zu beobachten, während die Änderungen repliziert werden. Die Änderungen werden erst in die anderen Regionen repliziert, nachdem sie in der Quellregion in die Datenbank eingetragen wurden.

Globale Tabellen mit MRSC-Konfiguration unterstützen keine Transaktionsvorgänge und geben einen Fehler zurück, wenn solche Operationen in einem MRSC-Replikat aufgerufen werden.

Lese- und Schreibdurchsatz

Modus bereitgestellter Kapazität

Eine Replikation verbraucht Schreibkapazität. Replikate, die für bereitgestellte Kapazität konfiguriert sind, können Anfragen drosseln, wenn die Kombination aus Anwendungsschreibdurchsatz und Replikationsschreibdurchsatz die bereitgestellte Schreibkapazität übersteigt. Bei globalen Tabellen, die den Bereitstellungsmodus verwenden, werden die Auto Scaling-Einstellungen für Lese- und Schreibkapazitäten zwischen den Replikaten synchronisiert.

Sie können die Lesekapazitätseinstellungen für jedes Replikat in einer globalen Tabelle unabhängig konfigurieren, indem Sie den `ProvisionedThroughputOverride`Parameter auf Replikatebene verwenden. Standardmäßig werden Änderungen an der bereitgestellten Lesekapazität auf alle Replikate in der globalen Tabelle angewendet. Beim Hinzufügen eines neuen Replikats zu einer globalen Tabelle wird die Lesekapazität der Quelltabelle oder des Replikats als Anfangswert verwendet, sofern nicht ausdrücklich eine Überschreibung auf Replikatebene angegeben ist.

On-Demand-Modus

Bei globalen Tabellen, die für den On-Demand-Modus konfiguriert sind, wird die Schreibkapazität automatisch über alle Replikate hinweg synchronisiert. DynamoDB passt die Kapazität

automatisch an den Datenverkehr an, und es müssen keine replikatspezifischen Lese- oder Schreibkapazitätseinstellungen verwaltet werden.

Synchronisierung von Einstellungen

Einstellungen in globalen DynamoDB-Tabellen sind Konfigurationsparameter, mit denen verschiedene Aspekte des Tabellenverhaltens und der Replikation gesteuert werden können. Diese Einstellungen werden über die DynamoDB-Steuerebene verwaltet APIs und können beim Erstellen oder Ändern globaler Tabellen konfiguriert werden. In globale Tabellen werden bestimmte Einstellungen für alle Replikate automatisch synchronisiert, um die Konsistenz zu wahren und gleichzeitig Flexibilität für regionsspezifische Optimierungen zu ermöglichen. Wenn Sie wissen, welche Einstellungen synchronisiert werden und wie sich diese verhalten, können Sie Ihre globale Tabelle effizient konfigurieren. Je nach Art ihrer replikatsübergreifenden Synchronisation lassen sich diese Einstellungen in drei Hauptkategorien unterteilen.

Die folgenden Einstellungen werden immer zwischen Replikaten in einer globalen Tabelle synchronisiert:

- Kapazitätsmodus (bereitgestellte oder On-Demand-Kapazität)
- Bereitgestellte Schreibkapazität der Tabelle
- Schreib-Auto-Scaling der Tabelle
- Attributdefinition des Schlüsselschemas
- Definition des globalen sekundären Index (GSI)
- Bereitgestellte Schreibkapazität des GSI
- Schreib-Auto-Scaling des GSI
- Typ der serverseitigen Verschlüsselung (Server-Side Encryption, SSE)
- Streams-Definition im MREC-Modus
- Time to Live (TTL)
- Warmdurchsatz
- Maximaler On-Demand-Schreibdurchsatz

Die folgenden Einstellungen werden zwischen Replikaten synchronisiert, können jedoch für ein Replikat außer Kraft gesetzt werden:

- Von der Tabelle bereitgestellte Lesekapazität

- Auto-Scaling für Tabellenlesevorgänge
- Vom GSI bereitgestellte Lesekapazität
- Auto-Scaling für GSI-Lesevorgänge
- Tabellenklasse
- Maximaler On-Demand-Lesedurchsatz

> **Note**
>
> Überschreibbare Werte werden geändert, wenn die Einstellung in einem anderen Replikat geändert wird. Sie haben zum Beispiel eine globale MREC-Tabelle mit Replikaten in USA Ost (Nord-Virginia) und USA West (Oregon). Für das Replikat „USA Ost“ (Nord-Virginia) wurde ein Lesedurchsatz von 200 bereitgestellt. RCUs Für das Replikat in der Region USA West (Oregon) wurde der bereitgestellte Lesedurchsatz auf 100 festgelegt. RCUs Wenn Sie die Einstellung für den bereitgestellten Lesedurchsatz auf dem Replikat USA Ost (Nord-Virginia) von 200 RCUs auf 300 aktualisieren RCUs, wird der neue Wert für den bereitgestellten Lesedurchsatz auch auf das Replikat in USA West (Oregon) angewendet. Dadurch wird die Einstellung für den bereitgestellten Lesedurchsatz für das Replikat USA West (Oregon) vom überschriebenen Wert 100 auf den neuen Wert 300 geändert. RCUs RCUs

Die folgenden Einstellungen werden niemals zwischen Replikaten in einer globalen Tabelle synchronisiert:

- Löschschutz
- Point-in-time Wiederherstellung
- Tags (Markierungen)
- Aktivierung von Table CloudWatch Contributor Insights
- Aktivierung von GSI Contributor Insights CloudWatch
- Definition der Kinesis-Datenströme
- Ressourcenrichtlinien
- Streams-Definition im MREC-Modus

Alle anderen Einstellungen werden nicht zwischen Replikaten synchronisiert.

DynamoDB Accelerator (DAX)

Bei Schreibvorgängen in Replikate von globalen Tabellen wird DynamoDB unter Umgehung von DynamoDB Accelerator (DAX) direkt aktualisiert. In der Folge kann es zu veralteten DAX-Caches kommen, da der DAX-Cache nicht durch die Schreibvorgänge aktualisiert wird. DAX-Caches, die für Replikate von globalen Tabellen konfiguriert sind, werden erst bei Ablauf der TTL des Cache aktualisiert.

Überlegungen zur Verwaltung globaler Tabellen

Sie können eine Tabelle, die zum Hinzufügen des neuen Replikats einer globalen Tabelle verwendet wurde, erst 24 Stunden nach der Erstellung des neuen Replikats löschen.

Wenn Sie eine AWS Region deaktivieren, die globale Tabellenreplikate enthält, werden diese Replikate 20 Stunden nach der Deaktivierung der Region dauerhaft in Tabellen mit nur einer Region konvertiert.

Tutorials: Erstellen von globalen Tabellen

Dieser Abschnitt enthält step-by-step Anweisungen zum Erstellen globaler DynamoDB-Tabellen, die für Ihren bevorzugten Konsistenzmodus konfiguriert sind. Wählen Sie je nach Anforderungen der Anwendung entweder den Modus „Multi-Region Eventual Consistency (MREC)“ oder „Multi-Region Strong Consistency (MRSC)“ aus.

Globale MREC-Tabellen bieten eine geringere Schreiblatenz und sorgen letztendlich für Konsistenz über alle AWS-Regionen hinweg. Globale MRSC-Tabellen bieten strikt konsistente Lesevorgänge in allen Regionen und ermöglichen etwas höhere Schreiblatenzen als MREC-Tabellen. Wählen Sie den Konsistenzmodus, der den Anforderungen Ihrer Anwendung an Datenkonsistenz, Latenz und Verfügbarkeit am ehesten entspricht.

Themen

- Erstellen einer globalen Tabelle, die für den MREC-Modus konfiguriert ist
- Erstellen einer globalen Tabelle, die für den MRSC-Modus konfiguriert ist

Erstellen einer globalen Tabelle, die für den MREC-Modus konfiguriert ist

In diesem Abschnitt erfahren Sie, wie Sie eine globale Tabelle im MREC-Modus (Multi-Region Eventual Consistency) erstellen. MREC ist der Standard-Konsistenzmodus für globale Tabellen und ermöglicht Schreibvorgänge mit geringer Latenz und asynchroner Replikation über alle AWS-

Regionen hinweg. Änderungen, die in einer Region an einem Element vorgenommen werden, werden in der Regel innerhalb einer Sekunde auf alle anderen Regionen repliziert. Dadurch eignet sich der MREC-Modus ideal für Anwendungen, die einer geringen Schreiblatenz Priorität einräumen und tolerieren können, dass aus verschiedenen Regionen kurzzeitig Datenversionen mit geringen Unterschieden zurückgegeben werden.

Sie können globale MREC-Tabellen mit Replikaten in jeder AWS Region erstellen, in der DynamoDB verfügbar ist, und jederzeit Replikate hinzufügen oder entfernen. In den folgenden Beispielen wird gezeigt, wie eine globale DynamoDB-Tabelle mit Replikaten in mehreren Regionen erstellt wird.

Erstellen einer globalen Tabelle im MREC-Modus mit der DynamoDB-Konsole

Gehen Sie wie folgt vor, um eine globale Tabelle mit der AWS-Managementkonsole zu erstellen. Im folgenden Beispiel wird eine globale Tabelle mit Replikattabellen in den USA und Europa erstellt.

1. Melden Sie sich bei der an AWS-Managementkonsole und öffnen Sie die DynamoDB-Konsole unter. https://console.aws.amazon.com/dynamodb/
2. Wählen Sie für dieses Beispiel in der Regionsauswahl in der Navigationsleiste die Region USA Ost (Ohio) aus.
3. Klicken Sie im Navigationsbereich auf der linken Seite der Konsole auf Tabellen.
4. Wählen Sie Tabelle erstellen aus.
5. Gehen Sie auf der Seite Tabelle erstellen wie folgt vor:

    a. Geben Sie unter Tabellenname **`Music`** ein.

    b. Geben Sie unter Partitionsschlüssel den Wert **`Artist`** ein.

    c. Geben Sie unter (Sortierschlüssel) den Wert **`SongTitle`** ein.

    d. Behalten Sie die übrigen Standardeinstellungen bei und wählen Sie Tabelle erstellen aus.

    Diese neue Tabelle dient als erste Replikattabelle in einer neuen globalen Tabelle. Sie stellt den Prototyp für andere Replikattabellen dar, die Sie später hinzufügen.
6. Gehen Sie folgendermaßen vor, wenn die Tabelle aktiv wird:

    a. Wählen Sie in der Tabellenliste die Tabelle Musik aus.

    b. Wählen Sie die Registerkarte Globale Tabellen aus.

    c. Wählen Sie die Option Replikat erstellen aus.
7. Wählen Sie in der Dropdown-Liste Verfügbare Replikationsregionen den Eintrag USA West (Oregon) us-west-2 aus.

Die Konsole stellt sicher, dass die ausgewählte Region keine Tabelle mit demselben Namen enthält. Wenn eine Tabelle mit demselben Namen vorhanden ist, müssen Sie die vorhandene Tabelle löschen, bevor Sie eine neue Replikattabelle in der betreffenden Region erstellen können.

8. Wählen Sie die Option Replikat erstellen aus. Dies startet den Prozess der Erstellung von Tabellen in der Region „USA West (Oregon) us-west-2“.

   Die Registerkarte Globale Tabelle für die Tabelle Music (und für alle anderen Replikattabellen) zeigt, dass die Tabelle in mehreren Regionen repliziert wurde.

9. Fügen Sie eine weitere Region hinzu, indem Sie die vorherigen Schritte wiederholen, aber als Region Europa (Frankfurt) eu-central-1 auswählen.
10. So testen Sie die Replikation:

    a. Stellen Sie sicher, dass Sie die AWS-Managementkonsole Region USA Ost (Ohio) verwenden.
    b. Wählen Sie Tabellenelemente erkunden aus.
    c. Wählen Sie Element erstellen aus.
    d. Geben Sie **`item_1`** für Künstler und **`Song Value 1`** für ein SongTitle.
    e. Wählen Sie Element erstellen aus.

11. Überprüfen Sie die Replikation, indem Sie zu den anderen Regionen wechseln:

    a. Wählen Sie in der Regionsauswahl in der rechten oberen Ecke Europa (Frankfurt) aus.
    b. Vergewissern Sie sich, dass die Tabelle Music das von Ihnen erstellte Objekt enthält.
    c. Wiederholen Sie die Überprüfung für USA West (Oregon).

Erstellen einer globalen MREC-Tabelle mit AWS CLI oder Java

CLI

Das folgende Codebeispiel zeigt, wie globale DynamoDB-Tabellen mit letztendlicher multiregionaler Konsistenz (Multi-Region Eventual Consistency, MREC) verwaltet werden.

- Erstellen Sie eine Tabelle mit multiregionaler Replikation (MREC).
- Fügen Sie Elemente in eine Replikattabelle ein und rufen Sie sie ab.
- Entfernen Sie Replikate one-by-one.

- Führen Sie eine Bereinigung durch Löschen der Tabelle aus.

AWS CLI mit Bash-Skript

Erstellen Sie eine Tabelle mit multiregionaler Replikation.

```
# Step 1: Create a new table (MusicTable) in US East (Ohio), with DynamoDB
 Streams enabled (NEW_AND_OLD_IMAGES)
aws dynamodb create-table \
    --table-name MusicTable \
    --attribute-definitions \
        AttributeName=Artist,AttributeType=S \
        AttributeName=SongTitle,AttributeType=S \
    --key-schema \
        AttributeName=Artist,KeyType=HASH \
        AttributeName=SongTitle,KeyType=RANGE \
    --billing-mode PAY_PER_REQUEST \
    --stream-specification StreamEnabled=true,StreamViewType=NEW_AND_OLD_IMAGES \
    --region us-east-2

# Step 2: Create an identical MusicTable table in US East (N. Virginia)
aws dynamodb update-table --table-name MusicTable --cli-input-json \
'{
  "ReplicaUpdates":
  [
    {
      "Create": {
        "RegionName": "us-east-1"
      }
    }
  ]
}' \
--region us-east-2

# Step 3: Create a table in Europe (Ireland)
aws dynamodb update-table --table-name MusicTable --cli-input-json \
'{
  "ReplicaUpdates":
  [
    {
      "Create": {
        "RegionName": "eu-west-1"
      }
```

```
    }
  ]
}' \
--region us-east-2
```

Beschreiben Sie die multiregionale Tabelle.

```
# Step 4: View the list of replicas created using describe-table
aws dynamodb describe-table \
    --table-name MusicTable \
    --region us-east-2 \
    --query 'Table.
{TableName:TableName,TableStatus:TableStatus,MultiRegionConsistency:MultiRegionConsistenc
{Region:RegionName,Status:ReplicaStatus}}'
```

Fügen Sie Elemente in eine Replikattabelle ein.

```
# Step 5: To verify that replication is working, add a new item to the Music
 table in US East (Ohio)
aws dynamodb put-item \
    --table-name MusicTable \
    --item '{"Artist": {"S":"item_1"},"SongTitle": {"S":"Song Value 1"}}' \
    --region us-east-2
```

Rufen Sie Elemente aus Replikattabellen ab.

```
# Step 6: Wait for a few seconds, and then check to see whether the item has
 been
# successfully replicated to US East (N. Virginia) and Europe (Ireland)
aws dynamodb get-item \
    --table-name MusicTable \
    --key '{"Artist": {"S":"item_1"},"SongTitle": {"S":"Song Value 1"}}' \
    --region us-east-1

aws dynamodb get-item \
    --table-name MusicTable \
    --key '{"Artist": {"S":"item_1"},"SongTitle": {"S":"Song Value 1"}}' \
    --region eu-west-1
```

Entfernen Sie Replikate.

```
# Step 7: Delete the replica table in Europe (Ireland) Region
aws dynamodb update-table --table-name MusicTable --cli-input-json \
'{
  "ReplicaUpdates":
  [
    {
      "Delete": {
        "RegionName": "eu-west-1"
      }
    }
  ]
}' \
--region us-east-2

# Delete the replica table in US East (N. Virginia) Region
aws dynamodb update-table --table-name MusicTable --cli-input-json \
'{
  "ReplicaUpdates":
  [
    {
      "Delete": {
        "RegionName": "us-east-1"
      }
    }
  ]
}' \
--region us-east-2
```

Führen Sie eine Bereinigung durch Löschen der Tabelle aus.

```
# Clean up: Delete the primary table
aws dynamodb delete-table --table-name MusicTable --region us-east-2

echo "Global table demonstration complete."
```

- Weitere API-Informationen finden Sie in den folgenden Themen der AWS CLI - Befehlsreferenz.
  - CreateTable

- DeleteTable
- DescribeTable
- GetItem
- PutItem
- UpdateTable

Java

Im folgenden Codebeispiel wird gezeigt, wie globale DynamoDB-Tabellen mit Replikaten in mehreren Regionen erstellt und verwaltet werden.

- Erstellen Sie eine Tabelle mit einem globalen sekundären Index und DynamoDB Streams.
- Fügen Sie Replikate in verschiedenen Regionen hinzu, um eine globale Tabelle zu erstellen.
- Entfernen Sie Replikate aus einer globalen Tabelle.
- Fügen Sie Testelemente hinzu, um die regionsübergreifende Replikation zu überprüfen.
- Beschreiben Sie die globale Tabellenkonfiguration und den Replikatstatus.

SDK für Java 2.x

Erstellen Sie eine Tabelle mit Global Secondary Index und DynamoDB Streams mithilfe von. AWS SDK for Java 2.x

```
   public static CreateTableResponse createTableWithGSI(
       final DynamoDbClient dynamoDbClient, final String tableName, final String
indexName) {

       if (dynamoDbClient == null) {
           throw new IllegalArgumentException("DynamoDB client cannot be null");
       }
       if (tableName == null || tableName.trim().isEmpty()) {
           throw new IllegalArgumentException("Table name cannot be null or
empty");
       }
       if (indexName == null || indexName.trim().isEmpty()) {
           throw new IllegalArgumentException("Index name cannot be null or
empty");
       }
```

```
        try {
            LOGGER.info("Creating table: " + tableName + " with GSI: " +
indexName);

            CreateTableRequest createTableRequest = CreateTableRequest.builder()
                .tableName(tableName)
                .attributeDefinitions(
                    AttributeDefinition.builder()
                        .attributeName("Artist")
                        .attributeType(ScalarAttributeType.S)
                        .build(),
                    AttributeDefinition.builder()
                        .attributeName("SongTitle")
                        .attributeType(ScalarAttributeType.S)
                        .build())
                .keySchema(
                    KeySchemaElement.builder()
                        .attributeName("Artist")
                        .keyType(KeyType.HASH)
                        .build(),
                    KeySchemaElement.builder()
                        .attributeName("SongTitle")
                        .keyType(KeyType.RANGE)
                        .build())
                .billingMode(BillingMode.PAY_PER_REQUEST)
                .globalSecondaryIndexes(GlobalSecondaryIndex.builder()
                    .indexName(indexName)
                    .keySchema(KeySchemaElement.builder()
                        .attributeName("SongTitle")
                        .keyType(KeyType.HASH)
                        .build())
                    .projection(

Projection.builder().projectionType(ProjectionType.ALL).build())
                    .build())
                .streamSpecification(StreamSpecification.builder()
                    .streamEnabled(true)
                    .streamViewType(StreamViewType.NEW_AND_OLD_IMAGES)
                    .build())
                .build();

            CreateTableResponse response =
dynamoDbClient.createTable(createTableRequest);
            LOGGER.info("Table creation initiated. Status: "
```

```
                + response.tableDescription().tableStatus());

            return response;

        } catch (DynamoDbException e) {
            LOGGER.severe("Failed to create table: " + tableName + " - " +
e.getMessage());
            throw e;
        }
    }
```

Warten Sie, bis eine Tabelle aktiv wird mit. AWS SDK for Java 2.x

```
    public static void waitForTableActive(final DynamoDbClient dynamoDbClient,
final String tableName) {

        if (dynamoDbClient == null) {
            throw new IllegalArgumentException("DynamoDB client cannot be null");
        }
        if (tableName == null || tableName.trim().isEmpty()) {
            throw new IllegalArgumentException("Table name cannot be null or
empty");
        }

        try {
            LOGGER.info("Waiting for table to become active: " + tableName);

            try (DynamoDbWaiter waiter =
                DynamoDbWaiter.builder().client(dynamoDbClient).build()) {
                DescribeTableRequest request =
                    DescribeTableRequest.builder().tableName(tableName).build();

                waiter.waitUntilTableExists(request);
                LOGGER.info("Table is now active: " + tableName);
            }

        } catch (DynamoDbException e) {
            LOGGER.severe("Failed to wait for table to become active: " +
tableName + " - " + e.getMessage());
            throw e;
        }
    }
```

Fügen Sie ein Replikat hinzu, um eine globale Tabelle zu erstellen oder zu erweitern mit AWS SDK for Java 2.x.

```
   public static UpdateTableResponse addReplica(
       final DynamoDbClient dynamoDbClient,
       final String tableName,
       final Region replicaRegion,
       final String indexName,
       final Long readCapacity) {

       if (dynamoDbClient == null) {
           throw new IllegalArgumentException("DynamoDB client cannot be null");
       }
       if (tableName == null || tableName.trim().isEmpty()) {
           throw new IllegalArgumentException("Table name cannot be null or
empty");
       }
       if (replicaRegion == null) {
           throw new IllegalArgumentException("Replica region cannot be null");
       }
       if (indexName == null || indexName.trim().isEmpty()) {
           throw new IllegalArgumentException("Index name cannot be null or
empty");
       }
       if (readCapacity == null || readCapacity <= 0) {
           throw new IllegalArgumentException("Read capacity must be a positive
number");
       }

       try {
           LOGGER.info("Adding replica in region: " + replicaRegion.id() + " for
table: " + tableName);

           // Create a ReplicationGroupUpdate for adding a replica
           ReplicationGroupUpdate replicationGroupUpdate =
ReplicationGroupUpdate.builder()
               .create(builder -> builder.regionName(replicaRegion.id())
                   .globalSecondaryIndexes(ReplicaGlobalSecondaryIndex.builder()
                       .indexName(indexName)

.provisionedThroughputOverride(ProvisionedThroughputOverride.builder()
```

```
                            .readCapacityUnits(readCapacity)
                            .build())
                        .build())
                    .build())
                .build();

            UpdateTableRequest updateTableRequest = UpdateTableRequest.builder()
                .tableName(tableName)
                .replicaUpdates(replicationGroupUpdate)
                .build();

            UpdateTableResponse response =
dynamoDbClient.updateTable(updateTableRequest);
            LOGGER.info("Replica addition initiated in region: " +
replicaRegion.id());

            return response;

        } catch (DynamoDbException e) {
            LOGGER.severe("Failed to add replica in region: " +
replicaRegion.id() + " - " + e.getMessage());
            throw e;
        }
    }
```

Entfernen Sie ein Replikat aus einer globalen Tabelle mit. AWS SDK for Java 2.x

```
    public static UpdateTableResponse removeReplica(
        final DynamoDbClient dynamoDbClient, final String tableName, final Region
replicaRegion) {

        if (dynamoDbClient == null) {
            throw new IllegalArgumentException("DynamoDB client cannot be null");
        }
        if (tableName == null || tableName.trim().isEmpty()) {
            throw new IllegalArgumentException("Table name cannot be null or
empty");
        }
        if (replicaRegion == null) {
            throw new IllegalArgumentException("Replica region cannot be null");
        }
```

```
        try {
            LOGGER.info("Removing replica in region: " + replicaRegion.id() + "
for table: " + tableName);

            // Create a ReplicationGroupUpdate for removing a replica
            ReplicationGroupUpdate replicationGroupUpdate =
ReplicationGroupUpdate.builder()
                .delete(builder ->
builder.regionName(replicaRegion.id()).build())
                .build();

            UpdateTableRequest updateTableRequest = UpdateTableRequest.builder()
                .tableName(tableName)
                .replicaUpdates(replicationGroupUpdate)
                .build();

            UpdateTableResponse response =
dynamoDbClient.updateTable(updateTableRequest);
            LOGGER.info("Replica removal initiated in region: " +
replicaRegion.id());

            return response;

        } catch (DynamoDbException e) {
            LOGGER.severe("Failed to remove replica in region: " +
replicaRegion.id() + " - " + e.getMessage());
            throw e;
        }
    }
```

Fügen Sie Testelemente hinzu, um die Replikation mithilfe von AWS SDK for Java 2.x zu überprüfen.

```
    public static PutItemResponse putTestItem(
        final DynamoDbClient dynamoDbClient, final String tableName, final String
artist, final String songTitle) {

        if (dynamoDbClient == null) {
            throw new IllegalArgumentException("DynamoDB client cannot be null");
        }
        if (tableName == null || tableName.trim().isEmpty()) {
```

```
            throw new IllegalArgumentException("Table name cannot be null or
empty");
        }
        if (artist == null || artist.trim().isEmpty()) {
            throw new IllegalArgumentException("Artist cannot be null or empty");
        }
        if (songTitle == null || songTitle.trim().isEmpty()) {
            throw new IllegalArgumentException("Song title cannot be null or
empty");
        }

        try {
            LOGGER.info("Adding test item to table: " + tableName);

            Map<String,
software.amazon.awssdk.services.dynamodb.model.AttributeValue> item = new
HashMap<>();
            item.put(
                "Artist",

software.amazon.awssdk.services.dynamodb.model.AttributeValue.builder()
                    .s(artist)
                    .build());
            item.put(
                "SongTitle",

software.amazon.awssdk.services.dynamodb.model.AttributeValue.builder()
                    .s(songTitle)
                    .build());

            PutItemRequest putItemRequest =
                PutItemRequest.builder().tableName(tableName).item(item).build();

            PutItemResponse response = dynamoDbClient.putItem(putItemRequest);
            LOGGER.info("Test item added successfully");

            return response;

        } catch (DynamoDbException e) {
            LOGGER.severe("Failed to add test item to table: " + tableName + " -
" + e.getMessage());
            throw e;
        }
    }
```

Beschreiben Sie die globale Tabellenkonfiguration und die Verwendung von AWS SDK for Java 2.x Replikaten.

```
   public static DescribeTableResponse describeTable(final DynamoDbClient
dynamoDbClient, final String tableName) {

       if (dynamoDbClient == null) {
           throw new IllegalArgumentException("DynamoDB client cannot be null");
       }
       if (tableName == null || tableName.trim().isEmpty()) {
           throw new IllegalArgumentException("Table name cannot be null or
empty");
       }

       try {
           LOGGER.info("Describing table: " + tableName);

           DescribeTableRequest request =
               DescribeTableRequest.builder().tableName(tableName).build();

           DescribeTableResponse response =
dynamoDbClient.describeTable(request);

           LOGGER.info("Table status: " + response.table().tableStatus());
           if (response.table().replicas() != null
               && !response.table().replicas().isEmpty()) {
               LOGGER.info("Number of replicas: " +
response.table().replicas().size());
               response.table()
                   .replicas()
                   .forEach(replica -> LOGGER.info(
                       "Replica region: " + replica.regionName() + ", Status: "
+ replica.replicaStatus()));
           }

           return response;

       } catch (ResourceNotFoundException e) {
           LOGGER.severe("Table not found: " + tableName + " - " +
e.getMessage());
           throw e;
```

```
        } catch (DynamoDbException e) {
            LOGGER.severe("Failed to describe table: " + tableName + " - " +
e.getMessage());
            throw e;
        }
    }
```

Vollständiges Beispiel für globale Tabellenoperationen unter Verwendung von AWS SDK for Java 2.x.

```
    public static void exampleUsage(final Region sourceRegion, final Region
replicaRegion) {

        String tableName = "Music";
        String indexName = "SongTitleIndex";
        Long readCapacity = 15L;

        // Create DynamoDB client for the source region
        try (DynamoDbClient dynamoDbClient =
            DynamoDbClient.builder().region(sourceRegion).build()) {

            try {
                // Step 1: Create the initial table with GSI and streams
                LOGGER.info("Step 1: Creating table in source region: " +
sourceRegion.id());
                createTableWithGSI(dynamoDbClient, tableName, indexName);

                // Step 2: Wait for table to become active
                LOGGER.info("Step 2: Waiting for table to become active");
                waitForTableActive(dynamoDbClient, tableName);

                // Step 3: Add replica in destination region
                LOGGER.info("Step 3: Adding replica in region: " +
replicaRegion.id());
                addReplica(dynamoDbClient, tableName, replicaRegion, indexName,
readCapacity);

                // Step 4: Wait a moment for replica creation to start
                Thread.sleep(5000);

                // Step 5: Describe table to view replica information
                LOGGER.info("Step 5: Describing table to view replicas");
```

```
                describeTable(dynamoDbClient, tableName);

                // Step 6: Add a test item to verify replication
                LOGGER.info("Step 6: Adding test item to verify replication");
                putTestItem(dynamoDbClient, tableName, "TestArtist", "TestSong");

                LOGGER.info("Global table setup completed successfully!");
                LOGGER.info("You can verify replication by checking the item in
region: " + replicaRegion.id());

                // Step 7: Remove replica and clean up table
                LOGGER.info("Step 7: Removing replica from region: " +
replicaRegion.id());
                removeReplica(dynamoDbClient, tableName, replicaRegion);
                DeleteTableResponse deleteTableResponse =
dynamoDbClient.deleteTable(
                    DeleteTableRequest.builder().tableName(tableName).build());
                LOGGER.info("MREC global table demonstration completed
successfully!");

            } catch (InterruptedException e) {
                Thread.currentThread().interrupt();
                throw new RuntimeException("Thread was interrupted", e);
            } catch (DynamoDbException e) {
                LOGGER.severe("DynamoDB operation failed: " + e.getMessage());
                throw e;
            }
        }
    }
```

- Weitere API-Informationen finden Sie in den folgenden Themen der AWS SDK for Java 2.x - API-Referenz.
  - CreateTable
  - DescribeTable
  - PutItem
  - UpdateTable

Erstellen einer globalen Tabelle, die für den MRSC-Modus konfiguriert ist

In diesem Abschnitt erfahren Sie, wie Sie eine globale Tabelle im MRSC-Modus (Multi-Region Strong Consistency) erstellen. Globale MRSC-Tabellen replizieren geänderte Elemente synchron über Regionen hinweg und stellen so sicher, dass durch strikt konsistente Lesevorgänge in jedem Replikat immer die aktuelle Version eines Elements zurückgegeben wird. Wenn Sie eine Tabelle mit einer einzigen Region in eine globale MRSC-Tabelle konvertieren, müssen Sie sicherstellen, dass die Tabelle leer ist. Die Konvertierung einer Tabelle mit nur einer Region in eine globale MRSC-Tabelle mit vorhandenen Elementen wird nicht unterstützt. Stellen Sie sicher, dass während des Konvertierungsvorgangs keine Daten in die Tabelle geschrieben werden.

Sie können eine globale MRSC-Tabelle mit drei Replikaten oder zwei Replikaten und einem Witness konfigurieren. Beim Erstellen einer globalen MRSC-Tabelle wählen Sie die Regionen aus, in denen Replikate und ein optionaler Witness bereitgestellt werden. Im folgenden Beispiel wird eine globale MRSC-Tabelle mit Replikaten in den Regionen USA Ost (Nord-Virginia) und USA Ost (Ohio) und einem Witness in der Region USA West (Oregon) erstellt.

> **Note**
>
> Stellen Sie vor dem Erstellen einer globalen Tabelle sicher, dass die Service-Quota-Durchsatzgrenzwerte in allen Zielregionen einheitlich sind, da dies für die Erstellung einer globalen Tabelle erforderlich ist. Weitere Informationen zu den Grenzwerten für den Durchsatz für globale Tabellen finden Sie unter Globale Tabellenkontingente.

Erstellen einer globalen Tabelle im MRSC-Modus mit der DynamoDB-Konsole

Gehen Sie wie folgt vor, um eine globale Tabelle mit der AWS-Managementkonsole zu erstellen.

1. Melden Sie sich bei der an AWS-Managementkonsole und öffnen Sie die DynamoDB-Konsole unter. https://console.aws.amazon.com/dynamodb/
2. Wählen Sie in der Navigationsleiste Regionsauswahl eine Region aus, in der globale Tabellen mit MRSC unterstützt werden, wie z. B. **`us-east-2`**.
3. Wählen Sie im Navigationsbereich Tables (Tabellen) aus.
4. Wählen Sie Create table (Tabelle erstellen) aus.
5. Gehen Sie auf der Seite Tabelle erstellen wie folgt vor:

    a. Geben Sie unter Tabellenname **`Music`** ein.

b. Geben Sie unter Partitionsschlüssel den Wert **`Artist`** ein und behalten Sie die Standardeinstellung Zeichenfolge bei.

c. Geben Sie unter Sortierschlüssel den Wert **`SongTitle`** ein und behalten Sie die Standardeinstellung Zeichenfolge bei.

d. Behalten Sie die übrigen Standardeinstellungen bei und wählen Sie Tabelle erstellen aus.

   Diese neue Tabelle dient als erste Replikattabelle in einer neuen globalen Tabelle. Sie stellt den Prototyp für andere Replikattabellen dar, die Sie später hinzufügen.

6. Warten Sie, bis die Tabelle aktiv wird, und wählen Sie sie dann aus der Tabellenliste aus.
7. Wählen Sie die Registerkarte Globale Tabellen und anschließend Replikat erstellen aus.
8. Gehen Sie auf der Seite Replikat erstellen wie folgt vor:

   a. Wählen Sie unter Multiregionale Konsistenz die Option Starke Konsistenz aus.

   b. Wählen Sie unter Replikationsregion 1 die Option **`US East (N. Virginia) us-east-1`** aus.

   c. Wählen Sie unter Replikationsregion 2 die Option **`US West (Oregon) us-west-2`** aus.

   d. Aktivieren Sie Option Als Witness konfigurieren für die Region „USA West (Oregon)“.

   e. Wählen Sie die Option Replikat erstellen aus.

9. Warten Sie, bis der Vorgang zur Erstellung des Replikats und des Witness abgeschlossen ist. Der Replikatstatus lautet Aktiv, wenn die Tabelle einsatzbereit ist.

Erstellen einer globalen MRSC-Tabelle mit oder Java AWS CLI

Bevor Sie beginnen, stellen Sie sicher, dass Ihr IAM-Prinzipal über die erforderlichen Berechtigungen verfügt, um eine globale MRSC-Tabelle mit einer Witness-Region zu erstellen.

Mit der folgenden IAM-Beispielrichtlinie können Sie eine DynamoDB-Tabelle (`MusicTable`) in der Region „USA Ost (Ohio)“ mit einem Replikat in der Region „USA Ost (Nord-Virginia)“ und einer Witness-Region in der Region „USA West (Oregon)“ erstellen:

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
```

```
            "Effect": "Allow",
            "Action": [
                "dynamodb:CreateTable",
                "dynamodb:CreateTableReplica",
                "dynamodb:CreateGlobalTableWitness",
                "dynamodb:DescribeTable",
                "dynamodb:UpdateTable",
                "dynamodb:DeleteTable",
                "dynamodb:DeleteTableReplica",
                "dynamodb:DeleteGlobalTableWitness",
                "dynamodb:Scan",
                "dynamodb:Query",
                "dynamodb:UpdateItem",
                "dynamodb:PutItem",
                "dynamodb:GetItem",
                "dynamodb:DeleteItem",
                "dynamodb:BatchWriteItem"
            ],
            "Resource": [
                "arn:aws:dynamodb:us-east-1:123456789012:table/MusicTable",
                "arn:aws:dynamodb:us-east-2:123456789012:table/MusicTable",
                "arn:aws:dynamodb:us-west-2:123456789012:table/MusicTable"
            ]
        },
        {
            "Effect": "Allow",
            "Action": "iam:CreateServiceLinkedRole",
            "Resource": "arn:aws:iam::*:role/aws-service-role/
replication.dynamodb.amazonaws.com/AWSServiceRoleForDynamoDBReplication",
            "Condition": {
                "StringLike": {
                    "iam:AWSServiceName": "replication.dynamodb.amazonaws.com"
                }
            }
        }
    ]
}
```

In den folgenden Codebeispielen wird gezeigt, wie globale DynamoDB-Tabellen mit Multi-Region Strong Consistency (MRSC) erstellt und verwaltet werden.

- Erstellen einer Tabelle mit multiregionaler starker Konsistenz

- Überprüfen Sie die MRSC-Konfiguration und den Replikatstatus.
- Testen Sie die starke Konsistenz in allen Regionen mit sofortigen Lesevorgängen.
- Führen Sie bedingte Schreibvorgänge mit MRSC-Garantien durch.
- Bereinigen Sie die globalen MRSC-Tabellenressourcen.

Bash

AWS CLI mit Bash-Skript

Erstellen Sie eine Tabelle mit multiregionaler starker Konsistenz (Multiregion Strong Consistency, MRSC).

```
# Step 1: Create a new table in us-east-2 (primary region for MRSC)
# Note: Table must be empty when enabling MRSC
aws dynamodb create-table \
    --table-name MusicTable \
    --attribute-definitions \
        AttributeName=Artist,AttributeType=S \
        AttributeName=SongTitle,AttributeType=S \
    --key-schema \
        AttributeName=Artist,KeyType=HASH \
        AttributeName=SongTitle,KeyType=RANGE \
    --billing-mode PAY_PER_REQUEST \
    --region us-east-2

# Wait for table to become active
aws dynamodb wait table-exists --table-name MusicTable --region us-east-2

# Step 2: Add replica and witness with Multi-Region Strong Consistency
# MRSC requires exactly three replicas in supported regions
aws dynamodb update-table \
    --table-name MusicTable \
    --replica-updates '[{"Create": {"RegionName": "us-east-1"}}]' \
    --global-table-witness-updates '[{"Create": {"RegionName": "us-west-2"}}]' \
    --multi-region-consistency STRONG \
    --region us-east-2
```

Überprüfen Sie die MRSC-Konfiguration und den Replikatstatus.

```
# Verify the global table configuration and MRSC setting
```

```
aws dynamodb describe-table \
    --table-name MusicTable \
    --region us-east-2 \
    --query 'Table.
{TableName:TableName,TableStatus:TableStatus,MultiRegionConsistency:MultiRegionConsistenc
{Region:RegionName,Status:ReplicaStatus}}'
```

Testen Sie die starke Konsistenz mit sofortigen Lesevorgängen in allen Regionen.

```
# Write an item to the primary region
aws dynamodb put-item \
    --table-name MusicTable \
    --item '{"Artist": {"S":"The Beatles"},"SongTitle": {"S":"Hey Jude"},"Album":
 {"S":"The Beatles 1967-1970"},"Year": {"N":"1968"}}' \
    --region us-east-2

# Read the item from replica region to verify strong consistency (cannot read or
 write to witness)
# No wait time needed - MRSC provides immediate consistency
echo "Reading from us-east-1 (immediate consistency):"
aws dynamodb get-item \
    --table-name MusicTable \
    --key '{"Artist": {"S":"The Beatles"},"SongTitle": {"S":"Hey Jude"}}' \
    --consistent-read \
    --region us-east-1
```

Führen Sie bedingte Schreibvorgänge mit MRSC-Garantien durch.

```
# Perform a conditional update from a different region
# This demonstrates that conditions work consistently across all regions
aws dynamodb update-item \
    --table-name MusicTable \
    --key '{"Artist": {"S":"The Beatles"},"SongTitle": {"S":"Hey Jude"}}' \
    --update-expression "SET #rating = :rating" \
    --condition-expression "attribute_exists(Artist)" \
    --expression-attribute-names '{"#rating": "Rating"}' \
    --expression-attribute-values '{":rating": {"N":"5"}}' \
    --region us-east-1
```

Bereinigen Sie die globalen MRSC-Tabellenressourcen.

```
# Remove replica tables (must be done before deleting the primary table)
aws dynamodb update-table \
    --table-name MusicTable \
    --replica-updates '[{"Delete": {"RegionName": "us-east-1"}}]' \
    --global-table-witness-updates '[{"Delete": {"RegionName": "us-west-2"}}]' \
    --region us-east-2

# Wait for replicas to be deleted
echo "Waiting for replicas to be deleted..."
sleep 30

# Delete the primary table
aws dynamodb delete-table \
    --table-name MusicTable \
    --region us-east-2
```

- Weitere API-Informationen finden Sie in den folgenden Themen der AWS CLI - Befehlsreferenz.
  - CreateTable
  - DeleteTable
  - DescribeTable
  - GetItem
  - PutItem
  - UpdateItem
  - UpdateTable

Java

SDK für Java 2.x

Erstellen Sie eine regionale Tabelle, die für die MRSC-Konvertierung bereit ist, mit. AWS SDK for Java 2.x

```
   public static CreateTableResponse createRegionalTable(final DynamoDbClient
dynamoDbClient, final String tableName) {

        if (dynamoDbClient == null) {
            throw new IllegalArgumentException("DynamoDB client cannot be null");
```

```
        }
        if (tableName == null || tableName.trim().isEmpty()) {
            throw new IllegalArgumentException("Table name cannot be null or
empty");
        }

        try {
            LOGGER.info("Creating regional table: " + tableName + " (must be
empty for MRSC)");

            CreateTableRequest createTableRequest = CreateTableRequest.builder()
                .tableName(tableName)
                .attributeDefinitions(
                    AttributeDefinition.builder()
                        .attributeName("Artist")
                        .attributeType(ScalarAttributeType.S)
                        .build(),
                    AttributeDefinition.builder()
                        .attributeName("SongTitle")
                        .attributeType(ScalarAttributeType.S)
                        .build())
                .keySchema(
                    KeySchemaElement.builder()
                        .attributeName("Artist")
                        .keyType(KeyType.HASH)
                        .build(),
                    KeySchemaElement.builder()
                        .attributeName("SongTitle")
                        .keyType(KeyType.RANGE)
                        .build())
                .billingMode(BillingMode.PAY_PER_REQUEST)
                .build();

            CreateTableResponse response =
dynamoDbClient.createTable(createTableRequest);
            LOGGER.info("Regional table creation initiated. Status: "
                + response.tableDescription().tableStatus());

            return response;

        } catch (DynamoDbException e) {
            LOGGER.severe("Failed to create regional table: " + tableName + " - "
+ e.getMessage());
            throw DynamoDbException.builder()
```

```
                .message("Failed to create regional table: " + tableName)
                .cause(e)
                .build();
        }
    }
```

Konvertiert eine regionale Tabelle in eine MRSC-Tabelle mit Replikaten und Zeugen unter Verwendung von. AWS SDK for Java 2.x

```
    public static UpdateTableResponse convertToMRSCWithWitness(
        final DynamoDbClient dynamoDbClient,
        final String tableName,
        final Region replicaRegion,
        final Region witnessRegion) {

        if (dynamoDbClient == null) {
            throw new IllegalArgumentException("DynamoDB client cannot be null");
        }
        if (tableName == null || tableName.trim().isEmpty()) {
            throw new IllegalArgumentException("Table name cannot be null or
empty");
        }
        if (replicaRegion == null) {
            throw new IllegalArgumentException("Replica region cannot be null");
        }
        if (witnessRegion == null) {
            throw new IllegalArgumentException("Witness region cannot be null");
        }

        try {
            LOGGER.info("Converting table to MRSC with replica in " +
replicaRegion.id() + " and witness in "
                + witnessRegion.id());

            // Create replica update using ReplicationGroupUpdate
            ReplicationGroupUpdate replicaUpdate =
ReplicationGroupUpdate.builder()
                .create(CreateReplicationGroupMemberAction.builder()
                    .regionName(replicaRegion.id())
                    .build())
                .build();
```

```
            // Create witness update
            GlobalTableWitnessGroupUpdate witnessUpdate =
GlobalTableWitnessGroupUpdate.builder()
                .create(CreateGlobalTableWitnessGroupMemberAction.builder()
                    .regionName(witnessRegion.id())
                    .build())
                .build();

            UpdateTableRequest updateTableRequest = UpdateTableRequest.builder()
                .tableName(tableName)
                .replicaUpdates(List.of(replicaUpdate))
                .globalTableWitnessUpdates(List.of(witnessUpdate))
                .multiRegionConsistency(MultiRegionConsistency.STRONG)
                .build();

            UpdateTableResponse response =
dynamoDbClient.updateTable(updateTableRequest);
            LOGGER.info("MRSC conversion initiated. Status: "
                + response.tableDescription().tableStatus());
            LOGGER.info("UpdateTableResponse full object: " + response);
            return response;

        } catch (DynamoDbException e) {
            LOGGER.severe("Failed to convert table to MRSC: " + tableName + " - "
+ e.getMessage());
            throw DynamoDbException.builder()
                .message("Failed to convert table to MRSC: " + tableName)
                .cause(e)
                .build();
        }
    }
```

Beschreiben Sie eine globale MRSC-Tabellenkonfiguration mit. AWS SDK for Java 2.x

```
    public static DescribeTableResponse describeMRSCTable(final DynamoDbClient
dynamoDbClient, final String tableName) {

        if (dynamoDbClient == null) {
            throw new IllegalArgumentException("DynamoDB client cannot be null");
        }
        if (tableName == null || tableName.trim().isEmpty()) {
```

```
            throw new IllegalArgumentException("Table name cannot be null or
empty");
        }

        try {
            LOGGER.info("Describing MRSC global table: " + tableName);

            DescribeTableRequest request =
                DescribeTableRequest.builder().tableName(tableName).build();

            DescribeTableResponse response =
dynamoDbClient.describeTable(request);

            LOGGER.info("Table status: " + response.table().tableStatus());
            LOGGER.info("Multi-region consistency: " +
response.table().multiRegionConsistency());

            if (response.table().replicas() != null
                && !response.table().replicas().isEmpty()) {
                LOGGER.info("Number of replicas: " +
response.table().replicas().size());
                response.table()
                    .replicas()
                    .forEach(replica -> LOGGER.info(
                        "Replica region: " + replica.regionName() + ", Status: "
+ replica.replicaStatus()));
            }

            if (response.table().globalTableWitnesses() != null
                && !response.table().globalTableWitnesses().isEmpty()) {
                LOGGER.info("Number of witnesses: "
                    + response.table().globalTableWitnesses().size());
                response.table()
                    .globalTableWitnesses()
                    .forEach(witness -> LOGGER.info(
                        "Witness region: " + witness.regionName() + ", Status: "
+ witness.witnessStatus()));
            }

            return response;

        } catch (ResourceNotFoundException e) {
            LOGGER.severe("Table not found: " + tableName + " - " +
e.getMessage());
```

```
            throw DynamoDbException.builder()
                .message("Table not found: " + tableName)
                .cause(e)
                .build();
        } catch (DynamoDbException e) {
            LOGGER.severe("Failed to describe table: " + tableName + " - " +
e.getMessage());
            throw DynamoDbException.builder()
                .message("Failed to describe table: " + tableName)
                .cause(e)
                .build();
        }
    }
```

Fügen Sie mit AWS SDK for Java 2.x Testelemente hinzu, um die starke Konsistenz von MRSC zu überprüfen.

```
    public static PutItemResponse putTestItem(
        final DynamoDbClient dynamoDbClient,
        final String tableName,
        final String artist,
        final String songTitle,
        final String album,
        final String year) {

        if (dynamoDbClient == null) {
            throw new IllegalArgumentException("DynamoDB client cannot be null");
        }
        if (tableName == null || tableName.trim().isEmpty()) {
            throw new IllegalArgumentException("Table name cannot be null or
empty");
        }
        if (artist == null || artist.trim().isEmpty()) {
            throw new IllegalArgumentException("Artist cannot be null or empty");
        }
        if (songTitle == null || songTitle.trim().isEmpty()) {
            throw new IllegalArgumentException("Song title cannot be null or
empty");
        }

        try {
            LOGGER.info("Adding test item to MRSC global table: " + tableName);
```

```
            Map<String, AttributeValue> item = new HashMap<>();
            item.put("Artist", AttributeValue.builder().s(artist).build());
            item.put("SongTitle", AttributeValue.builder().s(songTitle).build());

            if (album != null && !album.trim().isEmpty()) {
                item.put("Album", AttributeValue.builder().s(album).build());
            }
            if (year != null && !year.trim().isEmpty()) {
                item.put("Year", AttributeValue.builder().n(year).build());
            }

            PutItemRequest putItemRequest =
                PutItemRequest.builder().tableName(tableName).item(item).build();

            PutItemResponse response = dynamoDbClient.putItem(putItemRequest);
            LOGGER.info("Test item added successfully with strong consistency");

            return response;

        } catch (DynamoDbException e) {
            LOGGER.severe("Failed to add test item to table: " + tableName + " -
" + e.getMessage());
            throw DynamoDbException.builder()
                .message("Failed to add test item to table: " + tableName)
                .cause(e)
                .build();
        }
    }
```

Lesen Sie Elemente mit konsistenten Lesevorgängen aus MRSC-Replikaten unter Verwendung von. AWS SDK for Java 2.x

```
    public static GetItemResponse getItemWithConsistentRead(
        final DynamoDbClient dynamoDbClient, final String tableName, final String
artist, final String songTitle) {

        if (dynamoDbClient == null) {
            throw new IllegalArgumentException("DynamoDB client cannot be null");
        }
        if (tableName == null || tableName.trim().isEmpty()) {
```

```
            throw new IllegalArgumentException("Table name cannot be null or
empty");
        }
        if (artist == null || artist.trim().isEmpty()) {
            throw new IllegalArgumentException("Artist cannot be null or empty");
        }
        if (songTitle == null || songTitle.trim().isEmpty()) {
            throw new IllegalArgumentException("Song title cannot be null or
empty");
        }

        try {
            LOGGER.info("Reading item from MRSC global table with consistent
read: " + tableName);

            Map<String, AttributeValue> key = new HashMap<>();
            key.put("Artist", AttributeValue.builder().s(artist).build());
            key.put("SongTitle", AttributeValue.builder().s(songTitle).build());

            GetItemRequest getItemRequest = GetItemRequest.builder()
                .tableName(tableName)
                .key(key)
                .consistentRead(true)
                .build();

            GetItemResponse response = dynamoDbClient.getItem(getItemRequest);

            if (response.hasItem()) {
                LOGGER.info("Item found with strong consistency - no wait time
needed");
            } else {
                LOGGER.info("Item not found");
            }

            return response;

        } catch (DynamoDbException e) {
            LOGGER.severe("Failed to read item from table: " + tableName + " - "
+ e.getMessage());
            throw DynamoDbException.builder()
                .message("Failed to read item from table: " + tableName)
                .cause(e)
                .build();
        }
```

```
    }
```

Führen Sie bedingte Aktualisierungen mit MRSC-Garantien durch, indem Sie. AWS SDK for Java 2.x

```
    public static UpdateItemResponse performConditionalUpdate(
        final DynamoDbClient dynamoDbClient,
        final String tableName,
        final String artist,
        final String songTitle,
        final String rating) {

        if (dynamoDbClient == null) {
            throw new IllegalArgumentException("DynamoDB client cannot be null");
        }
        if (tableName == null || tableName.trim().isEmpty()) {
            throw new IllegalArgumentException("Table name cannot be null or
empty");
        }
        if (artist == null || artist.trim().isEmpty()) {
            throw new IllegalArgumentException("Artist cannot be null or empty");
        }
        if (songTitle == null || songTitle.trim().isEmpty()) {
            throw new IllegalArgumentException("Song title cannot be null or
empty");
        }
        if (rating == null || rating.trim().isEmpty()) {
            throw new IllegalArgumentException("Rating cannot be null or empty");
        }

        try {
            LOGGER.info("Performing conditional update on MRSC global table: " +
tableName);

            Map<String, AttributeValue> key = new HashMap<>();
            key.put("Artist", AttributeValue.builder().s(artist).build());
            key.put("SongTitle", AttributeValue.builder().s(songTitle).build());

            Map<String, String> expressionAttributeNames = new HashMap<>();
            expressionAttributeNames.put("#rating", "Rating");
```

```
            Map<String, AttributeValue> expressionAttributeValues = new
HashMap<>();
            expressionAttributeValues.put(
                ":rating", AttributeValue.builder().n(rating).build());

            UpdateItemRequest updateItemRequest = UpdateItemRequest.builder()
                .tableName(tableName)
                .key(key)
                .updateExpression("SET #rating = :rating")
                .conditionExpression("attribute_exists(Artist)")
                .expressionAttributeNames(expressionAttributeNames)
                .expressionAttributeValues(expressionAttributeValues)
                .build();

            UpdateItemResponse response =
dynamoDbClient.updateItem(updateItemRequest);
            LOGGER.info("Conditional update successful - demonstrates strong
consistency");

            return response;

        } catch (ConditionalCheckFailedException e) {
            LOGGER.warning("Conditional check failed: " + e.getMessage());
            throw e;
        } catch (DynamoDbException e) {
            LOGGER.severe("Failed to perform conditional update: " + tableName +
" - " + e.getMessage());
            throw DynamoDbException.builder()
                .message("Failed to perform conditional update: " + tableName)
                .cause(e)
                .build();
        }
    }
```

Warten Sie, bis MRSC-Repliken und Zeugen die Nutzung aktivieren. AWS SDK for Java 2.x

```
    public static void waitForMRSCReplicasActive(
        final DynamoDbClient dynamoDbClient, final String tableName, final int
maxWaitTimeSeconds)
        throws InterruptedException {

        if (dynamoDbClient == null) {
```

```
            throw new IllegalArgumentException("DynamoDB client cannot be null");
        }
        if (tableName == null || tableName.trim().isEmpty()) {
            throw new IllegalArgumentException("Table name cannot be null or
empty");
        }
        if (maxWaitTimeSeconds <= 0) {
            throw new IllegalArgumentException("Max wait time must be positive");
        }

        try {
            LOGGER.info("Waiting for MRSC replicas and witnesses to become
active: " + tableName);

            final long startTime = System.currentTimeMillis();
            final long maxWaitTimeMillis = maxWaitTimeSeconds * 1000L;
            int backoffSeconds = 5; // Start with 5 second intervals
            final int maxBackoffSeconds = 30; // Cap at 30 seconds

            while (System.currentTimeMillis() - startTime < maxWaitTimeMillis) {
                DescribeTableResponse response =
describeMRSCTable(dynamoDbClient, tableName);

                boolean allActive = true;
                StringBuilder statusReport = new StringBuilder();

                if (response.table().multiRegionConsistency() == null
                    || !MultiRegionConsistency.STRONG
                        .toString()

.equals(response.table().multiRegionConsistency().toString())) {
                    allActive = false;
                    statusReport
                        .append("MultiRegionConsistency: ")
                        .append(response.table().multiRegionConsistency())
                        .append(" ");
                }
                if (response.table().replicas() == null
                    || response.table().replicas().isEmpty()) {
                    allActive = false;
                    statusReport.append("No replicas found. ");
                }
                if (response.table().globalTableWitnesses() == null
                    || response.table().globalTableWitnesses().isEmpty()) {
```

```
                    allActive = false;
                    statusReport.append("No witnesses found. ");
                }

                // Check table status
                if (!"ACTIVE".equals(response.table().tableStatus().toString()))
{
                    allActive = false;
                    statusReport
                        .append("Table: ")
                        .append(response.table().tableStatus())
                        .append(" ");
                }

                // Check replica status
                if (response.table().replicas() != null) {
                    for (var replica : response.table().replicas()) {
                        if (!"ACTIVE".equals(replica.replicaStatus().toString()))
{
                            allActive = false;
                            statusReport
                                .append("Replica(")
                                .append(replica.regionName())
                                .append("): ")
                                .append(replica.replicaStatus())
                                .append(" ");
                        }
                    }
                }

                // Check witness status
                if (response.table().globalTableWitnesses() != null) {
                    for (var witness : response.table().globalTableWitnesses()) {
                        if (!"ACTIVE".equals(witness.witnessStatus().toString()))
{
                            allActive = false;
                            statusReport
                                .append("Witness(")
                                .append(witness.regionName())
                                .append("): ")
                                .append(witness.witnessStatus())
                                .append(" ");
                        }
                    }
```

```
                }

                if (allActive) {
                    LOGGER.info("All MRSC replicas and witnesses are now active:
" + tableName);
                    return;
                }

                LOGGER.info("Waiting for MRSC components to become active.
Status: " + statusReport.toString());
                LOGGER.info("Next check in " + backoffSeconds + " seconds...");

                tempWait(backoffSeconds);

                // Exponential backoff with cap
                backoffSeconds = Math.min(backoffSeconds * 2, maxBackoffSeconds);
            }

            throw DynamoDbException.builder()
                .message("Timeout waiting for MRSC replicas to become active
after " + maxWaitTimeSeconds + " seconds")
                .build();

        } catch (DynamoDbException | InterruptedException e) {
            LOGGER.severe("Failed to wait for MRSC replicas to become active: " +
tableName + " - " + e.getMessage());
            throw e;
        }
    }
```

Bereinigen Sie MRSC-Replikate und Zeugen mithilfe von. AWS SDK for Java 2.x

```
    public static UpdateTableResponse cleanupMRSCReplicas(
        final DynamoDbClient dynamoDbClient,
        final String tableName,
        final Region replicaRegion,
        final Region witnessRegion) {

        if (dynamoDbClient == null) {
            throw new IllegalArgumentException("DynamoDB client cannot be null");
        }
        if (tableName == null || tableName.trim().isEmpty()) {
```

```
            throw new IllegalArgumentException("Table name cannot be null or
empty");
        }
        if (replicaRegion == null) {
            throw new IllegalArgumentException("Replica region cannot be null");
        }
        if (witnessRegion == null) {
            throw new IllegalArgumentException("Witness region cannot be null");
        }

        try {
            LOGGER.info("Cleaning up MRSC replicas and witnesses for table: " +
tableName);

            // Remove replica using ReplicationGroupUpdate
            ReplicationGroupUpdate replicaUpdate =
ReplicationGroupUpdate.builder()
                .delete(DeleteReplicationGroupMemberAction.builder()
                    .regionName(replicaRegion.id())
                    .build())
                .build();

            // Remove witness
            GlobalTableWitnessGroupUpdate witnessUpdate =
GlobalTableWitnessGroupUpdate.builder()
                .delete(DeleteGlobalTableWitnessGroupMemberAction.builder()
                    .regionName(witnessRegion.id())
                    .build())
                .build();

            UpdateTableRequest updateTableRequest = UpdateTableRequest.builder()
                .tableName(tableName)
                .replicaUpdates(List.of(replicaUpdate))
                .globalTableWitnessUpdates(List.of(witnessUpdate))
                .build();

            UpdateTableResponse response =
dynamoDbClient.updateTable(updateTableRequest);
            LOGGER.info("MRSC cleanup initiated - removing replica and witness.
Response: " + response);

            return response;

        } catch (DynamoDbException e) {
```

```
            LOGGER.severe("Failed to cleanup MRSC replicas: " + tableName + " - "
+ e.getMessage());
            throw DynamoDbException.builder()
                .message("Failed to cleanup MRSC replicas: " + tableName)
                .cause(e)
                .build();
        }
    }
```

Vollständige Demonstration des MRSC-Workflows mithilfe von. AWS SDK for Java 2.x

```
    public static void demonstrateCompleteMRSCWorkflow(
        final DynamoDbClient primaryClient,
        final DynamoDbClient replicaClient,
        final String tableName,
        final Region replicaRegion,
        final Region witnessRegion)
        throws InterruptedException {

        if (primaryClient == null) {
            throw new IllegalArgumentException("Primary DynamoDB client cannot be
null");
        }
        if (replicaClient == null) {
            throw new IllegalArgumentException("Replica DynamoDB client cannot be
null");
        }
        if (tableName == null || tableName.trim().isEmpty()) {
            throw new IllegalArgumentException("Table name cannot be null or
empty");
        }
        if (replicaRegion == null) {
            throw new IllegalArgumentException("Replica region cannot be null");
        }
        if (witnessRegion == null) {
            throw new IllegalArgumentException("Witness region cannot be null");
        }

        try {
            LOGGER.info("=== Starting Complete MRSC Workflow Demonstration ===");

            // Step 1: Create an empty single-Region table
```

```
            LOGGER.info("Step 1: Creating empty single-Region table");
            createRegionalTable(primaryClient, tableName);

            // Use the existing GlobalTableOperations method for basic table
waiting
            LOGGER.info("Intermediate step: Waiting for table [" + tableName + "]
to become active before continuing");
            GlobalTableOperations.waitForTableActive(primaryClient, tableName);

            // Step 2: Convert to MRSC with replica and witness
            LOGGER.info("Step 2: Converting to MRSC with replica and witness");
            convertToMRSCWithWitness(primaryClient, tableName, replicaRegion,
witnessRegion);

            // Wait for MRSC conversion to complete using MRSC-specific waiter
            LOGGER.info("Waiting for MRSC conversion to complete...");
            waitForMRSCReplicasActive(primaryClient, tableName);

            LOGGER.info("Intermediate step: Waiting for table [" + tableName + "]
to become active before continuing");
            GlobalTableOperations.waitForTableActive(primaryClient, tableName);

            // Step 3: Verify MRSC configuration
            LOGGER.info("Step 3: Verifying MRSC configuration");
            describeMRSCTable(primaryClient, tableName);

            // Step 4: Test strong consistency with data operations
            LOGGER.info("Step 4: Testing strong consistency with data
operations");

            // Add test item to primary region
            putTestItem(primaryClient, tableName, "The Beatles", "Hey Jude", "The
Beatles 1967-1970", "1968");

            // Immediately read from replica region (no wait needed with MRSC)
            LOGGER.info("Reading from replica region immediately (strong
consistency):");
            GetItemResponse getResponse =
                getItemWithConsistentRead(replicaClient, tableName, "The
Beatles", "Hey Jude");

            if (getResponse.hasItem()) {
                LOGGER.info("# Strong consistency verified - item immediately
available in replica region");
```

```
            } else {
                LOGGER.warning("# Item not found in replica region");
            }

            // Test conditional update from replica region
            LOGGER.info("Testing conditional update from replica region:");
            performConditionalUpdate(replicaClient, tableName, "The Beatles",
"Hey Jude", "5");
            LOGGER.info("# Conditional update successful - demonstrates strong
consistency");

            // Step 5: Cleanup
            LOGGER.info("Step 5: Cleaning up resources");
            cleanupMRSCReplicas(primaryClient, tableName, replicaRegion,
witnessRegion);

            // Wait for cleanup to complete using basic table waiter
            LOGGER.info("Waiting for replica cleanup to complete...");
            GlobalTableOperations.waitForTableActive(primaryClient, tableName);

            // "Halt" until replica/witness cleanup is complete
            DescribeTableResponse cleanupVerification =
describeMRSCTable(primaryClient, tableName);
            int backoffSeconds = 5; // Start with 5 second intervals
            while (cleanupVerification.table().multiRegionConsistency() != null)
{
                LOGGER.info("Waiting additional time (" + backoffSeconds + "
seconds) for MRSC cleanup to complete...");
                tempWait(backoffSeconds);

                // Exponential backoff with cap
                backoffSeconds = Math.min(backoffSeconds * 2, 30);
                cleanupVerification = describeMRSCTable(primaryClient,
tableName);
            }

            // Delete the primary table
            deleteTable(primaryClient, tableName);

            LOGGER.info("=== MRSC Workflow Demonstration Complete ===");
            LOGGER.info("");
            LOGGER.info("Key benefits of Multi-Region Strong Consistency
(MRSC):");
```

```
            LOGGER.info("- Immediate consistency across all regions (no eventual
consistency delays)");
            LOGGER.info("- Simplified application logic (no need to handle
eventual consistency)");
            LOGGER.info("- Support for conditional writes and transactions across
regions");
            LOGGER.info("- Consistent read operations from any region without
waiting");

        } catch (DynamoDbException | InterruptedException e) {
            LOGGER.severe("MRSC workflow failed: " + e.getMessage());
            throw e;
        }
    }
```

- Weitere API-Informationen finden Sie in den folgenden Themen der AWS SDK for Java 2.x - API-Referenz.
  - CreateTable
  - DeleteTable
  - DescribeTable
  - GetItem
  - PutItem
  - UpdateItem
  - UpdateTable

Sicherheit von globalen DynamoDB-Tabellen

Globale Tabellenreplikate sind DynamoDB-Tabellen. Sie verwenden also dieselben Methoden zur Steuerung des Zugriffs auf Replikate wie für Tabellen mit einer einzelnen Region, einschließlich AWS Identity and Access Management (IAM-) Identitätsrichtlinien und ressourcenbasierter Richtlinien.

In diesem Thema wird beschrieben, wie Sie globale DynamoDB-Tabellen mithilfe von IAM-Berechtigungen und AWS Key Management Service ()AWS KMS-Verschlüsselung sichern. Sie erfahren mehr über die serviceverknüpften Rollen (SLR), die regionsübergreifende Replikation und auto-scaling ermöglichen, über die IAM-Berechtigungen, die zum Erstellen, Aktualisieren und Löschen globaler Tabellen erforderlich sind, und über die Unterschiede zwischen Multi-Region Eventual Consistency (MREC) und Multi-Region Strong Consistency (MRSC) -Tabellen. Außerdem

erfahren Sie mehr über Verschlüsselungsschlüssel, mit denen Sie die regionsübergreifende Replikation sicher verwalten können. AWS KMS

Dienstbezogene Rollen für globale Tabellen

Globale DynamoDB-Tabellen basieren auf serviceverknüpften Rollen (SLRs), um regionsübergreifende Replikations- und Autoskalierungsfunktionen zu verwalten.

Sie müssen diese Rollen nur einmal pro Konto einrichten. AWS Nach der Erstellung dienen dieselben Rollen für alle globalen Tabellen in Ihrem Konto. Weitere Informationen zu serviceverknüpften Rollen finden Sie unter Verwenden serviceverknüpfter Rollen im IAM-Benutzerhandbuch.

Mit dem Replikationsservice verknüpfte Rolle

Amazon DynamoDB erstellt automatisch die `AWSServiceRoleForDynamoDBReplication` serviceverknüpfte Rolle (SLR), wenn Sie Ihre erste globale Tabelle erstellen. Diese Rolle verwaltet die regionsübergreifende Replikation für Sie.

Wenn Sie ressourcenbasierte Richtlinien auf Replikate anwenden, stellen Sie sicher, dass Sie dem SLR-Prinzipal keine der im SLR-Prinzipal definierten Berechtigungen verweigern, da dadurch die Replikation unterbrochen wird. `AWSServiceRoleForDynamoDBReplicationPolicy` Wenn Sie die erforderlichen SLR-Berechtigungen verweigern, wird die Replikation zu und von den betroffenen Replikaten beendet, und der Status der Replikattabelle ändert sich in `REPLICATION_NOT_AUTHORIZED`.

- Bei globalen Multi-Region Eventual Consistency (MREC) -Tabellen gilt: Wenn ein Replikat länger als 20 Stunden im `REPLICATION_NOT_AUTHORIZED` Status verbleibt, wird das Replikat unwiderruflich in eine DynamoDB-Tabelle mit nur einer Region konvertiert.
- Bei globalen Tabellen mit starker Konsistenz in mehreren Regionen (Multiregion Strong Consistency, MRSC) führt das Verweigern der erforderlichen Berechtigungen zu Schreib- und stark konsistenten Lesevorgängen. `AccessDeniedException` Bleibt ein Replikat länger als sieben Tage im `REPLICATION_NOT_AUTHORIZED` Status, kann nicht mehr auf das Replikat zugegriffen werden, und Schreib- und Strongly Consistent-Lesevorgänge schlagen weiterhin mit einem Fehler fehl. Einige Managementoperationen wie das Löschen von Replikaten werden erfolgreich abgeschlossen.

Mit dem Auto Scaling Service verknüpfte Rolle

Bei der Konfiguration einer globalen Tabelle für den Modus mit bereitgestellter Kapazität muss Auto Scaling für die globale Tabelle konfiguriert werden. DynamoDB Auto Scaling verwendet den AWS Application Auto Scaling Scaling-Dienst, um die bereitgestellte Durchsatzkapazität auf Ihren globalen Tabellenreplikaten dynamisch anzupassen. Der Application Auto Scaling Scaling-Dienst erstellt eine serviceverknüpfte Rolle (SLR) mit dem Namen. AWSServiceRoleForApplicationAutoScaling_DynamoDBTable Diese serviceverknüpfte Rolle wird automatisch in Ihrem AWS Konto erstellt, wenn Sie Auto Scaling für eine DynamoDB-Tabelle zum ersten Mal konfigurieren. Es ermöglicht Application Auto Scaling, die bereitgestellte Tabellenkapazität zu verwalten und CloudWatch Alarme zu erstellen.

Achten Sie bei der Anwendung ressourcenbasierter Richtlinien auf Replikate darauf, dass Sie keine im SLR-Prinzipal für Application Auto Scaling definierten Berechtigungen verweigern, da dadurch die Auto Scaling-Funktionalität unterbrochen wird. AWSApplicationAutoscalingDynamoDBTablePolicy

Beispiel für IAM-Richtlinien für serviceverknüpfte Rollen

Eine IAM-Richtlinie mit der folgenden Bedingung hat keine Auswirkungen auf die erforderlichen Berechtigungen für die DynamoDB-Replikations-SLR und AWS Auto Scaling SLR. Diese Bedingung kann zu ansonsten weitgehend restriktiven Richtlinien hinzugefügt werden, um zu verhindern, dass Replikation oder Auto Scaling unbeabsichtigt unterbrochen werden.

Ausgenommen der erforderlichen SLR-Berechtigungen aus den Verweigerungsrichtlinien

Das folgende Beispiel zeigt, wie servicebezogene Rollenprinzipale von Deny-Statements ausgeschlossen werden können:

```
"Condition": {
    "StringNotEquals": {
        "aws:PrincipalArn": [
            "arn:aws::iam::111122223333:role/aws-service-role/
replication.dynamodb.amazonaws.com/AWSServiceRoleForDynamoDBReplication",
            "arn:aws::iam::111122223333:role/aws-service-role/dynamodb.application-
autoscaling.amazonaws.com/AWSServiceRoleForApplicationAutoScaling_DynamoDBTable"
        ]
    }
}
```

Wie globale Tabellen IAM verwenden AWS

In den folgenden Abschnitten werden die erforderlichen Berechtigungen für verschiedene globale Tabellenoperationen beschrieben und Richtlinienbeispiele bereitgestellt, die Ihnen bei der Konfiguration des entsprechenden Zugriffs für Ihre Benutzer und Anwendungen helfen sollen.

> **Note**
>
> Alle beschriebenen Berechtigungen müssen auf die spezifische Tabellenressource ARN in den betroffenen Regionen angewendet werden. Die Tabellenressource ARN folgt dem Format`arn:aws:dynamodb:region:account-id:table/table-name`, in dem Sie Ihre tatsächlichen Werte für Region, Konto-ID und Tabellennamen angeben müssen.

Themen

- Globale Tabellen erstellen und Replikate hinzufügen
- Aktualisieren von globalen Tabellen
- Löschen globaler Tabellen und Entfernen von Replikaten

Globale Tabellen erstellen und Replikate hinzufügen

Globale DynamoDB-Tabellen unterstützen zwei Konsistenzmodi: Multi-Region Eventual Consistency (MREC) und Multi-Region Strong Consistency (MRSC). Globale MREC-Tabellen können mehrere Replikate in einer beliebigen Anzahl von Regionen enthalten und sorgen so für Konsistenz. Globale MRSC-Tabellen erfordern genau drei Regionen (drei Replikate oder zwei Replikate und ein Zeuge) und bieten eine hohe Konsistenz mit einem Nullpunktziel (RPO).

Die zum Erstellen globaler Tabellen erforderlichen Berechtigungen hängen davon ab, ob Sie eine globale Tabelle mit oder ohne Zeugen erstellen.

Berechtigungen für die Erstellung globaler Tabellen

Die folgenden Berechtigungen sind sowohl für die anfängliche Erstellung globaler Tabellen als auch für das spätere Hinzufügen von Replikaten erforderlich. Diese Berechtigungen gelten sowohl für globale Tabellen mit Multi-Region Eventual Consistency (MREC) als auch für Multi-Region Strong Consistency (MRSC).

- Globale Tabellen erfordern eine regionsübergreifende Replikation, die DynamoDB über die `AWSServiceRoleForDynamoDBReplication`serviceverknüpfte Rolle (SLR) verwaltet. Die

folgende Berechtigung ermöglicht DynamoDB, diese Rolle automatisch zu erstellen, wenn Sie zum ersten Mal eine globale Tabelle erstellen:

  - `iam:CreateServiceLinkedRole`

- Um mithilfe der `UpdateTable`API eine globale Tabelle zu erstellen oder ein Replikat hinzuzufügen, benötigen Sie die folgende Berechtigung für die Quelltabellenressource:
  - `dynamodb:UpdateTable`
- Sie benötigen die folgenden Berechtigungen für die Tabellenressource in den Regionen, damit die Replikate hinzugefügt werden können:
  - `dynamodb:CreateTable`
  - `dynamodb:CreateTableReplica`
  - `dynamodb:Query`
  - `dynamodb:Scan`
  - `dynamodb:UpdateItem`
  - `dynamodb:PutItem`
  - `dynamodb:GetItem`
  - `dynamodb:DeleteItem`
  - `dynamodb:BatchWriteItem`

Zusätzliche Berechtigungen für globale MRSC-Tabellen, die einen Zeugen verwenden

Wenn Sie eine globale MRSC-Tabelle (Multi-Region Strong Consistency) mit einer Zeugenregion erstellen, benötigen Sie die folgenden Berechtigungen für die Tabellenressource in allen beteiligten Regionen (einschließlich der Replikatregionen und der Zeugenregion):

- `dynamodb:CreateGlobalTableWitness`

Beispiel für IAM-Richtlinien zum Erstellen globaler Tabellen

Erstellung einer globalen MREC- oder MRSC-Tabelle für drei Regionen

Die folgende identitätsbasierte Richtlinie ermöglicht es Ihnen, eine globale MREC- oder MRSC-Tabelle mit dem Namen „users“ in drei Regionen zu erstellen, einschließlich der Erstellung der erforderlichen mit dem DynamoDB-Replikationsdienst verknüpften Rolle.

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Sid": "AllowCreatingUsersGlobalTable",
      "Effect": "Allow",
      "Action": [
        "dynamodb:CreateTable",
        "dynamodb:CreateTableReplica",
        "dynamodb:UpdateTable",
        "dynamodb:Query",
        "dynamodb:Scan",
        "dynamodb:UpdateItem",
        "dynamodb:PutItem",
        "dynamodb:GetItem",
        "dynamodb:DeleteItem",
        "dynamodb:BatchWriteItem"
      ],
      "Resource": [
        "arn:aws:dynamodb:us-east-1:123456789012:table/users",
        "arn:aws:dynamodb:us-east-2:123456789012:table/users",
        "arn:aws:dynamodb:us-west-2:123456789012:table/users"
      ]
    },
    {
      "Sid": "AllowCreatingSLR",
      "Effect": "Allow",
      "Action": [
        "iam:CreateServiceLinkedRole"
      ],
      "Resource": [
        "arn:aws:iam::123456789012:role/aws-service-role/
replication.dynamodb.amazonaws.com/AWSServiceRoleForDynamoDBReplication"
      ]
    }
  ]
}
```

Beschränken Sie die Erstellung globaler MREC- oder MRSC-Tabellen auf bestimmte Regionen

Die folgende identitätsbasierte Richtlinie ermöglicht es Ihnen, mithilfe des RequestedRegion Bedingungsschlüssels aws: globale DynamoDB-Tabellen in bestimmten Regionen zu erstellen, einschließlich der Erstellung der erforderlichen mit dem DynamoDB-Replikationsdienst verknüpften Rolle.

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Sid": "AllowAddingReplicasToSourceTable",
      "Effect": "Allow",
      "Action": [
        "dynamodb:UpdateTable"
      ],
      "Resource": "*",
      "Condition": {
        "StringEquals": {
          "aws:RequestedRegion": [
            "us-east-1"
          ]
        }
      }
    },
    {
      "Sid": "AllowCreatingReplicas",
      "Effect": "Allow",
      "Action": [
        "dynamodb:CreateTable",
        "dynamodb:CreateTableReplica",
        "dynamodb:UpdateTable",
        "dynamodb:Query",
        "dynamodb:Scan",
        "dynamodb:UpdateItem",
        "dynamodb:PutItem",
        "dynamodb:GetItem",
        "dynamodb:DeleteItem",
        "dynamodb:BatchWriteItem"
      ],
```

```
      "Resource": "*",
      "Condition": {
        "StringEquals": {
          "aws:RequestedRegion": [
            "us-east-2",
            "us-west-2"
          ]
        }
      }
    },
    {
      "Sid": "AllowCreatingSLR",
      "Effect": "Allow",
      "Action": [
        "iam:CreateServiceLinkedRole"
      ],
      "Resource": [
        "arn:aws:iam::123456789012:role/aws-service-role/
replication.dynamodb.amazonaws.com/AWSServiceRoleForDynamoDBReplication"
      ]
    }
  ]
}
```

Erstellen einer globalen MRSC-Tabelle mit Zeuge

Die folgende identitätsbasierte Richtlinie ermöglicht es Ihnen, eine globale DynamoDB-MRSC-Tabelle mit dem Namen „users“ mit Replikaten in us-east-1 und us-east-2 und einem Zeugen in us-west-2 zu erstellen, einschließlich der Erstellung der erforderlichen mit dem DynamoDB-Replikationsdienst verknüpften Rolle.

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Sid": "AllowCreatingUsersGlobalTableWithWitness",
      "Effect": "Allow",
      "Action": [
        "dynamodb:CreateTable",
```

```
        "dynamodb:CreateTableReplica",
        "dynamodb:CreateGlobalTableWitness",
        "dynamodb:UpdateTable",
        "dynamodb:Query",
        "dynamodb:Scan",
        "dynamodb:UpdateItem",
        "dynamodb:PutItem",
        "dynamodb:GetItem",
        "dynamodb:DeleteItem",
        "dynamodb:BatchWriteItem"
      ],
      "Resource": [
        "arn:aws:dynamodb:us-east-1:123456789012:table/users",
        "arn:aws:dynamodb:us-east-2:123456789012:table/users"
      ]
    },
    {
      "Sid": "AllowCreatingSLR",
      "Effect": "Allow",
      "Action": [
        "iam:CreateServiceLinkedRole"
      ],
      "Resource": [
        "arn:aws:iam::123456789012:role/aws-service-role/
replication.dynamodb.amazonaws.com/AWSServiceRoleForDynamoDBReplication"
      ]
    }
  ]
}
```

Beschränkung der Erstellung von MRSC-Zeugen auf bestimmte Regionen

Diese identitätsbasierte Richtlinie ermöglicht es Ihnen, eine globale MRSC-Tabelle mit Replikaten zu erstellen, die auf bestimmte Regionen beschränkt sind, indem Sie den RequestedRegion Bedingungsschlüssel aws: verwenden und uneingeschränkte Zeugenerstellung für alle Regionen verwenden, einschließlich der Erstellung der erforderlichen, mit dem DynamoDB-Replikationsdienst verknüpften Rolle.

JSON

```
{
```

```
  "Version":"2012-10-17",
  "Statement": [
    {
      "Sid": "AllowCreatingReplicas",
      "Effect": "Allow",
      "Action": [
        "dynamodb:CreateTable",
        "dynamodb:CreateTableReplica",
        "dynamodb:UpdateTable",
        "dynamodb:Query",
        "dynamodb:Scan",
        "dynamodb:UpdateItem",
        "dynamodb:PutItem",
        "dynamodb:GetItem",
        "dynamodb:DeleteItem",
        "dynamodb:BatchWriteItem"
      ],
      "Resource": "*",
      "Condition": {
        "StringEquals": {
          "aws:RequestedRegion": [
            "us-east-1",
            "us-east-2"
          ]
        }
      }
    },
    {
      "Sid": "AllowCreatingWitness",
      "Effect": "Allow",
      "Action": [
        "dynamodb:CreateGlobalTableWitness"
      ],
      "Resource": "*"
    },
    {
      "Sid": "AllowCreatingSLR",
      "Effect": "Allow",
      "Action": [
        "iam:CreateServiceLinkedRole"
      ],
      "Resource": [
        "arn:aws:iam::123456789012:role/aws-service-role/
replication.dynamodb.amazonaws.com/AWSServiceRoleForDynamoDBReplication"
```

```
      ]
    }
  ]
}
```

Aktualisieren von globalen Tabellen

Um die Replikateinstellungen für eine bestehende globale Tabelle mithilfe der UpdateTableAPI zu ändern, benötigen Sie die folgende Berechtigung für die Tabellenressource in der Region, in der Sie den API-Aufruf tätigen:

- `dynamodb:UpdateTable`

Sie können auch andere globale Tabellenkonfigurationen aktualisieren, z. B. Richtlinien für auto Skalierung und Time-to-Live-Einstellungen. Für diese zusätzlichen Aktualisierungsvorgänge sind die folgenden Berechtigungen erforderlich:

- Um eine Replica-Auto-Scaling-Richtlinie mit der UpdateTableReplicaAutoScalingAPI zu aktualisieren, benötigen Sie die folgenden Berechtigungen für die Tabellenressource in allen Regionen, die Replikate enthalten:
  - `application-autoscaling:DeleteScalingPolicy`
  - `application-autoscaling:DeleteScheduledAction`
  - `application-autoscaling:DeregisterScalableTarget`
  - `application-autoscaling:DescribeScalableTargets`
  - `application-autoscaling:DescribeScalingActivities`
  - `application-autoscaling:DescribeScalingPolicies`
  - `application-autoscaling:DescribeScheduledActions`
  - `application-autoscaling:PutScalingPolicy`
  - `application-autoscaling:PutScheduledAction`
  - `application-autoscaling:RegisterScalableTarget`
- Um die Time-to-Live-Einstellungen mit der UpdateTimeToLiveAPI zu aktualisieren, benötigen Sie die folgenden Berechtigungen für die Tabellenressource in allen Regionen, die Replikate enthalten:
  - `dynamodb:UpdateTimeToLive`

Beachten Sie, dass Time to Live (TTL) nur für globale Tabellen unterstützt wird, die mit Multi-Region Eventual Consistency (MREC) konfiguriert sind. Weitere Informationen zur Funktionsweise globaler Tabellen mit TTL finden Sie unter So funktionieren globale DynamoDB-Tabellen.

Löschen globaler Tabellen und Entfernen von Replikaten

Um eine globale Tabelle zu löschen, müssen Sie alle Replikate entfernen. Die für diesen Vorgang erforderlichen Berechtigungen hängen davon ab, ob Sie eine globale Tabelle mit oder ohne Zeugenregion löschen.

Berechtigungen zum Löschen globaler Tabellen und zum Entfernen von Replikaten

Die folgenden Berechtigungen sind sowohl für das Entfernen einzelner Replikate als auch für das vollständige Löschen globaler Tabellen erforderlich. Durch das Löschen einer globalen Tabellenkonfiguration wird nur die Replikationsbeziehung zwischen Tabellen in verschiedenen Regionen entfernt. Die zugrunde liegende DynamoDB-Tabelle in der letzten verbleibenden Region wird nicht gelöscht. Die Tabelle in der letzten Region existiert weiterhin als standardmäßige DynamoDB-Tabelle mit denselben Daten und Einstellungen. Diese Berechtigungen gelten sowohl für globale Tabellen mit Multi-Region Eventual Consistency (MREC) als auch für Multi-Region Strong Consistency (MRSC).

- Um mithilfe der `UpdateTable`API Replikate aus einer globalen Tabelle zu entfernen, benötigen Sie die folgende Berechtigung für die Tabellenressource in der Region, von der aus Sie den API-Aufruf tätigen:
  - `dynamodb:UpdateTable`
- Sie benötigen die folgenden Berechtigungen für die Tabellenressource in jeder Region, in der Sie ein Replikat entfernen:
  - `dynamodb:DeleteTable`
  - `dynamodb:DeleteTableReplica`

Zusätzliche Berechtigungen für globale MRSC-Tabellen, die einen Zeugen verwenden

Um eine globale MRSC-Tabelle (Multiregion Strong Consistency) mit einem Zeugen zu löschen, benötigen Sie in allen beteiligten Regionen (einschließlich der Replikatregionen und der Zeugenregion) die folgenden Berechtigungen für die Tabellenressource:

- `dynamodb:DeleteGlobalTableWitness`

Beispiele für IAM-Richtlinien zum Löschen globaler Tabellenreplikate

Löschen globaler Tabellenreplikate

Diese identitätsbasierte Richtlinie ermöglicht es Ihnen, eine globale DynamoDB-Tabelle mit dem Namen „users“ und ihre Replikate in drei Regionen zu löschen:

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "dynamodb:UpdateTable",
        "dynamodb:DeleteTable",
        "dynamodb:DeleteTableReplica"
      ],
      "Resource": [
        "arn:aws:dynamodb:us-east-1:123456789012:table/users",
        "arn:aws:dynamodb:us-east-2:123456789012:table/users",
        "arn:aws:dynamodb:us-west-2:123456789012:table/users"
      ]
    }
  ]
}
```

Löschen einer globalen MRSC-Tabelle mit einem Zeugen

Mit dieser identitätsbasierten Richtlinie können Sie das Replikat und den Zeugen einer globalen MRSC-Tabelle mit dem Namen „users“ löschen:

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
```

```
      "Action": [
        "dynamodb:UpdateTable",
        "dynamodb:DeleteTable",
        "dynamodb:DeleteTableReplica",
        "dynamodb:DeleteGlobalTableWitness"
      ],
      "Resource": [
        "arn:aws:dynamodb:us-east-1:123456789012:table/users",
        "arn:aws:dynamodb:us-east-2:123456789012:table/users"
      ]
    }
  ]
}
```

Wie verwenden globale Tabellen AWS KMS

Wie alle DynamoDB-Tabellen verschlüsseln Replikate Daten im Ruhezustand immer mit Verschlüsselungsschlüsseln, die im AWS Key Management Service () gespeichert sind.AWS KMS

Alle Replikate in einer globalen Tabelle müssen mit demselben Typ von KMS-Schlüssel (AWS -eigener Schlüssel, von AWS verwalteter Schlüssel oder kundenseitig verwalteter Schlüssel) konfiguriert werden.

> **Important**
>
> DynamoDB benötigt Zugriff auf den Verschlüsselungsschlüssel des Replikats, um ein Replikat zu löschen. Wenn Sie einen kundenseitig verwalteten Schlüssel, der zur Verschlüsselung eines Replikats verwendet wird, deaktivieren oder löschen möchten, weil Sie das Replikat löschen, sollten Sie zuerst das Replikat löschen, warten, bis sich der Tabellenstatus in einem der verbleibenden Replikate in `ACTIVE` ändert, und dann den Schlüssel deaktivieren oder löschen.

Wenn Sie bei einer globalen MREC-Tabelle den Zugriff von DynamoDB auf einen kundenseitig verwalteten Schlüssel, der zur Verschlüsselung eines Replikats verwendet wird, deaktivieren oder entziehen, wird die Replikation zum und vom Replikat beendet und der Replikatstatus ändert sich in `INACCESSIBLE_ENCRYPTION_CREDENTIALS`. Wenn ein Replikat in einer globalen MREC-Tabelle länger als 20 Stunden im Status `INACCESSIBLE_ENCRYPTION_CREDENTIALS` verbleibt, wird das Replikat unwiderruflich in eine DynamoDB-Tabelle mit einer einzelnen Region konvertiert.

Wenn Sie bei einer globalen MRSC-Tabelle den Zugriff von DynamoDB auf einen kundenseitig verwalteten Schlüssel, der zur Verschlüsselung eines Replikats verwendet wird, deaktivieren oder entziehen, wird die Replikation zum und vom Replikat beendet, bei dem Versuch, strikt konsistente Lesevorgänge in das Replikat zu schreiben, tritt ein Fehler auf, und der Replikatstatus ändert sich in `INACCESSIBLE_ENCRYPTION_CREDENTIALS`. Wenn ein Replikat in einer globalen MRSC-Tabelle länger als sieben Tage im `INACCESSIBLE_ENCRYPTION_CREDENTIALS` Status verbleibt, wird das Replikat je nach den widerrufenen Berechtigungen archiviert oder es kann dauerhaft nicht mehr darauf zugegriffen werden.

DynamoDB-Tabellen für mehrere Konten

Globale Tabellen mit mehreren Konten replizieren Ihre DynamoDB-Tabellendaten automatisch über mehrere AWS Regionen und mehrere AWS Konten hinweg, um die Ausfallsicherheit zu verbessern, Workloads auf Kontoebene zu isolieren und unterschiedliche Sicherheits- und Governance-Kontrollen anzuwenden. Jede Replikattabelle befindet sich in einem eigenen AWS Konto, wodurch eine Fehlerisolierung sowohl auf Regions- als auch auf Kontoebene ermöglicht wird. Sie können Replikate auch an Ihre AWS Organisationsstruktur anpassen. Globale Tabellen mit mehreren Konten bieten zusätzliche Isolations-, Verwaltungs- und Sicherheitsvorteile im Vergleich zu globalen Tabellen mit demselben Konto.

Globale Tabellen mit mehreren Konten bieten die folgenden Vorteile:

- Automatisches Replizieren von DynamoDB-Tabellendaten für Konten und Regionen Ihrer AWS Wahl
- Verbessern Sie Sicherheit und Governance, indem Sie Daten kontenübergreifend mit unterschiedlichen Richtlinien, Richtlinien und Compliance-Grenzen replizieren
- Verbessern Sie die betriebliche Stabilität und die Fehlerisolierung auf Kontoebene, indem Sie Replikate auf separaten Konten platzieren AWS
- Richten Sie die Workloads nach Geschäftseinheit oder Eigentümer aus, wenn Sie eine Strategie für mehrere Konten verwenden
- Vereinfachen Sie die Kostenzuweisung, indem Sie jedes Replikat seinem jeweiligen Konto in Rechnung stellen AWS

Weitere Informationen finden Sie unter Vorteile der Verwendung mehrerer AWS Konten. Wenn für Ihre Workloads keine Replikation mehrerer Konten erforderlich ist oder Sie eine einfachere Replikatverwaltung mit lokalen Überschreibungen wünschen, können Sie weiterhin globale Tabellen mit demselben Konto verwenden.

Sie können globale Tabellen mit mehreren Konten konfigurieren. Multi-Region Eventual Consistency (MREC) Für konfigurierte globale Tabellen unterstützen das Modell mit mehreren Konten Multi-Region Strong Consistency (MRSC) nicht.

Themen

- So funktionieren globale DynamoDB-Tabellen
- Tutorials: Globale Tabellen mit mehreren Konten erstellen
- Sicherheit von globalen DynamoDB-Tabellen

So funktionieren globale DynamoDB-Tabellen

Globale Tabellen mit mehreren Konten erweitern die vollständig verwalteten, serverlosen, regionsübergreifenden und multiaktiven Funktionen von DynamoDB auf mehrere Konten. AWS Globale Tabellen mit mehreren Konten replizieren Daten über AWS Regionen und Konten hinweg und bieten dabei dieselbe aktive und aktive Funktionalität wie globale Tabellen mit demselben Konto. Wenn Sie in ein Replikat schreiben, repliziert DynamoDB die Daten auf alle anderen Replikate.

Zu den wichtigsten Unterschieden zu globalen Tabellen mit demselben Konto gehören:

- Die Replikation mehrerer Konten wird für globale Tabellen mit Multi-Region Eventual Consistency (MREC) unterstützt.
- Sie können Replikate nur hinzufügen, indem Sie mit einer Tabelle mit einer einzigen Region beginnen. Die Konvertierung einer vorhandenen globalen Tabelle mit demselben Konto in eine Konfiguration mit mehreren Konten wird nicht unterstützt. Um zu migrieren, müssen Sie vorhandene Replikate löschen, um zu einer Tabelle mit nur einer Region zurückzukehren, bevor Sie eine neue globale Tabelle mit mehreren Konten erstellen.
- Jedes Replikat muss sich in einem separaten Konto befinden. AWS Für eine globale Tabelle mit mehreren Konten und N Replikaten benötigen Sie N Konten.
- Globale Tabellen mit mehreren Konten verwenden standardmäßig einheitliche Tabelleneinstellungen für alle Replikate. Alle Replikate verwenden automatisch dieselbe Konfiguration (wie Durchsatzmodus, TTL und PITR), und im Gegensatz zu globalen Tabellen mit demselben Konto können diese Einstellungen nicht pro Replikat außer Kraft gesetzt werden.
- Kunden müssen in ihren Ressourcenrichtlinien Replikationsberechtigungen für den DynamoDB-Dienstprinzipal für globale Tabellen bereitstellen.

Globale Tabellen mit mehreren Konten verwenden dieselbe zugrunde liegende Replikationstechnologie wie globale Tabellen mit demselben Konto. Tabelleneinstellungen werden automatisch auf alle regionalen Replikate repliziert, und Kunden können die Einstellungen nicht für jedes Replikat überschreiben oder anpassen. Dies gewährleistet eine konsistente Konfiguration und ein vorhersehbares Verhalten für mehrere AWS Konten, die an derselben globalen Tabelle teilnehmen.

Einstellungen in globalen DynamoDB-Tabellen definieren, wie sich eine Tabelle verhält und wie Daten regionsübergreifend repliziert werden. Diese Einstellungen werden während der Tabellenerstellung oder APIs beim Hinzufügen eines neuen regionalen Replikats über die DynamoDB-Steuerebene konfiguriert.

Bei der Erstellung einer globalen Tabelle mit mehreren Konten müssen Kunden `GlobalTableSettingsReplicationMode = ENABLED` für jedes regionale Replikat Einstellungen vornehmen. Dadurch wird sichergestellt, dass in einer Region vorgenommene Konfigurationsänderungen automatisch auf alle anderen Regionen übertragen werden, die Teil der globalen Tabelle sind.

Sie können die Replikation der Einstellungen nach der Tabellenerstellung aktivieren. Dies unterstützt das Szenario, in dem eine Tabelle ursprünglich als regionale Tabelle erstellt und später zu einer globalen Tabelle mit mehreren Konten aktualisiert wurde.

Synchronisierte Einstellungen

Die folgenden Tabelleneinstellungen werden immer für alle Replikate in einer globalen Tabelle mit mehreren Konten synchronisiert:

> **Note**
>
> Im Gegensatz zu globalen Tabellen mit demselben Konto lassen globale Tabellen mit mehreren Konten keine regionsspezifischen Überschreibungen für diese Einstellungen zu. Die einzige Ausnahme besteht darin, dass Überschreibungen für Richtlinien zur auto-scaling von Lesevorgängen (Tabellen und GSIs) zulässig sind, da es sich um separate externe Ressourcen handelt.

- Kapazitätsmodus (bereitgestellte oder On-Demand-Kapazität)
- Von der Tabelle bereitgestellte Lese- und Schreibkapazität
- auto Skalierung beim Lesen und Schreiben von Tabellen

- Definition des lokalen sekundären Index (LSI)
- Definition des globalen sekundären Index (GSI)
- Von GSI bereitgestellte Lese- und Schreibkapazität
- GSI Auto Scaling mit Lese- und Schreibzugriff
- Streams-Definition im MREC-Modus
- Time to Live (TTL)
- Warmdurchsatz
- Maximaler Lese- und Schreibdurchsatz auf Abruf

Nicht synchronisierte Einstellungen

Die folgenden Einstellungen werden nicht zwischen Replikaten synchronisiert und müssen für jede Replikattabelle in jeder Region unabhängig konfiguriert werden.

- Tabellenklasse
- Typ der serverseitigen Verschlüsselung (Server-Side Encryption, SSE)
- KMS-Schlüssel-ID für serverseitige Verschlüsselung (SSE)
- Löschschutz
- Kinesis Data Streams (KDSD)
- Tags (Markierungen)
- Ressourcenrichtlinie
- Tabelle Cloudwatch-Contributor Insights (CCI)
- Einblicke in GSI Cloudwatch—Contributor (CCI)

Überwachen

In globalen Tabellen, die für Multi-Region Eventual Consistency (MREC) konfiguriert sind, wird die Metrik veröffentlicht. `ReplicationLatency` CloudWatch Diese Metrik misst die verstrichene Zeit zwischen dem Schreiben eines Elements in eine Replikattabelle und dem Erscheinen dieses Elements in einem anderen Replikat in der globalen Tabelle. `ReplicationLatency` wird in Millisekunden ausgedrückt und für jedes Paar aus Quell- und Zielregion ausgegeben.

Typische `ReplicationLatency` Werte hängen von der Entfernung zwischen den ausgewählten AWS Regionen sowie von anderen Variablen wie Workload-Typ und Durchsatz ab. Beispiel: Ein

Quellreplikat in der Region USA West (Nordkalifornien) (us-west-1) weist im Vergleich zur Region Afrika (Kapstadt) (af-south-1) einen niedrigeren Wert für `ReplicationLatency` auf als die Region USA West (Oregon) (us-west-2).

Ein erhöhter Wert für `ReplicationLatency` könnte darauf hinweisen, dass Updates von einem Replikat nicht in einem angemessenen Zeitraum an andere Replikattabellen verteilt werden. In diesem Fall können Sie die Lese- und Schreibaktivität Ihrer Anwendung vorübergehend in eine andere AWS Region umleiten.

Behandlung von Problemen mit der Replikationslatenz in globalen Tabellen mit mehreren Konten

Wenn aufgrund von vom Kunden verursachten Problemen mit einer Replikattabelle `ReplicationLatency` mehr als 3 Stunden vergangen sind, sendet DynamoDB eine Benachrichtigung, in der der Kunde aufgefordert wird, das zugrunde liegende Problem zu beheben. Zu den häufigsten kundenbedingten Problemen, die eine Replikation verhindern können, gehören:

- Entfernen der erforderlichen Berechtigungen aus der Ressourcenrichtlinie der Replikattabelle
- Abmeldung von einer AWS Region, in der ein Replikat der globalen Tabelle mit mehreren Konten gehostet wird
- Verweigerung der für die Entschlüsselung von Daten erforderlichen AWS KMS-Schlüsselberechtigungen für die Tabelle

DynamoDB sendet innerhalb von 3 Stunden bei erhöhter Replikationslatenz eine erste Benachrichtigung, gefolgt von einer zweiten Benachrichtigung nach 20 Stunden, falls das Problem weiterhin besteht. Wenn das Problem nicht innerhalb des erforderlichen Zeitfensters behoben wird, trennt DynamoDB das Replikat automatisch von der globalen Tabelle. Das betroffene Replikat wird dann in eine regionale Tabelle konvertiert.

Tutorials: Globale Tabellen mit mehreren Konten erstellen

Dieser Abschnitt enthält step-by-step Anweisungen zum Erstellen globaler DynamoDB-Tabellen, die sich über mehrere AWS Konten erstrecken.

Erstellen Sie mit der DynamoDB-Konsole eine globale Tabelle mit mehreren Konten

Gehen Sie wie folgt vor, um eine globale Tabelle mit mehreren Konten mithilfe von zu erstellen. AWS-Managementkonsole Im folgenden Beispiel wird eine globale Tabelle mit Replikattabellen in den Vereinigte Staaten erstellt.

1. Melden Sie sich bei der an AWS-Managementkonsole und öffnen Sie die DynamoDB-Konsole unter https://console.aws.amazon.com/dynamodb/für das erste Konto (sagen wir*111122223333*).
2. Wählen Sie für dieses Beispiel in der Regionsauswahl in der Navigationsleiste die Region USA Ost (Ohio) aus.
3. Klicken Sie im Navigationsbereich auf der linken Seite der Konsole auf Tabellen.
4. Wählen Sie Tabelle erstellen aus.
5. Gehen Sie auf der Seite Tabelle erstellen wie folgt vor:
   a. Geben Sie unter Tabellenname **`MusicTable`** ein.
   b. Geben Sie unter Partitionsschlüssel den Wert **`Artist`** ein.
   c. Geben Sie unter (Sortierschlüssel) den Wert **`SongTitle`** ein.
   d. Behalten Sie die übrigen Standardeinstellungen bei und wählen Sie Tabelle erstellen aus.
6. Fügen Sie der Tabelle die folgende Ressourcenrichtlinie hinzu

```
{
"Version": "2012-10-17",
"Statement": [
    {
        "Sid": "DynamoDBActionsNeededForSteadyStateReplication",
        "Effect": "Allow",
        "Action": [
            "dynamodb:ReadDataForReplication",
            "dynamodb:WriteDataForReplication",
            "dynamodb:ReplicateSettings"
        ],
        "Resource": "arn:aws:dynamodb:us-east-2:111122223333:table/MusicTable",
        "Principal": {"Service": ["replication.dynamodb.amazonaws.com"]},
        "Condition": {
            "StringEquals": {
                "aws:SourceAccount": ["444455556666","111122223333"],
                "aws:SourceArn": [
                    "arn:aws:dynamodb:us-east-1:444455556666:table/MusicTable",
                    "arn:aws:dynamodb:us-east-2:111122223333:table/MusicTable"
                ]
            }
        }
    },
    {
```

```
        "Sid": "AllowTrustedAccountsToJoinThisGlobalTable",
        "Effect": "Allow",
        "Action": [
            "dynamodb:AssociateTableReplica"
        ],
        "Resource": "arn:aws:dynamodb:us-east-2:111122223333:table/MusicTable",
        "Principal": {"AWS": ["444455556666"]}
    }
]
}
```

7. Diese neue Tabelle dient als erste Replikattabelle in einer neuen globalen Tabelle. Sie stellt den Prototyp für andere Replikattabellen dar, die Sie später hinzufügen.
8. Warten Sie, bis die Tabelle aktiv wird. Navigieren Sie für die neu erstellte Tabelle auf der Registerkarte Globale Tabellen zu Einstellungen > Replikation und klicken Sie auf Aktivieren.
9. Melden Sie sich von diesem Konto ab (*111122223333*hier).
10. Melden Sie sich bei der an AWS-Managementkonsole und öffnen Sie die DynamoDB-Konsole unter https://console.aws.amazon.com/dynamodb/für das zweite Konto (z. B.). *444455556666*
11. Wählen Sie für dieses Beispiel in der Navigationsleiste in der Regionsauswahl die Option US East (Nord-Virginia) aus.
12. Die Konsole stellt sicher, dass die ausgewählte Region keine Tabelle mit demselben Namen enthält. Wenn eine Tabelle mit demselben Namen vorhanden ist, müssen Sie die vorhandene Tabelle löschen, bevor Sie eine neue Replikattabelle in der betreffenden Region erstellen können.
13. Wählen Sie in der Dropdownliste neben Tabelle erstellen die Option Aus einem anderen Konto erstellen
14. Gehen Sie auf der Seite Tabelle aus einem anderen Konto erstellen wie folgt vor:
    a. Fügen Sie **arn:aws:dynamodb:us-east-2:*111122223333*:table/MusicTable** als Tabelle arn für die Quelltabelle hinzu.
    b. Fügen Sie in der Replikattabelle ARNs erneut **arn:aws:dynamodb:us-east-2:*111122223333*:table/MusicTable** den ARN der Quelltabelle hinzu. Wenn bereits mehrere Replikate als Teil einer globalen Tabelle mit mehreren Konten vorhanden sind, müssen Sie jedes vorhandene Replikat dem ReplicaTable ARN hinzufügen.
    c. Behalten Sie die anderen Standardeinstellungen bei und wählen Sie Submit.
15. Auf der Registerkarte „Globale Tabellen“ für die Tabelle „Musik“ (und für alle anderen Replikattabellen) wird angezeigt, dass die Tabelle in mehreren Regionen repliziert wurde.

16. So testen Sie die Replikation:

    a. Sie können jede Region, in der ein Replikat vorhanden ist, für diese Tabelle verwenden
    b. Wählen Sie Tabellenelemente erkunden aus.
    c. Wählen Sie Element erstellen aus.
    d. Geben Sie **`item_1`** für Künstler und **`Song Value 1`** für ein SongTitle.
    e. Wählen Sie Element erstellen aus.
    f. Überprüfen Sie die Replikation, indem Sie zu den anderen Regionen wechseln:
    g. Vergewissern Sie sich, dass die Tabelle Music das von Ihnen erstellte Objekt enthält.

Erstellen Sie eine globale Tabelle mit mehreren Konten mithilfe der AWS CLI

Die folgenden Beispiele zeigen, wie Sie mithilfe von eine globale Tabelle mit mehreren Konten erstellen. AWS CLI Diese Beispiele veranschaulichen den vollständigen Arbeitsablauf für die Einrichtung der kontenübergreifenden Replikation.

CLI

Verwenden Sie die folgenden AWS CLI Befehle, um eine globale Tabelle mit mehreren Konten und kontenübergreifender Replikation zu erstellen.

```
# STEP 1: Setting resource policy for the table in account 111122223333

cat > /tmp/source-resource-policy.json << 'EOF'
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "DynamoDBActionsNeededForSteadyStateReplication",
            "Effect": "Allow",
            "Action": [
                "dynamodb:ReadDataForReplication",
                "dynamodb:WriteDataForReplication",
                "dynamodb:ReplicateSettings"
            ],
            "Resource": "arn:aws:dynamodb:us-east-2:111122223333:table/MusicTable",
            "Principal": {"Service": ["replication.dynamodb.amazonaws.com"]},
            "Condition": {
                "StringEquals": {
                    "aws:SourceAccount": ["444455556666","111122223333"],
```

```
                    "aws:SourceArn": [
                        "arn:aws:dynamodb:us-east-1:444455556666:table/MusicTable",
                        "arn:aws:dynamodb:us-east-2:111122223333:table/MusicTable"
                    ]
                }
            }
        },
        {
            "Sid": "AllowTrustedAccountsToJoinThisGlobalTable",
            "Effect": "Allow",
            "Action": [
                "dynamodb:AssociateTableReplica"
            ],
            "Resource": "arn:aws:dynamodb:us-east-2:111122223333:table/MusicTable",
            "Principal": {"AWS": ["444455556666"]}
        }
    ]
}
EOF

# Step 2: Create a new table (MusicTable) in US East (Ohio),
#   with DynamoDB Streams enabled (NEW_AND_OLD_IMAGES),
#   and Settings Replication ENABLED on the account 111122223333

aws dynamodb create-table \
    --table-name MusicTable \
    --attribute-definitions \
        AttributeName=Artist,AttributeType=S \
        AttributeName=SongTitle,AttributeType=S \
    --key-schema \
        AttributeName=Artist,KeyType=HASH \
        AttributeName=SongTitle,KeyType=RANGE \
    --billing-mode PAY_PER_REQUEST \
    --stream-specification StreamEnabled=true,StreamViewType=NEW_AND_OLD_IMAGES \
    --global-table-settings-replication-mode ENABLED \
    --resource-policy file:///tmp/source-resource-policy.json \
    --region us-east-2


# Step 3: Creating replica table in account 444455556666

# Resource policy for account 444455556666
cat > /tmp/dest-resource-policy.json << 'EOF'
{
```

```
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "DynamoDBActionsNeededForSteadyStateReplication",
            "Effect": "Allow",
            "Action": [
                "dynamodb:ReadDataForReplication",
                "dynamodb:WriteDataForReplication",
                "dynamodb:ReplicateSettings"
            ],
            "Resource": "arn:aws:dynamodb:us-east-1:444455556666:table/MusicTable",
            "Principal": {"Service": ["replication.dynamodb.amazonaws.com"]},
            "Condition": {
                "StringEquals": {
                    "aws:SourceAccount": ["444455556666","111122223333"],
                    "aws:SourceArn": [
                        "arn:aws:dynamodb:us-east-1:444455556666:table/MusicTable",
                        "arn:aws:dynamodb:us-east-2:111122223333:table/MusicTable"
                    ]
                }
            }
        }
    ]
}
EOF

# Execute the replica table creation
aws dynamodb create-table \
    --table-name MusicTable \
    --global-table-source-arn "arn:aws:dynamodb:us-east-2:111122223333:table/
MusicTable" \
    --resource-policy file:///tmp/dest-resource-policy.json \
    --global-table-settings-replication-mode ENABLED \
    --region us-east-1

# Step 4: View the list of replicas created using describe-table
aws dynamodb describe-table \
    --table-name MusicTable \
    --region us-east-2 \
    --query 'Table.
{TableName:TableName,TableStatus:TableStatus,MultiRegionConsistency:MultiRegionConsistency,R
{Region:RegionName,Status:ReplicaStatus}}'
```

```
# Step 5: To verify that replication is working, add a new item to the Music table
 in US East (Ohio)
aws dynamodb put-item \
    --table-name MusicTable \
    --item '{"Artist": {"S":"item_1"},"SongTitle": {"S":"Song Value 1"}}' \
    --region us-east-2

# Step 6: Wait for a few seconds, and then check to see whether the item has been
# successfully replicated to US East (N. Virginia) and Europe (Ireland)
aws dynamodb get-item \
    --table-name MusicTable \
    --key '{"Artist": {"S":"item_1"},"SongTitle": {"S":"Song Value 1"}}' \
    --region us-east-1

aws dynamodb get-item \
    --table-name MusicTable \
    --key '{"Artist": {"S":"item_1"},"SongTitle": {"S":"Song Value 1"}}' \
    --region us-east-2

# Step 7: Delete the replica table in US East (N. Virginia) Region
aws dynamodb delete-table \
    --table-name MusicTable \
    --region us-east-1

# Clean up: Delete the primary table
aws dynamodb delete-table \
    --table-name MusicTable \
    --region us-east-2
```

Sicherheit von globalen DynamoDB-Tabellen

Globale Tabellenreplikate sind DynamoDB-Tabellen. Sie verwenden also dieselben Methoden zur Steuerung des Zugriffs auf Replikate wie für Tabellen mit einer einzelnen Region, einschließlich AWS Identity and Access Management (IAM-) Identitätsrichtlinien und ressourcenbasierter Richtlinien. In diesem Thema wird beschrieben, wie Sie globale DynamoDB-Tabellen mit mehreren Konten mithilfe von IAM-Berechtigungen und AWS Key Management Service () -Verschlüsselung sichern.AWS KMS Sie erfahren mehr über die ressourcenbasierten Richtlinien und serviceverknüpften Rollen (SLR), die eine regionsübergreifende kontenübergreifende Replikation und auto-scaling ermöglichen, sowie über die IAM-Berechtigungen, die zum Erstellen, Aktualisieren und Löschen globaler Tabellen für MREC-Tabellen (Multi-Region Eventual Consistency) erforderlich sind. Außerdem erfahren Sie mehr

über Verschlüsselungsschlüssel, mit denen Sie die regionsübergreifende Replikation sicher verwalten können. AWS KMS

Es enthält detaillierte Informationen zu den ressourcenbasierten Richtlinien und Berechtigungen, die für die Einrichtung der konto- und regionsübergreifenden Tabellenreplikation erforderlich sind. Das Verständnis dieses Sicherheitsmodells ist für Kunden, die sichere, kontoübergreifende Datenreplikationslösungen implementieren müssen, von entscheidender Bedeutung.

Serviceprinzipalautorisierung für die Replikation

Die globalen Tabellen von DynamoDB mit mehreren Konten verwenden einen eigenen Autorisierungsansatz, da die Replikation über Kontogrenzen hinweg durchgeführt wird. Dies erfolgt mithilfe des Replikationsdienstprinzipals von DynamoDB:. `replication.dynamodb.amazonaws.com` Jedes teilnehmende Konto muss diesen Prinzipal in der Ressourcenrichtlinie der Replikattabelle explizit zulassen und ihm damit Berechtigungen geben, die durch Quellkontextbedingungen für Schlüssel wie`aws:SourceAccount`, `aws:SourceArn` usw. auf bestimmte Replikate beschränkt werden können. Weitere Informationen finden Sie unter AWS globale Bedingungsschlüssel. Die Berechtigungen sind bidirektional, was bedeutet, dass sich alle Replikate gegenseitig ausdrücklich Berechtigungen gewähren müssen, bevor die Replikation für ein bestimmtes Replikatpaar eingerichtet werden kann.

Die folgenden Dienstprinzipalberechtigungen sind für die kontenübergreifende Replikation unerlässlich:

- `dynamodb:ReadDataForReplication`gewährt die Möglichkeit, Daten für Replikationszwecke zu lesen. Mit dieser Berechtigung können Änderungen an einem Replikat gelesen und an andere Replikate weitergegeben werden.
- `dynamodb:WriteDataForReplication`ermöglicht das Schreiben replizierter Daten in Zieltabellen. Mit dieser Berechtigung können Änderungen zwischen allen Replikaten in der globalen Tabelle synchronisiert werden.
- `dynamodb:ReplicateSettings`ermöglicht die Synchronisation von Tabelleneinstellungen zwischen Replikaten und sorgt so für eine konsistente Konfiguration in allen beteiligten Tabellen.

Jedes Replikat muss allen anderen Replikaten und sich selbst die oben genannten Berechtigungen gewähren — d. h. die Quellkontextbedingungen müssen den gesamten Satz von Replikaten umfassen, aus dem die globale Tabelle besteht. Diese Berechtigungen werden für jedes neue Replikat überprüft, wenn es zu einer globalen Tabelle mit mehreren Konten hinzugefügt wird.

Dadurch wird überprüft, ob Replikationsvorgänge nur vom autorisierten DynamoDB-Dienst und nur zwischen den vorgesehenen Tabellen ausgeführt werden.

Mit Diensten verknüpfte Rollen für globale Tabellen mit mehreren Konten

DynamoDB-Tabellen mit mehreren Konten replizieren Einstellungen über alle Replikate hinweg, sodass jedes Replikat identisch eingerichtet wird, einen konsistenten Durchsatz aufweist und ein nahtloses Failover-Erlebnis bietet. Die Replikation von Einstellungen wird über die `ReplicateSettings` Berechtigungen des Dienstprinzipals gesteuert, aber wir verlassen uns auch auf dienstverknüpfte Rollen (SLRs), um bestimmte kontenübergreifende Funktionen für die regionsübergreifende Replikation und auto-scaling zu verwalten. Diese Rollen werden nur einmal pro Konto eingerichtet. AWS Nach der Erstellung dienen dieselben Rollen für alle globalen Tabellen in Ihrem Konto. Weitere Informationen zu serviceverknüpften Rollen finden Sie unter Verwenden von serviceverknüpften Rollen im IAM-Benutzerhandbuch.

Mit dem Dienst verknüpfte Rolle für die Verwaltung von Einstellungen

Amazon DynamoDB erstellt automatisch die AWSService RoleForDynamo DBGlobal TableSettingsManagement serviceverknüpfte Rolle (SLR), wenn Sie Ihr erstes globales Tabellenreplikat mit mehreren Konten im Konto erstellen. Diese Rolle verwaltet die kontoübergreifende regionsübergreifende Replikation von Einstellungen für Sie.

Wenn Sie ressourcenbasierte Richtlinien auf Replikate anwenden, stellen Sie sicher, dass Sie keine der im SLR-Prinzipal definierten Berechtigungen verweigern, da dies die Einstellungsverwaltung beeinträchtigen und die Replikation beeinträchtigen könnte, wenn der Durchsatz zwischen den Replikaten nicht übereinstimmt oder. `AWSServiceRoleForDynamoDBGlobalTableSettingsManagement` GSIs Wenn Sie die erforderlichen SLR-Berechtigungen verweigern, wird die Replikation zu und von den betroffenen Replikaten möglicherweise beendet, und der Status der Replikattabelle ändert sich auf. `REPLICATION_NOT_AUTHORIZED` Wenn bei globalen Tabellen mit mehreren Konten ein Replikat länger als 20 Stunden im `REPLICATION_NOT_AUTHORIZED` Status verbleibt, wird das Replikat unwiderruflich in eine DynamoDB-Tabelle mit einer einzigen Region konvertiert. Die Spiegelreflexkamera hat die folgenden Berechtigungen:

- `application-autoscaling:DeleteScalingPolicy`
- `application-autoscaling:DescribeScalableTargets`
- `application-autoscaling:DescribeScalingPolicies`
- `application-autoscaling:DeregisterScalableTarget`

- `application-autoscaling:PutScalingPolicy`
- `application-autoscaling:RegisterScalableTarget`

Mit dem Auto Scaling Service verknüpfte Rolle

Bei der Konfiguration einer globalen Tabelle für den Modus mit bereitgestellter Kapazität muss Auto Scaling für die globale Tabelle konfiguriert werden. DynamoDB Auto Scaling verwendet den AWS Application Auto Scaling Scaling-Dienst, um die bereitgestellte Durchsatzkapazität auf Ihren globalen Tabellenreplikaten dynamisch anzupassen. Der Application Auto Scaling Scaling-Dienst erstellt eine serviceverknüpfte Rolle (SLR) mit dem Namen. AWSServiceRoleForApplicationAutoScaling_DynamoDBTable Diese serviceverknüpfte Rolle wird automatisch in Ihrem AWS Konto erstellt, wenn Sie Auto Scaling für eine DynamoDB-Tabelle zum ersten Mal konfigurieren. Es ermöglicht Application Auto Scaling, die bereitgestellte Tabellenkapazität zu verwalten und CloudWatch Alarme zu erstellen.

Wenn Sie ressourcenbasierte Richtlinien auf Replikate anwenden, stellen Sie sicher, dass Sie dem Application Auto Scaling SLR-Prinzipal keine in der AWSApplicationAutoscalingDynamoDBTableRichtlinie definierten Berechtigungen verweigern, da dadurch die Auto-Scaling-Funktionalität unterbrochen wird.

AWS Wie globale Tabellen IAM verwenden

In den folgenden Abschnitten werden die erforderlichen Berechtigungen für verschiedene globale Tabellenoperationen beschrieben und Richtlinienbeispiele bereitgestellt, die Ihnen bei der Konfiguration des entsprechenden Zugriffs für Ihre Benutzer und Anwendungen helfen sollen.

> **Note**
>
> Alle beschriebenen Berechtigungen müssen auf die spezifische Tabellenressource ARN in den betroffenen Regionen angewendet werden. Die Tabellenressource ARN folgt dem Format`arn:aws:dynamodb:region:account-id:table/table-name`, in dem Sie Ihre tatsächlichen Werte für Region, Konto-ID und Tabellennamen angeben müssen.

Die folgenden step-by-step Themen behandeln wir in den folgenden Abschnitten:

- Globale Tabellen mit mehreren Konten erstellen und Replikate hinzufügen
- Aktualisierung einer globalen Tabelle mit mehreren Konten

- Löschen globaler Tabellen und Entfernen von Replikaten

Globale Tabellen erstellen und Replikate hinzufügen

Berechtigungen für die Erstellung globaler Tabellen

Wenn ein neues Replikat zu einer regionalen Tabelle hinzugefügt wird, um eine globale Tabelle mit mehreren Konten zu bilden, oder zu einer vorhandenen globalen Tabelle mit mehreren Konten, muss der IAM-Prinzipal, der die Aktion ausführt, von allen vorhandenen Mitgliedern autorisiert werden. Alle vorhandenen Mitglieder müssen in ihrer Tabellenrichtlinie die folgenden Berechtigungen erteilen, damit das Replikat erfolgreich hinzugefügt werden kann:

- `dynamodb:AssociateTableReplica`- Mit dieser Berechtigung können Tabellen zu einer globalen Tabellenkonfiguration zusammengefügt werden. Dies ist die grundlegende Berechtigung, die den anfänglichen Aufbau der Replikationsbeziehung ermöglicht.

Durch diese präzise Steuerung können nur autorisierte Konten an der globalen Tabelleneinrichtung teilnehmen.

Beispiel für IAM-Richtlinien zum Erstellen globaler Tabellen

Beispiel für IAM-Richtlinien für ein Setup mit zwei Replikaten

Die Einrichtung globaler Tabellen mit mehreren Konten folgt einem bestimmten Autorisierungsablauf, der eine sichere Replikation gewährleistet. Lassen Sie uns anhand eines praktischen Szenarios untersuchen, wie dies in der Praxis funktioniert, in dem ein Kunde eine globale Tabelle mit zwei Replikaten einrichten möchte. Das erste Replikat (ReplicA) befindet sich in Konto A in der Region ap-east-1, während sich das zweite Replikat (ReplicAB) in Konto B in der Region eu-south-1 befindet.

- Im Quellkonto (Konto A) beginnt der Prozess mit der Erstellung der primären Replikattabelle. Der Kontoadministrator muss dieser Tabelle eine ressourcenbasierte Richtlinie beifügen, die dem Zielkonto (Konto B) ausdrücklich die erforderlichen Berechtigungen erteilt, um die Zuordnung durchzuführen. Diese Richtlinie autorisiert auch den DynamoDB-Replikationsdienst, wichtige Replikationsaktionen durchzuführen.
- Das Zielkonto (Konto B) folgt einem ähnlichen Prozess, indem es beim Erstellen des Replikats eine entsprechende ressourcenbasierte Richtlinie anhängt und auf den ARN der Quelltabelle verweist, der zur Erstellung des Replikats verwendet werden soll. Diese Richtlinie spiegelt die von Konto A erteilten Berechtigungen wider und schafft so eine vertrauenswürdige bidirektionale

Beziehung. Bevor die Replikation eingerichtet wird, validiert DynamoDB diese kontoübergreifenden Berechtigungen, um sicherzustellen, dass die richtige Autorisierung vorhanden ist.

So richten Sie dieses Setup ein:

- Der Administrator von Konto A muss zuerst die ressourcenbasierte Richtlinie an ReplicAA anhängen. Diese Richtlinie gewährt Konto B und dem DynamoDB-Replikationsdienst ausdrücklich die erforderlichen Berechtigungen.
- In ähnlicher Weise muss der Administrator von Konto B im Aufruf create table Replica B, das auf Replikat A als Quelltabelle verweist, eine entsprechende Richtlinie an Replicab anhängen, wobei die Kontoreferenzen umgekehrt werden, um Konto A die entsprechenden Berechtigungen zu gewähren.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "DynamoDBActionsNeededForSteadyStateReplication",
            "Effect": "Allow",
            "Action": [
                "dynamodb:ReadDataForReplication",
                "dynamodb:WriteDataForReplication",
                "dynamodb:ReplicateSettings"
            ],
            "Resource": "arn:aws:dynamodb:ap-east-1:A:table/ReplicaA",
            "Principal": {"Service": ["replication.dynamodb.amazonaws.com"]},
            "Condition": {
                "StringEquals": {
                    "aws:SourceAccount": [ "A", "B" ],
                    "aws:SourceArn": [
                        "arn:aws:dynamodb:ap-east-1:A:table/ReplicaA",
                        "arn:aws:dynamodb:eu-south-1:B:table/ReplicaB"
                    ]
                }
            }
        },
        {
            "Sid": "AllowTrustedAccountsToJoinThisGlobalTable",
            "Effect": "Allow",
            "Action": [
```

```
                "dynamodb:AssociateTableReplica"
            ],
            "Resource": "arn:aws:dynamodb:ap-east-1:A:table/ReplicaA",
            "Principal": {"AWS": ["B"]}
        }
    ]
}
```

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "DynamoDBActionsNeededForSteadyStateReplication",
            "Effect": "Allow",
            "Action": [
                "dynamodb:ReadDataForReplication",
                "dynamodb:WriteDataForReplication",
                "dynamodb:ReplicateSettings"
            ],
            "Resource": "arn:aws:dynamodb:eu-south-1:B:table/ReplicaB",
            "Principal": {"Service": ["replication.dynamodb.amazonaws.com"]},
            "Condition": {
                "StringEquals": {
                    "aws:SourceAccount": [ "A", "B" ],
                    "aws:SourceArn": [
                        "arn:aws:dynamodb:ap-east-1:A:table/ReplicaA",
                        "arn:aws:dynamodb:eu-south-1:B:table/ReplicaB"
                    ]
                }
            }
        }
    ]
}
```

Beispiel für IAM-Richtlinien für ein Setup mit drei Replikaten

In diesem Setup haben wir jeweils 3 Replikate ReplicA, Replicab und ReplicaC in Konto A, Konto B und Konto C. Replikat A ist das erste Replikat, das als regionale Tabelle beginnt und der dann Replicab und ReplicaC hinzugefügt werden.

- Der Administrator von Konto A muss zuerst ReplicA die ressourcenbasierte Richtlinie zuordnen, sodass die Replikation mit allen Mitgliedern möglich ist und den IAM-Prinzipalen von Konto B und Konto C das Hinzufügen von Replikaten ermöglicht wird.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "DynamoDBActionsNeededForSteadyStateReplication",
            "Effect": "Allow",
            "Action": [
                "dynamodb:ReadDataForReplication",
                "dynamodb:WriteDataForReplication",
                "dynamodb:ReplicateSettings"
            ],
            "Resource": "arn:aws:dynamodb:ap-east-1:A:table/ReplicaA",
            "Principal": {"Service": ["replication.dynamodb.amazonaws.com"]},
            "Condition": {
                "StringEquals": {
                    "aws:SourceAccount": [ "A", "B", "C" ],
                    "aws:SourceArn": [
                        "arn:aws:dynamodb:ap-east-1:A:table/ReplicaA",
                        "arn:aws:dynamodb:eu-south-1:B:table/ReplicaB",
                        "arn:aws:dynamodb:us-east-1:C:table/ReplicaC"
                    ]
                }
            }
        },
        {
            "Sid": "AllowTrustedAccountsToJoinThisGlobalTable",
            "Effect": "Allow",
            "Action": [
                "dynamodb:AssociateTableReplica"
            ],
            "Resource": "arn:aws:dynamodb:ap-east-1:A:table/ReplicaA",
            "Principal": { "AWS": [ "B", "C" ] }
        }
    ]
}
```

- Der Administrator von Konto B muss ein Replikat (Replikat B) hinzufügen, das auf Replica A als Quelle verweist. Für Replikat B gilt die folgende Richtlinie, die die Replikation zwischen allen Mitgliedern ermöglicht und es Konto C ermöglicht, ein Replikat hinzuzufügen:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "DynamoDBActionsNeededForSteadyStateReplication",
            "Effect": "Allow",
            "Action": [
                "dynamodb:ReadDataForReplication",
                "dynamodb:WriteDataForReplication",
                "dynamodb:ReplicateSettings"
            ],
            "Resource": "arn:aws:dynamodb:eu-south-1:B:table/ReplicaB",
            "Principal": {"Service": ["replication.dynamodb.amazonaws.com"]},
            "Condition": {
                "StringEquals": {
                    "aws:SourceAccount": [ "A", "B", "C" ],
                    "aws:SourceArn": [
                        "arn:aws:dynamodb:ap-east-1:A:table/ReplicaA",
                        "arn:aws:dynamodb:eu-south-1:B:table/ReplicaB",
                        "arn:aws:dynamodb:us-east-1:C:table/ReplicaC"
                    ]
                }
            }
        },
        {
            "Sid": "AllowTrustedAccountsToJoinThisGlobalTable",
            "Effect": "Allow",
            "Action": [
                "dynamodb:AssociateTableReplica"
            ],
            "Resource": "arn:aws:dynamodb:eu-south-1:B:table/ReplicaB",
            "Principal": { "AWS": [ "C" ] }
        }
    ]
}
```

- Schließlich erstellt der Administrator von Konto C ein Replikat mit der folgenden Richtlinie, die Replikationsberechtigungen zwischen allen Mitgliedern gewährt. Die Richtlinie erlaubt nicht, dass weitere Replikate hinzugefügt werden.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "DynamoDBActionsNeededForSteadyStateReplication",
            "Effect": "Allow",
            "Action": [
                "dynamodb:ReadDataForReplication",
                "dynamodb:WriteDataForReplication",
                "dynamodb:ReplicateSettings"
            ],
            "Resource": "arn:aws:dynamodb:us-east-1:C:table/ReplicaC",
            "Principal": {"Service": ["replication.dynamodb.amazonaws.com"]},
            "Condition": {
                "StringEquals": {
                    "aws:SourceAccount": [ "A", "B" ],
                    "aws:SourceArn": [
                        "arn:aws:dynamodb:ap-east-1:A:table/ReplicaA",
                        "arn:aws:dynamodb:eu-south-1:B:table/ReplicaB"
                    ]
                }
            }
        }
    ]
}
```

Aktualisierung einer globalen Tabelle mit mehreren Konten

Um die Replikateinstellungen für eine bestehende globale Tabelle mithilfe der UpdateTable API zu ändern, benötigen Sie die folgende Berechtigung für die Tabellenressource in der Region, in der Sie den API-Aufruf tätigen: `dynamodb:UpdateTable`

Sie können auch andere globale Tabellenkonfigurationen aktualisieren, z. B. Richtlinien für auto Skalierung und Time-to-Live-Einstellungen. Für diese zusätzlichen Aktualisierungsvorgänge sind die folgenden Berechtigungen erforderlich:

Um die Time-to-Live-Einstellungen mit der `UpdateTimeToLive` API zu aktualisieren, benötigen Sie die folgenden Berechtigungen für die Tabellenressource in allen Regionen, die Replikate enthalten: `dynamodb:UpdateTimeToLive`

Um eine Replica-Auto-Scaling-Richtlinie mit der `UpdateTableReplicaAutoScaling` API zu aktualisieren, benötigen Sie die folgenden Berechtigungen für die Tabellenressource in allen Regionen, die Replikate enthalten:

- `application-autoscaling:DeleteScalingPolicy`
- `application-autoscaling:DeleteScheduledAction`
- `application-autoscaling:DeregisterScalableTarget`
- `application-autoscaling:DescribeScalableTargets`
- `application-autoscaling:DescribeScalingActivities`
- `application-autoscaling:DescribeScalingPolicies`
- `application-autoscaling:DescribeScheduledActions`
- `application-autoscaling:PutScalingPolicy`
- `application-autoscaling:PutScheduledAction`
- `application-autoscaling:RegisterScalableTarget`

> **Note**
>
> Sie müssen `dynamodb:ReplicateSettings` Berechtigungen für alle Replikatregionen und Konten bereitstellen, damit die Aktualisierungstabelle erfolgreich aktualisiert werden kann. Wenn ein Replikat nicht berechtigt ist, Einstellungen auf ein beliebiges Replikat in der globalen Tabelle mit `AccessDeniedException` mehreren Konten zu replizieren, schlagen alle Aktualisierungsvorgänge für alle Replikate fehl, bis die Berechtigungen repariert sind.

Löschen globaler Tabellen und Entfernen von Replikaten

Um eine globale Tabelle zu löschen, müssen Sie alle Replikate entfernen. Im Gegensatz zu Global Table mit demselben Konto können Sie diese Option nicht verwenden, `UpdateTable` um eine Replikattabelle in einer entfernten Region zu löschen. Jedes Replikat muss über die `DeleteTable` API aus dem Konto gelöscht werden, das es steuert.

Berechtigungen zum Löschen globaler Tabellen und zum Entfernen von Replikaten

Die folgenden Berechtigungen sind sowohl für das Entfernen einzelner Replikate als auch für das vollständige Löschen globaler Tabellen erforderlich. Durch das Löschen einer globalen Tabellenkonfiguration wird nur die Replikationsbeziehung zwischen Tabellen in verschiedenen Regionen entfernt. Die zugrunde liegende DynamoDB-Tabelle in der letzten verbleibenden Region wird nicht gelöscht. Die Tabelle in der letzten Region existiert weiterhin als standardmäßige DynamoDB-Tabelle mit denselben Daten und Einstellungen.

Sie benötigen die folgenden Berechtigungen für die Tabellenressource in jeder Region, in der Sie ein Replikat entfernen:

- `dynamodb:DeleteTable`
- `dynamodb:DeleteTableReplica`

Wie verwenden globale Tabellen AWS KMS

Wie alle DynamoDB-Tabellen verschlüsseln globale Tabellenreplikate Daten im Ruhezustand immer mit Verschlüsselungsschlüsseln, die im AWS Key Management Service () gespeichert sind.AWS KMS

> **Note**
>
> Im Gegensatz zu globalen Tabellen mit demselben Konto können verschiedene Replikate in einer globalen Tabelle mit mehreren Konten mit unterschiedlichen Schlüsseltypen konfiguriert werden (AWS eigener Schlüssel oder vom Kunden AWS KMS verwalteter Schlüssel). Globale Tabellen mit mehreren Konten unterstützen keine verwalteten Schlüssel. AWS

Globale Tabellen mit mehreren Konten, die verwendet werden, CMKs erfordern, dass die Schlüsselrichtlinie jedes Replikats dem DynamoDB-Replikationsdienstprinzipal (`replication.dynamodb.amazonaws.com`) Berechtigungen für den Zugriff auf den Schlüssel für die Replikation und die Einstellungsverwaltung erteilt. Die folgenden Berechtigungen sind erforderlich:

- `kms:Decrypt`
- `kms:ReEncrypt*`
- `kms:GenerateDataKey*`
- `kms:DescribeKey`

Wichtig

DynamoDB benötigt Zugriff auf den Verschlüsselungsschlüssel des Replikats, um ein Replikat zu löschen. Wenn Sie einen vom Kunden verwalteten Schlüssel, der zur Verschlüsselung eines Replikats verwendet wird, deaktivieren oder löschen möchten, weil Sie das Replikat löschen, sollten Sie zuerst das Replikat löschen, warten, bis die Tabelle aus der Replikationsgruppe entfernt wurde, indem Sie describe in einem der anderen Replikate aufrufen, und dann den Schlüssel deaktivieren oder löschen.

Wenn Sie den Zugriff von DynamoDB auf einen vom Kunden verwalteten Schlüssel, der zur Verschlüsselung eines Replikats verwendet wird, deaktivieren oder entziehen, wird die Replikation zum und vom Replikat beendet und der Replikatstatus wird auf geändert`INACCESSIBLE_ENCRYPTION_CREDENTIALS`. Wenn ein Replikat länger als 20 Stunden im `INACCESSIBLE_ENCRYPTION_CREDENTIALS` Status verbleibt, wird das Replikat irreversibel in eine DynamoDB-Tabelle mit einer einzigen Region konvertiert.

Beispiel für eine Richtlinie AWS KMS

Die AWS KMS Richtlinie ermöglicht DynamoDB den Zugriff auf beide AWS KMS Schlüssel für die Replikation zwischen den Replikaten A und B. Die AWS KMS Schlüssel, die dem DynamoDB-Replikat in jedem Konto zugeordnet sind, müssen mit der folgenden Richtlinie aktualisiert werden:

```
{
    "Version": "2012-10-17",
    "Statement": [
      {
        "Effect": "Allow",
        "Principal": { "Service": "replication.dynamodb.amazonaws.com" },
        "Action": [
            "kms:Decrypt",
            "kms:ReEncrypt*",
            "kms:GenerateDataKey*",
            "kms:DescribeKey"
        ],
        "Condition": {
            "StringEquals": {
                "aws:SourceAccount": [ "A", "B" ],
                "aws:SourceArn": [
                    "arn:aws:dynamodb:ap-east-1:A:table/ReplicaA",
                    "arn:aws:dynamodb:eu-south-1:B:table/ReplicaB"
                ]
            }
```

```
        }
      }
    ]
}
```

Grundlegendes zur Amazon DynamoDB-Fakturierung für globale Tabellen

In diesem Leitfaden wird die Funktionsweise der DynamoDB-Fakturierung für globale Tabellen beschrieben und es werden die Komponenten ermittelt, die zu den Kosten globaler Tabellen beitragen, und anhand eines Beispiels veranschaulicht.

Globale Amazon-DynamoDB-Tabellen sind eine vollständig verwaltete, serverlose, multiregionale und multiaktive Datenbank. Globale Tabellen sind auf eine Verfügbarkeit von 99,999 % ausgelegt und sorgen so für stabilere Anwendungen und eine höhere Geschäftskontinuität. Globale Tabellen replizieren Ihre DynamoDB-Tabellen automatisch in den AWS Regionen Ihrer Wahl, sodass Sie eine schnelle, lokale Lese- und Schreibleistung erzielen können.

Funktionsweise

Das Fakturierungsmodell für globale Tabellen unterscheidet sich von dem für DynamoDB-Tabellen mit einer Region. Schreibvorgänge in DynamoDB-Tabellen mit einer Region werden in den folgenden Einheiten abgerechnet:

- Write Request Units (WRUs) für den On-Demand-Kapazitätsmodus, bei dem für jeden Schreibvorgang bis zu 1 KB eine WRU berechnet wird
- Schreibkapazitätseinheiten (WCUs) für den Bereitstellungsmodus, bei dem eine WCU einen Schreibvorgang pro Sekunde für bis zu 1 KB ermöglicht

Wenn Sie eine globale Tabelle durch Hinzufügen einer Replikattabelle zu einer vorhandenen Tabelle mit einer Region erstellen, wird die Tabelle mit einer Region in eine Replikattabelle umgewandelt. Das bedeutet, dass sich auch die Einheiten für die Abrechnung von Schreibvorgängen in die Tabelle ändern. Schreibvorgänge in Replikattabellen mit einer Region werden in den folgenden Einheiten abgerechnet:

- Replicated Write Request Units (rWRUs) für den On-Demand-Kapazitätsmodus, bei dem für jeden Schreibvorgang bis zu 1 KB eine RWRU pro Replikattabelle berechnet wird
- Replizierte Schreibkapazitätseinheiten (rWCUs) für den Bereitstellungskapazitätsmodus, bei dem eine WCU pro Replikattabelle einen Schreibvorgang pro Sekunde für bis zu 1 KB ermöglicht

Aktualisierungen globaler sekundärer Indizes (GSIs) werden mit denselben Einheiten abgerechnet wie DynamoDB-Tabellen mit einer Region, auch wenn es sich bei der Basistabelle für die GSI um eine Replikattabelle handelt. Aktualisierungsvorgänge für GSIs werden anhand der folgenden Einheiten abgerechnet:

- Write Request Units (WRUs) für den On-Demand-Kapazitätsmodus, bei dem für jeden Schreibvorgang bis zu 1 KB eine WRU berechnet wird
- Schreibkapazitätseinheiten (WCUs) für den Bereitstellungsmodus, bei dem eine WCU einen Schreibvorgang pro Sekunde für bis zu 1 KB ermöglicht

Für replizierte Schreibeinheiten (r WCUs und rWRUs) gilt der gleiche Preis wie für Schreibeinheiten mit einer Region (und). WCUs WRUs Für globale Tabellen fallen Gebühren für multiregionale Datenübertragungen an, da Daten regionsübergreifend repliziert werden. Gebühren für replizierte Schreibvorgänge (rWCU oder rWRU) fallen in jeder Region an, deren globale Tabelle eine Replikattabelle enthält.

Für Lesevorgänge aus Tabellen mit nur einer Region und aus Replikattabellen werden die folgenden Einheiten verwendet:

- Read Request Units (RRUs) für den On-Demand-Kapazitätsmodus, bei dem für jeden stark konsistenten Lesevorgang bis zu 4 KB eine RRU berechnet wird
- Lesekapazitätseinheiten (RCUs) für bereitgestellte Tabellen, bei denen eine RCU einen stark konsistenten Lesevorgang pro Sekunde für bis zu 4 KB ermöglicht

Konsistenzmodi und Fakturierung

Die replizierten Schreibeinheiten (r WCUs und rWRUs), die zur Abrechnung von Schreibvorgängen verwendet werden, sind sowohl für die Modi Multi-Region Strong Consistency (MRSC) als auch für Multi-Region Eventual Consistency (MREC) identisch. Für globale Tabellen, die den MRSC-Modus (Multi-Region Strong Consistency) verwenden und mit einem Zeugen konfiguriert sind, fallen keine Kosten für replizierte Schreibeinheiten (r WCUs und rWRUs), Speicherkosten oder Datenübertragungskosten für die Replikation zum Zeugen an.

Beispiel für die Fakturierung von DynamoDB-Tabellen

Im Folgenden sehen Sie ein Beispielszenario über mehrere Tage, das die Abrechnung von Schreibanforderungen in globalen Tabellen in der Praxis darstellt. (Beachten Sie, dass hier nur Schreibanforderungen berücksichtigt werden und keine Gebühren für die Tabellenwiederherstellung

und die regionsübergreifende Datenübertragung, die im vorliegenden Beispiel zusätzlich anfallen würden):

Tag 1 – Tabelle mit einer Region: Sie haben eine DynamoDB-On-Demand-Tabelle mit einer Region namens „Table_A“ in der Region „us-west-2“. Sie schreiben 100 Elemente mit 1 KB in Table_A. Für diese Schreibvorgänge mit nur einer Region wird eine Schreibanforderungseinheit (WRU) je 1 KB berechnet. Für Tag 1 fallen demnach folgende Gebühren an:

- 100 WRUs in der Region US-West-2 für Schreibvorgänge in einer Region

Die Gesamtzahl der am ersten Tag berechneten Anforderungseinheiten: 100. WRUs

Tag 2 – Erstellen einer globalen Tabelle: Sie erstellen eine globale Tabelle, indem Sie Table_A in der Region „us-east-2“ ein Replikat hinzufügen. Table_A ist jetzt eine globale Tabelle mit zwei Replikattabellen: eine in der Region „us-west-2“ und eine in der Region „us-east-2“. Sie schreiben 150 Elemente mit 1 KB in die Replikattabelle in der Region „us-west-2“. Für Tag 2 fallen demnach folgende Gebühren an:

- 150 r WRUs in der Region US-West-2 für replizierte Schreibvorgänge
- 150 r WRUs in der Region us-east-2 für replizierte Schreibvorgänge

Gesamtzahl der am zweiten Tag berechneten Anforderungseinheiten: 300 r. WRUs

Tag 3 – Hinzufügen eines globalen sekundären Index: Sie fügen der Replikattabelle in der Region „us-east-2“ einen globalen sekundären Index (GSI) hinzu, der alle Attribute aus der Basistabelle (Replikattabelle) projiziert. Die globale Tabelle erstellt automatisch den globalen Index für die Replikattabelle in der Region „us-west-2“ für Sie. Sie schreiben 200 Elemente mit 1 KB in die Replikattabelle in der Region „us-west-2“. Für Tag 3 fallen demnach folgende Gebühren an:

- • 200 r WRUs in der Region US-West-2 für replizierte Schreibvorgänge
- • 200 WRUs in der Region US-West-2 für GSI-Updates
- • 200 r WRUs in der Region us-east-2 für replizierte Schreibvorgänge
- • 200 WRUs in der Region US-East-2 für GSI-Updates

Gesamtzahl der am 3. Tag berechneten Schreibanforderungseinheiten: 400 WRUs und 400 r. WRUs

Die gesamten Gebühren für Schreibeinheiten für alle drei Tage betragen 500 WRUs (100 WRU an Tag 1 + 400 WRUs an Tag 3) und 700 R WRUs (300 R WRUs an Tag 2 + 400 R WRUs an Tag 3).

Zusammenfassend lässt sich sagen, dass Schreibvorgänge in Replikattabellen in allen Regionen, die eine Replikattabelle enthalten, in replizierten Schreibeinheiten abgerechnet werden. Wenn Sie über globale Sekundärindizes verfügen, werden Ihnen Schreibeinheiten für Aktualisierungen GSIs in allen Regionen berechnet, die einen globalen Index enthalten (was in einer globalen Tabelle für alle Regionen steht, die eine Replikattabelle enthalten).

Versionen von globalen DynamoDB-Tabellen

Es gibt zwei Versionen von globalen DynamoDB-Tabellen: Version 2019.11.21 (Aktuell) und Version 2017.11.29 (Legacy). Wir empfehlen für globale Tabellen die Verwendung der Version 2019.11.21 (Aktuell), da sie einfacher zu verwenden ist, in mehr Regionen unterstützt wird und für die meisten Workloads kostengünstiger ist als Version 2017.11.29 (Legacy).

Ermitteln der Version einer globalen Tabelle

Ermitteln der Version mit dem AWS CLI

Identifizieren des Replikats einer globalen Tabelle der Version 2019.11.21 (Aktuell)

Rufen Sie den Befehl `describe-table` auf, um festzustellen, ob es sich bei einer Tabelle um ein Replikat von Version 2019.11.21 (Aktuell) handelt. Wenn die Ausgabe das `GlobalTableVersion`-Attribut mit dem Wert „2019.11.21“ enthält, handelt es sich bei der Tabelle um das Replikat einer globalen Tabelle der Version 2019.11.21 (Aktuell).

Beispiel eines CLI-Befehls für `describe-table`:

```
aws dynamodb describe-table \
--table-name users \
--region us-east-2
```

Wenn die (verkürzte) Ausgabe das `GlobalTableVersion`-Attribut mit dem Wert „2019.11.21“ enthält, handelt es sich bei der Tabelle um das Replikat einer globalen Tabelle der Version 2019.11.21 (Aktuell).

```
{
    "Table": {
        "AttributeDefinitions": [
            {
                "AttributeName": "id",
                "AttributeType": "S"
```

```
            },
            {
                "AttributeName": "name",
                "AttributeType": "S"
            }
        ],
        "TableName": "users",
        ...
        "GlobalTableVersion": "2019.11.21",
        "Replicas": [
            {
                "RegionName": "us-west-2",
                "ReplicaStatus": "ACTIVE",
            }
        ],
        ...
    }
}
```

Identifizieren des Replikats einer globalen Tabelle der Version 2017.11.29 (Legacy)

Die Version 2017.11.29 (Legacy) verwendet einen speziellen Befehlssatz für die Verwaltung der globalen Tabelle. Rufen Sie den Befehl `describe-global-table` auf, um festzustellen, ob es sich bei einer Tabelle um ein Replikat von Version 2017.11.29 (Legacy) handelt. Wenn Sie eine erfolgreiche Antwort erhalten, handelt es sich bei der globalen Tabelle um ein Replikat der Version 2017.11.29 (Legacy). Wenn der Befehl `describe-global-table` den Fehler `GlobalTableNotFoundException` zurückgibt, handelt es sich bei der Tabelle nicht um ein Replikat der Version 2017.11.29 (Legacy).

Beispiel eines CLI-Befehls für `describe-global-table`:

```
aws dynamodb describe-global-table \
--table-name users \
--region us-east-2
```

Wenn der Befehl eine erfolgreiche Antwort zurückgibt, handelt es sich bei der globalen Tabelle um ein Replikat der Version 2017.11.29 (Legacy).

```
{
    "GlobalTableDescription": {
        "ReplicationGroup": [
```

```
            {
                "RegionName": "us-west-2"
            },
            {
                "RegionName": "us-east-2"
            }
        ],
        "GlobalTableArn": "arn:aws:dynamodb::123456789012:global-table/users",
        "CreationDateTime": "2025-06-10T13:55:53.630000-04:00",
        "GlobalTableStatus": "ACTIVE",
        "GlobalTableName": "users"
    }
}
```

Ermitteln der Version mithilfe der DynamoDB-Konsole

Gehen Sie wie folgt vor, um die Version des Replikats einer globalen Tabelle zu ermitteln:

1. Öffnen Sie die DynamoDB-Konsole zu Hausehttps://console.aws.amazon.com/dynamodb/.
2. Klicken Sie im Navigationsbereich auf der linken Seite der Konsole auf Tabellen.
3. Wählen Sie die globale Tabelle aus, deren Version Sie ermitteln möchten.
4. Wählen Sie die Registerkarte Globale Tabellen aus.

   Im Abschnitt Zusammenfassung wird die Version der verwendeten globalen Tabellen angezeigt.

Unterschiede im Verhalten zwischen der Legacy- und der aktuellen Version

In der folgenden Liste werden die Verhaltensunterschiede zwischen der Legacy-Version und der aktuellen Version globaler Tabellen beschrieben.

- Version 2019.11.21 (Aktuell) verbraucht im Vergleich zu Version 2017.11.29 (Legacy) bei mehreren Operationen in DynamoDB-Datenbanken weniger Schreibkapazität und ist daher für die meisten Kunden kostengünstiger. Die Unterschiede in Bezug auf DynamoDB-Operationen im Einzelnen:
  - Das Aufrufen PutItemeines 1-KB-Elements in einer Region und das Replizieren in andere Regionen erfordert 2 r WRUs pro Region für 2017.11.29 (Legacy), aber nur 1 RWRU für 2019.11.21 (Aktuell).
  - UpdateItemZum Aufrufen eines 1-KB-Elements sind 2 r WRUs in der Quellregion und 1 RWRU pro Zielregion für 2017.11.29 (Legacy) erforderlich, aber nur 1 RWRU für Quell- und Zielregion für 2019.11.21 (Aktuell).

- Der Aufruf DeleteItemfür ein 1-KB-Element erfordert 1 RWRU in der Quellregion und 2 r WRUs pro Zielregion für 2017.11.29 (Legacy), aber nur 1 RWRU sowohl für die Quell- als auch für die Zielregion für 2019.11.21 (Aktuell).

Die folgende Tabelle zeigt den rWRU-Verbrauch der Tabellen-Versionen 2017.11.29 (Legacy) und 2019.11.21 (Aktuell) für ein 1 KB großes Element in zwei Regionen.

Operation	2017.11.29 (Legacy)	2019.11.21 (Aktuell)	Einsparungen
PutItem	4 r WRUs	2 oder WRUs	50 %
UpdateItem	3 oder WRUs	2 oder WRUs	33%
DeleteItem	3 r WRUs	2 oder WRUs	33%

- Version 2017.11.29 (Legacy) ist nur in Version 11 verfügbar. AWS-Regionen Version 2019.11.21 (Aktuell) hingegen ist in allen AWS-Regionen verfügbar.
- Sie erstellen globale Tabellen der Version 2017.11.29 (Legacy), indem Sie zuerst einen Satz leerer Regionaltabellen erstellen und dann die API aufrufen, um die CreateGlobalTableglobale Tabelle zu bilden. Sie erstellen globale Tabellen der Version 2019.11.21 (aktuell), indem Sie die UpdateTableAPI aufrufen, um einer vorhandenen Regionaltabelle ein Replikat hinzuzufügen.
- Version 2017.11.29 (Legacy) erfordert, dass Sie alle Replikate in der Tabelle leeren, bevor Sie einer neuen Region (auch während ihrer Erstellung) ein Replikat hinzufügen. Version 2019.11.21 (Aktuell) unterstützt Sie beim Hinzufügen und Entfernen von Replikaten zu Regionen in einer Tabelle, die bereits Daten enthält.
- Version 2017.11.29 (Legacy) verwendet die folgenden speziellen Steuerungsebenen für die Verwaltung von Replikaten: APIs
  - CreateGlobalTable
  - DescribeGlobalTable
  - DescribeGlobalTableSettings
  - ListGlobalTables
  - UpdateGlobalTable
  - UpdateGlobalTableSettings

  Version 2019.11.21 (Aktuell) verwendet das und zur Verwaltung von Replikaten. DescribeTableUpdateTable APIs

- In Version 2017.11.29 (Legacy) werden zwei DynamoDB-Streams-Datensätze für jeden Schreibvorgang veröffentlicht. In Version 2019.11.21 (Aktuell) wird nur ein DynamoDB-Streams-Datensatz für jeden Schreibvorgang veröffentlicht.
- In Version 2017.11.29 (Legacy) werden die Attribute `aws:rep:deleting`, `aws:rep:updateregion` und `aws:rep:updatetime` gefüllt und aktualisiert. In Version 2019.11.21 (Aktuell) werden dieser Attribute nicht gefüllt und aktualisiert.
- In Version 2017.11.29 (Legacy) werden die Verwenden von Time to Live (TTL) in DynamoDB-Einstellungen nicht replikatsübergreifend synchronisiert. In Version 2019.11.21 (Aktuell) werden die TTL-Einstellungen replikatsübergreifend synchronisiert.
- In Version 2017.11.29 (Legacy) werden TTL-Löschvorgänge nicht auf andere Replikate übertragen. In Version 2019.11.21 (Aktuell) werden TTL-Löschvorgänge auf alle Replikate übertragen.
- In Version 2017.11.29 (Legacy) werden die Auto-Scaling-Einstellungen nicht replikatsübergreifend synchronisiert. In Version 2019.11.21 (Aktuell) werden die Auto-Scaling-Einstellungen replikatsübergreifend aktualisiert.
- In Version 2017.11.29 (Legacy) werden die Einstellungen für den globalen sekundären Index (GSI) nicht replikatsübergreifend synchronisiert. In Version 2019.11.21 (Aktuell) werden die GSI-Einstellungen replikatsübergreifend synchronisiert.
- In Version 2017.11.29 (Legacy) werden die Einstellungen für die Verschlüsselung im Ruhezustand nicht replikatsübergreifend synchronisiert. In Version 2019.11.21 (Aktuell) werden die Einstellungen für die Verschlüsselung im Ruhezustand replikatsübergreifend synchronisiert.
- In Version 2017.11.29 (Legacy) wird die `PendingReplicationCount`-Metrik veröffentlicht. In Version 2019.11.21 (Aktuell) wird diese Metrik nicht veröffentlicht.

Aktualisieren auf die aktuelle Version

Erforderliche Berechtigungen für die Aktualisierung von globalen Tabellen

Für eine Aktualisierung auf die Version 2019.11.21 (Aktuell) aktualisieren müssen Sie in allen Regionen mit Replikaten über die Berechtigung `dynamodb:UpdateGlobalTableversion` verfügen. Diese Berechtigungen werden zusätzlich zu den Berechtigungen benötigt, die für den Zugriff auf die DynamoDB-Konsole und die Anzeige von Tabellen erforderlich sind.

Über die folgende IAM-Richtlinie werden Berechtigungen zum Aktualisieren einer globalen Tabelle auf Version 2019.11.21 (Aktuell) erteilt.

```
{
    "version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "dynamodb:UpdateGlobalTableversion",
            "Resource": "*"
        }
    ]
}
```

Über die folgende IAM-Richtlinie wird die Berechtigung erteilt, nur die globale Tabelle `Music` mit Replikaten in zwei Regionen auf die Version 2019.11.21 (Aktuell) zu aktualisieren.

```
{
    "version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "dynamodb:UpdateGlobalTableversion",
            "Resource": [
                "arn:aws:dynamodb::123456789012:global-table/Music",
                "arn:aws:dynamodb:ap-southeast-1:123456789012:table/Music",
                "arn:aws:dynamodb:us-east-2:123456789012:table/Music"
            ]
        }
    ]
}
```

Informationen Fakten zum Ablauf der Aktualisierung

- Während der Aktualisierung wird in allen Replikaten globaler Tabellen die Verarbeitung des Lese- und Schreibdatenverkehr fortgesetzt.
- Die Aktualisierung dauert je nach Tabellengröße und Anzahl der Replikate zwischen einigen Minuten und mehreren Stunden.
- Während des Upgrade-Vorgangs ändert sich der Wert von von TableStatusauf. `ACTIVE` `UPDATING` Sie können den Status der Tabelle anzeigen, indem Sie die DescribeTableAPI aufrufen oder die Tabellenansicht in der DynamoDB-Konsole verwenden.

- Die Einstellungen für die bereitgestellte Kapazität für eine globale Tabelle werden während der Aktualisierung vom Auto-Scaling-Prozess nicht angepasst. Es wird dringend empfohlen, während der Aktualisierung der Tabelle den Modus On-Demand-Kapazität zu verwenden.
- Wenn Sie sich bei der Aktualisierung für den Modus bereitgestellter Kapazität mit Auto Scaling entscheiden, müssen Sie in Ihren Richtlinien den minimalen Lese- und Schreibdurchsatz erhöhen, um dem erwarteten Anstieg des Datenverkehrs während der Dauer der Aktualisierung Rechnung zu tragen.
- Es kann vorkommen, dass die Metrik `ReplicationLatency` vorübergehend Latenzspitzen meldet oder dass die Meldung von Metrikdaten während der Aktualisierung ganz ausbleibt. Weitere Informationen finden Sie unter the section called “ReplicationLatency”.
- Sobald der Aktualisierungsvorgang abgeschlossen ist, wird der Status der Tabelle zurück zu `ACTIVE`.

Verhalten von DynamoDB Streams vor, während und nach der Aktualisierung

Operation	Replikatregion	Verhalten vor der Aktualisierung	Verhalten während der Aktualisierung	Verhalten nach der Aktualisierung
Put oder Update	Quelle	Das Auffüllen von Zeitstempeln erfolgt mithilfe von. UpdateItem	Die Population mit Zeitstempeln erfolgt unter Verwendung von. PutItem	Es wird kein für den Kunden sichtbarer Zeitstempel generiert.
		Es werden zwei Streams-Datensätze generiert. Der erste Datensatz enthält die vom Kunden geschriebenen Attribute. Der zweite Datensatz enthält die	Es werden zwei Streams-Datensätze generiert. Der erste Datensatz enthält die vom Kunden geschriebenen Attribute. Der zweite Datensatz enthält die	Es wird ein einziger Streams-Datensatz generiert, der die vom Kunden geschriebenen Attribute enthält.

Operation	Replikatregion	Verhalten vor der Aktualisierung	Verhalten während der Aktualisierung	Verhalten nach der Aktualisierung
		`aws:rep:* -` Attribute.	`aws:rep:* -` Attribute.	
		Für jeden Schreibvorgang durch den Kunden WCUs werden zwei R verbraucht.	Für jeden Schreibvorgang durch den Kunden WCUs werden zwei R verbraucht.	Für jeden Schreibvorgang durch den Kunden wird eine rWCU verbraucht.
		`ReplicationLatency` und `PendingReplicationCount` Metriken werden in veröffentlicht CloudWatch.	`ReplicationLatency` und `PendingReplicationCount` Metriken werden in veröffentlicht CloudWatch.	`ReplicationLatency` Die Metrik ist veröffentlicht in CloudWatch.
	Ziel	Die Replikation erfolgt mit PutItem.	Die Replikation erfolgt unter Verwendung von PutItem.	Die Replikation erfolgt unter Verwendung von PutItem.

Operation	Replikatregion	Verhalten vor der Aktualisierung	Verhalten während der Aktualisierung	Verhalten nach der Aktualisierung
		Es wird ein einzelner Streams-Datensatz generiert, der sowohl die vom Kunden geschriebenen Attribute als auch die `aws:rep:* -` Attribute enthält.	Es wird ein einzelner Streams-Datensatz generiert, der sowohl die vom Kunden geschriebenen Attribute als auch die `aws:rep:* -` Attribute enthält.	Es wird ein einzelner Streams-Datensatz generiert, der nur die vom Kunden geschriebenen Attribute und keine Replikationsattribute enthält.
		Es wird eine rWCU verbraucht, wenn das Element in der Zielregion existiert. Zwei rWCUs werden verbraucht, wenn der Artikel in der Zielregion nicht existiert.	Es wird eine rWCU verbraucht, wenn das Element in der Zielregion existiert. Zwei rWCUs werden verbraucht, wenn der Artikel in der Zielregion nicht existiert.	Für jeden Schreibvorgang durch den Kunden wird eine rWCU verbraucht.

Operation	Replikatregion	Verhalten vor der Aktualisierung	Verhalten während der Aktualisierung	Verhalten nach der Aktualisierung
		`ReplicationLatency` und `PendingReplicationCount` Metriken werden in veröffentlicht CloudWatch.	`ReplicationLatency` und `PendingReplicationCount` Metriken werden in veröffentlicht CloudWatch.	`ReplicationLatency` Die Metrik ist veröffentlicht in CloudWatch.
Löschen	Quelle	Löschen Sie alle Elemente mit kleinerem Zeitstempel mithilfe von DeleteItem.	Löschen Sie alle Elemente mit kleinerem Zeitstempel mithilfe von. DeleteItem	Löschen Sie alle Elemente mit kleinerem Zeitstempel mithilfe von. DeleteItem
		Es wird ein einzelner Streams-Datensatz generiert, der sowohl die vom Kunden geschrieb enen Attribute als auch die `aws:rep:* -` Attribute enthält.	Es wird ein einzelner Streams-Datensatz generiert, der sowohl die vom Kunden geschrieb enen Attribute als auch die `aws:rep:* -` Attribute enthält.	Es wird ein einzelner Streams-Datensatz generiert, der die vom Kunden geschriebenen Attribute enthält.

Operation	Replikatregion	Verhalten vor der Aktualisierung	Verhalten während der Aktualisierung	Verhalten nach der Aktualisierung
		Für jeden Löschvorgang durch den Kunden wird eine rWCU verbraucht.	Für jeden Löschvorgang durch den Kunden wird eine rWCU verbraucht.	Für jeden Löschvorgang durch den Kunden wird eine rWCU verbraucht.
		`ReplicationLatency` und `PendingReplicationCount` Metriken werden in CloudWatch veröffentlicht.	`ReplicationLatency` und `PendingReplicationCount` Metriken werden in veröffentlicht CloudWatch.	`ReplicationLatency` Die Metrik ist veröffentlicht in CloudWatch.
	Ziel	Ein Löschvorgang findet in zwei Phasen statt: • In Phase 1 wird UpdateItem das Löschkennzeichen gesetzt. • DeleteItem Löscht in Phase 2 das Element.	Löscht das Element mit. DeleteItem	Löscht das Element mit. DeleteItem

Operation	Replikatregion	Verhalten vor der Aktualisierung	Verhalten während der Aktualisierung	Verhalten nach der Aktualisierung
		Es werden zwei Streams-Datensätze generiert. Der erste Datensatz enthält die am `aws:rep:deleting` -Feld vorgenommene Änderung. Der zweite Datensatz enthält die vom Kunden geschriebenen Attribute und die `aws:rep:*` -Attribute.	Es wird ein einzelner Stream-Datensatz generiert, der die vom Kunden geschriebenen Attribute enthält.	Es wird ein einzelner Stream-Datensatz generiert, der die vom Kunden geschriebenen Attribute enthält.
		Für jeden gelöschten Kunden WCUs werden zwei R verbraucht.	Für jeden Löschvorgang durch den Kunden wird eine rWCU verbraucht.	Für jeden Löschvorgang durch den Kunden wird eine rWCU verbraucht.

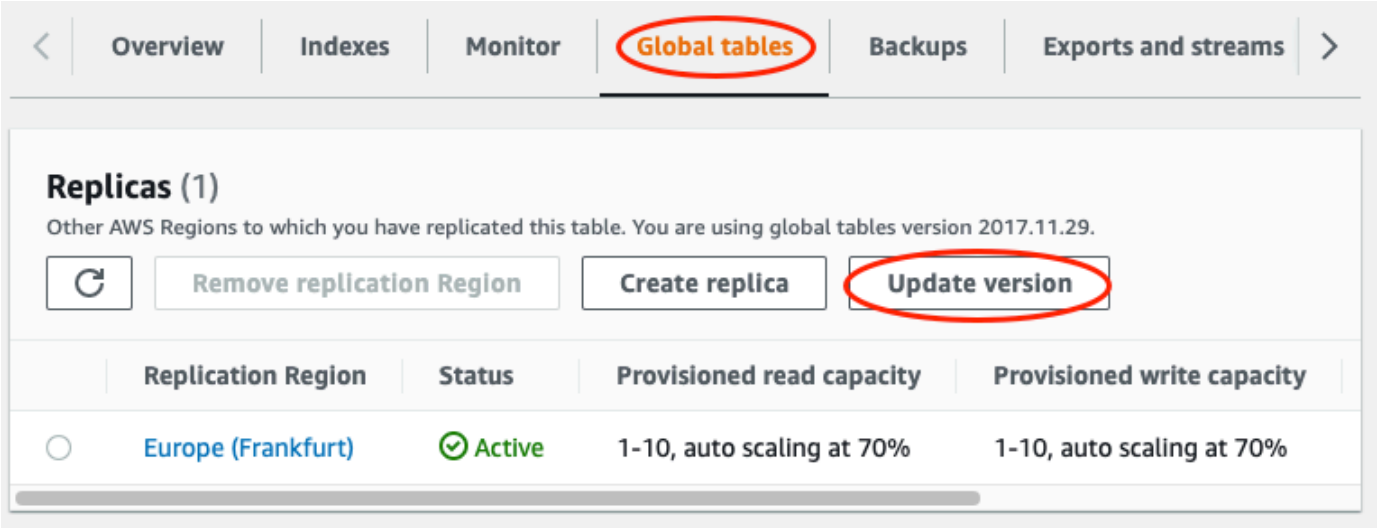
Operation	Replikatregion	Verhalten vor der Aktualisierung	Verhalten während der Aktualisierung	Verhalten nach der Aktualisierung
		`ReplicationLatency` und `PendingReplicationCount` Metriken werden in veröffentlicht CloudWatch.	`ReplicationLatency` Die Metrik ist veröffentlicht in CloudWatch.	`ReplicationLatency` Die Metrik ist veröffentlicht in CloudWatch.

Aktualisieren auf Version 2019.11.21 (Aktuell)

Führen Sie die folgenden Schritte aus, um Ihre Version für globale DynamoDB-Tabellen mithilfe der AWS-Managementkonsole zu aktualisieren.

So aktualisieren globale Tabellen auf Version 2019.11.21 (Aktuell)

1. Öffnen Sie die DynamoDB-Konsole zu Hausehttps://console.aws.amazon.com/dynamodb/.
2. Wählen Sie im Navigationsbereich auf der linken Seite der Konsole die Option Tabellen und dann die globale Tabelle aus, die Sie auf Version 2019.11.21 (Aktuell) aktualisieren möchten.
3. Wählen Sie die Registerkarte Globale Tabellen aus.
4. Wählen Sie Update version (Version aktualisieren) aus.



5. Lesen Sie sich die neuen Anforderungen durch und stimmen Sie ihnen zu, wählen Sie dann Update version Version aktualisieren.
6. Nach Abschluss des Aktualisierungsvorgangs wird die in der Konsole angezeigte Version für globale Tabellen in 2019.11.21 geändert.

Bewährte Methoden für globale Tabellen

In den folgenden Abschnitten werden bewährte Methoden für die Bereitstellung und Verwendung globaler Tabellen beschrieben.

Version

Es gibt zwei Versionen von globalen DynamoDB-Tabellen: Version 2019.11.21 (Aktuell) und Version 2017.11.29 (Legacy). Sie sollten nach Möglichkeit die Version 2019.11.21 (Aktuell) verwenden.

Löschschutz

Sie sollten den Löschschutz für globale Tabellenreplikate aktivieren, die Sie vor versehentlichem Löschen schützen möchten. Sie müssen den Löschschutz für jedes Replikat aktivieren.

Benutzen AWS CloudFormation

CloudFormation unterstützt derzeit nicht die stapelübergreifende Koordination von Ressourcen wie globalen Tabellen. Wenn Sie jedes Replikat einer globalen Tabelle in einem separaten regionalen Stack definieren, treten Fehler auf, die auf festgestellte Abweichungen zwischen den Stacks bei der

Durchführung von Replikataktualisierungen zurückzuführen sind. Um dieses Problem zu vermeiden, sollten Sie eine Region als Referenzregion für die Bereitstellung der globalen Tabellen auswählen und alle Replikate der globalen Tabelle im Stack dieser Region definieren.

> **Important**
>
> Sie können eine Ressource des Typs `AWS::DynamoDB::Table` nicht in eine Ressource des Typs `AWS::DynamoDB::GlobalTable` konvertieren, indem Sie den Typ in Ihrer Vorlage ändern. Der Versuch, eine Tabelle mit einer einzigen Region in eine globale Tabelle zu konvertieren, indem ihr CloudFormation Ressourcentyp geändert wird, kann dazu führen, dass Ihre DynamoDB-Tabelle gelöscht wird.

Stattdessen können Sie die `AWS::DynamoDB::GlobalTable`-Ressource verwenden, um eine neue Tabelle in einer einzelnen Region zu erstellen. Diese Tabelle wird wie jede andere Tabelle mit einer einzelnen Region bereitgestellt. Wenn Sie den Stack später aktualisieren, um einer Ressource weitere Regionen hinzuzufügen, werden der Tabelle Replikate hinzugefügt, und sie wird sicher in eine globale Tabelle konvertiert.

Wenn Sie bereits über eine `AWS::DynamoDB::Table`-Ressource verfügen, die Sie in eine `AWS::DynamoDB::GlobalTable`-Ressource konvertieren möchten, werden folgende Schritte zur Konvertierung des Ressourcentyps empfohlen:

1. Legen Sie die Richtlinie zum Löschen von `AWS::DynamoDB::Table` fest, die beibehalten werden soll.
2. Entfernen Sie die Tabelle aus der Stack-Definition.
3. Fügen Sie der Tabelle mit einer Region in der AWS Konsole Replikate hinzu und konvertieren Sie sie in eine globale Tabelle.
4. Importieren Sie die globale Tabelle als neue `AWS::DynamoDB::GlobalTable`-Ressource in den Stack.

Backups und Wiederherstellung Point-in-Time

Die Aktivierung von Automated Backups and Point-in-Time Recovery (PITR) für ein Replikat in einer globalen Tabelle kann ausreichen, um Ihre Disaster Recovery-Ziele zu erreichen. Mit -Backup erstellte AWS Replikat-Backups können automatisch regionsübergreifend repliziert werden, um die Ausfallsicherheit zu erhöhen. Berücksichtigen Sie bei der Wahl der Backup- und PITR-Strategie die

Ziele des Notfallwiederherstellungsplans vor dem Hintergrund der Hochverfügbarkeit in mehreren Regionen.

Planen einer hohen Verfügbarkeit in mehreren Regionen

Vorgeschriebene Anleitungen zur Bereitstellung globaler Tabellen finden Sie unter Best Practices for DynamoDB global table design.

Arbeiten mit Elementen und Attributen in DynamoDB

In Amazon DynamoDB ist ein Element eine Sammlung von Attributen. Jedes Attribut verfügt über einen Namen und einen Wert. Ein Attributwert kann eine Skalarfunktion, eine Gruppe oder ein Dokumenttyp sein. Weitere Informationen finden Sie unter Amazon DynamoDB: Funktionsweise.

DynamoDB bietet vier Operationen für grundlegende CRUD-Funktionalität (erstellen/lesen/aktualisieren/löschen). Alle diese Operationen sind atomar.

- `PutItem` – ein Element erstellen.
- `GetItem` – ein Element lesen.
- `UpdateItem` – ein Element aktualisieren.
- `DeleteItem` – ein Element löschen.

Jede dieser Operationen erfordert, dass Sie den Primärschlüssel des Elements angeben, mit dem Sie arbeiten möchten. Um z. B. ein Element mit `GetItem` zu lesen, müssen Sie den Partitions- und den Sortierschlüssel (sofern zutreffend) für das Element angeben.

Zusätzlich zu den vier grundlegenden CRUD-Operationen bietet DynamoDB außerdem Folgendes:

- `BatchGetItem` – Liest bis zu 100 Elemente aus einer oder mehreren Tabellen.
- `BatchWriteItem` – erstellt oder löscht bis zu 25 Elemente in einer oder mehreren Tabellen.

Diese Stapeloperationen kombinieren mehrere CRUD-Operationen in einer einzigen Anforderung. Darüber hinaus können die Stapeloperationen Elemente parallel lesen und schreiben, um Antwortlatenzen zu minimieren.

In diesem Abschnitt wird beschrieben, wie Sie diese Operationen verwenden. Außerdem wird auf verwandte Themen, wie z. B. bedingte Updates und unteilbare Zähler, eingegangen. Dieser Abschnitt enthält auch Beispielcode, der die verwendet AWS SDKs.

Themen

- Elementgrößen und -formate in DynamoDB
- Lesen eines Elements
- Schreiben eines Elements
- Rückgabewerte
- Batch-Vorgänge
- Unteilbare Zähler
- Bedingte Schreibvorgänge
- Verwenden von Ausdrücken in DynamoDB
- Verwenden von Time to Live (TTL) in DynamoDB
- Abfragen von Tabellen in DynamoDB
- Scannen von Tabellen in DynamoDB
- PartiQL – Eine SQL-kompatible Abfragesprache für Amazon DynamoDB
- Arbeiten mit Elementen: Java
- Arbeiten mit Elementen: .NET

Elementgrößen und -formate in DynamoDB

DynamoDB-Tabellen sind schemalos, mit Ausnahme des Primärschlüssels, sodass alle Elemente in einer Tabelle verschiedene Attribute, Größen und Datentypen haben können.

Die Gesamtgröße eines Elements ist die Summe der Längenwerte seiner Attributnamen und Werte, zuzüglich eventuell zutreffender Overhead, wie nachfolgend beschrieben. Sie können die folgenden Richtlinien zum Schätzen von Attributgrößen verwenden:

- Die Zeichenfolgen sind Unicode mit binärer UTF-8-Kodierung. Die Größe einer Zeichenkette ist (Anzahl der UTF-8-kodierten Byte des Attributnamens) + (Anzahl der UTF-8-kodierten Byte).
- Zahlen sind variable Länge mit bis zu 38 signifikanten Ziffern. Nullen am Anfang und am Ende werden abgeschnitten. Die Größe einer Zahl ist ca. (Anzahl der UTF-8-kodierten Byte des Attributnamens) + (1 Byte pro zwei signifikanten Ziffern) + (1 Byte).
- Ein binärer Wert muss im Base64-Format kodiert werden, bevor er an DynamoDB gesendet werden kann, aber die reine Bytelänge des Werts wird für die Größenberechnung verwendet. Die Größe eines binären Attributs ist (Anzahl der UTF-8-kodierten Byte des Attributnamens) + (Anzahl der reinen Byte).

- Die Größe eines Null-Attributs oder eines Booleschen Attributs ist (Anzahl der UTF-8-kodierten Byte des Attributnamens) + (1 Byte).
- Ein Attribut vom Typ `List` oder `Map` erfordert 3 Byte Overhead, unabhängig von dessen Inhalten. Die Größe einer `List` oder `Map` ist (Anzahl der UTF-8-kodierten Byte des Attributnamens) + Summe (Größe der verschachtelten Elemente) + (3 Byte). Die Größe einer leeren `List` oder `Map` ist (Anzahl der UTF-8-kodierten Byte des Attributnamens) + (3 Byte).
- Jedes `List`- oder `Map`-Element benötigt außerdem einen Overhead von 1 Byte.

> **Note**
>
> Wir empfehlen, dass Sie eher kürzere Attributnamen als längere wählen. Dies hilft Ihnen, den benötigten Speicherplatz zu reduzieren, kann aber auch den RCU/WCUs Speicherverbrauch verringern.

Für Speicher-Abrechnungszwecke enthält jedes Element einen Speicherzuschlag pro Element, der von den Funktionen abhängt, die Sie aktiviert haben.

- Alle Elemente in DynamoDB benötigen 100 Byte Speicheraufwand für die Indizierung.
- Einige DynamoDB-Funktionen (globale Tabellen, Transaktionen, Änderungsdatenerfassung für Kinesis Data Streams mit DynamoDB) erfordern zusätzlichen Speicheraufwand, um vom System erstellte Attribute zu berücksichtigen, die sich aus der Aktivierung dieser Funktionen ergeben. Globale Tabellen erfordern beispielsweise zusätzliche 48 Byte Speicheraufwand.

Lesen eines Elements

Verwenden Sie zum Lesen eines Elements von einer DynamoDB-Tabelle die Operation `GetItem`. Sie müssen den Namen der Tabelle sowie den Primärschlüssel des gewünschten Elements angeben.

Example

Das folgende AWS CLI Beispiel zeigt, wie ein Element aus der `ProductCatalog` Tabelle gelesen wird.

```
aws dynamodb get-item \
    --table-name ProductCatalog \
    --key '{"Id":{"N":"1"}}'
```

> **Note**
>
> Mit `GetItem` müssen Sie den Primärschlüssel vollständig und nicht nur teilweise angeben. Wenn eine Tabelle über einen zusammengesetzten Primärschlüssel (Partitions- und Sortierschlüssel) verfügt, müssen Sie einen Wert für den Partitionsschlüssel und einen Wert für den Sortierschlüssel angeben.

Eine `GetItem`-Anforderung führt standardmäßig einen Eventually Consistent-Lesevorgang durch. Sie können den Parameter `ConsistentRead` verwenden, um stattdessen einen Strongly Consistent-Lesevorgang anzufordern. (Dadurch werden zusätzliche Lesekapazitätseinheiten verbraucht, es wird jedoch die meiste up-to-date Version des Elements zurückgegeben.)

`GetItem` gibt alle Attribute des Elements zurück. Sie können einen Projektionsausdruck verwenden, um nur einige der Attribute zurückzugeben. Weitere Informationen finden Sie unter Verwenden von Projektionsausdrücken in DynamoDB.

Um die Anzahl der von `GetItem` verbrauchten Lesekapazitätseinheiten zurückzugeben, legen Sie den Parameter `ReturnConsumedCapacity` auf `TOTAL` fest.

Example

Das folgende Beispiel AWS Command Line Interface (AWS CLI) zeigt einige der optionalen `GetItem` Parameter.

```
aws dynamodb get-item \
    --table-name ProductCatalog \
    --key '{"Id":{"N":"1"}}' \
    --consistent-read \
    --projection-expression "Description, Price, RelatedItems" \
    --return-consumed-capacity TOTAL
```

Schreiben eines Elements

Zum Erstellen, Aktualisieren oder Löschen eines Elements in einer DynamoDB-Tabelle verwenden Sie eine der folgenden Operationen:

- `PutItem`
- `UpdateItem`
- `DeleteItem`

Für diese Operationen müssen Sie jeweils den Primärschlüssel vollständig und nicht nur teilweise angeben. Wenn eine Tabelle über einen zusammengesetzten Primärschlüssel (Partitions- und Sortierschlüssel) verfügt, müssen Sie einen Wert für den Partitionsschlüssel und einen Wert für den Sortierschlüssel angeben.

Um die Anzahl der von einer dieser Operationen verbrauchten Schreibkapazitätseinheiten zurückzugeben, legen Sie den Parameter `ReturnConsumedCapacity` auf einen der folgenden Werte fest:

- `TOTAL` – Gibt die Gesamtanzahl der verbrauchten Schreibkapazitätseinheiten zurück.
- `INDEXES` – Gibt die Gesamtzahl der verbrauchten Schreibkapazitätseinheiten mit Zwischensummen für die Tabelle und alle sekundären Indizes zurück, die vom Vorgang betroffen waren.
- `NONE` – Es werden keine Schreibkapazitätsdetails zurückgegeben. (Dies ist die Standardeinstellung.)

PutItem

`PutItem` erstellt ein neues Element. Wenn ein Element mit demselben Schlüssel bereits in der Tabelle vorhanden ist, wird es durch das neue Element ersetzt.

Example

Schreiben Sie ein neues Element in die `Thread`-Tabelle. Der Primärschlüssel für die `Thread`-Tabelle besteht aus `ForumName` (Partitionsschlüssel) und `Subject` (Sortierschlüssel).

```
aws dynamodb put-item \
    --table-name Thread \
    --item file://item.json
```

Die Argumente für `--item` werden in der Datei `item.json` gespeichert.

```
{
    "ForumName": {"S": "Amazon DynamoDB"},
    "Subject": {"S": "New discussion thread"},
    "Message": {"S": "First post in this thread"},
    "LastPostedBy": {"S": "fred@example.com"},
    "LastPostDateTime": {"S": "201603190422"}
}
```

UpdateItem

Wenn kein Element mit dem angegebenen Schlüssel vorhanden ist, erstellt `UpdateItem` ein neues Element. Andernfalls werden die Attribute eines vorhandenen Elements geändert.

Sie verwenden einen Aktualisierungsausdruck, um die zu ändernden Attribute und deren neue Werte anzugeben. Weitere Informationen finden Sie unter Verwenden von Aktualisierungsausdrücken in DynamoDB.

Innerhalb des Aktualisierungsausdrucks verwenden Sie die Ausdrucksattributwerte als Platzhalter für die tatsächlichen Werte. Weitere Informationen finden Sie unter Verwenden von Ausdrucksattributwerten in DynamoDB.

Example

Ändern Sie verschiedene Attribute im Element `Thread`. Der optionale Parameter `ReturnValues` zeigt das Element so an, wie es nach der Aktualisierung dargestellt wird. Weitere Informationen finden Sie unter Rückgabewerte.

```
aws dynamodb update-item \
    --table-name Thread \
    --key file://key.json \
    --update-expression "SET Answered = :zero, Replies = :zero, LastPostedBy
 = :lastpostedby" \
    --expression-attribute-values file://expression-attribute-values.json \
    --return-values ALL_NEW
```

Die Argumente für `--key` werden in der Datei `key.json` gespeichert.

```
{
    "ForumName": {"S": "Amazon DynamoDB"},
    "Subject": {"S": "New discussion thread"}
}
```

Die Argumente für `--expression-attribute-values` werden in der Datei `expression-attribute-values.json` gespeichert.

```
{
    ":zero": {"N":"0"},
    ":lastpostedby": {"S":"barney@example.com"}
```

```
}
```

DeleteItem

`DeleteItem` löscht das Element mit dem angegebenen Schlüssel.

Example

Das folgende AWS CLI Beispiel zeigt, wie das `Thread` Element gelöscht wird.

```
aws dynamodb delete-item \
    --table-name Thread \
    --key file://key.json
```

Rückgabewerte

In einigen Fällen soll DynamoDB bestimmte Attributwerte so zurückgeben, wie sie vor oder nach der Änderung erschienen. Die Operationen `PutItem`, `UpdateItem` und `DeleteItem` verfügen über einen Parameter `ReturnValues`, mit dem Sie Attributwerte zurückgeben können, bevor oder nachdem diese geändert wurden.

Der Standardwert für `ReturnValues` ist `NONE`. Dies bedeutet, dass DynamoDB keine Informationen über Attribute zurückgibt, die geändert wurden.

Nachfolgend sind die anderen gültigen Einstellungen für `ReturnValues` nach DynamoDB-API-Operation sortiert aufgeführt.

PutItem

- `ReturnValues`: `ALL_OLD`
  - Wenn Sie ein vorhandenes Element überschreiben, gibt `ALL_OLD` das gesamte Element so zurück, wie es vor dem Überschreiben dargestellt wurde.
  - Wenn Sie ein nicht vorhandenes Element schreiben, hat `ALL_OLD` keine Auswirkung.

UpdateItem

Die häufigste Nutzung für `UpdateItem` ist das Aktualisieren eines vorhandenen Elements. `UpdateItem` führt eigentlich eine upsert-Operation aus. Dies bedeutet, dass das Element automatisch erstellt wird, wenn es noch nicht vorhanden ist.

- `ReturnValues`: `ALL_OLD`
  - Wenn Sie ein vorhandenes Element aktualisieren, gibt `ALL_OLD` das gesamte Element so zurück, wie es vor dem Aktualisieren dargestellt wurde.
  - Wenn Sie ein nicht vorhandenes Element aktualisieren (Upsert-Operation), hat `ALL_OLD` keine Auswirkung.
- `ReturnValues`: `ALL_NEW`
  - Wenn Sie ein vorhandenes Element aktualisieren, gibt `ALL_NEW` das gesamte Element so zurück, wie es nach dem Aktualisieren dargestellt wird.
  - Wenn Sie ein nicht vorhandenes Element aktualisieren (Upsert-Operation), gibt `ALL_NEW` das gesamte Element zurück.
- `ReturnValues`: `UPDATED_OLD`
  - Wenn Sie ein vorhandenes Element aktualisieren, gibt `UPDATED_OLD` nur die aktualisierten Attribute so zurück, wie sie vor dem Aktualisieren dargestellt wurden.
  - Wenn Sie ein nicht vorhandenes Element aktualisieren (Upsert-Operation), hat `UPDATED_OLD` keine Auswirkung.
- `ReturnValues`: `UPDATED_NEW`
  - Wenn Sie ein vorhandenes Element aktualisieren, gibt `UPDATED_NEW` nur die betroffenen Attribute so zurück, wie sie nach dem Aktualisieren dargestellt werden.
  - Wenn Sie ein nicht vorhandenes Element aktualisieren (Upsert-Operation), gibt `UPDATED_NEW` nur die aktualisierten Attribute so zurück, wie sie nach dem Aktualisieren dargestellt werden.

DeleteItem

- `ReturnValues`: `ALL_OLD`
  - Wenn Sie ein vorhandenes Element löschen, gibt `ALL_OLD` das gesamte Element so zurück, wie es vor dem Löschen dargestellt wurde.
  - Wenn Sie ein nicht vorhandenes Element löschen, gibt `ALL_OLD` keine Daten zurück.

Batch-Vorgänge

Für Anwendungen, die mehrere Elemente lesen oder schreiben müssen, stellt DynamoDB die Operationen `BatchGetItem` und `BatchWriteItem` bereit. Mithilfe dieser Operationen können Sie die Anzahl der Netzläufe von Ihrer Anwendung für DynamoDB reduzieren. Außerdem führt

DynamoDB die einzelnen Lese- oder Schreibvorgänge parallel aus. Ihre Anwendungen profitieren von dieser Parallelität, ohne Gleichzeitigkeit oder Threads verwalten zu müssen.

Die Stapeloperationen sind im Wesentlichen Wrapper für mehrere Lese- oder Schreibanforderungen. Wenn eine `BatchGetItem`-Anforderung z. B. fünf Elemente enthält, führt DynamoDB fünf `GetItem`-Operationen in Ihrem Namen aus. Wenn eine `BatchWriteItem`-Anforderung zwei PUT-Anforderungen und vier DELETE-Anforderungen enthält, führt DynamoDB zwei `PutItem`- und vier `DeleteItem`-Anforderungen aus.

Im Allgemeinen treten bei einer Stapeloperation nur Fehler auf, wenn alle im Stapel enthaltenen Anforderungen fehlschlagen. Angenommen, Sie führen eine `BatchGetItem`-Operation aus, doch bei einer der `GetItem`-Einzelanforderungen des Stapels tritt ein Fehler auf. In diesem Fall gibt `BatchGetItem` die Schlüssel und Daten der fehlgeschlagenen `GetItem`-Anforderung zurück. Die anderen `GetItem`-Anforderungen des Stapels sind davon nicht betroffen.

BatchGetItem

Eine einzelne `BatchGetItem`-Operation kann bis zu 10 individuelle `GetItem`-Anforderungen enthalten und bis zu 16 MB Daten abrufen. Darüber hinaus kann eine `BatchGetItem`-Operation Elemente aus mehreren Tabellen abrufen.

Example

Rufen Sie zwei Elemente aus der `Thread`-Tabelle mit einem Projektionsausdruck ab, um nur einige der Attribute zurückzugeben.

```
aws dynamodb batch-get-item \
    --request-items file://request-items.json
```

Die Argumente für `--request-items` werden in der Datei `request-items.json` gespeichert.

```
{
    "Thread": {
        "Keys": [
            {
                "ForumName":{"S": "Amazon DynamoDB"},
                "Subject":{"S": "DynamoDB Thread 1"}
            },
            {
                "ForumName":{"S": "Amazon S3"},
```

```
                "Subject":{"S": "S3 Thread 1"}
            }
        ],
        "ProjectionExpression":"ForumName, Subject, LastPostedDateTime, Replies"
    }
}
```

BatchWriteItem

Die `BatchWriteItem`-Operation kann bis zu 25 individuelle `PutItem` und `DeleteItem`-Anforderungen enthalten und bis zu 16 MB Daten schreiben. (Die maximale Größe eines einzelnen Elements beträgt 400 KB.) Darüber hinaus kann eine `BatchWriteItem`-Operation Elemente in mehreren Tabellen einfügen oder daraus löschen.

> **Note**
>
> `BatchWriteItem` unterstützt keine `UpdateItem`-Anforderungen.

Example

Fügen Sie der `ProductCatalog`-Tabelle zwei Elemente hinzu.

```
aws dynamodb batch-write-item \
    --request-items file://request-items.json
```

Die Argumente für `--request-items` werden in der Datei `request-items.json` gespeichert.

```
{
    "ProductCatalog": [
        {
            "PutRequest": {
                "Item": {
                    "Id": { "N": "601" },
                    "Description": { "S": "Snowboard" },
                    "QuantityOnHand": { "N": "5" },
                    "Price": { "N": "100" }
                }
            }
        },
        {
```

```
            "PutRequest": {
                "Item": {
                    "Id": { "N": "602" },
                    "Description": { "S": "Snow shovel" }
                }
            }
        }
    ]
}
```

Unteilbare Zähler

Mit der `UpdateItem`-Operation können Sie einen unteilbaren Zähler implementieren. Hierbei handelt es sich um ein numerisches Attribut, das erhöht wird, und zwar ohne Bedingung und ohne Konflikte mit anderen Schreibanforderungen. (Alle Schreibanforderungen werden in der Reihenfolge angewendet, in der sie empfangen wurden.) Mit einem unteilbaren Zähler sind die Updates nicht idempotent. Mit anderen Worten, der numerische Wert wird bei jedem Aufruf von `UpdateItem` erhöht oder verringert. Wenn der zur Aktualisierung des unteilbaren Zählers verwendete Inkrementwert positiv ist, kann dies zu einer Überzählung führen. Wenn der Inkrementwert negativ ist, kann dies zu einer Unterzählung führen.

Sie können einen unteilbaren Zähler verwenden, um die Anzahl der Besucher einer Website zu verfolgen. In diesem Fall erhöht Ihre Anwendung einen numerischen Wert, unabhängig vom aktuellen Wert. Wenn bei einer `UpdateItem`-Operation ein Fehler auftritt, kann die Anwendung die Operation einfach wiederholen. Das bringt zwar das Risiko mit sich, den Zähler zweimal zu aktualisieren, doch eine leichte Unter- oder Überzählung der Websitebesucher ist tolerierbar.

Ein unteilbarer Zähler ist nicht geeignet, wenn eine Überzählung oder Unterzählung nicht toleriert werden kann (z. B. in einer Bankanwendung). In diesem Fall ist es sicherer, ein bedingtes Update anstelle eines unteilbaren Zählers zu verwenden.

Weitere Informationen finden Sie unter Vergrößern und Verkleinern numerischer Attribute.

Example

Im folgenden AWS CLI Beispiel wird der Wert `Price` eines Produkts um 5 erhöht. In diesem Beispiel war bekannt, dass das Element existiert, bevor der Zähler aktualisiert wurde. Da `UpdateItem` nicht idempotent ist, steigt `Price` bei jedem Ausführen dieses Codes an.

```
aws dynamodb update-item \
```

```
    --table-name ProductCatalog \
    --key '{"Id": { "N": "601" }}' \
    --update-expression "SET Price = Price + :incr" \
    --expression-attribute-values '{":incr":{"N":"5"}}' \
    --return-values UPDATED_NEW
```

Bedingte Schreibvorgänge

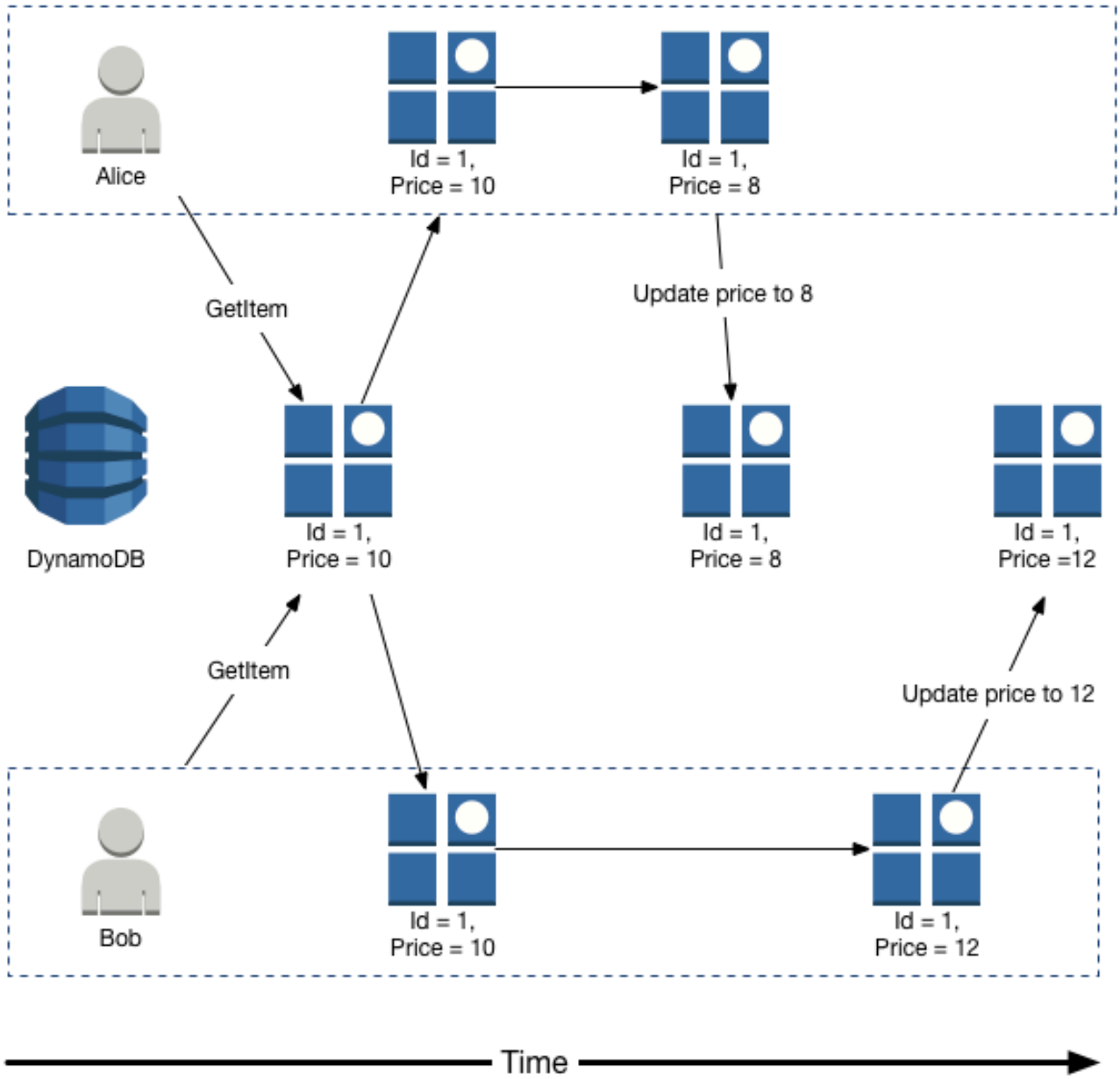
Standardmäßig sind die DynamoDB-Schreiboperationen (`PutItem`,`DeleteItem`) bedingungslos: Jede Operation überschreibt ein vorhandenes Element, das den angegebenen Primärschlüssel hat.

DynamoDB unterstützt optional bedingte Schreibvorgänge für diese Operationen. Ein bedingter Schreibvorgang wird nur dann erfolgreich ausgeführt, wenn die Elementattribute eine oder mehrere erwartete Bedingungen erfüllen. Andernfalls wird ein Fehler zurückgegeben.

Bedingte Schreibvorgänge vergleichen ihre Bedingungen mit der zuletzt aktualisierten Version des Elements. Beachten Sie, dass der bedingte Schreibvorgang kein vorheriges Element findet, wenn das Element zuvor noch nicht vorhanden war oder wenn der letzte erfolgreiche Vorgang für dieses Element ein Löschvorgang war.

Bedingte Schreibvorgänge sind in vielen Situationen hilfreich. Eine `PutItem`-Operation soll beispielsweise nur erfolgreich abgeschlossen werden, wenn noch kein Element mit demselben Primärschlüssel vorhanden ist. Sie können eine `UpdateItem`-Operation auch daran hindern, ein Element zu ändern, wenn eines seiner Attribute einen bestimmten Wert aufweist.

Bedingte Schreibvorgänge sind hilfreich, wenn mehrere Benutzer versuchen, dasselbe Element zu ändern. Betrachten Sie das folgende Diagramm, in dem zwei Benutzer (Alice und Bob) mit demselben Element aus einer DynamoDB-Tabelle arbeiten:



Angenommen, Alice verwendet das, AWS CLI um das Attribut auf 8 zu aktualisieren. `Price`

```
aws dynamodb update-item \
    --table-name ProductCatalog \
    --key '{"Id":{"N":"1"}}' \
    --update-expression "SET Price = :newval" \
```

```
    --expression-attribute-values file://expression-attribute-values.json
```

Die Argumente für `--expression-attribute-values` werden in der Datei `expression-attribute-values.json` gespeichert:

```
{
    ":newval":{"N":"8"}
}
```

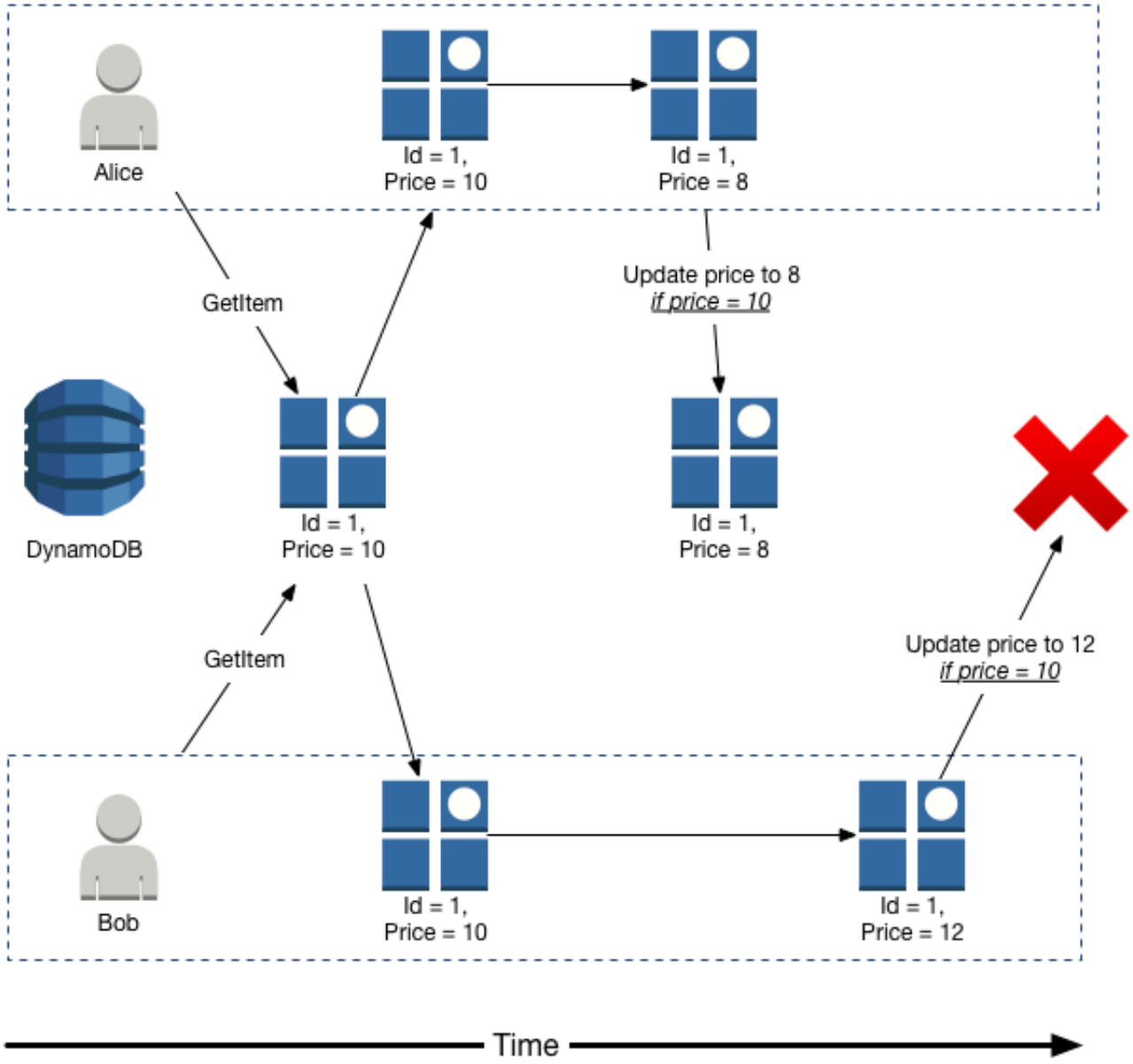
Angenommen, Bob erstellt später eine ähnliche `UpdateItem`-Anforderung, ändert den `Price` jedoch in 12. Für Bob sieht der Parameter `--expression-attribute-values` wie folgt aus:

```
{
    ":newval":{"N":"12"}
}
```

Bobs Anforderung wird erfolgreich ausgeführt, aber das Update, das Alice vorher vorgenommen hatte, geht verloren.

Für eine bedingte `PutItem`-, `DeleteItem`- oder `UpdateItem`-Anforderung geben Sie einen Bedingungsausdruck an. Ein Bedingungsausdruck ist eine Zeichenfolge, die Attributnamen, bedingte Operatoren und integrierte Funktionen enthält. Der gesamte Ausdruck muss mit True ausgewertet werden. Andernfalls schlägt sie fehl.

Betrachten Sie jetzt das folgende Diagramm, das zeigt, wie bedingte Schreibvorgänge verhindern würden, dass die von Alice vorgenommene Aktualisierung überschrieben wird:



Alice versucht zunächst, `Price` auf 8 zu aktualisieren, jedoch nur wenn der aktuelle `Price` 10 beträgt.

```
aws dynamodb update-item \
    --table-name ProductCatalog \
    --key '{"Id":{"N":"1"}}' \
    --update-expression "SET Price = :newval" \
```

```
    --condition-expression "Price = :currval" \
    --expression-attribute-values file://expression-attribute-values.json
```

Die Argumente für `--expression-attribute-values` werden in der Datei `expression-attribute-values.json` gespeichert.

```
{
    ":newval":{"N":"8"},
    ":currval":{"N":"10"}
}
```

Das Update von Alice wird erfolgreich ausgeführt, da die Bedingung mit True ausgewertet wird.

Als Nächstes versucht Bob, den `Price` auf 12 zu aktualisieren, jedoch nur wenn der aktuelle `Price` 10 beträgt. Für Bob sieht der Parameter `--expression-attribute-values` wie folgt aus:

```
{
    ":newval":{"N":"12"},
    ":currval":{"N":"10"}
}
```

Da Alice den `Price` zuvor in 8 geändert hat, wird der Bedingungsausdruck mit False ausgewertet und Bobs Aktualisierung schlägt fehl.

Weitere Informationen finden Sie unter CLI-Beispiel für DynamoDB-Bedingungsausdrücke.

Idempotenz von bedingten Schreibvorgängen

Bedingte Schreibvorgänge können idempotent sein, wenn die bedingte Prüfung für dasselbe Attribut ausgeführt wird, das aktualisiert wird. Das bedeutet, dass DynamoDB den jeweiligen Schreibvorgang nur ausführt, wenn bestimmte Attributwerte in dem Element mit den zum Zeitpunkt der Anforderung erwarteten Werten übereinstimmen.

Angenommen, Sie erstellen eine `UpdateItem`-Anforderung, um den `Price` eines Elements um 3 zu erhöhen, jedoch nur wenn der `Price` aktuell 20 beträgt. Nachdem Sie die Anforderung senden und bevor Sie die Ergebnisse erhalten, tritt ein Netzwerkfehler auf und Sie wissen nicht, ob die Anforderung erfolgreich war. Da dieser bedingte Schreibvorgang idempotent ist, können Sie dieselbe `UpdateItem`-Anforderung wiederholen und DynamoDB aktualisiert das Element nur, wenn `Price` gegenwärtig 20 beträgt.

Von bedingten Schreibvorgängen verbrauchte Kapazitätseinheiten

Wenn `ConditionExpression` während eines bedingten Schreibvorgangs als falsch ausgewertet wird, verbraucht DynamoDB weiterhin Schreibkapazität aus der Tabelle. Die verbrauchte Menge hängt von der Größe des vorhandenen Elements ab (oder beträgt mindestens 1). Wenn beispielsweise ein vorhandenes Element 300 KB groß ist und das neue Element, das Sie erstellen oder aktualisieren möchten, 310 KB hat, entsprechen die verbrauchten Schreibkapazitätseinheiten 300, wenn die Bedingung fehlschlägt, und 310, wenn die Bedingung erfolgreich ist. Wenn es sich um ein neues Element handelt (kein vorhandenes Element), beträgt die verbrauchte Schreibkapazität 1, wenn die Bedingung fehlschlägt, und 310, wenn die Bedingung erfolgreich ist.

> **Note**
>
> Schreibvorgänge verbrauchen nur Schreibkapazitätseinheiten. Sie belegen keine Lesekapazitätseinheiten.

Tritt bei einem bedingten Schreibvorgang ein Fehler auf, wird eine `ConditionalCheckFailedException` zurückgegeben. In diesem Fall erhalten Sie in der Antwort keine Informationen über die verbrauchte Schreibkapazität.

Um die Anzahl von Schreibkapazitätseinheiten zurückzugeben, die während eines bedingten Schreibvorgangs verbraucht wurden, verwenden Sie den Parameter `ReturnConsumedCapacity`:

- `TOTAL` – Gibt die Gesamtanzahl der verbrauchten Schreibkapazitätseinheiten zurück.
- `INDEXES` – Gibt die Gesamtzahl der verbrauchten Schreibkapazitätseinheiten mit Zwischensummen für die Tabelle und alle sekundären Indizes zurück, die vom Vorgang betroffen waren.
- `NONE` – Es werden keine Schreibkapazitätsdetails zurückgegeben. (Dies ist die Standardeinstellung.)

> **Note**
>
> Im Unterschied zu einem globalen sekundären Index teilt ein lokaler sekundärer Index seine bereitgestellte Durchsatzkapazität mit der Tabelle. Die Lese- und Schreibaktivität auf einem lokalen sekundären Index verbraucht die bereitgestellte Durchsatzkapazität aus der Tabelle.

Verwenden von Ausdrücken in DynamoDB

In Amazon DynamoDB können Sie mithilfe von Ausdrücken angeben, welche Attribute aus einem Element gelesen werden sollen, Daten schreiben, wenn eine Bedingung erfüllt ist, angeben, wie ein Element aktualisiert werden soll, Abfragen definieren und die Ergebnisse einer Abfrage filtern.

In dieser Tabelle werden die grundlegende Ausdrucksgrammatik und die verfügbaren Ausdrucksarten beschrieben.

Ausdruckstyp	Description
Projektionsausdruck	Ein Projektionsausdruck identifiziert die Attribute, die Sie aus einem Element abrufen möchten GetItem, wenn Sie Operationen wie Abfragen oder Scannen verwenden.
Bedingungsausdruck	Ein Bedingungsausdruck bestimmt, welche Elemente geändert werden sollen, wenn Sie die DeleteItem Operationen PutItem UpdateItem, und verwenden.
Aktualisierungsausdruck	Ein Aktualisierungsausdruck gibt an, wie die Attribute eines Elements geändert UpdateIte m werden, z. B. durch das Festlegen eines Skalarwerts oder das Entfernen von Elementen aus einer Liste oder einer Map.
Schlüsselbedingungsausdruck	Ein Schlüsselbedingungsausdruck bestimmt, welche Elemente eine Abfrage aus einer Tabelle oder einem Index liest.
Filterausdruck	Ein Filterausdruck bestimmt, welche Elemente in den Abfrageergebnissen zurückgegeben werden. Alle anderen Ergebnisse werden verworfen.

Weitere Informationen zur Ausdruckssyntax und Details zu jedem Ausdruckstyp finden Sie in folgenden Abschnitten.

Themen

- Verweisen auf Elementattribute beim Verwenden von Ausdrücken in DynamoDB
- Ausdrucksattributnamen (Aliasse) in DynamoDB
- Verwenden von Ausdrucksattributwerten in DynamoDB
- Verwenden von Projektionsausdrücken in DynamoDB
- Verwenden von Aktualisierungsausdrücken in DynamoDB
- Bedingungs- und Filterausdrücke, Operatoren und Funktionen in DynamoDB
- CLI-Beispiel für DynamoDB-Bedingungsausdrücke

> **Note**
>
> Für die Abwärtskompatibilität unterstützt DynamoDB ebenfalls bedingte Parameter, die keine Ausdrücke verwenden. Weitere Informationen finden Sie unter Bedingte Legacy-Parameter für DynamoDB.
> Neue Anwendungen sollten Ausdrücke anstelle der Legacy-Parameter verwenden.

Verweisen auf Elementattribute beim Verwenden von Ausdrücken in DynamoDB

Dieser Abschnitt beschreibt, wie Sie sich auf die Elementattribute in einem Ausdruck in Amazon DynamoDB beziehen können. Sie können mit jedem beliebigen Attribut arbeiten, auch wenn es tief innerhalb mehrerer Listen und Zuordnungen verschachtelt ist.

Themen

- Attribute der obersten Stufe
- Verschachtelte Attribute
- Dokumentpfade

Ein Beispielelement: ProductCatalog

Bei den Beispielen auf dieser Seite wurde folgendes Musterelement in der `ProductCatalog`-Tabelle verwendet. (Diese Tabelle wird in Beispieltabellen und -daten zur Verwendung in DynamoDB beschrieben.)

```
{
```

```
    "Id": 123,
    "Title": "Bicycle 123",
    "Description": "123 description",
    "BicycleType": "Hybrid",
    "Brand": "Brand-Company C",
    "Price": 500,
    "Color": ["Red", "Black"],
    "ProductCategory": "Bicycle",
    "InStock": true,
    "QuantityOnHand": null,
    "RelatedItems": [
        341,
        472,
        649
    ],
    "Pictures": {
        "FrontView": "http://example.com/products/123_front.jpg",
        "RearView": "http://example.com/products/123_rear.jpg",
        "SideView": "http://example.com/products/123_left_side.jpg"
    },
    "ProductReviews": {
     "FiveStar": [
       "Excellent! Can't recommend it highly enough! Buy it!",
       "Do yourself a favor and buy this."
     ],
     "OneStar": [
       "Terrible product! Do not buy this."
     ]
    },
    "Comment": "This product sells out quickly during the summer",
    "Safety.Warning": "Always wear a helmet"
}
```

Beachten Sie Folgendes:

- Der Partitions-Schlüsselwert (`Id`) ist `123`. Es gibt keinen Sortierschlüssel.
- Die meisten Attribute verfügen über skalare Datentypen, z. B. `String`, `Number`, `Boolean` und `Null`.
- Ein Attribut (`Color`) ist ein `String Set`.
- Die folgenden Attribute sind Dokumentdatentypen:
  - Eine Liste von `RelatedItems`. Jedes Element ist eine `Id` für ein zugehöriges Produkt.

- Eine Zuordnung von `Pictures`. Jedes Element ist eine kurze Beschreibung eines Bildes, zusammen mit einem URL für die entsprechende Imagedatei.
- Eine Zuordnung von `ProductReviews`. Jedes Element repräsentiert eine Bewertung und eine Liste von Rezensionen dieser Bewertung. Zunächst wird diese Zuordnung mit 5-Sterne- und 1-Stern-Rezensionen gefüllt.

Attribute der obersten Stufe

Ein Attribut befindet sich auf oberster Ebene, wenn es nicht in einem anderen Attribut eingebettet ist. Für das `ProductCatalog`-Element lauten die Attribute auf oberster Ebene wie folgt:

- `Id`
- `Title`
- `Description`
- `BicycleType`
- `Brand`
- `Price`
- `Color`
- `ProductCategory`
- `InStock`
- `QuantityOnHand`
- `RelatedItems`
- `Pictures`
- `ProductReviews`
- `Comment`
- `Safety.Warning`

Alle diese Attribute auf oberster Ebene sind Skalare, mit Ausnahme von `Color` (Liste), `RelatedItems` (Liste), `Pictures` (Zuordnung) und `ProductReviews` (Zuordnung).

Verschachtelte Attribute

Ein Attribut gilt als verschachtelt, wenn es in einem anderen Attribut eingebettet ist. Um auf das verschachtelte Attribut zuzugreifen, verwenden Sie die Dereferenzierungsoperatoren:

- `[n]` – für Listenelemente
- `.` (Punkt) – für Zuordnungselemente

Zugriff auf Listenelemente

Der Dereferenzierungsoperator für das Listenelement ist [N], wobei n die Elementnummer ist. Listenelemente sind nullbasiert, sodass [0] das erste Element in der Liste darstellt, [1] das zweite und so weiter. Hier sind einige Beispiele:

- `MyList[0]`
- `AnotherList[12]`
- `ThisList[5][11]`

Das Element `ThisList[5]` ist selbst eine verschachtelte Liste. Daher bezieht sich `ThisList[5][11]` auf das zwölfte Element in dieser Liste.

Die Zahl innerhalb der eckigen Klammern muss eine nicht negative ganze Zahl sein. Daher sind die folgenden Ausdrücke ungültig:

- `MyList[-1]`
- `MyList[0.4]`

Zugriff auf Zuweisungselemente

Der Dereferenzierungsoperator für ein Zuordnungselement ist . (Punkt). Verwenden Sie einen Punkt als Trennzeichen zwischen den Elementen in einer Zuordnung:

- `MyMap.nestedField`
- `MyMap.nestedField.deeplyNestedField`

Dokumentpfade

In einem Ausdruck verwenden Sie einen Dokumentpfad, um DynamoDB mitzuteilen, wo ein Attribut zu finden ist. Für eine Attribut auf oberster Ebene ist der Dokumentpfad einfach der Attributname. Sie erstellen den Dokumentpfad für ein verschachteltes Attribut mithilfe des Dereferenzoperators.

Es folgen einige Beispiele für Dokumentpfade. (Beziehen Sie sich auf das Element, das in Verweisen auf Elementattribute beim Verwenden von Ausdrücken in DynamoDB gezeigt wird.)

- Ein skalares Attribut auf oberster Ebene.

  `Description`

- Ein Listenattribut auf oberster Ebene. (Dies gibt die gesamte Liste zurück und nicht nur einige der Elemente.)

  `RelatedItems`

- Das dritte Element aus der `RelatedItems`-Liste. (Denken Sie daran, dass Listenelemente nullbasiert sind.)

  `RelatedItems[2]`

- Das Bild des Produkts in der Vorderansicht

  `Pictures.FrontView`

- Alle 5-Sterne-Rezensionen

  `ProductReviews.FiveStar`

- Die erste 5-Sterne-Rezension

  `ProductReviews.FiveStar[0]`

> **Note**
>
> Die maximale Tiefe eines Dokumentpfads ist 32. Aus diesem Grund kann die Anzahl der Dereferenzoperatoren in einem Pfad diesen Grenzwert nicht überschreiten.

Sie können jeden Attributnamen in einem Dokumentpfad verwenden, sofern er die folgenden Anforderungen erfüllt:

- Das erste Zeichen ist `a-z` oder `A-Z` und/oder `0-9`.
- Das zweite Zeichen (falls vorhanden) ist `a-z`, `A-Z`.

> **Note**
>
> Wenn ein Attributname diesen Anforderungen nicht entspricht, müssen Sie einen Ausdrucksattributnamen als Platzhalter definieren.

Weitere Informationen finden Sie unter Ausdrucksattributnamen (Aliasse) in DynamoDB.

Ausdrucksattributnamen (Aliasse) in DynamoDB

Ein Ausdrucksattributname ist ein Alias (oder Platzhalter), den Sie in einem Amazon-DynamoDB-Ausdruck als Alternative zu einem tatsächlichen Attributnamen verwenden. Ein Ausdrucksattributname muss mit einem Rautezeichen (#) beginnen, gefolgt von einem oder mehreren alphanumerischen Zeichen. Der Unterstrich (_) ist ebenfalls zulässig.

Dieser Abschnitt beschreibt einige Situationen, in denen Sie Ausdrucksattributnamen verwenden müssen.

> **Note**
>
> Die Beispiele in diesem Abschnitt verwenden das AWS Command Line Interface (AWS CLI).

Themen

- Reservierte Wörter
- Attributnamen mit Sonderzeichen
- Verschachtelte Attribute
- Wiederholtes Verweisen auf Attributnamen

Reservierte Wörter

In einigen Fällen kann es erforderlich sein, einen Ausdruck mit einem Attributnamen zu erstellen, der mit einem für DynamoDB reservierten Wort in Konflikt steht. (Eine vollständige Liste der reservierten Wörter finden Sie unter Reservierte Wörter in DynamoDB.)

Das folgende Beispiel würde AWS CLI beispielsweise fehlschlagen, da es sich um ein reserviertes Wort `COMMENT` handelt.

```
aws dynamodb get-item \
    --table-name ProductCatalog \
    --key '{"Id":{"N":"123"}}' \
    --projection-expression "Comment"
```

Um dieses Problem zu vermeiden, können Sie `Comment` durch einen Ausdrucksattributnamen wie `#c` ersetzen. Das Rautezeichen # ist erforderlich und gibt an, dass es sich um einen Platzhalter für einen Attributnamen handelt. Das AWS CLI Beispiel würde jetzt wie folgt aussehen.

```
aws dynamodb get-item \
     --table-name ProductCatalog \
     --key '{"Id":{"N":"123"}}' \
     --projection-expression "#c" \
     --expression-attribute-names '{"#c":"Comment"}'
```

> **Note**
>
> Wenn ein Attributname mit einer Zahl beginnt, ein Leerzeichen oder ein reserviertes Wort enthält, müssen Sie einen Ausdrucksattributnamen verwenden, um den Namen des Attributs im Ausdruck zu ersetzen.

Attributnamen mit Sonderzeichen

In einem Ausdruck wird ein Punkt (".") als Trennzeichen in einem Dokumentpfad interpretiert. DynamoDB bietet allerdings die Möglichkeit, ein Punktzeichen und andere Sonderzeichen wie einen Bindestrich („-“) in einem Attributnamen zu verwenden. Dies kann in einigen Fällen zweideutig sein. Angenommen, Sie möchten das Attribut `Safety.Warning` aus einem `ProductCatalog`-Element abrufen (siehe Verweisen auf Elementattribute beim Verwenden von Ausdrücken in DynamoDB).

In diesem Beispiel gehen wir davon aus, dass Sie auf `Safety.Warning` mit einem Projektionsausdruck zugreifen möchten.

```
aws dynamodb get-item \
    --table-name ProductCatalog \
    --key '{"Id":{"N":"123"}}' \
    --projection-expression "Safety.Warning"
```

DynamoDB gibt statt der erwarteten Zeichenfolge („`Always wear a helmet`“) ein leeres Ergebnis zurück. Der Grund dafür ist, dass DynamoDB einen Punkt in einem Ausdruck als Dokumentpfadtrennzeichen interpretiert. In diesem Fall müssen Sie einen Ausdrucksattributnamen (z. B. `#sw`) als Ersatz für `Safety.Warning` definieren. Anschließend können Sie den folgenden Projektionsausdruck verwenden.

```
aws dynamodb get-item \
    --table-name ProductCatalog \
    --key '{"Id":{"N":"123"}}' \
    --projection-expression "#sw" \
    --expression-attribute-names '{"#sw":"Safety.Warning"}'
```

DynamoDB gibt dann das richtige Ergebnis zurück.

> **Note**
>
> Wenn ein Attributname einen Punkt („.“) oder einen Bindestrich („-“) enthält, müssen Sie einen Ausdrucksattributnamen verwenden, um den Namen dieses Attributs im Ausdruck zu ersetzen.

Verschachtelte Attribute

Angenommen, Sie möchten auf das verschachtelte Attribut `ProductReviews.OneStar` zugreifen. In einem Ausdrucksattributnamen behandelt DynamoDB den Punkt („.“) als Zeichen in einem Attributnamen. Um auf das verschachtelte Attribut zu verweisen, definieren Sie einen Ausdrucksattributnamen für jedes Element im Dokumentpfad:

- `#pr – ProductReviews`
- `#1star – OneStar`

Anschließend können Sie `#pr.#1star` für den folgenden Projektionsausdruck verwenden.

```
aws dynamodb get-item \
    --table-name ProductCatalog \
    --key '{"Id":{"N":"123"}}' \
    --projection-expression "#pr.#1star"  \
    --expression-attribute-names '{"#pr":"ProductReviews", "#1star":"OneStar"}'
```

DynamoDB gibt dann das richtige Ergebnis zurück.

Wiederholtes Verweisen auf Attributnamen

Ausdrucksattributnamen sind hilfreich, wenn Sie sich wiederholt auf den gleichen Attributnamen beziehen müssen. Betrachten Sie z. B. den folgenden Ausdruck zum Abrufen einiger Rezensionen aus einem `ProductCatalog`-Element.

```
aws dynamodb get-item \
    --table-name ProductCatalog \
    --key '{"Id":{"N":"123"}}' \
    --projection-expression "ProductReviews.FiveStar, ProductReviews.ThreeStar,
 ProductReviews.OneStar"
```

Um den Ausdruck prägnanter zu gestalten, können Sie `ProductReviews` durch einen Ausdrucksattributnamen wie `#pr` ersetzen. Der geänderte Ausdruck sieht wie folgt aus:

- `#pr.FiveStar, #pr.ThreeStar, #pr.OneStar`

```
aws dynamodb get-item \
    --table-name ProductCatalog \
    --key '{"Id":{"N":"123"}}' \
    --projection-expression "#pr.FiveStar, #pr.ThreeStar, #pr.OneStar" \
    --expression-attribute-names '{"#pr":"ProductReviews"}'
```

Wenn Sie einen Ausdrucksattributnamen definieren, müssen Sie ihn im gesamten Ausdruck einheitlich verwenden. Sie dürfen außerdem nicht das Symbol # weglassen.

Verwenden von Ausdrucksattributwerten in DynamoDB

Ausdrucksattributwerte in Amazon DynamoDB agieren als Variablen. Sie ersetzen die tatsächlichen Werte, die Sie vergleichen möchten – Werte, die Sie möglicherweise bis zur Laufzeit nicht kennen. Ein Ausdrucksattributwert muss mit einem Doppelpunkt (`:`) beginnen, gefolgt von einem oder mehreren alphanumerischen Zeichen.

Angenommen, Sie möchten alle `ProductCatalog`-Elemente, die in `Black` verfügbar sind und `500` oder weniger kosten, zurückgeben. Sie können eine `Scan`-Operation mit einem Filterausdruck verwenden, wie in diesem AWS Command Line Interface (AWS CLI)-Beispiel gezeigt:

```
aws dynamodb scan \
```

```
    --table-name ProductCatalog \
    --filter-expression "contains(Color, :c) and Price <= :p" \
    --expression-attribute-values file://values.json
```

Die Argumente für `--expression-attribute-values` werden in der Datei `values.json` gespeichert.

```
{
    ":c": { "S": "Black" },
    ":p": { "N": "500" }
}
```

Wenn Sie einen Ausdrucksattributwert definieren, müssen Sie ihn während des gesamten Ausdrucks einheitlich verwenden. Sie dürfen außerdem das Symbol `:` nicht weglassen.

Ausdrucksattributwerte werden mit Schlüsselbedingungsausdrücken, Bedingungsausdrücken, Aktualisierungsausdrücken und Filterausdrücken verwendet.

Verwenden von Projektionsausdrücken in DynamoDB

Um Daten aus einer Tabelle zu lesen, verwenden sie Operationen wie `GetItem`, `Query` oder `Scan`. Amazon DynamoDB gibt standardmäßig alle Elementattribute zurück. Verwenden Sie einen Projektionsausdruck, um nur einige und nicht alle Attribute abzurufen.

Ein Projektionsausdruck ist eine Zeichenfolge, mit der die gewünschten Attribute identifiziert werden. Zum Abrufen eines einzelnen Attributs geben Sie seinen Namen an. Für mehrere Attribute müssen die Namen durch Kommas getrennt werden.

Es folgen einige Beispiele für Projektionsausdrücke, basierend auf dem `ProductCatalog`-Element von Verweisen auf Elementattribute beim Verwenden von Ausdrücken in DynamoDB:

- Ein einzelnes Attribut auf oberster Ebene:

  ```
  Title
  ```

- Drei Attribute auf oberster Ebene. DynamoDB ruft die gesamte `Color`-Einstellung.

  ```
  Title, Price, Color
  ```

- Vier Attribute auf oberster Ebene. DynamoDB gibt den gesamten Inhalt von `RelatedItems` und `ProductReviews` zurück.

  ```
  Title, Description, RelatedItems, ProductReviews
  ```

> **Note**
>
> Der Projektionsausdruck wirkt sich nicht auf den Verbrauch des bereitgestellten Durchsatzes aus. DynamoDB ermittelt die verbrauchten Kapazitätseinheiten basierend auf der Elementgröße, nicht anhand der Menge der Daten, die an eine Anwendung zurückgegeben werden.

Reservierte Wörter und Sonderzeichen

DynamoDB verfügt über reservierte Wörter und Sonderzeichen. DynamoDB lässt diese reservierten Wörter und Sonderzeichen für Namen zu. Dennoch empfehlen wir, diese nicht zu verwenden, da Sie Aliasse für diese Namen definieren müssen, sobald Sie sie in einem Ausdruck verwenden. Eine vollständige Liste finden Sie hier: Reservierte Wörter in DynamoDB.

In folgenden Fällen müssen Sie Ausdrucksattributnamen anstelle des eigentlichen Namens verwenden:

- Der Attributname steht auf der Liste der reservierten Wörter von DynamoDB.
- Der Attributname erfüllt nicht die Anforderung, dass das erste Zeichen `a-z` oder `A-Z` und das zweite Zeichen (falls vorhanden) `a-Z`, `A-Z` oder `0-9` sein muss.
- Der Attributname enthält # (Raute) oder : (Doppelpunkt).

Das folgende AWS CLI Beispiel zeigt, wie ein Projektionsausdruck mit einer `GetItem` Operation verwendet wird. Dieser Projektionsausdruck ruft ein skalares Attribut auf oberster Ebene (`Description`), das erste Element in einer Liste (`RelatedItems[0]`) und eine Liste, die innerhalb einer Zuordnung verschachtelt ist (`ProductReviews.FiveStar`), ab.

```
aws dynamodb get-item \
    --table-name ProductCatalog \
    --key '"Id": { "N": "123" } \
    --projection-expression "Description, RelatedItems[0], ProductReviews.FiveStar"
```

In diesem Beispiel wird der folgende JSON zurückgegeben.

```
{
    "Item": {
        "Description": {
```

```
            "S": "123 description"
        },
        "ProductReviews": {
            "M": {
                "FiveStar": {
                    "L": [
                        {
                            "S": "Excellent! Can't recommend it highly enough! Buy it!"
                        },
                        {
                            "S": "Do yourself a favor and buy this."
                        }
                    ]
                }
            }
        },
        "RelatedItems": {
            "L": [
                {
                    "N": "341"
                }
            ]
        }
    }
}
```

Verwenden von Aktualisierungsausdrücken in DynamoDB

Die `UpdateItem`-Operation aktualisiert ein vorhandenes Elements oder fügt ein neues Element in die Tabelle ein, wenn es noch nicht existiert. Sie müssen den Schlüssel des Elements angeben, das Sie aktualisieren möchten. Außerdem müssen Sie einen Aktualisierungsausdruck angeben, der die zu ändernden Attribute sowie die Werte enthält, die Sie ihnen zuweisen möchten.

Ein Aktualisierungsausdruck gibt an, wie `UpdateItem` die Attribute eines Elements ändert, z. B. durch Festlegen eines Skalarwerts oder durch Entfernen von Elementen aus einer Liste oder Zuordnung.

Im Folgenden finden Sie eine Syntaxzusammenfassung für Aktualisierungsausdrücke.

```
update-expression ::=
    [ SET action [, action] ... ]
    [ REMOVE action [, action] ...]
```

```
    [ ADD action [, action] ... ]
    [ DELETE action [, action] ...]
```

Ein Aktualisierungsausdruck besteht aus einer oder mehreren Klauseln. Jede Klausel beginnt mit einem SET-, REMOVE-, ADD- oder DELETE-Schlüsselwort. Sie können jede dieser Klauseln in beliebiger Reihenfolge in einen Aktualisierungsausdruck einfügen. Jedes Aktionsschlüsselwort kann jedoch nur einmal angezeigt werden.

Jede Klausel umfasst eine oder mehrere Aktionen, die durch Komma getrennt sind. Jede Aktion stellt eine Datenänderung dar.

Die Beispiele in diesem Abschnitt basieren auf dem Element `ProductCatalog`, wie in Verwenden von Projektionsausdrücken in DynamoDB dargestellt.

Die folgenden Themen behandeln verschiedene Anwendungsfälle für die SET-Aktion.

Themen

- SET – Ändern oder Hinzufügen von Elementattributen
- REMOVE – Löschen von Attributen aus einem Element
- ADD – Aktualisieren von Zahlen und Sätzen
- DELETE – Entfernen von Elementen aus einem Satz
- Verwenden mehrerer Aktualisierungsausdrücke

SET – Ändern oder Hinzufügen von Elementattributen

Verwenden Sie die SET-Aktion in einem Aktualisierungsausdruck, um ein oder mehrere Attribute zu einem Element hinzuzufügen. Wenn diese Attribute bereits vorhanden sind, werden sie durch die neuen Werte überschrieben. Wenn Sie verhindern möchten, dass ein vorhandenes Attribut überschrieben wird, können Sie SET mit der Funktion `if_not_exists` verwenden. Die Funktion `if_not_exists` gilt spezifisch für die SET-Aktion und kann nur in einem Aktualisierungsausdruck verwendet werden.

Wenn Sie SET verwenden, um ein Listenelement zu aktualisieren, wird der Inhalt dieses Elements durch die neuen Daten ersetzt, die Sie angeben. Wenn das Element noch nicht vorhanden ist, fügt SET das neue Element am Ende der Liste an.

Wenn Sie mehrere Elemente in einer einzigen SET-Operation hinzufügen, werden die Elemente nach Elementnummer sortiert.

Sie können `SET` auch verwenden, um einen Wert zu einem Attribut des Typs `Number` zu addieren oder von diesem zu subtrahieren. Wenn Sie mehrere `SET`-Aktionen durchführen möchten, trennen Sie sie durch Komma.

Schauen Sie sich die folgende Syntaxzusammenfassung an:

- Das *`path`* Element ist der Dokumentpfad zu dem Element.
- Ein *`operand`*Element kann entweder ein Dokumentpfad zu einem Element oder eine Funktion sein.

```
set-action ::=
    path = value

value ::=
    operand
    | operand '+' operand
    | operand '-' operand

operand ::=
    path | function

function ::=
    if_not_exists (path, value)
```

Wenn das Element kein Attribut im angegebenen Pfad enthält, ergibt `if_not_exists` `value`. Andernfalls ergibt es `path`.

Die folgende `PutItem`-Operation erstellt ein Beispielelement, auf das wir uns in den Beispielen beziehen werden.

```
aws dynamodb put-item \
    --table-name ProductCatalog \
    --item file://item.json
```

Die Argumente für `--item` werden in der Datei `item.json` gespeichert. (Der Einfachheit halber werden nur wenige Elementattribute verwendet.)

```
{
    "Id": {"N": "789"},
```

```
    "ProductCategory": {"S": "Home Improvement"},
    "Price": {"N": "52"},
    "InStock": {"BOOL": true},
    "Brand": {"S": "Acme"}
}
```

Themen

- Ändern von Attributen
- Hinzufügen von Listen und Zuordnungen
- Hinzufügen von Elementen zu einer Liste
- Hinzufügen verschachtelter Zuordnungsattribute
- Vergrößern und Verkleinern numerischer Attribute
- Anfügen von Elementen zu einer Liste
- Verhindern der Überschreibung eines vorhandenen Attributs

Ändern von Attributen

Example

Aktualisieren Sie die Attribute `ProductCategory` und `Price`.

```
aws dynamodb update-item \
    --table-name ProductCatalog \
    --key '{"Id":{"N":"789"}}' \
    --update-expression "SET ProductCategory = :c, Price = :p" \
    --expression-attribute-values file://values.json \
    --return-values ALL_NEW
```

Die Argumente für `--expression-attribute-values` werden in der Datei `values.json` gespeichert.

```
{
    ":c": { "S": "Hardware" },
    ":p": { "N": "60" }
}
```

> **Note**
>
> In der Operation `UpdateItem` führt `--return-values ALL_NEW` dazu, dass DynamoDB das Element so zurückgibt, wie es nach der Aktualisierung erscheint.

Hinzufügen von Listen und Zuordnungen

Example

Hinzufügen einer neuen Liste und Zuordnung:

```
aws dynamodb update-item \
    --table-name ProductCatalog \
    --key '{"Id":{"N":"789"}}' \
    --update-expression "SET RelatedItems = :ri, ProductReviews = :pr" \
    --expression-attribute-values file://values.json \
    --return-values ALL_NEW
```

Die Argumente für `--expression-attribute-values` werden in der Datei `values.json` gespeichert.

```
{
    ":ri": {
        "L": [
            { "S": "Hammer" }
        ]
    },
    ":pr": {
        "M": {
            "FiveStar": {
                "L": [
                    { "S": "Best product ever!" }
                ]
            }
        }
    }
}
```

Hinzufügen von Elementen zu einer Liste

Example

Fügen Sie der Liste `RelatedItems` ein neues Attribut hinzu. (Denken Sie daran, dass Listenelemente nullbasiert sind, sodass [0] das erste Element in der Liste darstellt, [1] das zweite und so weiter.)

```
aws dynamodb update-item \
    --table-name ProductCatalog \
    --key '{"Id":{"N":"789"}}' \
    --update-expression "SET RelatedItems[1] = :ri" \
    --expression-attribute-values file://values.json \
    --return-values ALL_NEW
```

Die Argumente für `--expression-attribute-values` werden in der Datei `values.json` gespeichert.

```
{
    ":ri": { "S": "Nails" }
}
```

> **Note**
>
> Wenn Sie `SET` verwenden, um ein Listenelement zu aktualisieren, wird der Inhalt dieses Elements durch die neuen Daten ersetzt, die Sie angeben. Wenn das Element noch nicht vorhanden ist, fügt `SET` das neue Element am Ende der Liste an.
> Wenn Sie mehrere Elemente in einer einzigen `SET`-Operation hinzufügen, werden die Elemente nach Elementnummer sortiert.

Hinzufügen verschachtelter Zuordnungsattribute

Example

Fügen Sie einige verschachtelte Zuordnungsattribute hinzu.

```
aws dynamodb update-item \
    --table-name ProductCatalog \
    --key '{"Id":{"N":"789"}}' \
```

```
    --update-expression "SET #pr.#5star[1] = :r5, #pr.#3star = :r3" \
    --expression-attribute-names file://names.json \
    --expression-attribute-values file://values.json \
    --return-values ALL_NEW
```

Die Argumente für `--expression-attribute-names` werden in der Datei `names.json` gespeichert.

```
{
    "#pr": "ProductReviews",
    "#5star": "FiveStar",
    "#3star": "ThreeStar"
}
```

Die Argumente für `--expression-attribute-values` werden in der Datei `values.json` gespeichert.

```
{
    ":r5": { "S": "Very happy with my purchase" },
    ":r3": {
        "L": [
            { "S": "Just OK - not that great" }
        ]
    }
}
```

Important

Sie können die Attribute einer verschachtelten Map nicht aktualisieren, wenn die übergeordnete Map nicht existiert. Wenn Sie versuchen, ein verschachteltes Attribut zu aktualisieren (z. B.`ProductReviews.FiveStar`), obwohl die übergeordnete Zuordnung (`ProductReviews`) nicht vorhanden ist, gibt DynamoDB a `ValidationException` mit der Meldung „Der im Aktualisierungsausdruck angegebene Dokumentpfad ist für die Aktualisierung ungültig“ zurück.
Wenn Sie Elemente erstellen, für die die Attribute der verschachtelten Map später aktualisiert werden, initialisieren Sie leere Maps für die übergeordneten Attribute. Beispiel:

```
{
    "Id": {"N": "789"},
    "ProductReviews": {"M": {}},
```

```
    "Metadata": {"M": {}}
}
```

Auf diese Weise können Sie verschachtelte Attribute ohne Fehler aktualisieren. `ProductReviews.FiveStar`

Vergrößern und Verkleinern numerischer Attribute

Sie können ein vorhandenes numerisches Attribut vergrößern oder verkleinern. Dazu verwenden Sie die Operatoren + (plus) und - (minus).

Example

Verringern Sie den `Price` eines Elements.

```
aws dynamodb update-item \
    --table-name ProductCatalog \
    --key '{"Id":{"N":"789"}}' \
    --update-expression "SET Price = Price - :p" \
    --expression-attribute-values '{":p": {"N":"15"}}' \
    --return-values ALL_NEW
```

Um den `Price` zu erhöhen, verwenden Sie den Operator + im Aktualisierungsausdruck.

Anfügen von Elementen zu einer Liste

Sie können Elemente an das Ende einer Liste anfügen. Dazu verwenden Sie `SET` mit der Funktion `list_append`. (Beim Funktionsnamen muss die Groß- und Kleinschreibung beachtet werden.) Die Funktion `list_append` gilt spezifisch für die `SET`-Aktion und kann nur in einem Aktualisierungsausdruck verwendet werden. Die Syntax ist wie folgt.

- `list_append (`*`list1`*`, `*`list2`*`)`

Die Funktion nimmt zwei Listen als Eingabe und fügt alle Elemente von *`list2`* bis *`list1`* an.

Example

Unter Hinzufügen von Elementen zu einer Liste erstellen Sie die Liste `RelatedItems` und fügen ihr zwei Elemente hinzu: `Hammer` und `Nails`. Nun fügen Sie zwei weitere Elemente an das Ende von `RelatedItems` an.

```
aws dynamodb update-item \
    --table-name ProductCatalog \
    --key '{"Id":{"N":"789"}}' \
    --update-expression "SET #ri = list_append(#ri, :vals)" \
    --expression-attribute-names '{"#ri": "RelatedItems"}' \
    --expression-attribute-values file://values.json  \
    --return-values ALL_NEW
```

Die Argumente für `--expression-attribute-values` werden in der Datei `values.json` gespeichert.

```
{
    ":vals": {
        "L": [
            { "S": "Screwdriver" },
            {"S": "Hacksaw" }
        ]
    }
}
```

Am Schluss fügen Sie ein weiteres Element an den Anfang von `RelatedItems` an. Vertauschen Sie dazu die Reihenfolge der `list_append`-Elemente. (Beachten Sie, dass `list_append` die beiden Listen als Eingabe übernimmt und die zweite Liste an die erste anfügt.)

```
aws dynamodb update-item \
    --table-name ProductCatalog \
    --key '{"Id":{"N":"789"}}' \
    --update-expression "SET #ri = list_append(:vals, #ri)" \
    --expression-attribute-names '{"#ri": "RelatedItems"}' \
    --expression-attribute-values '{":vals": {"L": [ { "S": "Chisel" }]}}' \
    --return-values ALL_NEW
```

Das daraus resultierende Attribut `RelatedItems` enthält jetzt fünf Elemente in der folgenden Reihenfolge: `Chisel`, `Hammer`, `Nails`, `Screwdriver`, `Hacksaw`.

Verhindern der Überschreibung eines vorhandenen Attributs

Example

Legen Sie den `Price` eines Elements fest, jedoch nur dann, wenn das Element noch nicht über ein `Price`-Attribut verfügt. (Wenn `Price` bereits vorhanden ist, ändert sich nichts.)

```
aws dynamodb update-item \
    --table-name ProductCatalog \
    --key '{"Id":{"N":"789"}}' \
    --update-expression "SET Price = if_not_exists(Price, :p)" \
    --expression-attribute-values '{":p": {"N": "100"}}' \
    --return-values ALL_NEW
```

REMOVE – Löschen von Attributen aus einem Element

Verwenden Sie die `REMOVE`-Aktion in einem Aktualisierungsausdruck, um ein oder mehrere Attribute aus einem Element in Amazon DynamoDB zu entfernen. Wenn Sie mehrere `REMOVE`-Aktionen durchführen möchten, trennen Sie sie durch Komma.

Im Folgenden finden Sie eine Syntaxzusammenfassung für `REMOVE` in einem Aktualisierungsausdruck. Der einzige Operand ist der Dokumentpfad für das Attribut, das Sie entfernen möchten.

```
remove-action ::=
    path
```

Example

Löschen einiger Attribute aus einem Element. (Wenn die Attribute nicht vorhanden sind, ändert sich nichts.)

```
aws dynamodb update-item \
    --table-name ProductCatalog \
    --key '{"Id":{"N":"789"}}' \
    --update-expression "REMOVE Brand, InStock, QuantityOnHand" \
    --return-values ALL_NEW
```

Entfernen von Elementen aus einer Liste

Sie können `REMOVE` verwenden, um einzelne Element aus einer Liste zu löschen.

Example

Unter Anfügen von Elementen zu einer Liste haben Sie ein Listenattribut (`RelatedItems`) so geändert, dass es fünf Elemente enthält:

- `[0]—Chisel`

- `[1]—Hammer`
- `[2]—Nails`
- `[3]—Screwdriver`
- `[4]—Hacksaw`

Das folgende Beispiel AWS Command Line Interface (AWS CLI) löscht `Hammer` und `Nails` aus der Liste.

```
aws dynamodb update-item \
    --table-name ProductCatalog \
    --key '{"Id":{"N":"789"}}' \
    --update-expression "REMOVE RelatedItems[1], RelatedItems[2]" \
    --return-values ALL_NEW
```

Nach dem Entfernen von `Hammer` und `Nails` werden die verbleibenden Elemente verschoben. Die Liste enthält nun Folgendes:

- `[0]—Chisel`
- `[1]—Screwdriver`
- `[2]—Hacksaw`

ADD – Aktualisieren von Zahlen und Sätzen

> **Note**
>
> Im Allgemeinen empfehlen wir, idempotente Operationen `ADD` zu verwenden, `SET` anstatt sie sicherzustellen.

Verwenden Sie die `ADD`-Aktion in einem Aktualisierungsausdruck, um ein neues Attribut mit seinen zugehörigen Werten einem Element hinzuzufügen.

Wenn das Attribut bereits vorhanden ist, hängt das Verhalten von `ADD` vom Datentyp des Attributs ab.

- Wenn das vorhandene Attribut eine Zahl ist und der Wert, den Sie hinzufügen, ebenfalls eine Zahl, wird der Wert mathematisch zum vorhandenen Attribut addiert. (Wenn der Wert eine negative Zahl ist, wird er vom vorhandenen Attribut abgezogen.)

- Wenn es sich bei dem Attribut um einen Satz handelt und der Wert, den Sie hinzufügen, ebenfalls ein Satz ist, wird der Wert an den vorhandenen Satz angefügt.

> **Note**
>
> Die ADD-Aktion unterstützt nur die Datentypen "Zahl" und "Satz".

Wenn Sie mehrere ADD-Aktionen durchführen möchten, trennen Sie sie durch Komma.

Schauen Sie sich die folgende Syntaxzusammenfassung an:

- Das *path* Element ist der Dokumentpfad zu einem Attribut. Das Attribut muss entweder vom Datentyp `Number` oder "Satz" sein.
- Das *value* Element ist eine Zahl, die Sie dem Attribut hinzufügen möchten (für `Number` Datentypen), oder ein Satz, der an das Attribut angehängt werden soll (für Satztypen).

```
add-action ::=
    path value
```

Die folgenden Themen behandeln verschiedene Anwendungsfälle für die ADD-Aktion.

Themen

- Hinzufügen einer Zahl
- Hinzufügen von Elementen zu einem Satz

Hinzufügen einer Zahl

Angenommen, das Attribut `QuantityOnHand` ist nicht vorhanden. Im folgenden AWS CLI Beispiel wird der Wert `QuantityOnHand` auf 5 gesetzt.

```
aws dynamodb update-item \
    --table-name ProductCatalog \
    --key '{"Id":{"N":"789"}}' \
    --update-expression "ADD QuantityOnHand :q" \
    --expression-attribute-values '{":q": {"N": "5"}}' \
    --return-values ALL_NEW
```

Da `QuantityOnHand` nun vorhanden ist, können Sie das Beispiel erneut ausführen, um `QuantityOnHand` jedes Mal um 5 zu erhöhen.

Hinzufügen von Elementen zu einem Satz

Angenommen, das Attribut `Color` ist nicht vorhanden. Im folgenden AWS CLI -Beispiel wird `Color` auf einen Zeichenfolgensatz mit zwei Elementen festgelegt.

```
aws dynamodb update-item \
    --table-name ProductCatalog \
    --key '{"Id":{"N":"789"}}' \
    --update-expression "ADD Color :c" \
    --expression-attribute-values '{":c": {"SS":["Orange", "Purple"]}}' \
    --return-values ALL_NEW
```

Da `Color` nun vorhanden ist, können wir weitere Elemente hinzufügen.

```
aws dynamodb update-item \
    --table-name ProductCatalog \
    --key '{"Id":{"N":"789"}}' \
    --update-expression "ADD Color :c" \
    --expression-attribute-values '{":c": {"SS":["Yellow", "Green", "Blue"]}}' \
    --return-values ALL_NEW
```

DELETE – Entfernen von Elementen aus einem Satz

> **Important**
>
> Die `DELETE`-Aktion unterstützt nur den Datentyp `Set`.

Verwenden Sie die `DELETE`-Aktion in einem Aktualisierungsausdruck, um ein oder mehrere Elemente aus einem Satz zu entfernen. Wenn Sie mehrere `DELETE`-Aktionen durchführen möchten, trennen Sie sie durch Komma.

Schauen Sie sich die folgende Syntaxzusammenfassung an:

- Das *`path`* Element ist der Dokumentpfad zu einem Attribut. Das Attribut muss vom Datentyp "Satz" sein.
- Das *`subset`* ist ein oder mehrere Elemente, aus denen Sie löschen möchten*`path`*. Sie müssen einen Satztyp angeben*`subset`*.

```
delete-action ::=
    path subset
```

Example

Unter Hinzufügen von Elementen zu einem Satz erstellen Sie den `Color`-Zeichenfolgensatz. In diesem Beispiel werden einige der Elemente aus diesem Satz entfernt:

```
aws dynamodb update-item \
    --table-name ProductCatalog \
    --key '{"Id":{"N":"789"}}' \
    --update-expression "DELETE Color :p" \
    --expression-attribute-values '{":p": {"SS": ["Yellow", "Purple"]}}' \
    --return-values ALL_NEW
```

Verwenden mehrerer Aktualisierungsausdrücke

Sie können mehrere Aktionen in einem einzigen Aktualisierungsausdruck verwenden. Alle Attributverweise werden anhand des Status des Elements aufgelöst, bevor eine der Aktionen angewendet wird.

Example

Bei einem bestimmten Element `{"id": "1", "a": 1, "b": 2, "c": 3}` entfernt a und verschiebt der folgende Ausdruck die Werte von b und`c`:

```
aws dynamodb update-item \
    --table-name test \
    --key '{"id":{"S":"1"}}' \
    --update-expression "REMOVE a SET b = a, c = b" \
    --return-values ALL_NEW
```

Das Ergebnis ist`{"id": "1", "b": 1, "c": 2}`. Obwohl a es entfernt und im selben Ausdruck neu zugewiesen b wird, werden beide Verweise auf ihre ursprünglichen Werte aufgelöst.

Example

Wenn Sie den Wert eines Attributs ändern und ein anderes Attribut vollständig entfernen möchten, können Sie eine SET-Aktion und eine REMOVE-Aktion in einer einzelnen Anweisung verwenden. Dieser Vorgang würde den `Price`-Wert auf 15 verringern und gleichzeitig das `InStock`-Attribut aus dem Element entfernen.

```
aws dynamodb update-item \
    --table-name ProductCatalog \
    --key '{"Id":{"N":"789"}}' \
    --update-expression "SET Price = Price - :p REMOVE InStock" \
    --expression-attribute-values '{":p": {"N":"15"}}' \
    --return-values ALL_NEW
```

Example

Wenn Sie einer Liste etwas hinzufügen und gleichzeitig den Wert eines anderen Attributs ändern möchten, können Sie zwei SET-Aktionen in einer einzelnen Anweisung verwenden. Diese Operation würde dem `RelatedItems`-Listenattribut „Nails“ hinzufügen und den `Price`-Wert auf 21 setzen.

```
aws dynamodb update-item \
    --table-name ProductCatalog \
    --key '{"Id":{"N":"789"}}' \
    --update-expression "SET RelatedItems[1] = :newValue, Price = :newPrice" \
    --expression-attribute-values '{":newValue": {"S":"Nails"}, ":newPrice":
 {"N":"21"}}'  \
    --return-values ALL_NEW
```

Bedingungs- und Filterausdrücke, Operatoren und Funktionen in DynamoDB

Zum Bearbeiten der Daten in einer DynamoDB-Tabelle verwenden Sie die Operationen `PutItem`, `UpdateItem` und `DeleteItem`. Für diese Datenmanipulationsoperationen können Sie einen Bedingungsausdruck angeben, um zu ermitteln, welche Elemente geändert werden sollten. Wenn der Bedingungsausdruck „true“ ergibt, ist die Operation erfolgreich. Andernfalls schlägt sie fehl.

In diesem Abschnitt werden die integrierten Funktionen und Schlüsselwörter zum Schreiben von Filter- und Bedingungsausdrücken in Amazon DynamoDB erörtert. Ausführlichere Informationen zu Funktionen und Programmierung mit DynamoDB finden Sie unter Programmieren mit DynamoDB und AWS SDKs und in der DynamoDB-API-Referenz.

Themen

- Syntax für Filter- und Bedingungsausdrücke
- Durchführen von Vergleichen
- Funktionen
- Logische Auswertungen
- Klammern

- Priorität in Bedingungen

Syntax für Filter- und Bedingungsausdrücke

In der folgenden Syntaxzusammenfassung *operand* kann ein wie folgt lauten:

- Ein Attributname auf oberster Ebene, z. B. `Id`, `Title`, `Description` oder `ProductCategory`
- Ein Dokumentpfad, der auf ein verschachteltes Attribut verweist

```
condition-expression ::=
      operand comparator operand
    | operand BETWEEN operand AND operand
    | operand IN ( operand (',' operand (, ...) ))
    | function
    | condition AND condition
    | condition OR condition
    | NOT condition
    | ( condition )

comparator ::=
    =
    | <>
    | <
    | <=
    | >
    | >=

function ::=
    attribute_exists (path)
    | attribute_not_exists (path)
    | attribute_type (path, type)
    | begins_with (path, substr)
    | contains (path, operand)
    | size (path)
```

Durchführen von Vergleichen

Verwenden Sie diese Komparatoren, um einen Operanden mit einem einzelnen Wert zu vergleichen:

- *a* = *b* — Wahr, wenn gleich *a* ist *b*.
- *a* <> *b* — Wahr, wenn ungleich *a* ist *b*.

- *a* < *b*— Stimmt, wenn *a* es kleiner ist als*b*.
- *a* <= *b*— Wahr, wenn kleiner oder gleich *a* ist*b*.
- *a* > *b*— Stimmt, wenn größer als *a* ist*b*.
- *a* >= *b*— Wahr, wenn größer als oder gleich *a* ist*b*.

Verwenden Sie die Schlüsselwörter `BETWEEN` und `IN`, um einen Operanden mit einer Reihe von Werten oder einer Aufzählung von Werten zu vergleichen:

- *a* `BETWEEN` *b* `AND` *c*— Wahr*b*, wenn größer als oder gleich und kleiner als oder gleich *a* ist*c*.
- *a* `IN (`*b*`, `*c*`, `*d*`)` — Wahr, wenn *a* der Wert einem beliebigen Wert in der Liste entspricht, z. B. einem beliebigen Wert von *bc*, oder. *d* Die Liste kann bis zu 100 Werte enthalten, die durch Kommas getrennt sind.

Funktionen

Verwenden Sie die folgenden Funktionen, um zu bestimmen, ob ein Attribut in einem Element vorhanden ist, oder um den Wert eines Attributs zu bewerten. Bei Funktionsnamen wird zwischen Groß- und Kleinschreibung unterschieden. Bei einem verschachtelten Attribut müssen Sie den vollständigen Dokumentpfad angeben.

Funktion	Description
`attribute_exists (`*path*`)`	True, wenn das Element das von `path` angegebene Attribut enthält. Beispiel: Prüfung, ob ein Element in der Tabelle `Product` über ein Seitenansichtsbild verfügt. • `attribute_exists (#Pictures.#SideView)`
`attribute_not_exists (`*path*`)`	True, wenn das Element das von `path` angegebene Attribut nicht enthält. Beispiel: Prüfung, ob ein Element über ein `Manufacturer` -Attribut verfügt

Funktion	Description
	• `attribute_not_exists (Manufacturer)`

Funktion	Description
`attribute_type ( path, type)`	True, wenn das Attribut am angegebenen Pfad einen bestimmten Datentyp hat. Der Parameter `type` muss einer der folgenden Werte aufweisen: • `S` – Zeichenfolge • `SS` – Zeichenfolgensatz • `N` – Zahl • `NS` – Zahlensatz • `B` – Binary • `BS` – Binärzahlensatz • `BOOL` – Boolean • `NULL` – Nullwert • `L` – Liste • `M` – Zuordnung Sie müssen einen Ausdrucksattributwert für den Parameter `type` verwenden. Beispiel: Prüfung, ob das Attribut `QuantityOnHand` vom Typ Liste ist. In diesem Beispiel ist `:v_sub` ein Platzhalter für die Zeichenfolge `L`. • `attribute_type (ProductReviews.FiveStar, :v_sub)`

Funktion	Description
	Sie müssen einen Ausdrucksattributwert für den Parameter `type` verwenden.
`begins_with ( path, substr)`	True, wenn das von `path` angegebene Attribut mit einer bestimmten Teilzeichenfolge beginnt. Beispiel: Prüfung, ob die ersten Zeichen der URL zum Vorderansichtsbild `http://` lauten. • `begins_with (Pictures.FrontView, :v_sub)` Der Ausdrucksattributwert `:v_sub` ist ein Platzhalter für `http://`.

Funktion	Description
`contains (path, operand)`	True, wenn das von `path` angegebene Attribut Folgendes ist: • ein `String`, der eine bestimmte Teilzeich enfolge enthält • ein `Set`, der ein bestimmtes Element innerhalb des Satzes enthält • ein `List`, der ein bestimmtes Element innerhalb der Liste enthält Wenn das von `path` angegebene Attribut eine `String` ist, muss der `operand` `String` sein. Wenn das von `path` angegebene Attribut `Set` ist muss `operand` der Elementtyp des Satzes sein. Der Pfad und der Operand müssen unterschi edlich sein. Das heißt, bei `contains (a, a)` wird ein Fehler zurückgegeben. Beispiel: Prüfung, ob das Attribut `Brand` die Teilzeichenfolge `Company` enthält. • `contains (Brand, :v_sub)` Der Ausdrucksattributwert `:v_sub` ist ein Platzhalter für `Company`. Beispiel: Prüfung, ob das Produkt in rot verfügbar ist. • `contains (Color, :v_sub)`

Funktion	Description
	Der Ausdrucksattributwert `:v_sub` ist ein Platzhalter für `Red`.

Funktion	Description
`size (path)`	Gibt eine Zahl zurück, die für die Größe eines Attributs steht. Die folgenden sind gültige Datentypen, die mit `size` verwendet werden können. Wenn das Attribut vom Typ `String` ist, gibt `size` die Länge der Zeichenfolge zurück. Beispiel: Prüfung, ob die Zeichenfolge `Brand` kleiner oder gleich 20 Zeichen ist. Der Ausdrucksattributwert `:v_sub` ist ein Platzhalter für `20`. • `size (Brand) <= :v_sub` Wenn das Attribut vom Typ `Binary` ist, gibt `size` die Anzahl der Bytes im Attributwert zurück. Beispiel: Angenommen, das Element `ProductCatalog` verfügt über ein binäres Attribut mit der Bezeichnung `VideoClip`, das ein kurzes Video über das betreffende Produkt enthält. Der folgende Ausdruck überprüft, ob `VideoClip` 64.000 Byte überschreitet. Der Ausdrucksattributwert `:v_sub` ist ein Platzhalter für `64000`. • `size(VideoClip) > :v_sub` Wenn das Attribut vom Typ `Set` ist, gibt `size` die Anzahl der Elemente im Satz zurück.

Funktion	Description
	Beispiel: Prüfung, ob das Produkt in mehr als einer Farbe verfügbar ist. Der Ausdrucks attributwert `:v_sub` ist ein Platzhalter für `1`. • `size (Color) < :v_sub` Wenn das Attribut vom Typ `List` oder `Map` ist, gibt `size` die Anzahl der untergeordneten Elemente zurück. Beispiel: Prüfung, ob die Anzahl der `OneStar`-Rezensionen einen bestimmten Grenzwert überschritten hat. Der Ausdrucksattributwert `:v_sub` ist ein Platzhalter für `3`. • `size(ProductReviews.OneStar) > :v_sub`

Logische Auswertungen

Verwenden Sie die Schlüsselwörter `AND`, `OR` und `NOT`, um logische Auswertungen durchzuführen. In der folgenden Liste *a* und *b* stellen Bedingungen dar, die ausgewertet werden sollen.

- *a* `AND` *b*— Stimmt, wenn *a* und beide wahr *b* sind.
- *a* `OR` *b*— Stimmt, wenn entweder *a* oder *b* (oder beide) wahr sind.
- `NOT` *a*— Wahr, wenn falsch *a* ist. Falsch, wenn *a* es wahr ist.

Im Folgenden finden Sie ein Codebeispiel für AND in einer Operation.

```
dynamodb-local (*)> select * from exprtest where a > 3 and a < 5;
```

Klammern

Verwenden Sie Klammern, um die Priorität einer logischen Auswertung zu ändern. Nehmen wir zum Beispiel an, dass die Bedingungen *a* und wahr *b* sind und diese Bedingung falsch *c* ist. Der folgenden Ausdruck ergibt True:

- `a OR b AND c`

Wenn Sie jedoch eine Bedingung in Klammern setzen, wird diese zuerst ausgewertet. Folgendes ergibt z. B. False:

- `(a OR b) AND c`

> **Note**
>
> Sie können Klammern in einem Ausdruck schachteln. Die innersten Klammern werden zuerst ausgewertet.

Im Folgenden finden Sie ein Codebeispiel mit Klammern in einer logischen Auswertung.

```
dynamodb-local (*)> select * from exprtest where attribute_type(b, string)
or ( a = 5 and c = “coffee”);
```

Priorität in Bedingungen

DynamoDB wertet Bedingungen von links nach rechts entsprechend der folgenden Prioritätsregeln aus:

- `= <> < <= > >=`
- `IN`
- `BETWEEN`
- `attribute_exists attribute_not_exists begins_with contains`
- Klammern
- `NOT`
- `AND`

- `OR`

CLI-Beispiel für DynamoDB-Bedingungsausdrücke

Im Folgenden finden Sie einige AWS Command Line Interface (AWS CLI) Beispiele für die Verwendung von Bedingungsausdrücken. Diese Beispiele basieren auf der Tabelle `ProductCatalog`, die in Verweisen auf Elementattribute beim Verwenden von Ausdrücken in DynamoDB eingeführt wurde. Der Partitionsschlüssel für diese Tabelle lautet `Id`. Es gibt keinen Sortierschlüssel. Die folgende `PutItem`-Operation erstellt ein `ProductCatalog`-Beispielelement, auf das wir uns in den Beispielen beziehen werden:

```
aws dynamodb put-item \
    --table-name ProductCatalog \
    --item file://item.json
```

Die Argumente für `--item` werden in der Datei `item.json` gespeichert. (Der Einfachheit halber werden nur wenige Elementattribute verwendet.)

```
{
    "Id": {"N": "456" },
    "ProductCategory": {"S": "Sporting Goods" },
    "Price": {"N": "650" }
}
```

Themen

- Conditional Put (Bedingtes Setzen)
- Bedingte Löschungen
- Bedingte Aktualisierungen
- Beispiele für bedingte Ausdrücke

Conditional Put (Bedingtes Setzen)

Die `PutItem`-Operation überschreibt ein Element mit demselben Primärschlüssel (falls vorhanden). Wenn Sie dies vermeiden möchten, verwenden Sie einen Bedingungsausdruck. Dies ermöglicht das Fortsetzen des Schreibvorgangs nur dann, wenn das in Frage stehende Element nicht bereits über denselben Primärschlüssel verfügt.

Im folgenden Beispiel wird mit `attribute_not_exists()` überprüft, ob der Primärschlüssel in der Tabelle vorhanden ist, bevor versucht wird, den Schreibvorgang durchzuführen.

> **Note**
>
> Wenn Ihr Primärschlüssel sowohl aus einem Partitionsschlüssel (pk) als auch aus einem Sortierschlüssel (sk) besteht, überprüft der Parameter, ob `attribute_not_exists(pk)` UND `attribute_not_exists(sk)` als gesamte Anweisung als wahr oder falsch ausgewertet werden, bevor versucht wird, den Schreibvorgang durchzuführen.

```
aws dynamodb put-item \
    --table-name ProductCatalog \
    --item file://item.json \
    --condition-expression "attribute_not_exists(Id)"
```

Wenn der Bedingungsausdruck mit „falsch“ ausgewertet wird, gibt DynamoDB die folgende Fehlermeldung zurück: Die bedingte Anforderung ist fehlgeschlagen.

> **Note**
>
> Weitere Informationen zu `attribute_not_exists` und anderen Funktionen finden Sie unter Bedingungs- und Filterausdrücke, Operatoren und Funktionen in DynamoDB.

Bedingte Löschungen

Um ein bedingtes Löschen durchzuführen, nutzen Sie eine `DeleteItem`-Operation mit einem Bedingungsausdruck. Der Bedingungsausdruck muss mit "true" ausgewertet werden, damit die Operation erfolgreich ist; andernfalls schlägt sie fehl.

Betrachten Sie das oben definierte Element.

Angenommen, Sie möchten das Element löschen, aber nur unter den folgenden Bedingungen:

- Die `ProductCategory` ist entweder "Sportartikel" oder "Gartenzubehör".
- Der `Price` liegt zwischen 500 und 600.

Im folgenden Beispiel wird versucht, das Element zu löschen:

```
aws dynamodb delete-item \
    --table-name ProductCatalog \
    --key '{"Id":{"N":"456"}}' \
    --condition-expression "(ProductCategory IN (:cat1, :cat2)) and (Price between :lo
 and :hi)" \
    --expression-attribute-values file://values.json
```

Die Argumente für `--expression-attribute-values` werden in der Datei `values.json` gespeichert.

```
{
    ":cat1": {"S": "Sporting Goods"},
    ":cat2": {"S": "Gardening Supplies"},
    ":lo": {"N": "500"},
    ":hi": {"N": "600"}
}
```

> **Note**
>
> In dem Bedingungsausdruck weist der `:` (Doppelpunkt) auf einen Ausdrucksattributwert hin – Platzhalter für den tatsächlichen Wert. Weitere Informationen finden Sie unter Verwenden von Ausdrucksattributwerten in DynamoDB.
> Weitere Informationen zu `IN`, `AND` und anderen Schlüsselwörtern finden Sie unter Bedingungs- und Filterausdrücke, Operatoren und Funktionen in DynamoDB.

In diesem Beispiel wird der `ProductCategory`-Vergleich mit True, aber der `Price`-Vergleich mit False ausgewertet. Dadurch wird der Bedingungsausdruck mit "false" ausgewertet und die `DeleteItem`-Operation schlägt fehl.

Bedingte Aktualisierungen

Um eine bedingte Aktualisierung durchzuführen, nutzen Sie eine `UpdateItem`-Operation mit einem Bedingungsausdruck. Der Bedingungsausdruck muss mit "true" ausgewertet werden, damit die Operation erfolgreich ist; andernfalls schlägt sie fehl.

> **Note**
>
> `UpdateItem` unterstützt auch Aktualisierungsausdrücke, in denen Sie die gewünschten Änderungen festlegen, die Sie an einem Element durchführen möchten. Weitere Informationen finden Sie unter Verwenden von Aktualisierungsausdrücken in DynamoDB.

Angenommen, Sie beginnen mit dem oben definierten Element.

Das folgende Beispiel führt eine `UpdateItem`-Operation durch. Es versucht, den `Price` eines Produkts um 75 zu reduzieren – der Bedingungsausdruck verhindert jedoch die Aktualisierung, wenn der aktuelle `Price` unter 500 liegt.

```
aws dynamodb update-item \
    --table-name ProductCatalog \
    --key '{"Id": {"N": "456"}}' \
    --update-expression "SET Price = Price - :discount" \
    --condition-expression "Price > :limit" \
    --expression-attribute-values file://values.json
```

Die Argumente für `--expression-attribute-values` werden in der Datei `values.json` gespeichert.

```
{
    ":discount": { "N": "75"},
    ":limit": {"N": "500"}
}
```

Wenn der Start-`Price` 650 ist, reduziert die `UpdateItem`-Operation den `Price` auf 575. Wenn Sie die `UpdateItem`-Operation erneut ausführen, wird der `Price` auf 500 reduziert. Wenn Sie ein drittes Mal ausführen, wird der Bedingungsausdruck mit False ausgewertet und die Aktualisierung schlägt fehl.

> **Note**
>
> In dem Bedingungsausdruck weist der `:` (Doppelpunkt) auf einen Ausdrucksattributwert hin – Platzhalter für den tatsächlichen Wert. Weitere Informationen finden Sie unter Verwenden von Ausdrucksattributwerten in DynamoDB.

Weitere Informationen zu ">" und anderen Operatoren finden Sie unter Bedingungs- und Filterausdrücke, Operatoren und Funktionen in DynamoDB.

Beispiele für bedingte Ausdrücke

Weitere Hinweise zu den Funktionen, die in den folgenden Beispielen verwendet werden, finden Sie unter Bedingungs- und Filterausdrücke, Operatoren und Funktionen in DynamoDB. Weitere Informationen zum Angeben verschiedener Attributtypen in einem Ausdruck finden Sie unter Verweisen auf Elementattribute beim Verwenden von Ausdrücken in DynamoDB.

Überprüfen der Attribute in einem Element

Sie können das Vorhandensein (oder Fehlen) jedes Attributs überprüfen. Wenn der Bedingungsausdruck mit "true" ausgewertet wird, ist die Operation erfolgreich; andernfalls schlägt sie fehl.

Das folgenden Beispiel verwendet `attribute_not_exists` zum Löschen eines Produkts nur, wenn es noch nicht über ein `Price`-Attribut verfügt:

```
aws dynamodb delete-item \
    --table-name ProductCatalog \
    --key '{"Id": {"N": "456"}}' \
    --condition-expression "attribute_not_exists(Price)"
```

DynamoDB stellt auch eine `attribute_exists`-Funktion bereit. Das folgende Beispiel löscht ein Produkt nur dann, wenn es schlechte Bewertungen erhalten hat.

```
aws dynamodb delete-item \
    --table-name ProductCatalog \
    --key '{"Id": {"N": "456"}}' \
    --condition-expression "attribute_exists(ProductReviews.OneStar)"
```

Prüfung auf Attributtyp

Mit der `attribute_type`-Funktion können Sie den Datentyp eines Attributwerts überprüfen. Wenn der Bedingungsausdruck mit "true" ausgewertet wird, ist die Operation erfolgreich; andernfalls schlägt sie fehl.

Im folgenden Beispiel wird `attribute_type` nur dann zum Löschen eines Produkts verwendet, wenn es über ein `Color`-Attribut des Typs String Set verfügt.

```
aws dynamodb delete-item \
    --table-name ProductCatalog \
    --key '{"Id": {"N": "456"}}' \
    --condition-expression "attribute_type(Color, :v_sub)" \
    --expression-attribute-values file://expression-attribute-values.json
```

Die Argumente für `--expression-attribute-values` sind in der expression-attribute-values JSON-Datei gespeichert.

```
{
    ":v_sub":{"S":"SS"}
}
```

Prüfen des Startwerts der Zeichenfolge

Sie können überprüfen, ob ein Zeichenfolge-Attributwert mit einer bestimmten Teilzeichenfolge beginnt, indem Sie die `begins_with`-Funktion verwenden. Wenn der Bedingungsausdruck mit "true" ausgewertet wird, ist die Operation erfolgreich; andernfalls schlägt sie fehl.

Im folgenden Beispiel wird `begins_with` nur dann zum Löschen eines Produkts verwendet, wenn das `FrontView`-Element der `Pictures`-Karte mit einem bestimmten Wert beginnt.

```
aws dynamodb delete-item \
    --table-name ProductCatalog \
    --key '{"Id": {"N": "456"}}' \
    --condition-expression "begins_with(Pictures.FrontView, :v_sub)" \
    --expression-attribute-values file://expression-attribute-values.json
```

Die Argumente für `--expression-attribute-values` werden in der expression-attribute-values JSON-Datei gespeichert.

```
{
    ":v_sub":{"S":"http://"}
}
```

Prüfen auf ein Element in einem Satz

Sie können nach einem Element in einem Satz oder nach einer Teilzeichenfolge innerhalb einer Zeichenfolge suchen, indem Sie die `contains`-Funktion verwenden. Wenn der Bedingungsausdruck mit "true" ausgewertet wird, ist die Operation erfolgreich; andernfalls schlägt sie fehl.

Im folgenden Beispiel wird `contains` nur dann zum Löschen eines Produkts verwendet, wenn der `Color`-Zeichenfolgensatz ein Element mit einem bestimmten Wert aufweist.

```
aws dynamodb delete-item \
    --table-name ProductCatalog \
    --key '{"Id": {"N": "456"}}' \
    --condition-expression "contains(Color, :v_sub)" \
    --expression-attribute-values file://expression-attribute-values.json
```

Die Argumente für `--expression-attribute-values` werden in der expression-attribute-values JSON-Datei gespeichert.

```
{
    ":v_sub":{"S":"Red"}
}
```

Überprüfen der Größe eines Attributwerts

Sie können die Größe eines Attributwerts überprüfen, indem Sie die `size`-Funktion verwenden. Wenn der Bedingungsausdruck mit "true" ausgewertet wird, ist die Operation erfolgreich; andernfalls schlägt sie fehl.

Im folgenden Beispiel wird `size` nur dann zum Löschen eines Produkts verwendet, wenn die Größe des `VideoClip`-Binärattributs größer als `64000` Byte ist.

```
aws dynamodb delete-item \
    --table-name ProductCatalog \
    --key '{"Id": {"N": "456"}}' \
    --condition-expression "size(VideoClip) > :v_sub" \
    --expression-attribute-values file://expression-attribute-values.json
```

Die Argumente für `--expression-attribute-values` werden in der expression-attribute-values JSON-Datei gespeichert.

```
{
    ":v_sub":{"N":"64000"}
}
```

Verwenden von Time to Live (TTL) in DynamoDB

Time To Live (TTL) für DynamoDB ist eine kosteneffiziente Methode zum Löschen von Elementen, die nicht mehr relevant sind. TTL ermöglicht es Ihnen, einen Ablaufzeitstempel pro Element zu definieren, um festzulegen, wann ein Element nicht mehr benötigt wird. DynamoDB löscht abgelaufene Elemente automatisch innerhalb weniger Tage nach ihrem Ablauftermin, ohne dass dafür Schreibdurchsatz verbraucht wird.

Um TTL zu verwenden, aktivieren Sie es zunächst in einer Tabelle. Definieren Sie dann ein bestimmtes Attribut zum Speichern des TTL-Ablaufzeitstempels. Der Zeitstempel muss als Datentyp Zahl im Unix-Epochenzeitformat mit der Granularität Sekunden gespeichert werden. Elemente mit einem TTL-Attribut, das kein Zahlentyp ist, werden vom TTL-Prozess ignoriert. Immer, wenn ein Element erstellt oder aktualisiert wird, können Sie die Ablaufzeit berechnen und sie im TTL-Attribut speichern.

Elemente mit gültigen, abgelaufenen TTL-Attributen können vom System jederzeit gelöscht werden. In der Regel geschieht das innerhalb weniger Tage nach ihrem Ablauf. Sie können die abgelaufenen Elemente, deren Löschung noch aussteht, weiterhin aktualisieren. Dazu gehören auch das Ändern oder Entfernen ihrer TTL-Attribute. Wenn Sie ein abgelaufenes Element aktualisieren, empfiehlt es sich, einen Bedingungsausdruck zu verwenden, um sicherzustellen, dass das Element anschließend nicht gelöscht wurde. Verwenden Sie Filterausdrücke, um abgelaufene Elemente aus Scan- und Abfrageergebnissen zu entfernen.

Gelöschte Elemente funktionieren ähnlich wie Objekte, die bei typischen Löschvorgängen gelöscht wurden. Nach dem Löschen fließen Elemente als Servicelöschungen statt als Benutzerlöschungen in DynamoDB Streams ein und werden wie andere Löschvorgänge aus lokalen und globalen sekundären Indizes entfernt.

Wenn Sie Version 2019.11.21 der globalen Tabellen (aktuell) und außerdem die TTL-Funktion verwenden, repliziert DynamoDB TTL-Löschungen in alle Replikattabellen. Die anfängliche TTL-Löschung verbraucht keine Schreibkapazitätseinheiten (WCU) in der Region, in der die TTL abläuft. Die in die Replikattabellen replizierte TTL-Löschung verbraucht jedoch in jeder Region mit einem Replikat bei Verwendung bereitgestellter Kapazität eine Einheit für replizierte Schreibkapazität bzw. bei Verwendung von On-Demand-Kapazität eine replizierte Schreibeinheit. Dafür werden die entsprechenden Gebühren berechnet.

Weitere Informationen zu TTL finden Sie in den folgenden Themen:

Themen

- Aktivieren von Time to Live (TTL) in DynamoDB
- Berechnen der Time to Live (TTL) in DynamoDB
- Arbeiten mit abgelaufenen Elementen und Time to Live (TTL)

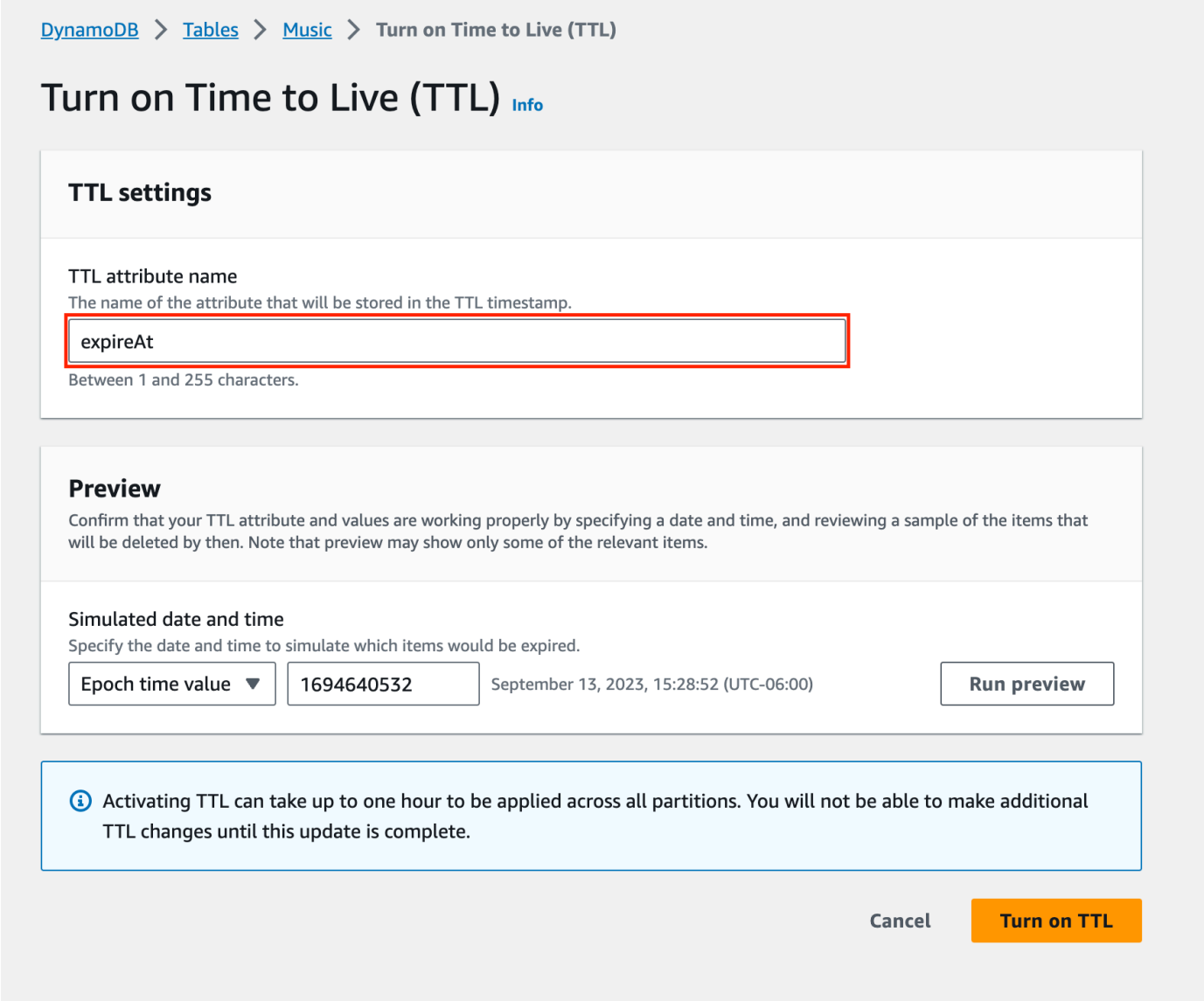
Aktivieren von Time to Live (TTL) in DynamoDB

> **Note**
>
> Um das Debugging und die Verifizierung des ordnungsgemäßen Betriebs der TTL-Funktion zu erleichtern, werden die Werte für die Element-TTL in DynamoDB-Diagnoseprotokollen in Klartext angegeben.

Sie können TTL in der Amazon DynamoDB DynamoDB-Konsole, der AWS Command Line Interface (AWS CLI), oder mithilfe der Amazon DynamoDB DynamoDB-API-Referenz mit einer der angegebenen Optionen aktivieren. AWS SDKs Es dauert ungefähr eine Stunde, um TTL auf allen Partitionen zu aktivieren.

DynamoDB TTL über die Konsole aktivieren AWS

1. Melden Sie sich bei der an AWS-Managementkonsole und öffnen Sie die DynamoDB-Konsole unter. https://console.aws.amazon.com/dynamodb/
2. Wählen Sie Tabellen und anschließend die Tabelle aus, die Sie ändern möchten.
3. Wählen Sie auf der Registerkarte Zusätzliche Einstellungen im Abschnitt Time to Live (TTL) die Option Einschalten aus, um TTL zu aktivieren.
4. Wenn Sie TTL für eine Tabelle aktivieren, müssen Sie bei DynamoDB einen bestimmten Attributnamen angeben, nach dem der Service sucht, wenn bestimmt wird, ob der Ablauf eines Elements veranlasst werden soll. Beim Namen des TTL-Attributs (siehe unten) wird zwischen Groß- und Kleinschreibung unterschieden. Außerdem muss er mit dem in Ihren Lese- und Schreibvorgängen definierten Attribut übereinstimmen. Andernfalls werden abgelaufene Elemente nicht gelöscht. Um das TTL-Attribut umzubenennen, müssen Sie TTL zunächst deaktivieren und dann mit dem neuen Attribut aktivieren. TTL verarbeitet Löschungen etwa 30 Minuten lang weiter, nachdem es deaktiviert wurde. Bei wiederhergestellten Tabellen muss TTL neu konfiguriert werden.



5. (Optional) Sie können einen Test durchführen, indem Sie Datum und Uhrzeit des Ablaufs simulieren und einige Elemente abgleichen. Dadurch erhalten Sie eine Beispielliste mit Elementen, die bestätigt, dass es Elemente mit dem angegebenen TTL-Attributnamen und der Ablaufzeit gibt.

Nachdem TTL aktiviert wurde, wird das TTL-Attribut mit TTL gekennzeichnet, wenn Sie Elemente in der DynamoDB-Konsole ansehen. Sie können den Termin anzeigen, an dem ein Element abläuft, indem Sie mit dem Mauszeiger über das Attribut fahren.

Aktivieren von DynamoDB TTL mithilfe der API

Python

Sie können TTL mithilfe des Vorgangs mit Code aktivieren. UpdateTimeToLive

```
import boto3


def enable_ttl(table_name, ttl_attribute_name):
    """
    Enables TTL on DynamoDB table for a given attribute name
        on success, returns a status code of 200
        on error, throws an exception

    :param table_name: Name of the DynamoDB table
    :param ttl_attribute_name: The name of the TTL attribute being provided to the
 table.
    """
    try:
        dynamodb = boto3.client('dynamodb')

        # Enable TTL on an existing DynamoDB table
        response = dynamodb.update_time_to_live(
            TableName=table_name,
            TimeToLiveSpecification={
                'Enabled': True,
                'AttributeName': ttl_attribute_name
            }
        )

        # In the returned response, check for a successful status code.
        if response['ResponseMetadata']['HTTPStatusCode'] == 200:
            print("TTL has been enabled successfully.")
        else:
            print(f"Failed to enable TTL, status code {response['ResponseMetadata']
['HTTPStatusCode']}")
    except Exception as ex:
        print("Couldn't enable TTL in table %s. Here's why: %s" % (table_name, ex))
        raise


# your values
```

```
enable_ttl('your-table-name', 'expirationDate')
```

Sie können überprüfen, ob TTL aktiviert ist, indem Sie den DescribeTimeToLiveVorgang verwenden, der den TTL-Status in einer Tabelle beschreibt. Der `TimeToLive`-Status ist entweder `ENABLED` oder `DISABLED`.

```
# create a DynamoDB client
dynamodb = boto3.client('dynamodb')

# set the table name
table_name = 'YourTable'

# describe TTL
response = dynamodb.describe_time_to_live(TableName=table_name)
```

JavaScript

Sie können TTL mithilfe des Vorgangs mit Code aktivieren. UpdateTimeToLiveCommand

```
import { DynamoDBClient, UpdateTimeToLiveCommand } from "@aws-sdk/client-dynamodb";

const enableTTL = async (tableName, ttlAttribute) => {

    const client = new DynamoDBClient({});

    const params = {
        TableName: tableName,
        TimeToLiveSpecification: {
            Enabled: true,
            AttributeName: ttlAttribute
        }
    };

    try {
        const response = await client.send(new UpdateTimeToLiveCommand(params));
        if (response.$metadata.httpStatusCode === 200) {
            console.log(`TTL enabled successfully for table ${tableName}, using
 attribute name ${ttlAttribute}.`);
        } else {
            console.log(`Failed to enable TTL for table ${tableName}, response
 object: ${response}`);
        }
```

```
        return response;
    } catch (e) {
        console.error(`Error enabling TTL: ${e}`);
        throw e;
    }
};

// call with your own values
enableTTL('ExampleTable', 'exampleTtlAttribute');
```

Aktivieren Sie Time to Live mit dem AWS CLI

1. Aktivieren Sie TTL in der `TTLExample`-Tabelle.

   ```
   aws dynamodb update-time-to-live --table-name TTLExample --time-to-live-
   specification "Enabled=true, AttributeName=ttl"
   ```

2. Beschreiben Sie TTL in der `TTLExample`-Tabelle.

   ```
   aws dynamodb describe-time-to-live --table-name TTLExample
   {
       "TimeToLiveDescription": {
           "AttributeName": "ttl",
           "TimeToLiveStatus": "ENABLED"
       }
   }
   ```

3. Fügen Sie der `TTLExample`-Tabelle mit dem Time-to-Live-Attributsatz ein Element mit der BASH-Shell und AWS CLI hinzu:

   ```
   EXP=`date -d '+5 days' +%s`
   aws dynamodb put-item --table-name "TTLExample" --item '{"id": {"N": "1"}, "ttl":
    {"N": "'$EXP'"}}'
   ```

In diesem Beispiel wurde mit dem aktuellen Datum begonnen und es wurden 5 Tage hinzugefügt, um ein Ablaufdatum zu erstellen. Anschließend wird das Ablaufdatum in ein Epoch-Zeitformat umgewandelt, um der Tabelle "`TTLExample`" ein Element hinzuzufügen.

> **Note**
>
> Eine Möglichkeit, Ablaufwerte für Time to Live festzulegen, besteht darin, die Anzahl von Sekunden zu berechnen, die dem Ablaufdatum hinzugefügt werden. Beispielsweise sind 5 Tage 432.000 Sekunden. Es ist jedoch häufig besser, mit einem Datum zu beginnen und dieses als Ausgangspunkt zu nehmen.

Es ist relativ einfach, die aktuelle Zeit im Epoch-Zeitformat zu erhalten, wie in folgenden Beispielen veranschaulicht.

- Linux-Terminal: `date +%s`
- Python: `import time; int(time.time())`
- Java: `System.currentTimeMillis() / 1000L`
- JavaScript: `Math.floor(Date.now() / 1000)`

Aktivieren Sie DynamoDB TTL mit CloudFormation

```
AWSTemplateFormatVersion: "2010-09-09"
Resources:
  TTLExampleTable:
    Type: AWS::DynamoDB::Table
    Description: "A DynamoDB table with TTL Specification enabled"
    Properties:
      AttributeDefinitions:
        - AttributeName: "Album"
          AttributeType: "S"
        - AttributeName: "Artist"
          AttributeType: "S"
      KeySchema:
        - AttributeName: "Album"
          KeyType: "HASH"
        - AttributeName: "Artist"
          KeyType: "RANGE"
      ProvisionedThroughput:
        ReadCapacityUnits: "5"
        WriteCapacityUnits: "5"
      TimeToLiveSpecification:
        AttributeName: "TTLExampleAttribute"
```

```
        Enabled: true
```

Weitere Informationen zur Verwendung von TTL in Ihren CloudFormation Vorlagen finden Sie hier.

Berechnen der Time to Live (TTL) in DynamoDB

Eine gängige Methode zur Implementierung von TTL besteht darin, eine Ablaufzeit für Elemente festzulegen, die auf dem Erstellungs- oder letzten Aktualisierungsdatum basiert. Dies kann durch Hinzufügen von Zeit zu den Zeitstempeln `createdAt` und `updatedAt` erreicht werden. Beispielsweise kann die TTL für neu erstellte Elemente auf `createdAt` + 90 Tage festgelegt werden. Wenn das Element aktualisiert wird, kann die TTL mit `updatedAt` + 90 Tage neu berechnet werden.

Die berechnete Ablaufzeit muss im Unix-Format in Sekunden angegeben werden. Damit Elemente für Ablauf und Löschung infrage kommen, darf die TTL nicht länger als fünf Jahre zurückliegen. Wenn Sie ein anderes Format verwenden, ignorieren die TTL-Prozesse das Element. Wenn Sie das gewünschte Ablaufdatum auf einen Zeitpunkt in der Zukunft legen, läuft das Element nach diesem Zeitpunkt ab. Stellen Sie das Ablaufdatum beispielsweise auf 1724241326 ein (das entspricht Montag, 21. August 2024, 11:55:26 (GMT)), läuft das Element nach der angegebenen Zeit ab.

Themen

- Erstellen eines Elements und Festlegen der Time to Live
- Aktualisieren eines Elements und der Time to Live

Erstellen eines Elements und Festlegen der Time to Live

Das folgende Beispiel zeigt, wie die Ablaufzeit berechnet wird, wenn ein neues Element erstellt wird. Dabei wird `expireAt` als TTL-Attributname verwendet. Eine Zuweisungsanweisung ruft die aktuelle Uhrzeit als Variable ab. In diesem Beispiel wird die Ablaufzeit als 90 Tage ab der aktuellen Uhrzeit berechnet. Die Zeit wird dann in das Unix-Format umgewandelt und als Ganzzahl-Datentyp im TTL-Attribut gespeichert.

Die folgenden Codebeispiele veranschaulichen, wie Sie ein Element mit TTL erstellen.

Java

SDK für Java 2.x

```
package com.amazon.samplelib.ttl;
```

```
import com.amazon.samplelib.CodeSampleUtils;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.AttributeValue;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.model.PutItemRequest;
import software.amazon.awssdk.services.dynamodb.model.PutItemResponse;
import software.amazon.awssdk.services.dynamodb.model.ResourceNotFoundException;

import java.util.HashMap;
import java.util.Map;
import java.util.Optional;

/**
 * Creates an item in a DynamoDB table with TTL attributes.
 * This class demonstrates how to add TTL expiration timestamps to DynamoDB
 items.
 */
public class CreateTTL {

    private static final String USAGE =
        """
            Usage:
                <tableName> <primaryKey> <sortKey> <region>
            Where:
                tableName - The Amazon DynamoDB table being queried.
                primaryKey - The name of the primary key. Also known as the hash
 or partition key.
                sortKey - The name of the sort key. Also known as the range
 attribute.
                region (optional) - The AWS region that the Amazon DynamoDB table
 is located in. (Default: us-east-1)
            """;
    private static final int DAYS_TO_EXPIRE = 90;
    private static final int SECONDS_PER_DAY = 24 * 60 * 60;
    private static final String PRIMARY_KEY_ATTR = "primaryKey";
    private static final String SORT_KEY_ATTR = "sortKey";
    private static final String CREATION_DATE_ATTR = "creationDate";
    private static final String EXPIRE_AT_ATTR = "expireAt";
    private static final String SUCCESS_MESSAGE = "%s PutItem operation with TTL
 successful.";
    private static final String TABLE_NOT_FOUND_ERROR = "Error: The Amazon
 DynamoDB table \"%s\" can't be found.";
```

```
    private final DynamoDbClient dynamoDbClient;

    /**
     * Constructs a CreateTTL instance with the specified DynamoDB client.
     *
     * @param dynamoDbClient The DynamoDB client to use
     */
    public CreateTTL(final DynamoDbClient dynamoDbClient) {
        this.dynamoDbClient = dynamoDbClient;
    }

    /**
     * Constructs a CreateTTL with a default DynamoDB client.
     */
    public CreateTTL() {
        this.dynamoDbClient = null;
    }

    /**
     * Main method to demonstrate creating an item with TTL.
     *
     * @param args Command line arguments
     */
    public static void main(final String[] args) {
        try {
            int result = new CreateTTL().processArgs(args);
            System.exit(result);
        } catch (Exception e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
    }

    /**
     * Process command line arguments and create an item with TTL.
     *
     * @param args Command line arguments
     * @return 0 if successful, non-zero otherwise
     * @throws ResourceNotFoundException If the table doesn't exist
     * @throws DynamoDbException If an error occurs during the operation
     * @throws IllegalArgumentException If arguments are invalid
     */
    public int processArgs(final String[] args) {
```

```
        // Argument validation (remove or replace this line when reusing this
code)
        CodeSampleUtils.validateArgs(args, new int[] {3, 4}, USAGE);

        final String tableName = args[0];
        final String primaryKey = args[1];
        final String sortKey = args[2];
        final Region region = Optional.ofNullable(args.length > 3 ? args[3] :
null)
            .map(Region::of)
            .orElse(Region.US_EAST_1);

        try (DynamoDbClient ddb = dynamoDbClient != null
            ? dynamoDbClient
            : DynamoDbClient.builder().region(region).build()) {
            final CreateTTL createTTL = new CreateTTL(ddb);
            createTTL.createItemWithTTL(tableName, primaryKey, sortKey);
            return 0;
        } catch (Exception e) {
            throw e;
        }
    }

    /**
     * Creates an item in the specified table with TTL attributes.
     *
     * @param tableName The name of the table
     * @param primaryKeyValue The value for the primary key
     * @param sortKeyValue The value for the sort key
     * @return The response from the PutItem operation
     * @throws ResourceNotFoundException If the table doesn't exist
     * @throws DynamoDbException If an error occurs during the operation
     */
    public PutItemResponse createItemWithTTL(
        final String tableName, final String primaryKeyValue, final String
sortKeyValue) {
        // Get current time in epoch second format
        final long createDate = System.currentTimeMillis() / 1000;

        // Calculate expiration time 90 days from now in epoch second format
        final long expireDate = createDate + (DAYS_TO_EXPIRE * SECONDS_PER_DAY);

        final Map<String, AttributeValue> itemMap = new HashMap<>();
        itemMap.put(
```

```
            PRIMARY_KEY_ATTR,
 AttributeValue.builder().s(primaryKeyValue).build());
        itemMap.put(SORT_KEY_ATTR,
 AttributeValue.builder().s(sortKeyValue).build());
        itemMap.put(
            CREATION_DATE_ATTR,
            AttributeValue.builder().n(String.valueOf(createDate)).build());
        itemMap.put(
            EXPIRE_AT_ATTR,
            AttributeValue.builder().n(String.valueOf(expireDate)).build());

        final PutItemRequest request =
            PutItemRequest.builder().tableName(tableName).item(itemMap).build();

        try {
            final PutItemResponse response = dynamoDbClient.putItem(request);
            System.out.println(String.format(SUCCESS_MESSAGE, tableName));
            return response;
        } catch (ResourceNotFoundException e) {
            System.err.format(TABLE_NOT_FOUND_ERROR, tableName);
            throw e;
        } catch (DynamoDbException e) {
            System.err.println(e.getMessage());
            throw e;
        }
    }
}
```

- Einzelheiten zur API finden Sie PutItemunter AWS SDK for Java 2.x API-Referenz.

JavaScript

SDK für JavaScript (v3)

```
import { DynamoDBClient, PutItemCommand } from "@aws-sdk/client-dynamodb";

export function createDynamoDBItem(table_name, region, partition_key, sort_key) {
    const client = new DynamoDBClient({
        region: region,
        endpoint: `https://dynamodb.${region}.amazonaws.com`
    });
```

```
    // Get the current time in epoch second format
    const current_time = Math.floor(new Date().getTime() / 1000);

    // Calculate the expireAt time (90 days from now) in epoch second format
    const expire_at = Math.floor((new Date().getTime() + 90 * 24 * 60 * 60 *
 1000) / 1000);

    // Create DynamoDB item
    const item = {
        'partitionKey': {'S': partition_key},
        'sortKey': {'S': sort_key},
        'createdAt': {'N': current_time.toString()},
        'expireAt': {'N': expire_at.toString()}
    };

    const putItemCommand = new PutItemCommand({
        TableName: table_name,
        Item: item,
        ProvisionedThroughput: {
            ReadCapacityUnits: 1,
            WriteCapacityUnits: 1,
        },
    });

    client.send(putItemCommand, function(err, data) {
        if (err) {
            console.log("Exception encountered when creating item %s, here's what
 happened: ", data, err);
            throw err;
        } else {
            console.log("Item created successfully: %s.", data);
            return data;
        }
    });
}

// Example usage (commented out for testing)
// createDynamoDBItem('your-table-name', 'us-east-1', 'your-partition-key-value',
 'your-sort-key-value');
```

- Einzelheiten zur API finden Sie PutItemin der AWS SDK für JavaScript API-Referenz.

Python

SDK für Python (Boto3)

```
from datetime import datetime, timedelta

import boto3


def create_dynamodb_item(table_name, region, primary_key, sort_key):
    """
    Creates a DynamoDB item with an attached expiry attribute.

    :param table_name: Table name for the boto3 resource to target when creating
 an item
    :param region: string representing the AWS region. Example: `us-east-1`
    :param primary_key: one attribute known as the partition key.
    :param sort_key: Also known as a range attribute.
    :return: Void (nothing)
    """
    try:
        dynamodb = boto3.resource("dynamodb", region_name=region)
        table = dynamodb.Table(table_name)

        # Get the current time in epoch second format
        current_time = int(datetime.now().timestamp())

        # Calculate the expiration time (90 days from now) in epoch second format
        expiration_time = int((datetime.now() + timedelta(days=90)).timestamp())

        item = {
            "primaryKey": primary_key,
            "sortKey": sort_key,
            "creationDate": current_time,
            "expireAt": expiration_time,
        }
        response = table.put_item(Item=item)

        print("Item created successfully.")
        return response
    except Exception as e:
        print(f"Error creating item: {e}")
        raise e
```

```
# Use your own values
create_dynamodb_item(
    "your-table-name", "us-west-2", "your-partition-key-value", "your-sort-key-
value"
)
```

- Einzelheiten zur API finden Sie PutItemin AWS SDK for Python (Boto3) API Reference.

Aktualisieren eines Elements und der Time to Live

Dies ist eine Fortsetzung des Beispiels aus dem vorherigen Abschnitt. Die Ablaufzeit kann neu berechnet werden, wenn das Element aktualisiert wird. Im folgenden Beispiel wird der Zeitstempel `expireAt` neu berechnet, sodass er 90 Tage nach der aktuellen Uhrzeit liegt.

Die folgenden Codebeispiele veranschaulichen, wie Sie die TTL eines Elements aktualisieren.

Java

SDK für Java 2.x

Aktualisieren Sie die TTL für ein vorhandenes DynamoDB-Element in einer Tabelle.

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.AttributeValue;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.model.ResourceNotFoundException;
import software.amazon.awssdk.services.dynamodb.model.UpdateItemRequest;
import software.amazon.awssdk.services.dynamodb.model.UpdateItemResponse;

import java.util.HashMap;
import java.util.Map;
import java.util.Optional;

    public UpdateItemResponse updateItemWithTTL(
        final String tableName, final String primaryKeyValue, final String
 sortKeyValue) {
        // Get current time in epoch second format
        final long currentTime = System.currentTimeMillis() / 1000;
```

```
        // Calculate expiration time 90 days from now in epoch second format
        final long expireDate = currentTime + (DAYS_TO_EXPIRE * SECONDS_PER_DAY);

        // Create the key map for the item to update
        final Map<String, AttributeValue> keyMap = new HashMap<>();
        keyMap.put(PRIMARY_KEY_ATTR,
AttributeValue.builder().s(primaryKeyValue).build());
        keyMap.put(SORT_KEY_ATTR,
AttributeValue.builder().s(sortKeyValue).build());

        // Create the expression attribute values
        final Map<String, AttributeValue> expressionAttributeValues = new
HashMap<>();
        expressionAttributeValues.put(
            ":c",
AttributeValue.builder().n(String.valueOf(currentTime)).build());
        expressionAttributeValues.put(
            ":e",
AttributeValue.builder().n(String.valueOf(expireDate)).build());

        final UpdateItemRequest request = UpdateItemRequest.builder()
            .tableName(tableName)
            .key(keyMap)
            .updateExpression(UPDATE_EXPRESSION)
            .expressionAttributeValues(expressionAttributeValues)
            .build();

        try {
            final UpdateItemResponse response =
dynamoDbClient.updateItem(request);
            System.out.println(String.format(SUCCESS_MESSAGE, tableName));
            return response;
        } catch (ResourceNotFoundException e) {
            System.err.format(TABLE_NOT_FOUND_ERROR, tableName);
            throw e;
        } catch (DynamoDbException e) {
            System.err.println(e.getMessage());
            throw e;
        }
    }
```

- Einzelheiten zur API finden Sie unter UpdateItem AWS SDK for Java 2.xAPI-Referenz.

JavaScript

SDK für JavaScript (v3)

```
import { DynamoDBClient, UpdateItemCommand } from "@aws-sdk/client-dynamodb";
import { marshall, unmarshall } from "@aws-sdk/util-dynamodb";

export const updateItem = async (tableName, partitionKey, sortKey, region = 'us-
east-1') => {
    const client = new DynamoDBClient({
        region: region,
        endpoint: `https://dynamodb.${region}.amazonaws.com`
    });

    const currentTime = Math.floor(Date.now() / 1000);
    const expireAt = Math.floor((Date.now() + 90 * 24 * 60 * 60 * 1000) / 1000);

    const params = {
        TableName: tableName,
        Key: marshall({
            partitionKey: partitionKey,
            sortKey: sortKey
        }),
        UpdateExpression: "SET updatedAt = :c, expireAt = :e",
        ExpressionAttributeValues: marshall({
            ":c": currentTime,
            ":e": expireAt
        }),
    };

    try {
        const data = await client.send(new UpdateItemCommand(params));
        const responseData = unmarshall(data.Attributes);
        console.log("Item updated successfully: %s", responseData);
        return responseData;
    } catch (err) {
        console.error("Error updating item:", err);
        throw err;
    }
}

// Example usage (commented out for testing)
// updateItem('your-table-name', 'your-partition-key-value', 'your-sort-key-
value');
```

- Einzelheiten zur API finden Sie UpdateItemin der AWS SDK für JavaScript API-Referenz.

Python

SDK für Python (Boto3)

```
from datetime import datetime, timedelta

import boto3


def update_dynamodb_item(table_name, region, primary_key, sort_key):
    """
    Update an existing DynamoDB item with a TTL.
    :param table_name: Name of the DynamoDB table
    :param region: AWS Region of the table - example `us-east-1`
    :param primary_key: one attribute known as the partition key.
    :param sort_key: Also known as a range attribute.
    :return: Void (nothing)
    """
    try:
        # Create the DynamoDB resource.
        dynamodb = boto3.resource("dynamodb", region_name=region)
        table = dynamodb.Table(table_name)

        # Get the current time in epoch second format
        current_time = int(datetime.now().timestamp())

        # Calculate the expireAt time (90 days from now) in epoch second format
        expire_at = int((datetime.now() + timedelta(days=90)).timestamp())

        table.update_item(
            Key={"partitionKey": primary_key, "sortKey": sort_key},
            UpdateExpression="set updatedAt=:c, expireAt=:e",
            ExpressionAttributeValues={":c": current_time, ":e": expire_at},
        )

        print("Item updated successfully.")
    except Exception as e:
        print(f"Error updating item: {e}")
```

```
# Replace with your own values
update_dynamodb_item(
    "your-table-name", "us-west-2", "your-partition-key-value", "your-sort-key-
value"
)
```

- Einzelheiten zur API finden Sie UpdateItemin AWS SDK for Python (Boto3) API Reference.

Die in dieser Einführung vorgestellten TTL-Beispiele sind ein Weg, um sicherzustellen, dass nur kürzlich aktualisierte Elemente in einer Tabelle gespeichert bleiben. Die Lebensdauer aktualisierter Elemente wird verlängert, wohingegen Elemente, die seit ihrer Erstellung nicht mehr aktualisiert wurden, ablaufen und kostenlos gelöscht werden. So wird Speicherplatz reduziert und die Tabellen werden auf dem neuesten Stand gehalten.

Arbeiten mit abgelaufenen Elementen und Time to Live (TTL)

Abgelaufene Elemente, deren Löschung noch aussteht, können aus Lese- und Schreibvorgängen herausgefiltert werden. Dies ist nützlich in Szenarien, in denen abgelaufene Daten nicht mehr gültig sind und nicht verwendet werden sollten. Wenn sie nicht gefiltert werden, werden sie weiterhin bei Lese- und Schreibvorgängen angezeigt, bis sie vom Hintergrundprozess gelöscht werden.

> **Note**
>
> Diese Elemente werden weiterhin auf die Speicher- und Lesekosten angerechnet, bis sie gelöscht werden.

TTL-Löschungen können in DynamoDB Streams identifiziert werden, jedoch nur in der Region, in der die Löschung stattgefunden hat. TTL-Löschungen, die in globale Tabellenbereiche repliziert werden, sind in DynamoDB-Streams in den Regionen, in die die Löschung repliziert wurde, nicht identifizierbar.

Filtern abgelaufener Elemente aus Lesevorgängen

Bei Lesevorgängen wie Scan und Abfrage kann ein Filterausdruck abgelaufene Elemente herausfiltern, deren Löschung noch aussteht. Wie im folgenden Codeausschnitt gezeigt, kann der

Filterausdruck Elemente herausfiltern, bei denen die TTL-Zeit gleich oder kleiner als die aktuelle Zeit ist. Der Python-SDK-Code enthält beispielsweise eine Zuweisungsanweisung, die die aktuelle Uhrzeit als Variable (`now`) abruft und sie in `int` für das Unix-Zeitformat konvertiert.

In den folgenden Codebeispielen wird gezeigt, wie TTL-Elemente abgefragt werden.

Java

SDK für Java 2.x

Abfragen eines gefilterten Ausdrucks zum Sammeln von TTL-Elementen in einer DynamoDB-Tabelle mithilfe von. AWS SDK for Java 2.x

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.AttributeValue;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.model.QueryRequest;
import software.amazon.awssdk.services.dynamodb.model.QueryResponse;
import software.amazon.awssdk.services.dynamodb.model.ResourceNotFoundException;

import java.util.Map;
import java.util.Optional;

        final QueryRequest request = QueryRequest.builder()
            .tableName(tableName)
            .keyConditionExpression(KEY_CONDITION_EXPRESSION)
            .filterExpression(FILTER_EXPRESSION)
            .expressionAttributeNames(expressionAttributeNames)
            .expressionAttributeValues(expressionAttributeValues)
            .build();

        try (DynamoDbClient ddb = dynamoDbClient != null
            ? dynamoDbClient
            : DynamoDbClient.builder().region(region).build()) {
            final QueryResponse response = ddb.query(request);
            System.out.println("Query successful. Found " + response.count() + "
 items that have not expired yet.");

            // Print each item
            response.items().forEach(item -> {
                System.out.println("Item: " + item);
            });
```

```
            return 0;
        } catch (ResourceNotFoundException e) {
            System.err.format(TABLE_NOT_FOUND_ERROR, tableName);
            throw e;
        } catch (DynamoDbException e) {
            System.err.println(e.getMessage());
            throw e;
        }
```

- Weitere API-Informationen finden Sie unter Abfragen in der AWS SDK for Java 2.x -API-Referenz.

JavaScript

SDK für JavaScript (v3)

Abfragen eines gefilterten Ausdrucks zum Sammeln von TTL-Elementen in einer DynamoDB-Tabelle mithilfe von. AWS SDK für JavaScript

```
import { DynamoDBClient, QueryCommand } from "@aws-sdk/client-dynamodb";
import { marshall, unmarshall } from "@aws-sdk/util-dynamodb";

export const queryFiltered = async (tableName, primaryKey, region = 'us-east-1')
 => {
    const client = new DynamoDBClient({
        region: region,
        endpoint: `https://dynamodb.${region}.amazonaws.com`
    });

    const currentTime = Math.floor(Date.now() / 1000);

    const params = {
        TableName: tableName,
        KeyConditionExpression: "#pk = :pk",
        FilterExpression: "#ea > :ea",
        ExpressionAttributeNames: {
            "#pk": "primaryKey",
            "#ea": "expireAt"
        },
        ExpressionAttributeValues: marshall({
```

```
            ":pk": primaryKey,
            ":ea": currentTime
        })
    };

    try {
        const { Items } = await client.send(new QueryCommand(params));
        Items.forEach(item => {
            console.log(unmarshall(item))
        });
        return Items;
    } catch (err) {
        console.error(`Error querying items: ${err}`);
        throw err;
    }
}

// Example usage (commented out for testing)
// queryFiltered('your-table-name', 'your-partition-key-value');
```

- Weitere API-Informationen finden Sie unter Abfragen in der AWS SDK für JavaScript -API-Referenz.

Python

SDK für Python (Boto3)

Abfragen eines gefilterten Ausdrucks zum Sammeln von TTL-Elementen in einer DynamoDB-Tabelle mithilfe von. AWS SDK für Python (Boto3)

```
from datetime import datetime

import boto3


def query_dynamodb_items(table_name, partition_key):
    """

    :param table_name: Name of the DynamoDB table
    :param partition_key:
    :return:
```

```
    """
    try:
        # Initialize a DynamoDB resource
        dynamodb = boto3.resource("dynamodb", region_name="us-east-1")

        # Specify your table
        table = dynamodb.Table(table_name)

        # Get the current time in epoch format
        current_time = int(datetime.now().timestamp())

        # Perform the query operation with a filter expression to exclude expired
 items
        # response = table.query(
        #
 KeyConditionExpression=boto3.dynamodb.conditions.Key('partitionKey').eq(partition_key),
        #
 FilterExpression=boto3.dynamodb.conditions.Attr('expireAt').gt(current_time)
        # )
        response = table.query(

 KeyConditionExpression=dynamodb.conditions.Key("partitionKey").eq(partition_key),

 FilterExpression=dynamodb.conditions.Attr("expireAt").gt(current_time),
        )

        # Print the items that are not expired
        for item in response["Items"]:
            print(item)

    except Exception as e:
        print(f"Error querying items: {e}")


# Call the function with your values
query_dynamodb_items("Music", "your-partition-key-value")
```

- Weitere API-Informationen finden Sie unter Abfragen in der API-Referenz zum AWS SDK für Python (Boto3).

Bedingtes Schreiben in abgelaufene Elemente

Ein Bedingungsausdruck kann verwendet werden, um Schreibvorgänge auf abgelaufene Elemente zu verhindern. Der folgende Codeausschnitt ist eine bedingte Aktualisierung, die prüft, ob die Ablaufzeit nach der aktuellen Zeit liegt. Falls ja, wird der Schreibvorgang fortgesetzt.

In den folgenden Codebeispielen wird gezeigt, wie die TTL eines Elements bedingungsabhängig aktualisiert wird.

Java

SDK für Java 2.x

Aktualisieren Sie die TTL für ein vorhandenes DynamoDB-Element in einer Tabelle mit einer Bedingung.

```
package com.amazon.samplelib.ttl;

import com.amazon.samplelib.CodeSampleUtils;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.AttributeValue;
import
 software.amazon.awssdk.services.dynamodb.model.ConditionalCheckFailedException;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.model.ResourceNotFoundException;
import software.amazon.awssdk.services.dynamodb.model.UpdateItemRequest;
import software.amazon.awssdk.services.dynamodb.model.UpdateItemResponse;

import java.util.Map;
import java.util.Optional;

/**
 * Updates an item in a DynamoDB table with TTL attributes using a conditional
 expression.
 * This class demonstrates how to conditionally update TTL expiration timestamps.
 */
public class UpdateTTLConditional {

    private static final String USAGE =
        """
            Usage:
```

```
                <tableName> <primaryKey> <sortKey> <region>
            Where:
                tableName - The Amazon DynamoDB table being queried.
                primaryKey - The name of the primary key. Also known as the hash
or partition key.
                sortKey - The name of the sort key. Also known as the range
attribute.
                region (optional) - The AWS region that the Amazon DynamoDB table
is located in. (Default: us-east-1)
            """;
   private static final int DAYS_TO_EXPIRE = 90;
   private static final int SECONDS_PER_DAY = 24 * 60 * 60;
   private static final String PRIMARY_KEY_ATTR = "primaryKey";
   private static final String SORT_KEY_ATTR = "sortKey";
   private static final String UPDATED_AT_ATTR = "updatedAt";
   private static final String EXPIRE_AT_ATTR = "expireAt";
   private static final String UPDATE_EXPRESSION = "SET " + UPDATED_AT_ATTR +
"=:c, " + EXPIRE_AT_ATTR + "=:e";
   private static final String CONDITION_EXPRESSION = "attribute_exists(" +
PRIMARY_KEY_ATTR + ")";
   private static final String SUCCESS_MESSAGE = "%s UpdateItem operation with
TTL successful.";
   private static final String CONDITION_FAILED_MESSAGE = "Condition check
failed. Item does not exist.";
   private static final String TABLE_NOT_FOUND_ERROR = "Error: The Amazon
DynamoDB table \"%s\" can't be found.";

   private final DynamoDbClient dynamoDbClient;

   /**
    * Constructs an UpdateTTLConditional with a default DynamoDB client.
    */
   public UpdateTTLConditional() {
       this.dynamoDbClient = null;
   }

   /**
    * Constructs an UpdateTTLConditional with the specified DynamoDB client.
    *
    * @param dynamoDbClient The DynamoDB client to use
    */
   public UpdateTTLConditional(final DynamoDbClient dynamoDbClient) {
       this.dynamoDbClient = dynamoDbClient;
   }
```

```
    /**
     * Main method to demonstrate conditionally updating an item with TTL.
     *
     * @param args Command line arguments
     */
    public static void main(final String[] args) {
        try {
            int result = new UpdateTTLConditional().processArgs(args);
            System.exit(result);
        } catch (Exception e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
    }

    /**
     * Process command line arguments and conditionally update an item with TTL.
     *
     * @param args Command line arguments
     * @return 0 if successful, non-zero otherwise
     * @throws ResourceNotFoundException If the table doesn't exist
     * @throws DynamoDbException If an error occurs during the operation
     * @throws IllegalArgumentException If arguments are invalid
     */
    public int processArgs(final String[] args) {
        // Argument validation (remove or replace this line when reusing this
code)
        CodeSampleUtils.validateArgs(args, new int[] {3, 4}, USAGE);

        final String tableName = args[0];
        final String primaryKey = args[1];
        final String sortKey = args[2];
        final Region region = Optional.ofNullable(args.length > 3 ? args[3] :
null)
            .map(Region::of)
            .orElse(Region.US_EAST_1);

        // Get current time in epoch second format
        final long currentTime = System.currentTimeMillis() / 1000;

        // Calculate expiration time 90 days from now in epoch second format
        final long expireDate = currentTime + (DAYS_TO_EXPIRE * SECONDS_PER_DAY);
```

```
        // Create the key map for the item to update
        final Map<String, AttributeValue> keyMap = Map.of(
            PRIMARY_KEY_ATTR, AttributeValue.builder().s(primaryKey).build(),
            SORT_KEY_ATTR, AttributeValue.builder().s(sortKey).build());

        // Create the expression attribute values
        final Map<String, AttributeValue> expressionAttributeValues = Map.of(
            ":c",
 AttributeValue.builder().n(String.valueOf(currentTime)).build(),
            ":e",
 AttributeValue.builder().n(String.valueOf(expireDate)).build());

        final UpdateItemRequest request = UpdateItemRequest.builder()
            .tableName(tableName)
            .key(keyMap)
            .updateExpression(UPDATE_EXPRESSION)
            .conditionExpression(CONDITION_EXPRESSION)
            .expressionAttributeValues(expressionAttributeValues)
            .build();

        try (DynamoDbClient ddb = dynamoDbClient != null
            ? dynamoDbClient
            : DynamoDbClient.builder().region(region).build()) {
            final UpdateItemResponse response = ddb.updateItem(request);
            System.out.println(String.format(SUCCESS_MESSAGE, tableName));
            return 0;
        } catch (ConditionalCheckFailedException e) {
            System.err.println(CONDITION_FAILED_MESSAGE);
            throw e;
        } catch (ResourceNotFoundException e) {
            System.err.format(TABLE_NOT_FOUND_ERROR, tableName);
            throw e;
        } catch (DynamoDbException e) {
            System.err.println(e.getMessage());
            throw e;
        }
    }
}
```

- Einzelheiten zur API finden Sie unter UpdateItemAPI-Referenz.AWS SDK for Java 2.x

JavaScript

SDK für JavaScript (v3)

Aktualisieren Sie die TTL für ein vorhandenes DynamoDB-Element in einer Tabelle mit einer Bedingung.

```
import { DynamoDBClient, UpdateItemCommand } from "@aws-sdk/client-dynamodb";
import { marshall, unmarshall } from "@aws-sdk/util-dynamodb";

export const updateItemConditional = async (tableName, partitionKey, sortKey,
 region = 'us-east-1', newAttribute = 'default-value') => {
    const client = new DynamoDBClient({
        region: region,
        endpoint: `https://dynamodb.${region}.amazonaws.com`
    });

    const currentTime = Math.floor(Date.now() / 1000);

    const params = {
        TableName: tableName,
        Key: marshall({
            artist: partitionKey,
            album: sortKey
        }),
        UpdateExpression: "SET newAttribute = :newAttribute",
        ConditionExpression: "expireAt > :expiration",
        ExpressionAttributeValues: marshall({
            ':newAttribute': newAttribute,
            ':expiration': currentTime
        }),
        ReturnValues: "ALL_NEW"
    };

    try {
        const response = await client.send(new UpdateItemCommand(params));
        const responseData = unmarshall(response.Attributes);
        console.log("Item updated successfully: ", responseData);
        return responseData;
    } catch (error) {
        if (error.name === "ConditionalCheckFailedException") {
            console.log("Condition check failed: Item's 'expireAt' is expired.");
        } else {
            console.error("Error updating item: ", error);
```

```
        }
        throw error;
    }
};

// Example usage (commented out for testing)
// updateItemConditional('your-table-name', 'your-partition-key-value', 'your-
sort-key-value');
```

- Einzelheiten zur API finden Sie UpdateItemin der AWS SDK für JavaScript API-Referenz.

Python

SDK für Python (Boto3)

Aktualisieren Sie die TTL für ein vorhandenes DynamoDB-Element in einer Tabelle mit einer Bedingung.

```
from datetime import datetime, timedelta

import boto3
from botocore.exceptions import ClientError


def update_dynamodb_item_ttl(table_name, region, primary_key, sort_key,
 ttl_attribute):
    """
    Updates an existing record in a DynamoDB table with a new or updated TTL
 attribute.

    :param table_name: Name of the DynamoDB table
    :param region: AWS Region of the table - example `us-east-1`
    :param primary_key: one attribute known as the partition key.
    :param sort_key: Also known as a range attribute.
    :param ttl_attribute: name of the TTL attribute in the target DynamoDB table
    :return:
    """
    try:
        dynamodb = boto3.resource("dynamodb", region_name=region)
        table = dynamodb.Table(table_name)

        # Generate updated TTL in epoch second format
```

```
        updated_expiration_time = int((datetime.now() +
 timedelta(days=90)).timestamp())

        # Define the update expression for adding/updating a new attribute
        update_expression = "SET newAttribute = :val1"

        # Define the condition expression for checking if 'expireAt' is not
 expired
        condition_expression = "expireAt > :val2"

        # Define the expression attribute values
        expression_attribute_values = {":val1": ttl_attribute, ":val2":
 updated_expiration_time}

        response = table.update_item(
            Key={"primaryKey": primary_key, "sortKey": sort_key},
            UpdateExpression=update_expression,
            ConditionExpression=condition_expression,
            ExpressionAttributeValues=expression_attribute_values,
        )

        print("Item updated successfully.")
        return response["ResponseMetadata"]["HTTPStatusCode"]  # Ideally a 200 OK
    except ClientError as e:
        if e.response["Error"]["Code"] == "ConditionalCheckFailedException":
            print("Condition check failed: Item's 'expireAt' is expired.")
        else:
            print(f"Error updating item: {e}")
    except Exception as e:
        print(f"Error updating item: {e}")


# replace with your values
update_dynamodb_item_ttl(
    "your-table-name",
    "us-east-1",
    "your-partition-key-value",
    "your-sort-key-value",
    "your-ttl-attribute-value",
)
```

- Einzelheiten zur API finden Sie UpdateItemin AWS SDK for Python (Boto3) API Reference.

Identifizieren gelöschter Elemente in DynamoDB Streams

Der Stream-Datensatz enthält ein Benutzeridentitätsfeld `Records[<index>].userIdentity`. Elemente, die durch den TTL-Prozess gelöscht wurden, haben die folgenden Felder:

```
Records[<index>].userIdentity.type
"Service"

Records[<index>].userIdentity.principalId
"dynamodb.amazonaws.com"
```

Der folgende JSON-Ausdruck zeigt den relevanten Teil eines einzelnen Stream-Datensatzes:

```
"Records": [
  {
 ...
  "userIdentity": {
  "type": "Service",
       "principalId": "dynamodb.amazonaws.com"
    }
   ...
 }
]
```

Abfragen von Tabellen in DynamoDB

Sie können die `Query`–API-Operation in Amazon DynamoDB verwenden, um Elemente auf der Basis von Primärschlüsselwerten zu finden.

Sie müssen den Namen des Partitionsschlüsselattributs und einen einzelnen Wert für dieses Attribut angeben. `Query` gibt alle Elemente mit diesem Partitionsschlüsselwert zurück. Sie können optional ein Sortierschlüsselattribut angeben und einen Vergleichsoperator verwenden, um die Suchergebnisse zu verfeinern.

Weitere Informationen zur Verwendung von `Query`, wie beispielsweise die Anforderungssyntax, Antwortparameter und zusätzliche Beispiele, finden Sie unter Abfragen in der API-Referenz zu Amazon DynamoDB.

Themen

- Schlüsselbedingungsausdrücke für die Abfrage-Operation in DynamoDB

- Filterausdrücke für die Abfrage-Operation in DynamoDB
- Paginieren von Tabellenabfrageergebnissen in DynamoDB
- Weitere Aspekte der Arbeit mit der Abfrage-Operation in DynamoDB

Schlüsselbedingungsausdrücke für die Abfrage-Operation in DynamoDB

Sie können alle Attributnamen in einem Schlüsselbedingungsausdruck verwenden, sofern das erste Zeichen `a-z` oder `A-Z` ist und die übrigen Zeichen (ab dem zweiten Zeichen, sofern vorhanden) `a-z`, `A-Z` oder `0-9` sind. Darüber hinaus müssen die Attributnamen nicht DynamoDB-reservierte Wörter sein. (Eine vollständige Liste finden Sie unter Reservierte Wörter in DynamoDB.) Wenn ein Attributname diesen Anforderungen nicht entspricht, müssen Sie einen Ausdruckattributnamen als Platzhalter definieren. Weitere Informationen finden Sie unter Ausdrucksattributnamen (Aliasse) in DynamoDB.

Für Objekte, die mit einem bestimmten Partitionsschlüsselwert versehen sind, speichert DynamoDB diese Elemente nah beieinander, sortiert nach Sortierschlüsselwerten. In einer `Query`-Operation ruft DynamoDB die Elemente sortiert ab und verarbeitet dann die Elemente mit `KeyConditionExpression` und allen `FilterExpression`, die möglicherweise vorhanden sind. Nur dann werden die `Query`-Ergebnisse an den Client zurückgesendet.

Die `Query`-Operation gibt immer eine Ergebnismenge zurück. Wenn keine übereinstimmenden Elemente gefunden werden, ist die Ergebnismenge leer.

`Query`-Ergebnisse werden immer von dem Sortierschlüsselwert sortiert. Wenn der Datentyp des Sortierschlüssels als `Number` festgelegt ist, werden die Ergebnisse in der numerischen Reihenfolge zurückgegeben. Andernfalls werden die Ergebnisse in der Reihenfolge der UTF-8-Bytes zurückgegeben. Standardmäßig ist die Sortierreihenfolge aufsteigend. Um die Reihenfolge umzukehren, setzen Sie den Parameter `ScanIndexForward` auf `false`.

Eine einzelne `Query`-Operation kann bis zu 1 MB an Daten abrufen. Diese Größenbeschränkung gilt, bevor ein `FilterExpression` oder `ProjectionExpression` den Ergebnissen zugeordnet wird. Wenn `LastEvaluatedKey` in der Antwort vorhanden ist und nicht Null ist, müssen Sie die Ergebnismenge wechseln (siehe Paginieren von Tabellenabfrageergebnissen in DynamoDB).

Beispiele für Schlüsselbedingungsausdrücke

Wenn Sie Suchkriterien angeben, verwenden Sie einen Schlüsselbedingungsausdruck – eine Zeichenkette, die die Elemente bestimmt, die von der Tabelle oder dem Index gelesen werden müssen.

Sie müssen den Namen und Wert des Partitionsschlüssels als Gleichheitsbedingung angeben. Sie können kein Nicht-Schlüssel-Attribut in einem Schlüsselbedingungsausdruck verwenden.

Sie können optional eine zweite Bedingung für den Sortierschlüssel (sofern vorhanden) angeben. Die Bedingung des Sortierschlüssels muss einen der folgenden Vergleichsoperatoren verwenden:

- *a* = *b* — wahr, wenn *a* das Attribut dem Wert entspricht *b*
- *a* < *b* — wahr, wenn *a* es kleiner ist als *b*
- *a* <= *b* — wahr, wenn kleiner oder gleich *a* ist *b*
- *a* > *b* — wahr, wenn größer *a* ist als *b*
- *a* >= *b* — wahr, wenn größer oder gleich *a* ist *b*
- *a* `BETWEEN` *b* `AND` *c* — wahr, wenn größer als oder gleich *b* und kleiner als oder gleich *a* ist *c*.

Folgende Funktionen werden ebenfalls unterstützt:

- `begins_with (`*a*`, `*substr*`)` – True, wenn der Wert von Attribut *a* mit einer bestimmten Teilkettenfolge beginnt.

Die folgenden Beispiele AWS Command Line Interface (AWS CLI) veranschaulichen die Verwendung von Ausdrücken für Schlüsselbedingungen. Diese Ausdrücke verwenden Platzhalter (z. B. `:name` und `:sub`) anstelle tatsächlicher Werte. Weitere Informationen erhalten Sie unter Ausdrucksattributnamen (Aliasse) in DynamoDB und Verwenden von Ausdrucksattributwerten in DynamoDB.

Example

Fragen Sie die `Thread`-Tabelle nach einem bestimmten `ForumName` (Partitionsschlüssel) ab. Alle Elemente mit diesem `ForumName`-Wert werden von der Abfrage gelesen, da der Sortierschlüssel (`Subject`) in `KeyConditionExpression` nicht enthalten ist.

```
aws dynamodb query \
    --table-name Thread \
    --key-condition-expression "ForumName = :name" \
    --expression-attribute-values  '{":name":{"S":"Amazon DynamoDB"}}'
```

Example

Fragen Sie die `Thread`-Tabelle nach einem bestimmten `ForumName` (Partitions-Schlüssel) ab, aber dieses Mal geben Sie nur die Elemente mit einem bestimmten `Subject` (Sortierschlüssel) zurück.

```
aws dynamodb query \
    --table-name Thread \
    --key-condition-expression "ForumName = :name and Subject = :sub" \
    --expression-attribute-values  file://values.json
```

Die Argumente für `--expression-attribute-values` werden in der Datei `values.json` gespeichert.

```
{
    ":name":{"S":"Amazon DynamoDB"},
    ":sub":{"S":"DynamoDB Thread 1"}
}
```

Example

Fragen Sie die `Reply`-Tabelle nach einer bestimmten `Id` (Partitionsschlüssel) ab, aber geben Sie nur die Elemente zurück, deren `ReplyDateTime` (Sortierschlüssel) mit bestimmten Zeichen beginnt.

```
aws dynamodb query \
    --table-name Reply \
    --key-condition-expression "Id = :id and begins_with(ReplyDateTime, :dt)" \
    --expression-attribute-values  file://values.json
```

Die Argumente für `--expression-attribute-values` werden in der Datei `values.json` gespeichert.

```
{
    ":id":{"S":"Amazon DynamoDB#DynamoDB Thread 1"},
    ":dt":{"S":"2015-09"}
}
```

Filterausdrücke für die Abfrage-Operation in DynamoDB

Wenn Sie die `Query`-Ergebnisse weiter eingrenzen müssen, können Sie optional einen Filterausdruck angeben. Ein Filterausdruck bestimmt, welche Elemente in den `Query`-Ergebnissen an Sie zurückgegeben werden. Alle anderen Ergebnisse werden verworfen.

Ein Filterausdruck wird angewendet, nachdem eine `Query` abgeschlossen ist, aber bevor die Ergebnisse zurückgegeben werden. Folglich verbraucht eine `Query` unabhängig davon, ob ein Filterausdruck vorhanden ist oder nicht, gleich viel Lesekapazität.

Eine `Query`-Operation kann bis zu 1 MB an Daten abrufen. Diese Größenbeschränkung gilt, bevor der Filterausdruck ausgewertet wird.

Ein Filterausdruck darf keine Partitionsschlüssel- oder Sortierschlüsselattribute enthalten. Sie müssen diese Attribute in dem Schlüsselbedingungsausdruck und nicht in dem Filterausdruck angeben.

Die Syntax für einen Filterausdruck ist der eines Schlüsselbedingungsausdrucks ähnlich. Filterausdrücke können dieselben Vergleichsoperatoren, Funktionen und logischen Operatoren wie ein Schlüsselbedingungsausdruck verwenden. Darüber hinaus können Filterausdrücke den Nicht-Gleichheitsoperator (<>), den`OR`-Operator, den`CONTAINS`-Operator, den `IN`-Operator, den `BEGINS_WITH`-Operator, den `BETWEEN`-Operator, den `EXISTS`-Operator und den`SIZE`-Operator verwenden. Weitere Informationen erhalten Sie unter Schlüsselbedingungsausdrücke für die Abfrage-Operation in DynamoDB und Syntax für Filter- und Bedingungsausdrücke.

Example

Im folgenden AWS CLI Beispiel wird die `Thread` Tabelle nach einem bestimmten `ForumName` (Partitionsschlüssel) und `Subject` (Sortierschlüssel) abgefragt. Von den gefundenen Elementen werden nur die beliebtesten Diskussionsthreads zurückgegeben, also nur die Threads mit mehr als einer bestimmten Anzahl von `Views`.

```
aws dynamodb query \
    --table-name Thread \
    --key-condition-expression "ForumName = :fn and Subject begins_with :sub" \
    --filter-expression "#v >= :num" \
    --expression-attribute-names '{"#v": "Views"}' \
    --expression-attribute-values file://values.json
```

Die Argumente für `--expression-attribute-values` werden in der Datei `values.json` gespeichert.

```
{
    ":fn":{"S":"Amazon DynamoDB"},
    ":sub":{"S":"DynamoDB Thread 1"},
    ":num":{"N":"3"}
}
```

Beachten Sie, dass `Views` ein reserviertes Wort in DynamoDB ist (siehe Reservierte Wörter in DynamoDB), daher verwendet dieses Beispiel `#v` als Platzhalter. Weitere Informationen finden Sie unter Ausdrucksattributnamen (Aliasse) in DynamoDB.

> **Note**
>
> Ein Filterausdruck entfernt Elemente aus der `Query`-Ergebnismenge. Verwenden Sie nach Möglichkeit `Query` nicht, wenn Sie erwarten, eine große Anzahl von Elementen abzurufen, jedoch auch die meisten dieser Elemente verwerfen zu müssen.

Paginieren von Tabellenabfrageergebnissen in DynamoDB

DynamoDB paginiert die Ergebnisse von `Query`-Operationen. Bei der Paginierung werden die `Query`-Ergebnisse in „Seiten“ mit Daten von einer Größe von 1 MB (oder weniger) unterteilt. Eine Anwendung kann die erste Ergebnisseite verarbeiten, dann die zweite Seite und so weiter.

Eine einzelne `Query` gibt nur einen Ergebnissatz zurück, der innerhalb des Grenzwerts von 1 MB liegt. Um zu bestimmen, ob es mehr Ergebnisse gibt, und diese seitenweise abzurufen, sollte eine Anwendung die folgenden Schritte ausführen:

1. Überprüfen Sie das `Query`-Low-Level-Ergebnis:
   - Wenn das Ergebnis ein `LastEvaluatedKey`-Element enthält und es ist ungleich null, fahren Sie mit Schritt 2 fort.
   - Wenn im Ergebnis kein `LastEvaluatedKey` vorhanden ist, sind keine Elemente mehr zum Abrufen vorhanden.
2. Erstellen Sie eine `Query` mit demselben `KeyConditionExpression`. Verwenden Sie jedoch dieses Mal den `LastEvaluatedKey`-Wert aus Schritt 1 als `ExclusiveStartKey`-Parameter in der neuen `Query`-Anforderung.
3. Führen Sie die neue `Query`-Anforderung aus.
4. Fahren Sie mit Schritt 1 fort.

Mit anderen Worten: Der `LastEvaluatedKey` einer `Query`-Antwort sollte als `ExclusiveStartKey` für die nächste `Query`-Anforderung verwendet werden. Wenn in einer `LastEvaluatedKey`-Antwort kein `Query`-Element vorhanden ist, haben Sie die letzte Ergebnisseite abgerufen. Wenn `LastEvaluatedKey` nicht leer ist, bedeutet dies nicht notwendigerweise, dass die Ergebnismenge

mehr Daten enthält. Die einzige Möglichkeit zu erfahren, dass das Ende des Ergebnissatzes erreicht wurde, ist, dass `LastEvaluatedKey` leer ist.

Sie können das verwenden AWS CLI , um sich dieses Verhalten anzusehen. Der AWS CLI sendet wiederholt `Query` Anfragen auf niedriger Ebene an DynamoDB, bis er in den Ergebnissen nicht mehr vorhanden `LastEvaluatedKey` ist. Betrachten Sie das folgende AWS CLI Beispiel, in dem Filmtitel aus einem bestimmten Jahr abgerufen werden.

```
aws dynamodb query --table-name Movies \
    --projection-expression "title" \
    --key-condition-expression "#y = :yyyy" \
    --expression-attribute-names '{"#y":"year"}' \
    --expression-attribute-values '{":yyyy":{"N":"1993"}}' \
    --page-size 5 \
    --debug
```

Normalerweise AWS CLI verarbeitet der die Seitennummerierung automatisch. In diesem Beispiel begrenzt der AWS CLI `--page-size` Parameter jedoch die Anzahl der Elemente pro Seite. Der `--debug`-Parameter gibt Low-Level-Informationen zu Anforderungen und Antworten aus.

Wenn Sie das Beispiel ausführen, sieht die erste Antwort von DynamoDB ungefähr folgendermaßen aus.

```
2017-07-07 11:13:15,603 - MainThread - botocore.parsers - DEBUG - Response body:
b'{"Count":5,"Items":[{"title":{"S":"A Bronx Tale"}},
{"title":{"S":"A Perfect World"}},{"title":{"S":"Addams Family Values"}},
{"title":{"S":"Alive"}},{"title":{"S":"Benny & Joon"}}],
"LastEvaluatedKey":{"year":{"N":"1993"},"title":{"S":"Benny & Joon"}},
"ScannedCount":5}'
```

Der `LastEvaluatedKey` in der Antwort gibt an, dass nicht alle Elemente abgerufen wurden. Das sendet AWS CLI dann eine weitere `Query` Anfrage an DynamoDB. Dieses Anforderungs- und Antwortmuster wird bis zur endgültigen Antwort fortgesetzt.

```
2017-07-07 11:13:16,291 - MainThread - botocore.parsers - DEBUG - Response body:
b'{"Count":1,"Items":[{"title":{"S":"What\'s Eating Gilbert
 Grape"}}],"ScannedCount":1}'
```

Das Fehlen von `LastEvaluatedKey` gibt an, dass keine abrufbaren Elemente mehr vorhanden sind.

> **Note**
>
> Sie AWS SDKs verarbeiten DynamoDB-Antworten auf niedriger Ebene (einschließlich des Vorhandenseins oder Fehlens von`LastEvaluatedKey`) und bieten verschiedene Abstraktionen für die Paginierung von Ergebnissen. `Query` Beispielsweise stellt SDK für die Java-Dokumentschnittstelle `java.util.Iterator`-Unterstützung bereit, sodass Sie die Ergebnisse nacheinander durchgehen können.
> Codebeispiele in verschiedenen Programmiersprachen finden Sie im Amazon-DynamoDB-Handbuch für erste Schritte und in der AWS SDK-Dokumentation für Ihre Sprache.

Sie können die Seitengröße auch reduzieren, indem Sie die Anzahl der Elemente in der Ergebnismenge mit dem `Limit`-Parameter der`Query`-Operation verwenden.

Weitere Informationen zur Verwendung der Abfrage mit DynamoDB finden Sie unter Abfragen von Tabellen in DynamoDB.

Weitere Aspekte der Arbeit mit der Abfrage-Operation in DynamoDB

In diesem Abschnitt werden weitere Aspekte der DynamoDB-Abfrageoperation behandelt, darunter die Beschränkung der Ergebnisgröße, das Zählen gescannter gegenüber zurückgegebener Elemente, die Überwachung des Lesekapazitätsverbrauchs und die Steuerung der Lesekonsistenz.

Begrenzung der Anzahl an Elementen in der Ergebnismenge

Die `Query`-Operation ermöglicht Ihnen, die Anzahl der Elemente einzuschränken, die gelesen werden. Legen Sie dazu den Parameter `Limit` auf die maximale Anzahl Elemente fest, die zurückgegeben werden sollen.

Beispiel: Sie führen eine `Query` für eine Tabelle mit einem `Limit`-Wert von 6 und ohne Filterausdruck durch. Das `Query`-Ergebnis enthält die ersten sechs Elemente der Tabelle, die mit dem Schlüsselbedingungsausdruck der Anforderung übereinstimmen.

Angenommen, Sie fügen nun einen Filterausdruck der `Query` hinzu. In diesem Fall liest DynamoDB bis zu sechs Elemente und gibt dann nur diejenigen zurück, die mit dem Filterausdruck übereinstimmen. Das `Query`-Endergebnis enthält sechs oder weniger Elemente, selbst wenn mehr Elemente mit dem Filterausdruck übereinstimmen würden, wenn DynamoDB weitere Elemente gelesen hätte.

Zählen der Elemente in den Ergebnissen

Zusätzlich zu den Elementen, die mit Ihren Kriterien übereinstimmen, enthält die `Query`-Antwort noch die folgenden Elemente:

- `ScannedCount` – die Anzahl der Elemente, die mit dem Schlüsselbedingungsausdruck übereinstimmten, bevor ein Filterausdruck (falls vorhanden) angewendet wurde.
- `Count` – die Anzahl der Elemente, die verbleiben, nachdem ein Filterausdruck (falls vorhanden) angewendet wurde.

> **Note**
>
> Wenn Sie keinen Filterausdruck verwenden, haben `ScannedCount` und `Count` den gleichen Wert.

Wenn der `Query`-Ergebnissatz größer als 1 MB ist, repräsentieren `ScannedCount` und `Count` nur eine Teilmenge der Gesamtelemente. Sie müssen mehrere `Query`-Operationen ausführen, um alle Ergebnisse abzurufen (siehe Paginieren von Tabellenabfrageergebnissen in DynamoDB).

Jede `Query`-Antwort enthält die `ScannedCount` und `Count` für die Elemente, die von der betreffenden `Query`-Anforderung verarbeitet wurden. Um eine Gesamtsumme für alle `Query`-Anforderungen zu erhalten, könnten Sie eine laufende Zählung der Werte `ScannedCount` und `Count` vornehmen.

Durch die Abfrage verbrauchte Kapazitätseinheiten

Sie können `Query` für jede Tabelle oder jeden sekundären Index verwenden, sofern Sie den Namen des Partitionsschlüsselattributs und einen einzelnen Wert für dieses Attribut angeben. `Query` gibt alle Elemente mit diesem Partitionsschlüsselwert zurück. Sie können optional ein Sortierschlüsselattribut angeben und einen Vergleichsoperator verwenden, um die Suchergebnisse zu verfeinern. `Query` API-Operationen verbrauchen Lesekapazitätseinheiten wie folgt.

Wenn Sie **`Query`** eine ...	DynamoDB verbraucht Lesekapazitätseinheiten von ...
Tabelle	Die bereitgestellte Lesekapazität der Tabelle.

Wenn Sie **`Query`** eine ...	DynamoDB verbraucht Lesekapazitätseinheiten von ...
Globaler sekundärer Index	Die bereitgestellte Lesekapazität des Index.
Lokaler sekundärer Index	Die bereitgestellte Lesekapazität der Basistabelle.

Standardmäßig gibt eine `Query`-Operation keine Daten über seinen Lesekapazitätsverbrauch zurück. Jedoch können Sie den `ReturnConsumedCapacity`-Parameter in einer `Query`-Anforderung angeben, um diese Information zu erhalten. Folgende sind die gültigen Einstellungen für `ReturnConsumedCapacity`:

- `NONE` – Es werden keine verbrauchten Kapazitätsdaten zurückgegeben. (Dies ist die Standardeinstellung.)
- `TOTAL` – Die Antwort enthält die zusammengefasste Anzahl der verbrauchten Lesekapazitätseinheiten.
- `INDEXES` – Die Antwort zeigt die zusammengefasste Anzahl der verbrauchten Lesekapazitätseinheiten, zusammen mit der verbrauchten Kapazität für jede aufgerufene Tabelle und jeden aufgerufenen Index, an.

DynamoDB berechnet die Anzahl von verbrauchten Lesekapazitätseinheiten basierend auf der Anzahl und Größe der Elemente, nicht der Menge der Daten, die an eine Anwendung zurückgegeben werden. Aus diesem Grund bleibt die Anzahl der verbrauchten Kapazitätseinheiten gleich, unabhängig davon, ob Sie alle Attribute (das Standardverhalten) oder nur einige von ihnen anfordern (mithilfe eines Projektionsausdrucks). Die Anzahl ist auch gleich, unabhängig davon, ob Sie einen Filterausdruck verwenden oder nicht. `Query` verbraucht eine Mindestlesekapazitätseinheit für einen strikt konsistenten Lesevorgang pro Sekunde oder zwei letztendlich konsistente Lesevorgänge pro Sekunde für ein Element von bis zu 4 KB. Wenn Sie ein Element lesen möchten, das größer als 4 KB ist, benötigt DynamoDB zusätzliche Leseanforderungseinheiten. Bei leeren Tabellen und sehr großen Tabellen mit einer geringen Anzahl an Partitionsschlüsseln können zusätzliche RCUs Gebühren anfallen, die über die abgefragte Datenmenge hinausgehen. Dies deckt die Kosten für die `Query`-Bearbeitung der Anfrage, auch wenn keine Daten vorhanden sind.

Lesekonsistenz für die Abfrage

Eine `Query`-Operation führt standardmäßig lEventually Consistent-Lesevorgänge aus. Dies bedeutet, dass die `Query`-Ergebnisse Änderungen aufgrund kürzlich abgeschlossener `PutItem` oder `UpdateItem`-Operationen, nicht wiedergeben könnten. Weitere Informationen finden Sie unter DynamoDB-Lesekonsistenz.

Wenn Sie Strongly Consistent-Lesevorgänge benötigen, legen Sie den `ConsistentRead`-Parameter in der `Query`-Anforderung auf `true` fest.

Scannen von Tabellen in DynamoDB

Eine `Scan`-Operation in Amazon DynamoDB liest jedes Element in einer Tabelle oder einem Sekundärindex. Standardmäßig gibt ein `Scan`-Vorgang für jedes Element in der Tabelle oder im Index alle Datenattribute zurück. Sie können den `ProjectionExpression`-Parameter verwenden, sodass `Scan` nur einige der Attribute und nicht alle zurückgibt.

`Scan` gibt immer einen Ergebnissatz zurück. Wenn keine übereinstimmenden Elemente gefunden werden, ist die Ergebnismenge leer.

Eine einzelne `Scan`-Anforderung kann maximal 1 MB Daten abrufen. DynamoDB kann optional einen Filterausdruck auf diese Daten anwenden, um die Ergebnisse einzugrenzen, bevor sie an den Benutzer zurückgegeben werden.

Themen

- Filterausdrücke für Scan
- Begrenzung der Anzahl an Elementen in der Ergebnismenge
- Paginierung der Ergebnisse
- Zählen der Elemente in den Ergebnissen
- Durch die Abfrage verbrauchte Kapazitätseinheiten
- Lesekonsistenz für Scan
- Parallele Scans

Filterausdrücke für Scan

Wenn Sie die `Scan`-Ergebnisse weiter eingrenzen müssen, können Sie optional einen Filterausdruck angeben. Ein Filterausdruck bestimmt, welche Elemente in den `Scan`-Ergebnissen an Sie zurückgegeben werden. Alle anderen Ergebnisse werden verworfen.

Ein Filterausdruck wird angewendet, nachdem eine `Scan` abgeschlossen ist, aber bevor die Ergebnisse zurückgegeben werden. Folglich verbraucht eine `Scan` unabhängig davon, ob ein Filterausdruck vorhanden ist oder nicht, gleich viel Lesekapazität.

Eine `Scan`-Operation kann bis zu 1 MB an Daten abrufen. Diese Größenbeschränkung gilt, bevor der Filterausdruck ausgewertet wird.

Mit `Scan` können Sie Attribute in einem Filterausdruck angeben—einschließlich Partitionsschlüsseln und Sortierschlüsselattributen.

Die Syntax für einen Filterausdruck ist identisch mit der für einen Bedingungsausdruck. Filterausdrücke können dieselben Vergleichsoperatoren, Funktionen und logischen Operatoren wie ein Bedingungsausdruck verwenden. Weitere Informationen zu logischen Operatoren finden Sie unter Bedingungs- und Filterausdrücke, Operatoren und Funktionen in DynamoDB.

Example

Das folgende Beispiel AWS Command Line Interface (AWS CLI) scannt die `Thread` Tabelle und gibt nur die Elemente zurück, die zuletzt von einem bestimmten Benutzer gepostet wurden.

```
aws dynamodb scan \
     --table-name Thread \
     --filter-expression "LastPostedBy = :name" \
     --expression-attribute-values '{":name":{"S":"User A"}}'
```

Begrenzung der Anzahl an Elementen in der Ergebnismenge

Die `Scan`-Operation ermöglicht es Ihnen, die Anzahl der Elemente einzuschränken, die in den Ergebnissen zurückgegeben werden. Legen Sie dafür vor dem Filtern der Ausdrucksauswertung den Parameter `Limit` auf die maximale Anzahl von Elementen fest, die die `Scan`-Operation zurückgeben soll.

Beispiel: Sie führen eine `Scan`-Operation für eine Tabelle mit einem `Limit`-Wert von 6 und ohne Filterausdruck durch. Die`Scan`-Ergebnis enthält die ersten sechs Elemente der Tabelle.

Angenommen, Sie fügen nun einen Filterausdruck der `Scan` hinzu. In diesem Fall wendet DynamoDB den entsprechenden Filterausdruck auf die sechs zurückgegebenen Elemente an, wobei die nicht übereinstimmenden verworfen werden. Das letzte `Scan`-Ergebnis enthält sechs Elemente oder weniger, je nachdem, wie viele Elemente gefiltert wurden.

Paginierung der Ergebnisse

DynamoDB paginiert die Ergebnisse von `Scan`-Operationen. Bei der Paginierung werden die `Scan`-Ergebnisse in „Seiten“ mit Daten von einer Größe von 1 MB (oder weniger) unterteilt. Eine Anwendung kann die erste Ergebnisseite verarbeiten, dann die zweite Seite und so weiter.

Eine einzelne `Scan`-Operation gibt nur einen Ergebnissatz zurück, der innerhalb des Grenzwerts von 1 MB liegt.

Um zu bestimmen, ob es mehr Ergebnisse gibt, und diese seitenweise abzurufen, sollten Anwendungen die folgenden Schritte ausführen:

1. Überprüfen Sie das `Scan`-Low-Level-Ergebnis:
   - Wenn das Ergebnis ein `LastEvaluatedKey`-Element enthält, fahren Sie mit Schritt 2 fort.
   - Wenn im Ergebnis kein `LastEvaluatedKey` vorhanden ist, sind keine Elemente mehr zum Abrufen vorhanden.
2. Erstellen Sie eine neue `Scan`-Anforderung mit den gleichen Parametern wie bei der vorherigen Anforderung. Verwenden Sie jedoch dieses Mal den `LastEvaluatedKey`-Wert aus Schritt 1 als `ExclusiveStartKey`-Parameter in der neuen `Scan`-Anforderung.
3. Führen Sie die neue `Scan`-Anforderung aus.
4. Fahren Sie mit Schritt 1 fort.

Mit anderen Worten: Der `LastEvaluatedKey` einer `Scan`-Antwort sollte als `ExclusiveStartKey` für die nächste `Scan`-Anforderung verwendet werden. Wenn in einer `Scan`-Antwort kein `LastEvaluatedKey`-Element vorhanden ist, haben Sie die letzte Ergebnisseite abgerufen. (Dass `LastEvaluatedKey` nicht vorhanden ist, ist der einzige Hinweis darauf, dass Sie das Ende des Ergebnissatzes erreicht haben.)

Sie können das verwenden AWS CLI , um sich dieses Verhalten anzusehen. Der AWS CLI sendet wiederholt `Scan` Anfragen auf niedriger Ebene an DynamoDB, bis er in den Ergebnissen nicht mehr vorhanden `LastEvaluatedKey` ist. Stellen Sie sich das folgende AWS CLI Beispiel vor, das die gesamte `Movies` Tabelle scannt, aber nur die Filme eines bestimmten Genres zurückgibt.

```
aws dynamodb scan \
    --table-name Movies \
    --projection-expression "title" \
    --filter-expression 'contains(info.genres,:gen)' \
```

```
    --expression-attribute-values '{":gen":{"S":"Sci-Fi"}}' \
    --page-size 100  \
    --debug
```

Normalerweise AWS CLI verarbeitet der die Seitennummerierung automatisch. In diesem Beispiel begrenzt der AWS CLI `--page-size` Parameter jedoch die Anzahl der Elemente pro Seite. Der `--debug`-Parameter gibt Low-Level-Informationen zu Anforderungen und Antworten aus.

> **Note**
>
> Ihre Paginierungsergebnisse unterscheiden sich auch basierend auf den Eingabeparametern, die Sie übergeben.
>
> - Die Verwendung von `aws dynamodb scan --table-name Prices --max-items 1` gibt `NextToken` zurück.
> - Die Verwendung von `aws dynamodb scan --table-name Prices --limit 1` gibt `LastEvaluatedKey` zurück.
>
> Beachten Sie auch, dass insbesondere die Verwendung von `--starting-token` den Wert `NextToken` erfordert.

Wenn Sie das Beispiel ausführen, sieht die erste Antwort von DynamoDB ungefähr folgendermaßen aus.

```
2017-07-07 12:19:14,389 - MainThread - botocore.parsers - DEBUG - Response body:
b'{"Count":7,"Items":[{"title":{"S":"Monster on the Campus"}},{"title":{"S":"+1"}},
{"title":{"S":"100 Degrees Below Zero"}},{"title":{"S":"About Time"}},{"title":
{"S":"After Earth"}},
{"title":{"S":"Age of Dinosaurs"}},{"title":{"S":"Cloudy with a Chance of Meatballs
 2"}}],
"LastEvaluatedKey":{"year":{"N":"2013"},"title":{"S":"Curse of
 Chucky"}},"ScannedCount":100}'
```

Der `LastEvaluatedKey` in der Antwort gibt an, dass nicht alle Elemente abgerufen wurden. Das sendet AWS CLI dann eine weitere `Scan` Anfrage an DynamoDB. Dieses Anforderungs- und Antwortmuster wird bis zur endgültigen Antwort fortgesetzt.

```
2017-07-07 12:19:17,830 - MainThread - botocore.parsers - DEBUG - Response body:
```

```
b'{"Count":1,"Items":[{"title":{"S":"WarGames"}}],"ScannedCount":6}'
```

Das Fehlen von `LastEvaluatedKey` gibt an, dass keine abrufbaren Elemente mehr vorhanden sind.

> **Note**
>
> Sie AWS SDKs verarbeiten DynamoDB-Antworten auf niedriger Ebene (einschließlich des Vorhandenseins oder Fehlens von`LastEvaluatedKey`) und bieten verschiedene Abstraktionen für die Paginierung von Ergebnissen. `Scan` Beispielsweise stellt SDK für die Java-Dokumentschnittstelle `java.util.Iterator`-Unterstützung bereit, sodass Sie die Ergebnisse nacheinander durchgehen können.
> Codebeispiele in verschiedenen Programmiersprachen finden Sie im Amazon-DynamoDB-Handbuch für erste Schritte und in der AWS SDK-Dokumentation für Ihre Sprache.

Zählen der Elemente in den Ergebnissen

Zusätzlich zu den Elementen, die mit Ihren Kriterien übereinstimmen, enthält die `Scan`-Antwort noch die folgenden Elemente:

- `ScannedCount` – Anzahl der ausgewerteten Elemente vor jedem `ScanFilter` angewendet wird. Ein hoher `ScannedCount`-Wert mit wenigen oder gar keinen `Count`-Ergebnissen weist auf eine ineffiziente `Scan`-Operation hin. Wenn Sie keinen Filter in der Anforderung verwendet haben, haben `ScannedCount` und `Count` denselben Wert.
- `Count` – die Anzahl der Elemente, die verbleiben, nachdem ein Filterausdruck (falls vorhanden) angewendet wurde.

> **Note**
>
> Wenn Sie keinen Filterausdruck verwenden, haben `ScannedCount` und `Count` denselben Wert.

Wenn der `Scan`-Ergebnissatz größer als 1 MB ist, repräsentieren `ScannedCount` und `Count` nur eine Teilmenge der Gesamtelemente. Sie müssen mehrere `Scan`-Operationen ausführen, um alle Ergebnisse abzurufen (siehe Paginierung der Ergebnisse).

Jede `Scan`-Antwort enthält die `ScannedCount` und `Count` für die Elemente, die von der betreffenden `Scan`-Anforderung verarbeitet wurden. Um eine Gesamtsumme für alle `Scan`-Anforderungen zu erhalten, könnten Sie eine laufende Zählung von `ScannedCount` und `Count` vornehmen.

Durch die Abfrage verbrauchte Kapazitätseinheiten

Sie können `Scan` für jede Tabelle und jeden Sekundärindex ausführen. `Scan`-Operationen verbrauchen Lesekapazitätseinheiten wie folgt.

Wenn Sie **`Scan`** eine ...	DynamoDB verbraucht Lesekapazitätseinheiten von ...
Tabelle	Die bereitgestellte Lesekapazität der Tabelle.
Globaler sekundärer Index	Die bereitgestellte Lesekapazität des Index.
Lokaler sekundärer Index	Die bereitgestellte Lesekapazität der Basistabelle.

> **Note**
>
> Kontoübergreifender Zugriff für sekundäre Indexscanvorgänge wird derzeit nicht für ressourcenbasierte Richtlinien unterstützt.

Standardmäßig gibt eine `Scan`-Operation keine Daten über seinen Lesekapazitätsverbrauch zurück. Jedoch können Sie den `ReturnConsumedCapacity`-Parameter in einer `Scan`-Anforderung angeben, um diese Information zu erhalten. Folgende sind die gültigen Einstellungen für `ReturnConsumedCapacity`:

- `NONE` – Es werden keine verbrauchten Kapazitätsdaten zurückgegeben. (Dies ist die Standardeinstellung.)
- `TOTAL` – Die Antwort enthält die zusammengefasste Anzahl der verbrauchten Lesekapazitätseinheiten.

- `INDEXES` – Die Antwort zeigt die zusammengefasste Anzahl der verbrauchten Lesekapazitätseinheiten, zusammen mit der verbrauchten Kapazität für jede aufgerufene Tabelle und jeden aufgerufenen Index, an.

DynamoDB berechnet die Anzahl von verbrauchten Lesekapazitätseinheiten basierend auf der Anzahl und Größe der Elemente, nicht der Menge der Daten, die an eine Anwendung zurückgegeben werden. Aus diesem Grund bleibt die Anzahl der verbrauchten Kapazitätseinheiten gleich, unabhängig davon, ob Sie alle Attribute (das Standardverhalten) oder nur einige von ihnen anfordern (mithilfe eines Projektionsausdrucks). Die Anzahl ist auch gleich, unabhängig davon, ob Sie einen Filterausdruck verwenden oder nicht. `Scan` verbraucht eine Mindestlesekapazitätseinheit für einen strikt konsistenten Lesevorgang pro Sekunde oder zwei letztendlich konsistente Lesevorgänge pro Sekunde für ein Element von bis zu 4 KB. Wenn Sie ein Element lesen möchten, das größer als 4 KB ist, benötigt DynamoDB zusätzliche Leseanforderungseinheiten. Bei leeren Tabellen und sehr großen Tabellen mit einer geringen Anzahl an Partitionsschlüsseln können zusätzliche RCUs Gebühren anfallen, die über die Menge der gescannten Daten hinausgehen. Damit werden die Kosten für die Bearbeitung der `Scan`-Anfrage gedeckt, auch wenn keine Daten vorhanden sind.

Lesekonsistenz für Scan

Eine `Scan`-Operation führt standardmäßig lEventually Consistent-Lesevorgänge aus. Dies bedeutet, dass die `Scan`-Ergebnisse Änderungen aufgrund kürzlich abgeschlossener `PutItem` oder `UpdateItem`-Operationen, nicht wiedergeben könnten. Weitere Informationen finden Sie unter DynamoDB-Lesekonsistenz.

Wenn Sie Strongly Consistent-Lesevorgänge benötigen, setzen Sie zum Zeitpunkt, an dem der `Scan` beginnt, den `ConsistentRead`-Parameter in der `Scan`-Anforderung auf `true`. Dadurch wird sichergestellt, dass alle Schreibvorgänge, die vor `Scan`-Beginn abgeschlossen wurden, in die `Scan`-Antwort aufgenommen werden.

Die Festlegung von `ConsistentRead` für `true` kann beim Backup von Tabellen oder in Replikationsszenarien mit DynamoDB Streams hilfreich sein. Sie verwenden zuerst `Scan`, während `ConsistentRead` auf "true" eingestellt ist, um eine konsistente Kopie der Daten in der Tabelle zu erhalten. Während der `Scan`-Operation zeichnet DynamoDB Streams alle zusätzlichen Schreibaktivitäten auf, die in der Tabelle ausgeführt werden. Nach Abschluss der `Scan` können Sie die Schreibaktivität vom Stream auf die Tabelle anwenden.

> **Note**
>
> Beachten Sie, dass eine `Scan`-Operation, bei der `ConsistentRead` auf `true` festgelegt ist, doppelt so viele Lesekapazitätseinheiten verbraucht wie mit dem `ConsistentRead`-Standardwert (`false`).

Parallele Scans

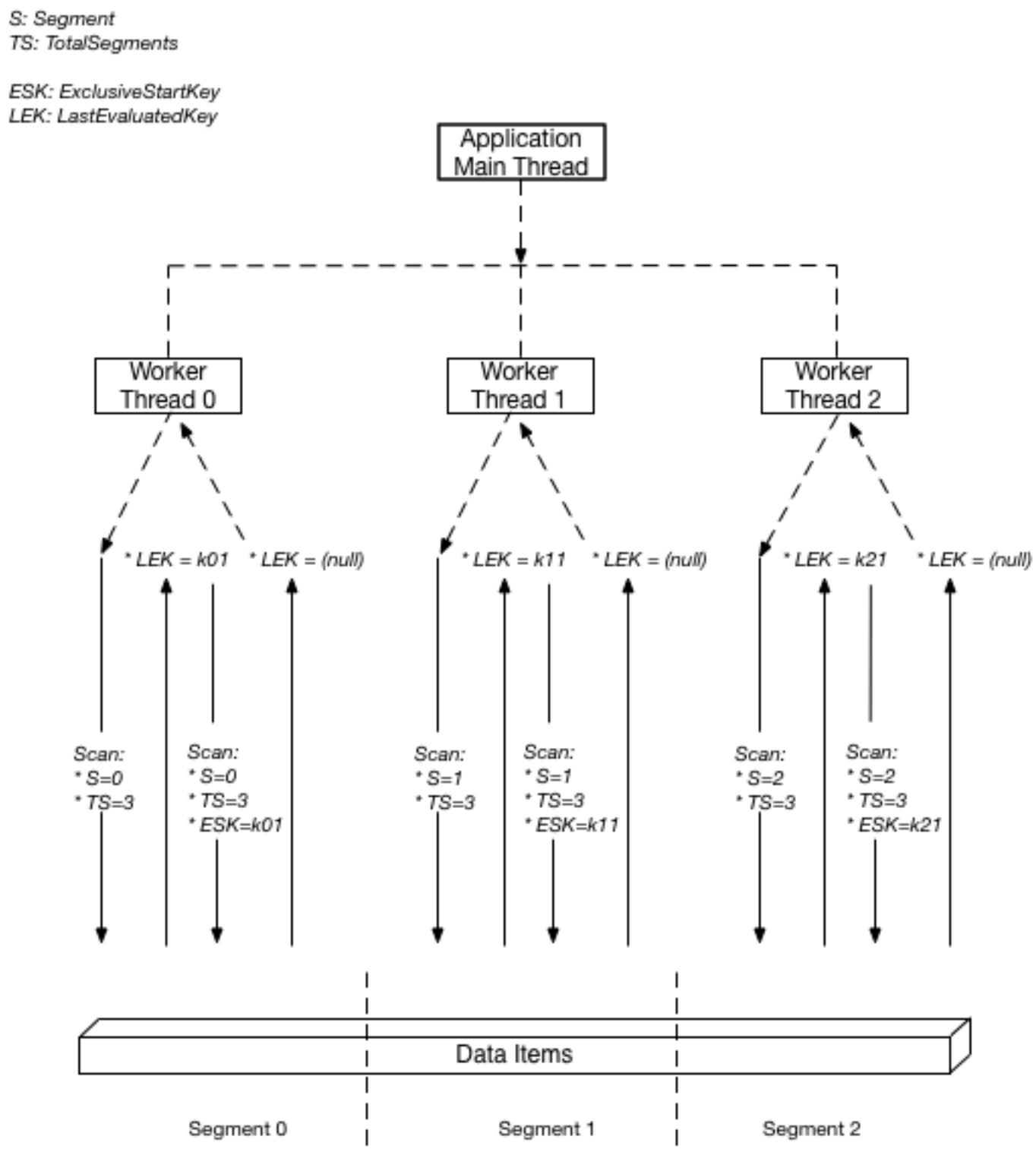
Standardmäßig verarbeitet die `Scan`-Operation Daten sequenziell. Amazon DynamoDB gibt Daten in Schritten von 1 MB an die Anwendung zurück, und eine Anwendung führt zusätzliche `Scan`-Operationen aus, um die nächsten 1 MB Daten abzurufen.

Je größer die gescannte Tabelle oder der Index, desto mehr Zeit benötigt `Scan` um zu beenden. Außerdem kann eine sequenzielle `Scan` möglicherweise die bereitgestellte Lesedurchsatzkapazität nicht immer vollständig nutzen: Auch wenn DynamoDB die Daten einer großen Tabelle über mehrere physische Partitionen verteilt, kann eine `Scan`-Operation nur jeweils eine Partition lesen. Aus diesem Grund wird der Durchsatz einer `Scan`-Operation durch den maximalen Durchsatz einer einzelnen Partition beschränkt.

Um diese Probleme zu beheben, kann die `Scan`-Operation eine Tabelle oder einen sekundären Index in mehrere Segmente logisch unterteilen, wobei mehrere Anwendungs-Worker die Segmente parallel scannen. Jeder Worker kann ein Thread (in Programmiersprachen, die Multithreading unterstützen) oder ein Betriebssystemprozess sein. Zum Durchführen eines parallelen Scans erstellt jeder Worker eine eigene `Scan`-Anforderung mit den folgenden Parametern:

- `Segment` – Ein Segment, das von einem bestimmten Worker gescannt werden soll. Jeder Worker sollte einen anderen Wert für `Segment` verwenden.
- `TotalSegments` – Die Gesamtanzahl der Segmente für den parallelen Scan. Dieser Wert muss mit der Anzahl der Worker übereinstimmen, die Ihre Anwendung verwenden wird.

Das folgende Diagramm zeigt, wie eine Multithread-Anwendung eine parallele `Scan`-Operation mit drei Grad Parallelität ausführt.



In diesem Diagramm ruft die Anwendung drei Threads auf und weist jedem Thread eine Zahl zu. (Segmente sind nullbasiert, sodass die erste Zahl immer 0 ist.) Jeder Thread erstellt eine `Scan`-Anforderung, mit der `Segment` auf die zugewiesene Zahl und `TotalSegments` auf 3 eingestellt wird.

Jeder Thread scannt ein zugewiesenes Segment, indem 1 MB Daten abgerufen werden, und gibt die Daten an den Haupt-Thread der Anwendung zurück.

DynamoDB weist Elemente Segmenten zu, indem es eine Hash-Funktion auf den Partitionsschlüssel jedes Elements anwendet. Bei einem bestimmten `TotalSegments` Wert werden alle Elemente mit demselben Partitionsschlüssel immer demselben zugewiesen. `Segment` Das bedeutet, dass in einer Tabelle, in der sich Element 1, Element 2 und Element 3 alle teilen `pk="account#123"` (aber unterschiedliche Sortierschlüssel haben), diese Elemente von demselben Worker verarbeitet werden, unabhängig von den Sortierschlüsselwerten oder der Größe der Elementsammlung.

Da die Segmentzuweisung ausschließlich auf dem Hash des Partitionsschlüssels basiert, können Segmente ungleichmäßig verteilt sein. Einige Segmente enthalten möglicherweise keine Elemente, während andere viele Partitionsschlüssel mit großen Elementsammlungen enthalten können. Daher garantiert eine Erhöhung der Gesamtzahl der Segmente keine schnellere Scanleistung, insbesondere wenn die Partitionsschlüssel nicht gleichmäßig über den Schlüsselraum verteilt sind.

Die Werte für `Segment` und `TotalSegments` gelten für einzelne `Scan`-Anforderungen. Sie können jederzeit andere Werte verwenden. Sie müssen ggf. mit diesen Werten und der Anzahl der verwendeten Worker experimentieren, bis Ihre Anwendung eine optimale Leistung erzielt.

> **Note**
>
> Ein paralleler Scan mit einer großen Anzahl von Workern kann leicht den gesamten bereitgestellten Durchsatz für die Tabelle oder den Index, die bzw. der gescannt wird, verbrauchen. Es ist am besten, solche Scans zu verhindern, wenn für die Tabelle oder den Index auch viele Lese- und Schreibaktivitäten von anderen Anwendungen erfolgen. Zur Steuerung der Datenmenge, die pro Anforderung zurückgegeben wird, verwenden Sie den Parameter `Limit`. Auf diese Weise können Sie Situationen verhindern, in denen ein Worker den gesamten bereitgestellten Durchsatz auf Kosten aller anderen Worker verbraucht.

PartiQL – Eine SQL-kompatible Abfragesprache für Amazon DynamoDB

Amazon DynamoDB unterstützt PartiQL, eine SQL-kompatible Abfragesprache, um Daten in Amazon DynamoDB auszuwählen, einzufügen, zu aktualisieren und zu löschen. Mit PartiQL können Sie problemlos mit DynamoDB-Tabellen interagieren und Ad-hoc-Abfragen mithilfe von NoSQL Workbench und DynamoDB for PartiQL ausführen. AWS-Managementkonsole AWS Command Line Interface APIs

PartiQL-Operationen bieten die gleiche Verfügbarkeit, Latenz und Performance wie die anderen DynamoDB-Datenebenen-Operationen.

In den folgenden Abschnitten wird die DynamoDB-Implementierung von PartiQL beschrieben.

Themen

- Was ist PartiQL?
- PartiQL in Amazon DynamoDB
- Erste Schritte mit PartiQL für DynamoDB
- PartiQL-Datentypen für DynamoDB
- PartiQL-Anweisungen für DynamoDB
- Verwenden von PartiQL-Funktionen mit DynamoDB
- PartiQL-Arithmetik-, Vergleichs- und logische Operatoren für DynamoDB
- Ausführen von Transaktionen mit PartiQL für DynamoDB
- Ausführen von Batchoperationen mit PartiQL für DynamoDB
- IAM-Sicherheitsrichtlinien mit PartiQL für DynamoDB

Was ist PartiQL?

PartiQL bietet SQL-kompatiblen Abfragezugriff über mehrere Datenspeicher mit strukturierten, halbstrukturierten und verschachtelten Daten hinweg. Es ist bei Amazon weit verbreitet und ist jetzt als Teil vieler AWS Dienste verfügbar, einschließlich DynamoDB.

Die PartiQL-Spezifikation und ein Tutorial zur Core-Abfragesprache finden Sie in der PartiQL-Dokumentation.

> **Note**
>
> - Amazon DynamoDB unterstützt eine Teilmenge der PartiQL-Abfragesprache.
> - Amazon DynamoDB unterstützt das Amazon-Ion-Datenformat oder Amazon-Ion-Literale nicht.

PartiQL in Amazon DynamoDB

Zum Ausführen von PartiQL-Abfragen in DynamoDB können Sie Folgendes verwenden:

- Die DynamoDB-Konsole
- NoSQL Workbench
- Das AWS Command Line Interface ()AWS CLI
- Die DynamoDB APIs

Weitere Informationen zum Verwenden dieser Methoden für den Zugriff auf DynamoDB finden Sie unter Zugreifen auf DynamoDB.

Erste Schritte mit PartiQL für DynamoDB

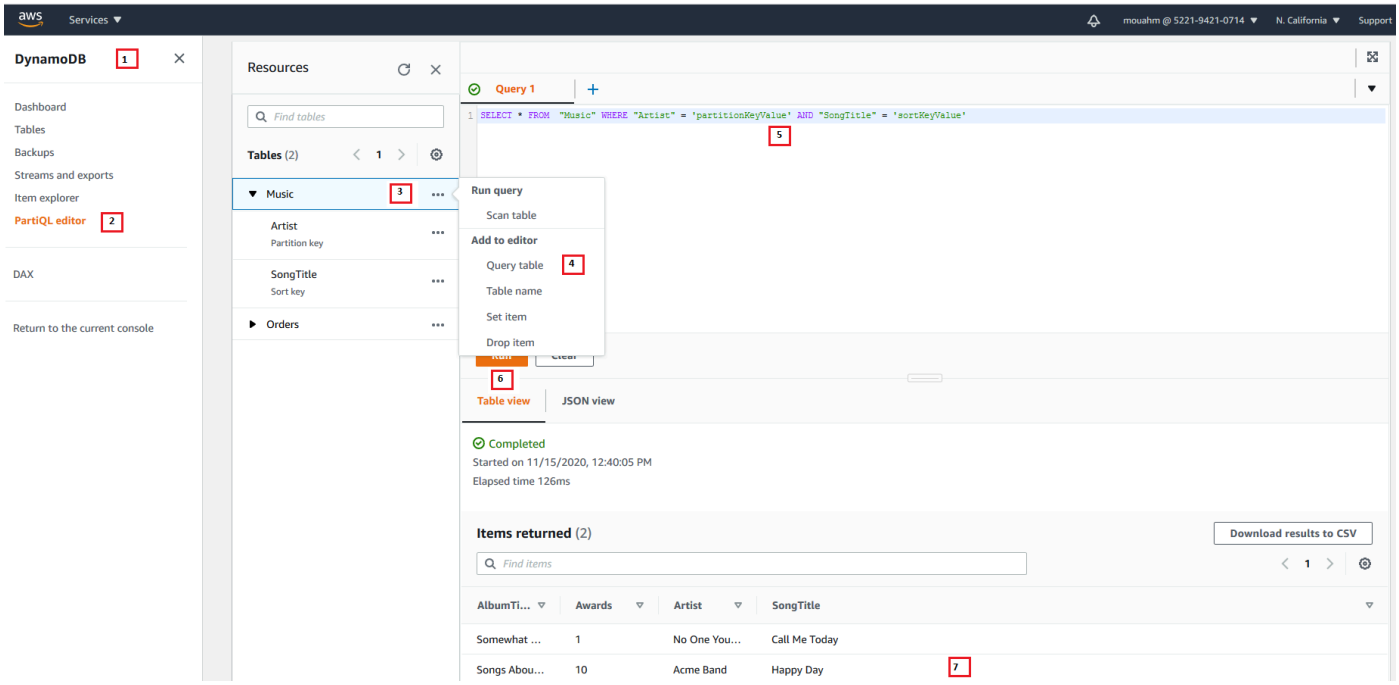
In diesem Abschnitt wird beschrieben, wie PartiQL für DynamoDB von der Amazon DynamoDB DynamoDB-Konsole, der AWS Command Line Interface ()AWS CLI und DynamoDB aus verwendet wird. APIs

In den folgenden Beispielen ist die DynamoDB-Tabelle, die im Tutorial Erste Schritte mit DynamoDB definiert ist, eine Voraussetzung.

Informationen zur Verwendung der DynamoDB-Konsole oder DynamoDB für den Zugriff auf DynamoDB finden Sie unter APIs Accessing DynamoDB. AWS Command Line Interface

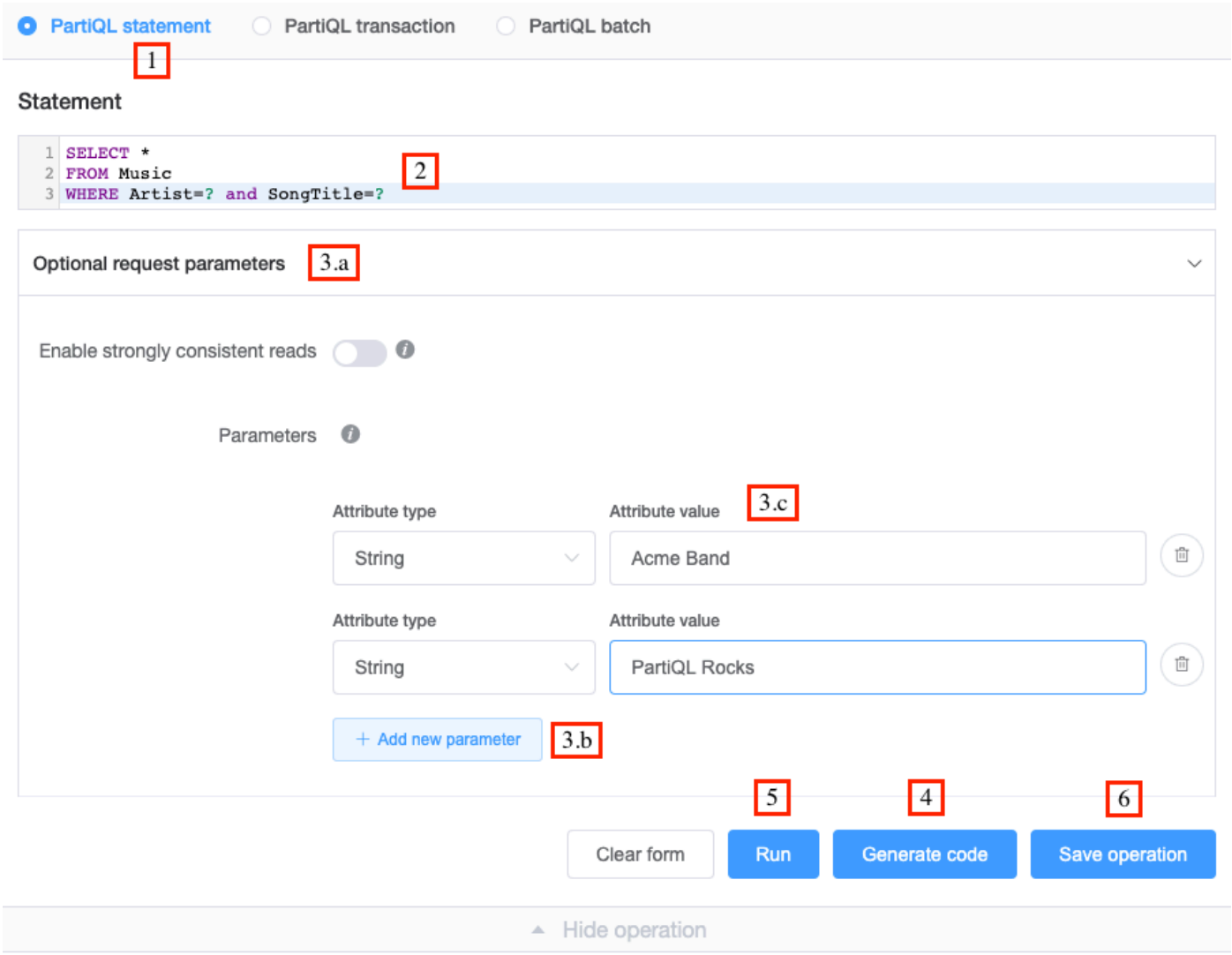
Um die NoSQL Workbench herunterzuladen und zu verwenden, um PartiQL for DynamoDB (PartiQL für DynamoDB) Anweisungen zu erstellen, wählen Sie PartiQL operations (PartiQL-Operationen) in der oberen rechten Ecke der NoSQL Workbench für DynamoDB Operation Builder.

Console



1. Melden Sie sich bei der an AWS-Managementkonsole und öffnen Sie die DynamoDB-Konsole unter. https://console.aws.amazon.com/dynamodb/
2. Wählen Sie im Navigationsbereich auf der linken Seite der Konsole PartiQL editor (PartiQL-Editor) aus.
3. Wählen Sie die Tabelle Music (Musik).
4. Klicken Sie auf die Query table (Abfragetabelle). Diese Aktion generiert eine Abfrage, die nicht zu einem vollständigen Tabellenscan führt.
5. ErsetzenSie `partitionKeyValue` mit dem Zeichenfolgen-Wert `Acme Band`. ErsetzenSie `sortKeyValue` mit dem Zeichenfolgen-Wert `Happy Day`.
6. Wählen Sie die Schaltfläche Run (Ausführen) aus.
7. Sie können die Ergebnisse der Abfrage anzeigen, indem Sie die Schaltflächen Table view (Tabellenansicht) oder JSON view (JSON-Ansicht) auswählen.

NoSQL workbench



1. Klicken Sie auf PartiQL statement (PartiQL-Anweisung).
2. Geben Sie die folgende PartiQL-SELECT-Anweisung ein

```
SELECT *
FROM Music
WHERE Artist=? and SongTitle=?
```

3. So geben Sie einen Wert für den `Artist`- und `SongTitle`-Parameter ein:

   a. Klicken Sie auf Optional request parameter (Optionale Anfrageparameter).

   b. Klicken Sie auf Add new parameters (Fügen Sie neue Parameter hinzu).

   c. Wählen Sie den Attributtyp string (Zeichenfolge) und den Wert `Acme Band`.

d. Wiederholen Sie die Schritte b und c, und wählen Sie den Typ string (Zeichenfolge) und den Wert `PartiQL Rocks`.

4. Falls Sie Code generieren möchten, wählen Sie Generate code (Code generieren) aus.

   Wählen Sie in den angezeigten Tabs Ihre gewünschte Sprache aus. Sie können diesen Code jetzt kopieren und in Ihrer Anwendung verwenden.

5. Falls die Operation sofort ausgeführt werden soll, wählen Sie Run (Ausführen).

AWS CLI

1. Erstellen Sie mit der INSERT-PartiQL-Anweisung ein Element in der Tabelle `Music`.

```
aws dynamodb execute-statement --statement "INSERT INTO Music  \
         VALUE  \
         {'Artist':'Acme Band','SongTitle':'PartiQL Rocks'}"
```

2. Rufen Sie mit der SELECT-PartiQL-Anweisung ein Element aus der Musiktabelle ab.

```
aws dynamodb execute-statement --statement "SELECT * FROM Music   \
                                            WHERE Artist='Acme Band' AND
 SongTitle='PartiQL Rocks'"
```

3. Aktualisieren Sie ein Element in der Tabelle `Music` mit der UPDATE-PartiQL-Anweisung.

```
aws dynamodb execute-statement --statement "UPDATE Music  \
                                            SET AwardsWon=1  \
                                            SET AwardDetail={'Grammys':[2020,
 2018]}  \
                                            WHERE Artist='Acme Band' AND
 SongTitle='PartiQL Rocks'"
```

   Fügen Sie einen Listenwert für ein Element in der `Music`-Tabelle ein

```
aws dynamodb execute-statement --statement "UPDATE Music  \
                                            SET AwardDetail.Grammys
 =list_append(AwardDetail.Grammys,[2016])  \
                                            WHERE Artist='Acme Band' AND
 SongTitle='PartiQL Rocks'"
```

Entfernen Sie einen Listenwert für ein Element in der `Music`-Tabelle

```
aws dynamodb execute-statement --statement "UPDATE Music  \
                                            REMOVE AwardDetail.Grammys[2]  \
                                            WHERE Artist='Acme Band' AND
 SongTitle='PartiQL Rocks'"
```

Fügen Sie ein neues Kartenelement für ein Element in der `Music`-Tabelle ein

```
aws dynamodb execute-statement --statement "UPDATE Music  \
                                            SET AwardDetail.BillBoard=[2020]  \
                                            WHERE Artist='Acme Band' AND
 SongTitle='PartiQL Rocks'"
```

Fügen Sie ein neues Zeichenfolgensatzattribut für ein Element in der `Music`-Tabelle hinzu.

```
aws dynamodb execute-statement --statement "UPDATE Music  \
                                            SET BandMembers =<<'member1',
 'member2'>>  \
                                            WHERE Artist='Acme Band' AND
 SongTitle='PartiQL Rocks'"
```

Aktualisieren Sie ein neues Zeichenfolgensatzattribut für ein Element in der `Music`-Tabelle-

```
aws dynamodb execute-statement --statement "UPDATE Music  \
                                            SET BandMembers
 =set_add(BandMembers, <<'newmember'>>)  \
                                            WHERE Artist='Acme Band' AND
 SongTitle='PartiQL Rocks'"
```

4. Löschen Sie ein Element aus dem`Music`-Tabelle mit der DELETE-PartiQL-Anweisung.

```
aws dynamodb execute-statement --statement "DELETE  FROM Music  \
    WHERE Artist='Acme Band' AND SongTitle='PartiQL Rocks'"
```

Java

```
import java.util.ArrayList;
import java.util.List;
```

```
import com.amazonaws.AmazonClientException;
import com.amazonaws.AmazonServiceException;
import software.amazon.dynamodb.AmazonDynamoDB;
import software.amazon.dynamodb.AmazonDynamoDBClientBuilder;
import software.amazon.dynamodb.model.AttributeValue;
import software.amazon.dynamodb.model.ConditionalCheckFailedException;
import software.amazon.dynamodb.model.ExecuteStatementRequest;
import software.amazon.dynamodb.model.ExecuteStatementResult;
import software.amazon.dynamodb.model.InternalServerErrorException;
import software.amazon.dynamodb.model.ItemCollectionSizeLimitExceededException;
import software.amazon.dynamodb.model.ProvisionedThroughputExceededException;
import software.amazon.dynamodb.model.RequestLimitExceededException;
import software.amazon.dynamodb.model.ResourceNotFoundException;
import software.amazon.dynamodb.model.TransactionConflictException;

public class DynamoDBPartiQGettingStarted {

    public static void main(String[] args) {
        // Create the DynamoDB Client with the region you want
        AmazonDynamoDB dynamoDB = createDynamoDbClient("us-west-1");

        try {
            // Create ExecuteStatementRequest
            ExecuteStatementRequest executeStatementRequest = new
 ExecuteStatementRequest();
            List<AttributeValue> parameters= getPartiQLParameters();

            //Create an item in the Music table using the INSERT PartiQL statement
            processResults(executeStatementRequest(dynamoDB, "INSERT INTO Music
 value {'Artist':?,'SongTitle':?}", parameters));

            //Retrieve an item from the Music table using the SELECT PartiQL
 statement.
            processResults(executeStatementRequest(dynamoDB, "SELECT * FROM Music
 where Artist=? and SongTitle=?", parameters));

            //Update an item in the Music table using the UPDATE PartiQL statement.
            processResults(executeStatementRequest(dynamoDB, "UPDATE Music
 SET AwardsWon=1 SET AwardDetail={'Grammys':[2020, 2018]}  where Artist=? and
 SongTitle=?", parameters));

            //Add a list value for an item in the Music table.
```

```
            processResults(executeStatementRequest(dynamoDB, "UPDATE Music SET
AwardDetail.Grammys =list_append(AwardDetail.Grammys,[2016])  where Artist=? and
SongTitle=?", parameters));

            //Remove a list value for an item in the Music table.
            processResults(executeStatementRequest(dynamoDB, "UPDATE Music REMOVE
AwardDetail.Grammys[2]   where Artist=? and SongTitle=?", parameters));

            //Add a new map member for an item in the Music table.
            processResults(executeStatementRequest(dynamoDB, "UPDATE Music set
AwardDetail.BillBoard=[2020] where Artist=? and SongTitle=?", parameters));

            //Add a new string set attribute for an item in the Music table.
            processResults(executeStatementRequest(dynamoDB, "UPDATE Music
SET BandMembers =<<'member1', 'member2'>> where Artist=? and SongTitle=?",
parameters));

            //update a string set attribute for an item in the Music table.
            processResults(executeStatementRequest(dynamoDB, "UPDATE Music SET
BandMembers =set_add(BandMembers, <<'newmember'>>) where Artist=? and SongTitle=?",
parameters));

            //Retrieve an item from the Music table using the SELECT PartiQL
statement.
            processResults(executeStatementRequest(dynamoDB, "SELECT * FROM Music
where Artist=? and SongTitle=?", parameters));

            //delete an item from the Music Table
            processResults(executeStatementRequest(dynamoDB, "DELETE  FROM Music
where Artist=? and SongTitle=?", parameters));
        } catch (Exception e) {
            handleExecuteStatementErrors(e);
        }
    }

    private static AmazonDynamoDB createDynamoDbClient(String region) {
        return AmazonDynamoDBClientBuilder.standard().withRegion(region).build();
    }

    private static List<AttributeValue> getPartiQLParameters() {
        List<AttributeValue> parameters = new ArrayList<AttributeValue>();
        parameters.add(new AttributeValue("Acme Band"));
        parameters.add(new AttributeValue("PartiQL Rocks"));
        return parameters;
```

```
   }

   private static ExecuteStatementResult executeStatementRequest(AmazonDynamoDB
client, String statement, List<AttributeValue> parameters ) {
       ExecuteStatementRequest request = new ExecuteStatementRequest();
       request.setStatement(statement);
       request.setParameters(parameters);
       return client.executeStatement(request);
   }

   private static void processResults(ExecuteStatementResult
executeStatementResult) {
       System.out.println("ExecuteStatement successful: "+
executeStatementResult.toString());

   }

   // Handles errors during ExecuteStatement execution. Use recommendations in
error messages below to add error handling specific to
   // your application use-case.
   private static void handleExecuteStatementErrors(Exception exception) {
       try {
           throw exception;
       } catch (ConditionalCheckFailedException ccfe) {
           System.out.println("Condition check specified in the operation failed,
review and update the condition " +
                                       "check before retrying. Error: " +
ccfe.getErrorMessage());
       } catch (TransactionConflictException tce) {
           System.out.println("Operation was rejected because there is an ongoing
transaction for the item, generally " +
                                       "safe to retry with exponential back-off.
Error: " + tce.getErrorMessage());
       } catch (ItemCollectionSizeLimitExceededException icslee) {
           System.out.println("An item collection is too large, you\'re using Local
Secondary Index and exceeded " +
                                       "size limit of items per
partition key. Consider using Global Secondary Index instead. Error: " +
icslee.getErrorMessage());
       } catch (Exception e) {
           handleCommonErrors(e);
       }
   }
```

```
    private static void handleCommonErrors(Exception exception) {
        try {
            throw exception;
        } catch (InternalServerErrorException isee) {
            System.out.println("Internal Server Error, generally safe to retry with
 exponential back-off. Error: " + isee.getErrorMessage());
        } catch (RequestLimitExceededException rlee) {
            System.out.println("Throughput exceeds the current throughput limit for
 your account, increase account level throughput before " +
                                        "retrying. Error: " +
 rlee.getErrorMessage());
        } catch (ProvisionedThroughputExceededException ptee) {
            System.out.println("Request rate is too high. If you're using a custom
 retry strategy make sure to retry with exponential back-off. " +
                                        "Otherwise consider reducing frequency of
 requests or increasing provisioned capacity for your table or secondary index.
 Error: " +
                                        ptee.getErrorMessage());
        } catch (ResourceNotFoundException rnfe) {
            System.out.println("One of the tables was not found, verify table exists
 before retrying. Error: " + rnfe.getErrorMessage());
        } catch (AmazonServiceException ase) {
            System.out.println("An AmazonServiceException occurred, indicates that
 the request was correctly transmitted to the DynamoDB " +
                                        "service, but for some reason, the service
 was not able to process it, and returned an error response instead. Investigate and
 " +
                                        "configure retry strategy. Error type: " +
 ase.getErrorType() + ". Error message: " + ase.getErrorMessage());
        } catch (AmazonClientException ace) {
            System.out.println("An AmazonClientException occurred, indicates that
 the client was unable to get a response from DynamoDB " +
                                        "service, or the client was unable to parse
 the response from the service. Investigate and configure retry strategy. "+
                                        "Error: " + ace.getMessage());
        } catch (Exception e) {
            System.out.println("An exception occurred, investigate and configure
 retry strategy. Error: " + e.getMessage());
        }
    }

}
```

Verwenden von parametrisierten Anweisungen

Anstatt Werte direkt in eine PartiQL-Anweisungszeichenfolge einzubetten, können Sie Fragezeichen-Platzhalter (?) verwenden und die Werte separat in das Feld eingeben. `Parameters` Jeder Wert `?` wird durch den entsprechenden Parameterwert ersetzt, und zwar in der Reihenfolge, in der er angegeben wurde.

Die Verwendung parametrisierter Anweisungen ist eine bewährte Methode, da sie die Anweisungsstruktur von den Datenwerten trennt, sodass Anweisungen leichter zu lesen und wiederzuverwenden sind. Außerdem müssen Attributwerte in der Anweisungszeichenfolge nicht manuell formatiert und maskiert werden.

Parametrisierte Anweisungen werden in `ExecuteStatement`, `BatchExecuteStatement`, und `ExecuteTransaction` -Operationen unterstützt.

In den folgenden Beispielen wird mithilfe parametrisierter Werte für den Partitionsschlüssel und den Sortierschlüssel ein Element aus der `Music` Tabelle abgerufen.

AWS CLI parameterized

```
aws dynamodb execute-statement \
    --statement "SELECT * FROM \"Music\" WHERE Artist=? AND SongTitle=?" \
    --parameters '[{"S": "Acme Band"}, {"S": "PartiQL Rocks"}]'
```

Java parameterized

```
List<AttributeValue> parameters = new ArrayList<>();
parameters.add(new AttributeValue("Acme Band"));
parameters.add(new AttributeValue("PartiQL Rocks"));

ExecuteStatementRequest request = new ExecuteStatementRequest()
    .withStatement("SELECT * FROM Music WHERE Artist=? AND SongTitle=?")
    .withParameters(parameters);

ExecuteStatementResult result = dynamoDB.executeStatement(request);
```

Python parameterized

```
response = dynamodb_client.execute_statement(
    Statement="SELECT * FROM Music WHERE Artist=? AND SongTitle=?",
```

```
    Parameters=[
        {'S': 'Acme Band'},
        {'S': 'PartiQL Rocks'}
    ]
)
```

> **Note**
>
> Das Java-Beispiel im vorherigen Abschnitt „Erste Schritte“ verwendet durchweg parametrisierte Anweisungen. Die `getPartiQLParameters()` Methode erstellt die Parameterliste, und jede Anweisung verwendet `?` Platzhalter anstelle von Inline-Werten.

PartiQL-Datentypen für DynamoDB

In der folgenden Tabelle sind die Datentypen aufgeführt, die Sie mit PartiQL für DynamoDB verwenden können.

DynamoDB-Datentyp	PartiQL-Repräsentation	Hinweise
`Boolean`	TRUE | FALSE	Die Groß- und Kleinschr eibung muss nicht berücksic htigt werden.
`Binary`	k. A.	Nur über Code unterstützt.
`List`	[Wert1, Wert2,...]	Es gibt keine Einschrän kungen hinsichtlich der Datentypen, die in einem List-Typ gespeichert werden können, und die Elemente in einer List müssen nicht vom gleichen Typ sein.
`Map`	{'name': Wert}	Es gibt keine Einschrän kungen hinsichtlich der Datentypen, die in einem Map-Typ gespeichert werden

DynamoDB-Datentyp	PartiQL-Repräsentation	Hinweise
		können, und die Elemente in einer Map müssen nicht vom gleichen Typ sein.
`Null`	NULL	Die Groß- und Kleinschreibung muss nicht berücksichtigt werden.
`Number`	1, 1,0, 1e0	Zahlen können positiv, negativ oder Null sein. Zahlen können auf 38 Stellen genau sein.
`Number Set`	<number1, number2><>	Die Elemente in einem Zahlensatz müssen vom Typ Number sein.
`String Set`	<<'string1', 'string2'>>	Die Elemente in einem Zeichenfolgensatz müssen vom Typ String sein.
`String`	Zeichenfolgenwert	Zum Angeben von Zeichenfolgen-Werten müssen einfache Anführungszeichen verwendet werden.

Beispiele

Die folgende Anweisung veranschaulicht, wie die folgenden Datentypen eingefügt werden: `String`, `Number`, `Map`, `List`, `Number Set` und `String Set`.

```
INSERT INTO TypesTable value {'primarykey':'1',
'NumberType':1,
'MapType' : {'entryname1': 'value', 'entryname2': 4},
'ListType': [1,'stringval'],
'NumberSetType':<<1,34,32,4.5>>,
'StringSetType':<<'stringval','stringval2'>>
}
```

Die folgende Anweisung veranschaulicht, wie neue Elemente in die `Map`, `List`, `Number Set` und `String Set`-Typen eingefügt werden und den Wert eines `Number`-Typ ändern.

```
UPDATE TypesTable
SET NumberType=NumberType + 100
SET MapType.NewMapEntry=[2020, 'stringvalue', 2.4]
SET ListType = LIST_APPEND(ListType, [4, <<'string1', 'string2'>>])
SET NumberSetType= SET_ADD(NumberSetType, <<345, 48.4>>)
SET StringSetType = SET_ADD(StringSetType, <<'stringsetvalue1', 'stringsetvalue2'>>)
WHERE primarykey='1'
```

Die folgende Anweisung veranschaulicht, wie neue Elemente aus den `Map`, `List`, `Number Set` und `String Set`-Typen entfernt werden und den Wert eines `Number`-Typ ändern.

```
UPDATE TypesTable
SET NumberType=NumberType - 1
REMOVE ListType[1]
REMOVE MapType.NewMapEntry
SET NumberSetType = SET_DELETE( NumberSetType, <<345>>)
SET StringSetType = SET_DELETE( StringSetType, <<'stringsetvalue1'>>)
WHERE primarykey='1'
```

Weitere Informationen finden Sie unter DynamoDB-Datentypen.

PartiQL-Anweisungen für DynamoDB

Amazon DynamoDB unterstützt die folgenden PartiQL-Anweisungen.

> **Note**
>
> DynamoDB unterstützt nicht alle PartiQL-Anweisungen.
> Diese Referenz enthält grundlegende Syntax- und Verwendungsbeispiele für PartiQL-Anweisungen, die Sie manuell mit dem AWS CLI oder APIs ausführen.

Data Manipulation Language (DML) ist die Gruppe von PartiQL-Anweisungen, die Sie zum Verwalten von Daten in DynamoDB-Tabellen verwenden. Sie verwenden DML-Anweisungen, um Daten in einer Tabelle hinzuzufügen, zu ändern oder zu löschen.

Die folgenden Anweisungen für DML- und Abfragesprachen werden unterstützt:

- PartiQL-Select-Anweisungen für DynamoDB
- Aktualisierungen für PartiQL-Anweisungen für DynamoDB
- PartiQL-Insert-Anweisungen für DynamoDB
- PartiQL-Delete-Anweisungen für DynamoDB

Ausführen von Transaktionen mit PartiQL für DynamoDB und Ausführen von Batchoperationen mit PartiQL für DynamoDB werden auch von PartiQL für DynamoDB unterstützt.

PartiQL-Select-Anweisungen für DynamoDB

Verwenden Sie die `SELECT`-Anweisung zum Abrufen von Daten aus einer Tabelle in Amazon DynamoDB.

Die Verwendung der `SELECT`-Anweisung kann zu einem vollständigen Tabellenscan führen, wenn in der WHERE-Klausel keine Gleichheitsbedingung oder IN-Bedingung mit einem Partitionsschlüssel angegeben ist. Die Scan-Operation untersucht jedes Element auf die angeforderten Werte und kann den bereitgestellten Durchsatz für eine große Tabelle oder Index in einer einzigen Operation verbrauchen.

Wenn Sie den vollständigen Tabellenscan in PartiQL vermeiden möchten, können Sie:

- Erstellen Sie Ihre `SELECT`-Anweisungen so, dass keine vollständigen Tabellenscans durchgeführt werden, indem Sie sicherstellen, dass die Bedingung der WHERE-Klausel entsprechend konfiguriert ist.
- Deaktivieren Sie vollständige Tabellenscans mithilfe der IAM-Richtlinie, die unter Beispiel: In PartiQL für DynamoDB Select-Anweisungen erlauben und vollständige Tabellenscan-Anweisungen verweigern im DynamoDB-Entwicklerhandbuch.

Weitere Informationen finden Sie unter Bewährte Methoden für das Abfragen und Scannen von Daten im DynamoDB-Entwicklerhandbuch.

Themen

- Syntax
- Parameters
- Beispiele

Syntax

```
SELECT expression  [, ...]
FROM table[.index]
[ WHERE condition ] [ [ORDER BY key [DESC|ASC] , ...]
```

Parameters

expression

(erforderlich) Eine Projektion aus dem *-Platzhalter oder eine Projektionsliste aus einem oder mehreren Attributnamen oder Dokumentpfaden aus dem Ergebnissatz. Ein Ausdruck kann aus Aufrufen an Verwenden von PartiQL-Funktionen mit DynamoDB oder Feldern bestehen, die von PartiQL-Arithmetik-, Vergleichs- und logische Operatoren für DynamoDB geändert werden.

table

(Erforderlich) Der abzufragende Tabellenname.

index

(Optional) Der Name des abzufragenden Indexes.

> **Note**
>
> Sie müssen dem Tabellennamen und dem Indexnamen doppelte Anführungszeichen hinzufügen, wenn Sie einen Index abfragen.
>
> ```
> SELECT *
> FROM "TableName"."IndexName"
> ```

condition

(Optional) Die Auswahlkriterien für die Abfrage.

> **Important**
>
> Um sicherzustellen, dass eine SELECT-Anweisung nicht zu einem vollständigen Tabellenscan führt, muss die WHERE-Klauselbedingung einen Partitionsschlüssel angeben. Verwenden Sie den Gleichheits- oder IN-Operator.

Angenommen, Sie haben eine `Orders`-Tabelle mit `OrderID`-Partitionsschlüssel und andere Nicht-Schlüsselattribute, einschließlich eines `Address`, würden die folgenden Anweisungen nicht zu einem vollständigen Tabellenscan führen:

```
SELECT *
FROM "Orders"
WHERE OrderID = 100

SELECT *
FROM "Orders"
WHERE OrderID = 100 and Address='some address'

SELECT *
FROM "Orders"
WHERE OrderID = 100 or OrderID = 200

SELECT *
FROM "Orders"
WHERE OrderID IN [100, 300, 234]
```

Folgende `SELECT`-Anweisungen führen jedoch zu einem vollständigen Tabellenscan:

```
SELECT *
FROM "Orders"
WHERE OrderID > 1

SELECT *
FROM "Orders"
WHERE Address='some address'

SELECT *
FROM "Orders"
WHERE OrderID = 100 OR Address='some address'
```

key

(Optional) Ein Hash-Schlüssel oder ein Sortierschlüssel, der zum Sortieren von ausgegebenen Ergebnissen verwendet werden soll. Die Standardreihenfolge ist aufsteigend (`ASC`) Geben Sie an.`DESC`, wenn die Ergebnisse in absteigender Reihenfolge neu abgestimmt werden sollen.

> **Note**
>
> Wenn Sie die `WHERE`-Klausel weglassen, werden alle Elemente in der Tabelle abgerufen.

Beispiele

Die folgende Abfrage gibt ein Element, falls vorhanden, aus der `Orders`-Tabelle zurück, indem der Partitionsschlüssel `OrderID` angegeben und der Gleichheitsoperator verwendet wird.

```
SELECT OrderID, Total
FROM "Orders"
WHERE OrderID = 1
```

Die folgende Abfrage gibt alle Elemente in der `Orders`-Tabelle mit einem bestimmten Partitionsschlüssel `OrderID` zurück, mithilde von Werten mit dem Operator OR.

```
SELECT OrderID, Total
FROM "Orders"
WHERE OrderID = 1 OR OrderID = 2
```

Die folgende Abfrage gibt alle Elemente in der `Orders`-Tabelle mit einem bestimmten Partitionsschlüssel `OrderID` zurück, mithilde von Werten mit dem Operator IN. Die zurückgegebenen Ergebnisse sind in absteigender Reihenfolge, basierend auf dem `OrderID`-Attributwert.

```
SELECT OrderID, Total
FROM "Orders"
WHERE OrderID IN [1, 2, 3] ORDER BY OrderID DESC
```

Die folgende Abfrage zeigt einen vollständigen Tabellenscan, der alle Elemente aus der `Orders`-Tabelle zurückgibt, deren `Total` größer als 500 ist, wobei `Total` ein Nicht-Schlüsselattribut ist.

```
SELECT OrderID, Total
FROM "Orders"
WHERE Total > 500
```

Die folgende Abfrage zeigt einen vollständigen Tabellenscan, der alle Elemente aus der `Orders`-Tabelle innerhalb eines bestimmten `Total`-Reihenfolgebereichs zurückgibt, wobei der IN-Operator und ein Nicht-Schlüsselattribut `Total` verwendet werden.

```
SELECT OrderID, Total
FROM "Orders"
WHERE Total IN [500, 600]
```

Die folgende Abfrage zeigt einen vollständigen Tabellenscan, der alle Elemente aus der `Orders`-Tabelle innerhalb eines bestimmten `Total`-Reihenfolgebereichs zurückgibt, wobei der BETWEEN-Operator und ein Nicht-Schlüsselattribut `Total` verwendet werden.

```
SELECT OrderID, Total
FROM "Orders"
WHERE Total BETWEEN 500 AND 600
```

Die folgende Abfrage gibt das erste Datum zurück, an dem ein Firestick-Gerät zum Überwachen verwendet wurde, indem der `CustomerID`-Partitionsschlüssel und der `MovieID`-Sortierschlüssel in der Bedingung der WHERE-Klausel angegeben und Dokumentpfade in der SELECT-Klausel verwendet werden.

```
SELECT Devices.FireStick.DateWatched[0]
FROM WatchList
WHERE CustomerID= 'C1' AND MovieID= 'M1'
```

Die folgende Abfrage zeigt einen vollständigen Tabellenscan, der die Liste der Elemente zurückgibt, bei denen ein Firestick-Gerät nach dem 24.12.19 zum ersten Mal unter Verwendung von Dokumentpfaden in der WHERE-Klausel Bedingung verwendet wurde.

```
SELECT Devices
FROM WatchList
WHERE Devices.FireStick.DateWatched[0] >= '12/24/19'
```

Aktualisierungen für PartiQL-Anweisungen für DynamoDB

Verwenden der `UPDATE`-Anweisung, um den Wert eines oder mehrerer Attribute innerhalb eines Elements in einer Amazon-DynamoDB-Tabelle zu ändern.

> **Note**
>
> Sie können nur ein Element gleichzeitig aktualisieren. Sie können keine einzelne DynamoDB-PartiQL-Anweisung ausgeben, die mehrere Elemente aktualisiert. Weitere Informationen zum

Aktualisieren mehrerer Elemente finden Sie unter Ausführen von Transaktionen mit PartiQL für DynamoDB oder Ausführen von Batchoperationen mit PartiQL für DynamoDB.

Themen

- Syntax
- Parameters
- Rückgabewert
- Beispiele

Syntax

```
UPDATE  table
[SET | REMOVE]  path  [=  data] […]
WHERE condition [RETURNING returnvalues]
<returnvalues>  ::= [ALL OLD | MODIFIED OLD | ALL NEW | MODIFIED NEW] *
```

Parameters

table

(erforderlich) Die Benutzertabelle mit den zu ändernden Daten.

path

(Erforderlich) Ein Attributname oder Dokumentpfad, der erstellt oder geändert werden soll.

data

(Erforderlich) Ein Attributwert oder das Ergebnis einer Operation.

Die unterstützten Operationen, die mit SET verwendet werden sollen:

- LIST_APPEND: fügt einem Listentyp einen Wert hinzu.
- SET_ADD: fügt einem Zahlen- oder Zeichenfolgensatz einen Wert hinzu.
- SET_DELETE: Entfernt einen Wert aus einem Zahlen- oder Zeichenfolgensatz.

condition

(erforderlich) Die Auswahlkriterien für die Elemente, die geändert werden sollen. Diese Bedingung muss auf einen einzelnen Primärschlüsselwert aufgelöst werden.

returnvalues

(Optional) Verwenden Sie `returnvalues`, wenn Sie die Elementattribute so abrufen möchten, wie sie vor oder nach der Aktualisierung angezeigt werden. Die gültigen Werte sind:

- `ALL OLD *` – Gibt alle Attribute des Elements so zurück, wie sie vor dem Aktualisieren dargestellt wurden.
- `MODIFIED OLD *` – Gibt nur die aktualisierten Attribute so zurück, wie sie vor dem Aktualisieren dargestellt wurden.
- `ALL NEW *` – Gibt alle Attribute des Elements zurück, wie sie nach dem Aktualisierungsvorgang angezeigt werden.
- `MODIFIED NEW *` – Gibt nur die aktualisierten Attribute so zurück, wie sie nach dem `UpdateItem` vorkommen.

Rückgabewert

Diese Anweisung gibt keinen Wert zurück, es sei denn der `returnvalues`-Parameter ist angegeben.

> **Note**
>
> Wenn die WHERE-Klausel der UPDATE-Anweisung für kein Element in der DynamoDB-Tabelle als wahr ausgewertet wird, wird `ConditionalCheckFailedException` zurückgegeben.

Beispiele

Aktualisieren Sie einen Attributwert in einem vorhandenen Element. Wenn das Attribut nicht vorhanden ist, wird es erstellt.

Mit der folgenden Abfrage wird ein Element in der `"Music"`-Tabelle durch Hinzufügen eines Attributs vom Typ „number“ (`AwardsWon`) und ein Attribut vom Typ „map“ (`AwardDetail`) enthalten.

```
UPDATE "Music"
SET AwardsWon=1
SET AwardDetail={'Grammys':[2020, 2018]}
WHERE Artist='Acme Band' AND SongTitle='PartiQL Rocks'
```

Sie können `RETURNING ALL OLD *` hinzufügen, um die Attribute so zurückzugeben, wie sie vor der `Update`-Operation erschienen.

```
UPDATE "Music"
SET AwardsWon=1
SET AwardDetail={'Grammys':[2020, 2018]}
WHERE Artist='Acme Band' AND SongTitle='PartiQL Rocks'
RETURNING ALL OLD *
```

Damit wird Folgendes zurückgegeben:

```
{
    "Items": [
        {
            "Artist": {
                "S": "Acme Band"
            },
            "SongTitle": {
                "S": "PartiQL Rocks"
            }
        }
    ]
}
```

Sie können `RETURNING ALL NEW *` hinzufügen, um die Attribute so zurückzugeben, wie sie nach der `Update`-Operation erschienen.

```
UPDATE "Music"
SET AwardsWon=1
SET AwardDetail={'Grammys':[2020, 2018]}
WHERE Artist='Acme Band' AND SongTitle='PartiQL Rocks'
RETURNING ALL NEW *
```

Damit wird Folgendes zurückgegeben:

```
{
    "Items": [
        {
            "AwardDetail": {
                "M": {
                    "Grammys": {
```

```
                        "L": [
                            {
                                "N": "2020"
                            },
                            {
                                "N": "2018"
                            }
                        ]
                    }
                }
            },
            "AwardsWon": {
                "N": "1"
            }
        }
    ]
}
```

Mit der folgenden Abfrage wird ein Element in der "`Music`"-Tabelle durch Anhängen an eine Liste `AwardDetail.Grammys` aktualisiert.

```
UPDATE "Music"
SET AwardDetail.Grammys =list_append(AwardDetail.Grammys,[2016])
WHERE Artist='Acme Band' AND SongTitle='PartiQL Rocks'
```

Mit der folgenden Abfrage wird ein Element in der "`Music`"-Tabelle durch Entfernen aus einer Liste `AwardDetail.Grammys` aktualisiert.

```
UPDATE "Music"
REMOVE AwardDetail.Grammys[2]
WHERE Artist='Acme Band' AND SongTitle='PartiQL Rocks'
```

Mit der folgenden Abfrage wird ein Element in der "`Music`"-Tabelle durch Hinzufügen von `BillBoard` zur Karte `AwardDetail` aktualisiert.

```
UPDATE "Music"
SET AwardDetail.BillBoard=[2020]
WHERE Artist='Acme Band' AND SongTitle='PartiQL Rocks'
```

Die folgende Abfrage aktualisiert ein Element in der "`Music`"-Tabelle durch Hinzufügen des Zeichenfolgensatzattributs `BandMembers`.

```
UPDATE "Music"
SET BandMembers =<<'member1', 'member2'>>
WHERE Artist='Acme Band' AND SongTitle='PartiQL Rocks'
```

Die folgende Abfrage aktualisiert ein Element in der `"Music"`-Tabelle, indem `newbandmember` zum Zeichenfolgensatz `BandMembers` hinzugefügt wird.

```
UPDATE "Music"
SET BandMembers =set_add(BandMembers, <<'newbandmember'>>)
WHERE Artist='Acme Band' AND SongTitle='PartiQL Rocks'
```

PartiQL-Delete-Anweisungen für DynamoDB

Verwenden der`DELETE`-Anweisung verwenden, um ein vorhandenes Element aus Ihrer Amazon-DynamoDB-Tabelle zu löschen.

> **Note**
>
> Sie können nur jeweils ein Element löschen. Sie können keine einzelne DynamoDB-PartiQL-Anweisung ausgeben, die mehrere Elemente löscht. Informationen zum Löschen mehrerer Elemente finden Sie unter Ausführen von Transaktionen mit PartiQL für DynamoDB oder Ausführen von Batchoperationen mit PartiQL für DynamoDB.

Themen

- Syntax
- Parameters
- Rückgabewert
- Beispiele

Syntax

```
DELETE FROM table
 WHERE condition [RETURNING returnvalues]
 <returnvalues>  ::= ALL OLD *
```

Parameters

table

(Erforderlich) Die DynamoDB-Tabelle, die das zu löschende Element enthält.

condition

(Erforderlich) Die Auswahlkriterien für das zu löschende Element; diese Bedingung muss auf einen einzelnen Primärschlüsselwert aufgelöst werden.

returnvalues

(Optional) Verwenden Sie `returnvalues`, wenn Sie die Elementattribute so erhalten möchten, wie sie vor dem Löschen dargestellt wurden. Die gültigen Werte sind:

- `ALL OLD *`- die Inhalte des alten Elements werden zurückgegeben.

Rückgabewert

Diese Anweisung gibt keinen Wert zurück, es sei denn der `returnvalues`-Parameter ist angegeben.

> **Note**
>
> Wenn in der DynamoDB-Tabelle kein Element mit demselben Primärschlüssel wie dem des Elements vorhanden ist, für das das DELETE ausgegeben wird, wird SUCCESS zurückgegeben, wobei 0 Elemente gelöscht wurden. Wenn die Tabelle ein Element mit demselben Primärschlüssel enthält, aber die Bedingung in der WHERE-Klausel der DELETE-Anweisung als falsch ausgewertet wird, wird `ConditionalCheckFailedException` zurückgegeben.

Beispiele

Mit der folgenden Abfrage wird ein Element in der Tabelle `"Music"` gelöscht.

```
DELETE FROM "Music" WHERE "Artist" = 'Acme Band' AND "SongTitle" = 'PartiQL Rocks'
```

Sie können den Parameter `RETURNING ALL OLD *` hinzufügen, um die gelöschten Daten zurückzugeben.

```
DELETE FROM "Music" WHERE "Artist" = 'Acme Band' AND "SongTitle" = 'PartiQL Rocks'
 RETURNING ALL OLD *
```

Die `Delete`-Anweisung gibt nun Folgendes zurück:

```
{
    "Items": [
        {
            "Artist": {
                "S": "Acme Band"
            },
            "SongTitle": {
                "S": "PartiQL Rocks"
            }
        }
    ]
}
```

PartiQL-Insert-Anweisungen für DynamoDB

Verwenden Sie die `INSERT`-Anweisung, um einer Tabelle in Amazon DynamoDB ein Element hinzuzufügen.

> **Note**
>
> Sie können nur ein Element gleichzeitig einfügen. Sie können keine einzelne DynamoDB-PartiQL-Anweisung ausgeben, die mehrere Elemente einfügt. Weitere Informationen zum Einfügen mehrerer Elemente finden Sie unter Ausführen von Transaktionen mit PartiQL für DynamoDB oder Ausführen von Batchoperationen mit PartiQL für DynamoDB.

Themen

- Syntax
- Parameters
- Rückgabewert
- Beispiele

Syntax

Fügen Sie ein einzelnes Element ein.

```
INSERT INTO table VALUE item
```

Parameters

table

(erforderlich) Die Tabelle, in der Sie die Daten einfügen möchten. Die Tabelle muss bereits vorhanden sein.

item

(Erforderlich) Ein gültiges DynamoDB Element, das als PartiQL-Tupel gezeigt wird. Sie müssen nur ein Element angeben, und bei jedem Attributnamen im Element wird die Groß-/Kleinschreibung beachtet und kann in PartiQL mit einfachen Anführungszeichen (`'...'`) gekennzeichnet werden.

Zeichenfolgenwerte werden in PartiQL auch mit einfachen Anführungszeichen (`'...'`) angegeben.

Rückgabewert

Diese Anweisung gibt keine Werte zurück.

> **Note**
>
> Wenn die DynamoDB-Tabelle bereits ein Element mit demselben Primärschlüssel wie der Primärschlüssel des einzufügenden Elements enthält, wird `DuplicateItemException` zurückgegeben.

Beispiele

```
INSERT INTO "Music" value {'Artist' : 'Acme Band','SongTitle' : 'PartiQL Rocks'}
```

Verwenden von PartiQL-Funktionen mit DynamoDB

PartiSQL in Amazon DynamoDB unterstützt die folgenden integrierten Varianten von SQL-Standardfunktionen.

> **Note**
>
> SQL-Funktionen, die nicht in dieser Liste enthalten sind, werden derzeit in DynamoDB nicht unterstützt.

Aggregationsfunktionen

- Verwenden der SIZE-Funktion mit PartiQL für Amazon DynamoDB

Konditionale Funktionen

- Verwenden der EXISTS-Funktion mit PartiQL für DynamoDB
- Verwenden der ATTRIBUTE_TYPE-Funktion mit PartiQL für DynamoDB
- Verwenden der BEGINS_WITH-Funktion mit PartiQL für DynamoDB
- Verwenden der CONTAINS-Funktion mit PartiQL für DynamoDB
- Verwenden der MISSING-Funktion mit PartiQL für DynamoDB

Verwenden der EXISTS-Funktion mit PartiQL für DynamoDB

Sie können EXISTS verwenden, um dieselbe Funktion wie `ConditionCheck` in der TransactWriteItems-API auszuführen. Die EXISTS-Funktion kann nur in Transaktionen verwendet werden.

Gibt bei Vorliegen eines Werts `TRUE` zurück, wenn der Wert eine nicht-leere Sammlung ist. Gibt andernfalls `FALSE` zurück.

> **Note**
>
> Diese Funktion kann nur in Transaktionsoperationen verwendet werden.

Syntax

```
EXISTS ( statement )
```

Argumente

statement

(Erforderlich) Die SELECT-Anweisung, die die Funktion auswertet.

> **Note**
>
> Die SELECT-Anweisung muss einen vollständigen Primärschlüssel und eine andere Bedingung angeben.

Rückgabetyp

`bool`

Beispiele

```
EXISTS(
    SELECT * FROM "Music"
    WHERE "Artist" = 'Acme Band' AND "SongTitle" = 'PartiQL Rocks')
```

Verwenden der BEGINS_WITH-Funktion mit PartiQL für DynamoDB

Gibt `TRUE` zurück, wenn das angegebene Attribut mit einer bestimmten Teilzeichenfolge beginnt.

Syntax

```
begins_with(path, value )
```

Argumente

path

(Erforderlich) Der zu verwendende Attributname oder Dokumentpfad.

value

(Erforderlich) Die Zeichenfolge, nach der gesucht werden soll.

Rückgabetyp

bool

Beispiele

```
SELECT * FROM "Orders" WHERE "OrderID"=1 AND begins_with("Address", '7834 24th')
```

Verwenden der MISSING-Funktion mit PartiQL für DynamoDB

Gibt TRUE zurück, wenn das Element das angegebene Attribut nicht enthält. Mit dieser Funktion können nur Gleichheits- und Ungleichheitsoperatoren verwendet werden.

Syntax

```
 attributename IS | IS NOT  MISSING
```

Argumente

attributename

(Erforderlich) Der Attributname, nach dem Sie suchen möchten.

Rückgabetyp

bool

Beispiele

```
SELECT * FROM Music WHERE "Awards" is MISSING
```

Verwenden der ATTRIBUTE_TYPE-Funktion mit PartiQL für DynamoDB

Gibt TRUE zurück, wenn das Attribut am angegebenen Pfad einen bestimmten Datentyp hat.

Syntax

```
attribute_type( attributename, type )
```

Argumente

attributename

(Erforderlich) Der zu verwendende Attributname.

type

(Erforderlich) Der Attributtyp, nach dem geprüft werden soll. Eine Liste der gültigen Werte finden Sie unter DynamoDB attribute_type.

Rückgabetyp

`bool`

Beispiele

```
SELECT * FROM "Music" WHERE attribute_type("Artist", 'S')
```

Verwenden der CONTAINS-Funktion mit PartiQL für DynamoDB

Gibt TRUE zurück, wenn das vom Pfad angegebene Attribut Folgendes ist:

- Eine Zeichenfolge, die eine bestimmte Teilzeichenfolge enthält
- Einen Satz, der ein bestimmtes Element innerhalb des Satzes enthält

Weitere Informationen finden Sie im Thema zur DynamoDB-Contains-Funktion.

Syntax

```
contains( path, substring )
```

Argumente

path

(Erforderlich) Der zu verwendende Attributname oder Dokumentpfad.

substring

(Erforderlich) Die Attribut-Teilzeichenfolge oder das Satz-Element, nach dem Sie suchen möchten Weitere Informationen finden Sie im Thema zur DynamoDB-Contains-Funktion.

Rückgabetyp

`bool`

Beispiele

```
SELECT * FROM "Orders" WHERE "OrderID"=1 AND contains("Address", 'Kirkland')
```

Verwenden der SIZE-Funktion mit PartiQL für Amazon DynamoDB

Gibt eine Zahl zurück, die für die Größe eines Attributs in Bytes steht. Die folgenden sind gültige Datentypen, die mit Größe verwendet werden können. Weitere Informationen finden Sie in der DynamoDB-size-Funktion.

Syntax

```
size( path)
```

Argumente

path

(Erforderlich) Der Attributname oder Dokumentpfad.

Weitere Informationen zu unterstützten Typen finden Sie im Thema DynamoDB-SIZE-Funktion.

Rückgabetyp

`int`

Beispiele

```
SELECT * FROM "Orders" WHERE "OrderID"=1 AND size("Image") >300
```

PartiQL-Arithmetik-, Vergleichs- und logische Operatoren für DynamoDB

PartiSQL in Amazon DynamoDB unterstützt die folgenden SQL-Standardoperatoren.

> **Note**
>
> SQL-Operatoren, die nicht in dieser Liste enthalten sind, werden derzeit in DynamoDB nicht unterstützt.

Arithmetische Operatoren

Operator	Description
+	Addition
-	Subtraktion

Vergleichsoperatoren

Operator	Description
=	gleich
<>	nicht gleich
!=	nicht gleich
>	Größer als
<	Kleiner als
>=	Größer als oder gleich
<=	Kleiner als oder gleich

Logische Operatoren

Operator	Description
`AND`	`TRUE`, wenn alle durch `AND` getrennten Bedingungen `TRUE` sind
`BETWEEN`	`TRUE`, wenn der Operand innerhalb des Vergleichsbereichs liegt. Dieser Operator umfasst die Unter- und Obergrenze der Operanden, auf die Sie ihn anwenden.
`IN`	`TRUE`wenn der Operand einem aus einer Liste von Ausdrücken entspricht (bei maximal 50 Hash-Attributwerten oder bei maximal 100 Nichtschlüsselattributwerten). Die Ergebnisse werden in Seiten mit bis zu 10 Elementen zurückgegeben. Wenn die `IN` Liste mehr Werte enthält, müssen Sie die in der Antwort `NextToken` zurückgegebenen Werte verwenden, um nachfolgende Seiten abzurufen .
`IS`	`TRUE`, wenn der Operand ein bestimmter PartiQL-Datentyp ist, einschließlich `NULL` oder `MISSING`
`NOT`	Kehrt den Wert eines gegebenen booleschen Ausdrucks um
`OR`	`TRUE`, wenn eine der von `OR` getrennten Bedingungen `TRUE` ist

Weitere Informationen zu logischen Operatoren finden Sie unter Durchführen von Vergleichen und Logische Auswertungen.

Ausführen von Transaktionen mit PartiQL für DynamoDB

In diesem Abschnitt wird die Verwendung von Transaktionen mit PartiQL für DynamoDB beschrieben. PartiQL-Transaktionen sind auf insgesamt 100 Anweisungen (Aktionen) begrenzt.

Weitere Informationen zu DynamoDB-Transaktionen finden Sie unter Verwalten komplexer Workflows mit DynamoDB-Transaktionen.

> **Note**
>
> Die gesamte Transaktion muss entweder aus Leseanweisungen oder Schreibanweisungen bestehen. Sie können nicht beides in einer Transaktion mischen. Die EXISTS-Funktion ist eine Ausnahme. Sie können es verwenden, um den Zustand bestimmter Attribute des Elements auf ähnliche Weise wie `ConditionCheck` bei der TransactWriteItemsAPI-Operation zu überprüfen.

Themen

- Syntax
- Parameters
- Rückgabewerte
- Beispiele

Syntax

```
[
   {
      "Statement":" statement ",
      "Parameters":[
         {
            " parametertype " : " parametervalue "
         }, ...]
   } , ...
]
```

Parameters

statement

(Erforderlich) Eine unterstützte PartiQL für DynamoDB-Anweisung.

> **Note**
>
> Die gesamte Transaktion muss entweder aus Leseanweisungen oder Schreibanweisungen bestehen. Sie können nicht beides in einer Transaktion mischen.

parametertype

(Optional) Ein DynamoDB-Typ, wenn Parameter bei der Angabe der PartiQL-Anweisung verwendet wurden.

parametervalue

(Optional) Ein Parameterwert, wenn Parameter bei der Angabe der PartiQL-Anweisung verwendet wurden.

Rückgabewerte

Diese Anweisung gibt keine Werte für Schreibvorgänge (INSERT, UPDATE oder DELETE) zurück. Sie gibt jedoch verschiedene Werte für Lesevorgänge (SELECT) basierend auf den in der WHERE-Klausel angegebenen Bedingungen zurück.

> **Note**
>
> Wenn eine der Singleton-Operationen INSERT, UPDATE oder DELETE einen Fehler zurückgibt, werden die Transaktionen mit einer `TransactionCanceledException`-Ausnahme abgebrochen, und der Code für den Abbruchgrund enthält die Fehler der einzelnen Singleton-Operationen.

Beispiele

Im folgenden Beispiel werden mehrere Anweisungen als Transaktion ausgeführt.

AWS CLI

1. Speichern Sie den folgenden JSON-Code in einer Datei mit dem Namen partiql.json.

```
[
    {
        "Statement": "EXISTS(SELECT * FROM \"Music\" where Artist='No One You
 Know' and SongTitle='Call Me Today' and Awards is  MISSING)"
    },
    {
        "Statement": "INSERT INTO Music value {'Artist':?,'SongTitle':'?'}",
        "Parameters": [{\"S\": \"Acme Band\"}, {\"S\": \"Best Song\"}]
    },
    {
        "Statement": "UPDATE \"Music\" SET AwardsWon=1 SET
 AwardDetail={'Grammys':[2020, 2018]}  where Artist='Acme Band' and
 SongTitle='PartiQL Rocks'"
    }
]
```

2. Führen Sie in der Eingabeaufforderung den folgenden Befehl aus.

```
aws dynamodb execute-transaction --transact-statements  file://partiql.json
```

Java

```
public class DynamoDBPartiqlTransaction {

    public static void main(String[] args) {
        // Create the DynamoDB Client with the region you want
        AmazonDynamoDB dynamoDB = createDynamoDbClient("us-west-2");

        try {
            // Create ExecuteTransactionRequest
            ExecuteTransactionRequest executeTransactionRequest =
 createExecuteTransactionRequest();
            ExecuteTransactionResult executeTransactionResult =
 dynamoDB.executeTransaction(executeTransactionRequest);
            System.out.println("ExecuteTransaction successful.");
            // Handle executeTransactionResult

        } catch (Exception e) {
```

```
            handleExecuteTransactionErrors(e);
        }
    }

    private static AmazonDynamoDB createDynamoDbClient(String region) {
        return AmazonDynamoDBClientBuilder.standard().withRegion(region).build();
    }

    private static ExecuteTransactionRequest createExecuteTransactionRequest() {
        ExecuteTransactionRequest request = new ExecuteTransactionRequest();

        // Create statements
        List<ParameterizedStatement> statements = getPartiQLTransactionStatements();

        request.setTransactStatements(statements);
        return request;
    }

    private static List<ParameterizedStatement> getPartiQLTransactionStatements() {
        List<ParameterizedStatement> statements = new
ArrayList<ParameterizedStatement>();

        statements.add(new ParameterizedStatement()
                               .withStatement("EXISTS(SELECT * FROM "Music" where
Artist='No One You Know' and SongTitle='Call Me Today' and Awards is  MISSING)"));

        statements.add(new ParameterizedStatement()
                               .withStatement("INSERT INTO "Music" value
{'Artist':'?','SongTitle':'?'}")
                               .withParameters(new AttributeValue("Acme Band"),new
AttributeValue("Best Song")));

        statements.add(new ParameterizedStatement()
                               .withStatement("UPDATE "Music" SET AwardsWon=1
SET AwardDetail={'Grammys':[2020, 2018]}  where Artist='Acme Band' and
SongTitle='PartiQL Rocks'"));

        return statements;
    }

    // Handles errors during ExecuteTransaction execution. Use recommendations in
error messages below to add error handling specific to
    // your application use-case.
    private static void handleExecuteTransactionErrors(Exception exception) {
```

```
        try {
            throw exception;
        } catch (TransactionCanceledException tce) {
            System.out.println("Transaction Cancelled, implies a client issue, fix
before retrying. Error: " + tce.getErrorMessage());
        } catch (TransactionInProgressException tipe) {
            System.out.println("The transaction with the given request token is
already in progress, consider changing " +
                "retry strategy for this type of error. Error: " +
tipe.getErrorMessage());
        } catch (IdempotentParameterMismatchException ipme) {
            System.out.println("Request rejected because it was retried with a
different payload but with a request token that was already used, " +
                "change request token for this payload to be accepted. Error: " +
ipme.getErrorMessage());
        } catch (Exception e) {
            handleCommonErrors(e);
        }
    }

    private static void handleCommonErrors(Exception exception) {
        try {
            throw exception;
        } catch (InternalServerErrorException isee) {
            System.out.println("Internal Server Error, generally safe to retry with
exponential back-off. Error: " + isee.getErrorMessage());
        } catch (RequestLimitExceededException rlee) {
            System.out.println("Throughput exceeds the current throughput limit for
your account, increase account level throughput before " +
                "retrying. Error: " + rlee.getErrorMessage());
        } catch (ProvisionedThroughputExceededException ptee) {
            System.out.println("Request rate is too high. If you're using a custom
retry strategy make sure to retry with exponential back-off. " +
                "Otherwise consider reducing frequency of requests or increasing
provisioned capacity for your table or secondary index. Error: " +
                ptee.getErrorMessage());
        } catch (ResourceNotFoundException rnfe) {
            System.out.println("One of the tables was not found, verify table exists
before retrying. Error: " + rnfe.getErrorMessage());
        } catch (AmazonServiceException ase) {
            System.out.println("An AmazonServiceException occurred, indicates that
the request was correctly transmitted to the DynamoDB " +
                "service, but for some reason, the service was not able to process
it, and returned an error response instead. Investigate and " +
```

```
                "configure retry strategy. Error type: " + ase.getErrorType() + ".
 Error message: " + ase.getErrorMessage());
        } catch (AmazonClientException ace) {
            System.out.println("An AmazonClientException occurred, indicates that
 the client was unable to get a response from DynamoDB " +
                "service, or the client was unable to parse the response from the
 service. Investigate and configure retry strategy. "+
                "Error: " + ace.getMessage());
        } catch (Exception e) {
            System.out.println("An exception occurred, investigate and configure
 retry strategy. Error: " + e.getMessage());
        }
    }

}
```

Das folgende Beispiel zeigt die verschiedenen Rückgabewerte, wenn DynamoDB-Elemente mit unterschiedlichen Bedingungen liest, die in der WHERE-Klausel angegeben sind.

AWS CLI

1. Speichern Sie den folgenden JSON-Code in einer Datei mit dem Namen partiql.json.

```
[
    // Item exists and projected attribute exists
    {
        "Statement": "SELECT * FROM "Music" WHERE Artist='No One You Know' and
 SongTitle='Call Me Today'"
    },
    // Item exists but projected attributes do not exist
    {
        "Statement": "SELECT non_existent_projected_attribute FROM "Music" WHERE
 Artist='No One You Know' and SongTitle='Call Me Today'"
    },
    // Item does not exist
    {
        "Statement": "SELECT * FROM "Music" WHERE Artist='No One I Know' and
 SongTitle='Call You Today'"
    }
]
```

2. Folgenden Befehl in der Eingabeaufforderung eingeben.

```
aws dynamodb execute-transaction --transact-statements  file://partiql.json
```

3. Die folgende Antwort wird ausgegeben:

```
{
    "Responses": [
        // Item exists and projected attribute exists
        {
            "Item": {
                "Artist":{
                    "S": "No One You Know"
                },
                "SongTitle":{
                    "S": "Call Me Today"
                }
            }
        },
        // Item exists but projected attributes do not exist
        {
            "Item": {}
        },
        // Item does not exist
        {}
    ]
}
```

Ausführen von Batchoperationen mit PartiQL für DynamoDB

In diesem Abschnitt wird die Verwendung von Stapel-Anweisungen mit PartiQL für DynamoDB beschrieben.

> **Note**
>
> - Der gesamte Stapel muss entweder aus Leseanweisungen oder Schreibanweisungen bestehen; Sie können nicht beides in einem Stapel mischen.
> - `BatchExecuteStatement` und `BatchWriteItem` können nicht mehr als 25 Anweisungen pro Stapel ausführen.

- `BatchExecuteStatement` verwendet `BatchGetItem`, was eine Liste von Primärschlüsseln in separaten Anweisungen verarbeitet.

Themen

- Syntax
- Parameters
- Beispiele

Syntax

```
[
  {
    "Statement": "SELECT pk FROM ProblemSet WHERE pk = 'p#9StkWHYTxm7x2AqSXcrfu7' AND
 sk = 'info'"
  },
  {
    "Statement": "SELECT pk FROM ProblemSet WHERE pk = 'p#isC2ChceGbxHgESc4szoTE' AND
 sk = 'info'"
  }
]
```

```
[
   {
      "Statement":" statement ",
      "Parameters":[
         {
            " parametertype " : " parametervalue "
         }, ...]
   } , ...
]
```

Parameters

statement

(Erforderlich) Eine unterstützte PartiQL für DynamoDB-Anweisung.

> **Note**
>
> - Der gesamte Stapel muss entweder aus Leseanweisungen oder Schreibanweisungen bestehen; Sie können nicht beides in einem Stapel mischen.
> - `BatchExecuteStatement` und `BatchWriteItem` können nicht mehr als 25 Anweisungen pro Stapel ausführen.

parametertype

(Optional) Ein DynamoDB-Typ, wenn Parameter bei der Angabe der PartiQL-Anweisung verwendet wurden.

parametervalue

(Optional) Ein Parameterwert, wenn Parameter bei der Angabe der PartiQL-Anweisung verwendet wurden.

Beispiele

AWS CLI

1. Speichern Sie den folgenden JSON in einer Datei namens partiql.json

```
[
   {
  "Statement": "INSERT INTO Music VALUE {'Artist':?,'SongTitle':?}",
   "Parameters": [{"S": "Acme Band"}, {"S": "Best Song"}]
 },
 {
  "Statement": "UPDATE Music SET AwardsWon=1, AwardDetail={'Grammys':[2020,
 2018]} WHERE Artist='Acme Band' AND SongTitle='PartiQL Rocks'"
    }
]
```

2. Führen Sie in der Eingabeaufforderung den folgenden Befehl aus.

```
aws dynamodb batch-execute-statement  --statements  file://partiql.json
```

Java

```
public class DynamoDBPartiqlBatch {

    public static void main(String[] args) {
        // Create the DynamoDB Client with the region you want
        AmazonDynamoDB dynamoDB = createDynamoDbClient("us-west-2");

        try {
            // Create BatchExecuteStatementRequest
            BatchExecuteStatementRequest batchExecuteStatementRequest =
 createBatchExecuteStatementRequest();
            BatchExecuteStatementResult batchExecuteStatementResult =
 dynamoDB.batchExecuteStatement(batchExecuteStatementRequest);
            System.out.println("BatchExecuteStatement successful.");
            // Handle batchExecuteStatementResult

        } catch (Exception e) {
            handleBatchExecuteStatementErrors(e);
        }
    }

    private static AmazonDynamoDB createDynamoDbClient(String region) {

        return AmazonDynamoDBClientBuilder.standard().withRegion(region).build();
    }

    private static BatchExecuteStatementRequest createBatchExecuteStatementRequest()
 {
        BatchExecuteStatementRequest request = new BatchExecuteStatementRequest();

        // Create statements
        List<BatchStatementRequest> statements = getPartiQLBatchStatements();

        request.setStatements(statements);
        return request;
    }

    private static List<BatchStatementRequest> getPartiQLBatchStatements() {
        List<BatchStatementRequest> statements = new
 ArrayList<BatchStatementRequest>();

        statements.add(new BatchStatementRequest()
```

```
                                .withStatement("INSERT INTO Music value
{'Artist':'Acme Band','SongTitle':'PartiQL Rocks'}"));

        statements.add(new BatchStatementRequest()
                                .withStatement("UPDATE Music set
AwardDetail.BillBoard=[2020] where Artist='Acme Band' and SongTitle='PartiQL
Rocks'"));

        return statements;
    }

    // Handles errors during BatchExecuteStatement execution. Use recommendations in
error messages below to add error handling specific to
    // your application use-case.
    private static void handleBatchExecuteStatementErrors(Exception exception) {
        try {
            throw exception;
        } catch (Exception e) {
            // There are no API specific errors to handle for BatchExecuteStatement,
common DynamoDB API errors are handled below
            handleCommonErrors(e);
        }
    }

    private static void handleCommonErrors(Exception exception) {
        try {
            throw exception;
        } catch (InternalServerErrorException isee) {
            System.out.println("Internal Server Error, generally safe to retry with
exponential back-off. Error: " + isee.getErrorMessage());
        } catch (RequestLimitExceededException rlee) {
            System.out.println("Throughput exceeds the current throughput limit for
your account, increase account level throughput before " +
                "retrying. Error: " + rlee.getErrorMessage());
        } catch (ProvisionedThroughputExceededException ptee) {
            System.out.println("Request rate is too high. If you're using a custom
retry strategy make sure to retry with exponential back-off. " +
                "Otherwise consider reducing frequency of requests or increasing
provisioned capacity for your table or secondary index. Error: " +
                ptee.getErrorMessage());
        } catch (ResourceNotFoundException rnfe) {
            System.out.println("One of the tables was not found, verify table exists
before retrying. Error: " + rnfe.getErrorMessage());
        } catch (AmazonServiceException ase) {
```

```
            System.out.println("An AmazonServiceException occurred, indicates that
 the request was correctly transmitted to the DynamoDB " +
                "service, but for some reason, the service was not able to process
 it, and returned an error response instead. Investigate and " +
                "configure retry strategy. Error type: " + ase.getErrorType() + ".
 Error message: " + ase.getErrorMessage());
        } catch (AmazonClientException ace) {
            System.out.println("An AmazonClientException occurred, indicates that
 the client was unable to get a response from DynamoDB " +
                "service, or the client was unable to parse the response from the
 service. Investigate and configure retry strategy. "+
                "Error: " + ace.getMessage());
        } catch (Exception e) {
            System.out.println("An exception occurred, investigate and configure
 retry strategy. Error: " + e.getMessage());
        }
    }

}
```

IAM-Sicherheitsrichtlinien mit PartiQL für DynamoDB

Die folgenden Berechtigungen sind erforderlich:

- Um Elemente mit PartiQL für DynamoDB zu lesen, müssen Sie über die `dynamodb:PartiQLSelect`-Berechtigung für die Tabelle oder den Index verfügen.
- Um Elemente mit PartiQL für DynamoDB einzufügen, müssen Sie über die `dynamodb:PartiQLInsert`-Berechtigung für die Tabelle oder den Index verfügen.
- Um Elemente mit PartiQL für DynamoDB zu aktualisieren, müssen Sie über die `dynamodb:PartiQLUpdate`-Berechtigung für die Tabelle oder den Index verfügen.
- Um Elemente mit PartiQL für DynamoDB zu löschen, müssen Sie über die `dynamodb:PartiQLDelete`-Berechtigung für die Tabelle oder den Index verfügen.

Beispiel: Alle PartiQL for DynamoDB-Anweisungen (Select/Insert/Update/Delete) in einer Tabelle zulassen

Mit der folgenden IAM-Richtlinie werden alle PartiQL für DynamoDB-Anweisungen in einer Tabelle erteilt.

JSON

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Effect":"Allow",
         "Action":[
            "dynamodb:PartiQLInsert",
            "dynamodb:PartiQLUpdate",
            "dynamodb:PartiQLDelete",
            "dynamodb:PartiQLSelect"
         ],
         "Resource":[
            "arn:aws:dynamodb:us-west-2:123456789012:table/Music"
         ]
      }
   ]
}
```

Beispiel: PartiQL für DynamoDB-Auswahlanweisungen für eine Tabelle zulassen

Mit der folgenden IAM-Richtlinie werden die Berechtigungen zum Ausführen der `select`-Anweisung für eine bestimmte Tabelle erteilt

JSON

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Effect":"Allow",
         "Action":[
            "dynamodb:PartiQLSelect"
         ],
         "Resource":[
            "arn:aws:dynamodb:us-west-2:123456789012:table/Music"
         ]
      }
   ]
```

```
}
```

Beispiel: PartiQL für DynamoDB-Einfügeanweisungen für einen Index zulassen

Mit der folgenden IAM-Richtlinie werden die Berechtigungen zum Ausführen des `insert`-Anweisung auf einen bestimmten Index.

JSON

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Effect":"Allow",
         "Action":[
            "dynamodb:PartiQLInsert"
         ],
         "Resource":[
            "arn:aws:dynamodb:us-west-2:123456789012:table/Music/index/index1"
         ]
      }
   ]
}
```

Beispiel: PartiQL für DynamoDB-Transaktionsanweisungen nur für eine Tabelle zulassen

Mit der folgenden IAM-Richtlinie werden die Berechtigungen nur zum Ausführen transaktionaler Anweisungen für eine bestimmte Tabelle erteilt.

JSON

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Effect":"Allow",
         "Action":[
            "dynamodb:PartiQLInsert",
            "dynamodb:PartiQLUpdate",
```

```
            "dynamodb:PartiQLDelete",
            "dynamodb:PartiQLSelect"
         ],
         "Resource":[
            "arn:aws:dynamodb:us-west-2:123456789012:table/Music"
         ],
         "Condition":{
            "StringEquals":{
               "dynamodb:EnclosingOperation":[
                  "ExecuteTransaction"
               ]
            }
         }
      }
   ]
}
```

Beispiel: Erlauben Sie PartiQL für nicht-transaktionale DynamoDB-Lese- und Schreibvorgänge und blockieren Sie PartiQL-transaktionale Lese- und Schreibvorgänge in einer Tabelle.

Die folgende IAM-Richtlinie gewährt Berechtigungen zum Ausführen von nicht-transaktionalen Lese-/Schreibvorgängen in PartiQL für DynamoDB, während transaktionale Lese-/Schreibvorgänge in PartiQL für DynamoDB blockiert werden.

JSON

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Effect":"Deny",
         "Action":[
            "dynamodb:PartiQLInsert",
            "dynamodb:PartiQLUpdate",
            "dynamodb:PartiQLDelete",
            "dynamodb:PartiQLSelect"
         ],
         "Resource":[
            "arn:aws:dynamodb:us-west-2:123456789012:table/Music"
         ],
         "Condition":{
```

```
            "StringEquals":{
               "dynamodb:EnclosingOperation":[
                  "ExecuteTransaction"
               ]
            }
         }
      },
      {
         "Effect":"Allow",
         "Action":[
            "dynamodb:PartiQLInsert",
            "dynamodb:PartiQLUpdate",
            "dynamodb:PartiQLDelete",
            "dynamodb:PartiQLSelect"
         ],
         "Resource":[
            "arn:aws:dynamodb:us-west-2:123456789012:table/Music"
         ]
      }
   ]
}
```

Beispiel: In PartiQL für DynamoDB Select-Anweisungen erlauben und vollständige Tabellenscan-Anweisungen verweigern

Mit der folgenden IAM-Richtlinie werden die Berechtigungen zum Ausführen der `select`-Anweisung für eine bestimmte Tabelle beim Blockieren von `select`-Anweisungen, die zu einem vollständigen Tabellenscan führen.

JSON

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Effect":"Deny",
         "Action":[
            "dynamodb:PartiQLSelect"
         ],
         "Resource":[
            "arn:aws:dynamodb:us-west-2:123456789012:table/WatchList"
```

```
            ],
            "Condition":{
               "Bool":{
                  "dynamodb:FullTableScan":[
                     "true"
                  ]
               }
            }
         },
         {
            "Effect":"Allow",
            "Action":[
               "dynamodb:PartiQLSelect"
            ],
            "Resource":[
               "arn:aws:dynamodb:us-west-2:123456789012:table/WatchList"
            ]
         }
      ]
}
```

Arbeiten mit Elementen: Java

Sie können die AWS SDK für Java Dokument-API verwenden, um typische Erstellungs-, Lese-, Aktualisierungs- und Löschvorgänge (CRUD) für Amazon DynamoDB DynamoDB-Elemente in einer Tabelle durchzuführen.

> **Note**
>
> Das SDK für Java stellt auch ein Objektpersistenzmodell bereit, mit dem Sie Ihre clientseitigen Klassen DynamoDB-Tabellen zuordnen können. Mit diesem Ansatz können Sie die Codemenge, die Sie schreiben müssen, verringern. Weitere Informationen finden Sie unter Java 1.x: Dynamo DBMapper.

Dieser Abschnitt enthält Java-Beispiele zur Durchführung verschiedener Java-Dokument-API-Elementaktionen und mehrere vollständige Arbeitsbeispiele.

Themen

- Einfügen eines Elements

- Abrufen eines Elements
- Batch Write: Einfügen und Löschen mehrerer Elemente
- Batch Get: Abrufen mehrerer Elemente
- Aktualisieren eines Elements
- Löschen eines Elements
- Beispiel: CRUD-Operationen mit der AWS SDK für Java Dokument-API
- Beispiel: Batch-Operationen mithilfe der AWS SDK für Java Dokument-API
- Beispiel: Umgang mit binären Typattributen mithilfe der AWS SDK für Java Dokument-API

Einfügen eines Elements

Die `putItem`-Methode speichert ein Element in einer Tabelle. Wenn das Element bereits vorhanden ist, wird das ganze Element ersetzt. Anstatt das gesamte Element zu ersetzen und nur spezifische Attribute zu aktualisieren, können Sie die Methode `updateItem` verwenden. Weitere Informationen finden Sie unter Aktualisieren eines Elements.

Java v2

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.AttributeValue;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.model.PutItemRequest;
import software.amazon.awssdk.services.dynamodb.model.PutItemResponse;
import software.amazon.awssdk.services.dynamodb.model.ResourceNotFoundException;
import java.util.HashMap;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-started.html
 *
 * To place items into an Amazon DynamoDB table using the AWS SDK for Java V2,
 * its better practice to use the
 * Enhanced Client. See the EnhancedPutItem example.
 */
```

```
public class PutItem {
    public static void main(String[] args) {
        final String usage = """

                Usage:
                    <tableName> <key> <keyVal> <albumtitle> <albumtitleval> <awards>
 <awardsval> <Songtitle> <songtitleval>

                Where:
                    tableName - The Amazon DynamoDB table in which an item is placed
 (for example, Music3).
                    key - The key used in the Amazon DynamoDB table (for example,
 Artist).
                    keyval - The key value that represents the item to get (for
 example, Famous Band).
                    albumTitle - The Album title (for example, AlbumTitle).
                    AlbumTitleValue - The name of the album (for example, Songs
 About Life ).
                    Awards - The awards column (for example, Awards).
                    AwardVal - The value of the awards (for example, 10).
                    SongTitle - The song title (for example, SongTitle).
                    SongTitleVal - The value of the song title (for example, Happy
 Day).
                **Warning** This program will  place an item that you specify into a
 table!
                """;

       if (args.length != 9) {
           System.out.println(usage);
           System.exit(1);
       }

       String tableName = args[0];
       String key = args[1];
       String keyVal = args[2];
       String albumTitle = args[3];
       String albumTitleValue = args[4];
       String awards = args[5];
       String awardVal = args[6];
       String songTitle = args[7];
       String songTitleVal = args[8];

       Region region = Region.US_EAST_1;
       DynamoDbClient ddb = DynamoDbClient.builder()
```

```
                .region(region)
                .build();

        putItemInTable(ddb, tableName, key, keyVal, albumTitle, albumTitleValue,
awards, awardVal, songTitle,
                songTitleVal);
        System.out.println("Done!");
        ddb.close();
    }

    public static void putItemInTable(DynamoDbClient ddb,
            String tableName,
            String key,
            String keyVal,
            String albumTitle,
            String albumTitleValue,
            String awards,
            String awardVal,
            String songTitle,
            String songTitleVal) {

        HashMap<String, AttributeValue> itemValues = new HashMap<>();
        itemValues.put(key, AttributeValue.builder().s(keyVal).build());
        itemValues.put(songTitle, AttributeValue.builder().s(songTitleVal).build());
        itemValues.put(albumTitle,
AttributeValue.builder().s(albumTitleValue).build());
        itemValues.put(awards, AttributeValue.builder().s(awardVal).build());

        PutItemRequest request = PutItemRequest.builder()
                .tableName(tableName)
                .item(itemValues)
                .build();

        try {
            PutItemResponse response = ddb.putItem(request);
            System.out.println(tableName + " was successfully updated. The request
id is "
                    + response.responseMetadata().requestId());

        } catch (ResourceNotFoundException e) {
            System.err.format("Error: The Amazon DynamoDB table \"%s\" can't be
found.\n", tableName);
            System.err.println("Be sure that it exists and that you've typed its
name correctly!");
```

```
            System.exit(1);
        } catch (DynamoDbException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
    }
}
```

Java v1

Dazu gehen Sie wie folgt vor:

1. Erstellen Sie eine Instance der `DynamoDB`-Klasse.
2. Erstellen Sie eine Instance der `Table`-Klasse, um die Tabelle zu repräsentieren, mit der Sie arbeiten möchten.
3. Erstellen Sie eine Instance der `Item`-Klasse, um das neue Element zu repräsentieren. Sie müssen den Primärschlüssel und die Attribute für das neue Element angeben.
4. Rufen Sie die `putItem`-Methode des `Table`-Objekts auf, das das `Item` verwendet, das Sie im vorangegangenen Schritt erstellt haben.

Im folgenden Java-Codebeispiel werden die vorherigen Aufgaben veranschaulicht. Der Code schreibt ein neues Element in die `ProductCatalog`-Tabelle.

Example

```
AmazonDynamoDB client = AmazonDynamoDBClientBuilder.standard().build();
DynamoDB dynamoDB = new DynamoDB(client);

Table table = dynamoDB.getTable("ProductCatalog");

// Build a list of related items
List<Number> relatedItems = new ArrayList<Number>();
relatedItems.add(341);
relatedItems.add(472);
relatedItems.add(649);

//Build a map of product pictures
Map<String, String> pictures = new HashMap<String, String>();
pictures.put("FrontView", "http://example.com/products/123_front.jpg");
pictures.put("RearView", "http://example.com/products/123_rear.jpg");
```

```
pictures.put("SideView", "http://example.com/products/123_left_side.jpg");

//Build a map of product reviews
Map<String, List<String>> reviews = new HashMap<String, List<String>>();

List<String> fiveStarReviews = new ArrayList<String>();
fiveStarReviews.add("Excellent! Can't recommend it highly enough!  Buy it!");
fiveStarReviews.add("Do yourself a favor and buy this");
reviews.put("FiveStar", fiveStarReviews);

List<String> oneStarReviews = new ArrayList<String>();
oneStarReviews.add("Terrible product!  Do not buy this.");
reviews.put("OneStar", oneStarReviews);

// Build the item
Item item = new Item()
    .withPrimaryKey("Id", 123)
    .withString("Title", "Bicycle 123")
    .withString("Description", "123 description")
    .withString("BicycleType", "Hybrid")
    .withString("Brand", "Brand-Company C")
    .withNumber("Price", 500)
    .withStringSet("Color",  new HashSet<String>(Arrays.asList("Red", "Black")))
    .withString("ProductCategory", "Bicycle")
    .withBoolean("InStock", true)
    .withNull("QuantityOnHand")
    .withList("RelatedItems", relatedItems)
    .withMap("Pictures", pictures)
    .withMap("Reviews", reviews);

// Write the item to the table
PutItemOutcome outcome = table.putItem(item);
```

Im vorangegangenen Beispiel verfügt das Element über Attribute, die skalar sind (`String`, `Number`, `Boolean`, `Null`), Sätze (`String Set`) sowie Dokumenttypen (`List`, `Map`).

Angeben eines optionalen Parameters

Neben den erforderlichen Parametern können Sie auch optionale Parameter für die `putItem`-Methode angeben. Zum Beispiel verwendet das folgende Java-Codebeispiel einen optionalen Parameter, um eine Bedingung für das Hochladen des Elements anzugeben. Wenn die von Ihnen angegebene Bedingung nicht erfüllt ist, wird eine ausgelöst. AWS SDK für Java

`ConditionalCheckFailedException` Das Codebeispiel gibt die folgenden optionalen Parameter in der `putItem`-Methode an:

- Ein `ConditionExpression`, der die Bedingungen für die Anforderung definiert. In dem Codebeispiel ist die Bedingung definiert, zu der das vorhandene Element mit demselben Primärschlüssel nur dann ersetzt wird, wenn es über ein ISBN-Attribut verfügt, das mit einem bestimmten Wert übereinstimmt.
- Eine Zuordnung für `ExpressionAttributeValues`, die in der Bedingung verwendet wird. In diesem Fall ist nur eine Ersetzung erforderlich: Der Platzhalter `:val` im Bedingungsausdruck wird zur Laufzeit durch den tatsächlichen zu prüfenden ISBN-Wert ersetzt.

Im folgenden Beispiel wird ein neues Buchelement mit diesen optionalen Parameter hinzugefügt.

Example

```
Item item = new Item()
    .withPrimaryKey("Id", 104)
    .withString("Title", "Book 104 Title")
    .withString("ISBN", "444-4444444444")
    .withNumber("Price", 20)
    .withStringSet("Authors",
        new HashSet<String>(Arrays.asList("Author1", "Author2")));

Map<String, Object> expressionAttributeValues = new HashMap<String, Object>();
expressionAttributeValues.put(":val", "444-4444444444");

PutItemOutcome outcome = table.putItem(
    item,
    "ISBN = :val", // ConditionExpression parameter
    null,          // ExpressionAttributeNames parameter - we're not using it for this
 example
    expressionAttributeValues);
```

PutItem und JSON-Dokumente

Sie können ein JSON-Dokument in einer DynamoDB-Tabelle als Attribut speichern. Verwenden Sie dazu die `withJSON`-Methode `Item`. Diese Methode analysiert das JSON-Dokument und ordnet jedes Element einem nativen DynamoDB-Datentyp zu.

Angenommen, Sie möchten das folgende JSON-Dokument speichern, welches Anbieter enthält, die Bestellungen für ein bestimmtes Produkt erfüllen können.

Example

```
{
    "V01": {
        "Name": "Acme Books",
        "Offices": [ "Seattle" ]
    },
    "V02": {
        "Name": "New Publishers, Inc.",
        "Offices": ["London", "New York"
        ]
    },
    "V03": {
        "Name": "Better Buy Books",
        "Offices": [ "Tokyo", "Los Angeles", "Sydney"
        ]
    }
}
```

Sie können die `withJSON`-Methode verwenden, um es in der `ProductCatalog`-Tabelle in einem `Map´`-Attribut mit dem Namen `VendorInfo` zu speichern. Das folgende Java-Codebeispiel veranschaulicht dies.

```
// Convert the document into a String.  Must escape all double-quotes.
String vendorDocument = "{"
    + "    \"V01\": {"
    + "        \"Name\": \"Acme Books\","
    + "        \"Offices\": [ \"Seattle\" ]"
    + "    },"
    + "    \"V02\": {"
    + "        \"Name\": \"New Publishers, Inc.\","
    + "        \"Offices\": [ \"London\", \"New York\"" + "]" + "},"
    + "    \"V03\": {"
    + "        \"Name\": \"Better Buy Books\","
    +          "\"Offices\": [ \"Tokyo\", \"Los Angeles\", \"Sydney\""
    + "            ]"
    + "        }"
    + "    }";

Item item = new Item()
    .withPrimaryKey("Id", 210)
    .withString("Title", "Book 210 Title")
    .withString("ISBN", "210-2102102102")
```

```
    .withNumber("Price", 30)
    .withJSON("VendorInfo", vendorDocument);

PutItemOutcome outcome = table.putItem(item);
```

Abrufen eines Elements

Um ein einzelnes Element abzurufen, verwenden Sie die `getItem`-Methode eines `Table`-Objekts. Dazu gehen Sie wie folgt vor:

1. Erstellen Sie eine Instance der `DynamoDB`-Klasse.
2. Erstellen Sie eine Instance der `Table`-Klasse, um die Tabelle zu repräsentieren, mit der Sie arbeiten möchten.
3. Rufen Sie die `getItem`-Methode für die `Table`-Instance auf. Sie müssen den Primärschlüssel des Elements angeben, das Sie abrufen möchten.

Im folgenden Java-Codebeispiel werden die vorherigen Schritte veranschaulicht. Im Codebeispiel wird das Element abgerufen, das den angegebenen Partitionsschlüssel aufweist.

```
AmazonDynamoDB client = AmazonDynamoDBClientBuilder.standard().build();
DynamoDB dynamoDB = new DynamoDB(client);

Table table = dynamoDB.getTable("ProductCatalog");

Item item = table.getItem("Id", 210);
```

Angeben eines optionalen Parameters

Neben den erforderlichen Parametern können Sie auch optionale Parameter für die `getItem`-Methode angeben. Zum Beispiel verwendet das folgende Java-Codebeispiel eine optionale Methode zum Abrufen nur einer bestimmten Liste von Attributen und zum Angeben von Strongly Consistent-Lesevorgängen. (Weitere Informationen zur Lesekonsistenz finden Sie unter DynamoDB-Lesekonsistenz.)

Sie können einen `ProjectionExpression` verwenden, um nur spezifische Attribute oder Elemente anstelle eines ganzen Elements abzurufen. Ein `ProjectionExpression` kann Attribute auf oberster Ebene oder verschachtelte Attribute mithilfe von Dokumentpfaden angeben. Weitere Informationen finden Sie unter Verwenden von Projektionsausdrücken in DynamoDB.

Mit den Parametern der `getItem`-Methode kann keine Read Consistency angegeben werden. Sie können jedoch eine `GetItemSpec` erstellen, die vollständigen Zugriff auf die Eingaben für die `GetItem`-Low-Level-Operation bietet. Durch das Codebeispiel unten wird eine `GetItemSpec` erstellt. Es verwendet diese Spezifikation als Eingabe für die `getItem`-Methode.

Example

```
GetItemSpec spec = new GetItemSpec()
    .withPrimaryKey("Id", 206)
    .withProjectionExpression("Id, Title, RelatedItems[0], Reviews.FiveStar")
    .withConsistentRead(true);

Item item = table.getItem(spec);

System.out.println(item.toJSONPretty());
```

Um ein `Item` in einem vom Menschen lesbaren Format zu drucken, verwenden Sie die Methode `toJSONPretty`. Die Ausgabe aus dem vorherigen Beispiel sieht wie folgt aus.

```
{
  "RelatedItems" : [ 341 ],
  "Reviews" : {
    "FiveStar" : [ "Excellent! Can't recommend it highly enough! Buy it!", "Do yourself
 a favor and buy this" ]
  },
  "Id" : 123,
  "Title" : "20-Bicycle 123"
}
```

GetItem und JSON-Dokumente

Im Abschnitt PutItem und JSON-Dokumente haben Sie ein JSON-Dokument in einem `Map`-Attribut mit dem Namen `VendorInfo` gespeichert. Sie können die `getItem`-Methode verwenden, um das gesamte Dokument im JSON-Format abzurufen. Sie können auch die Dokumentpfadnotation verwenden, um nur einige der Elemente des Dokuments abzurufen. Das folgende Java-Codebeispiel veranschaulicht diese Vorgehensweisen.

```
GetItemSpec spec = new GetItemSpec()
    .withPrimaryKey("Id", 210);
```

```
System.out.println("All vendor info:");
spec.withProjectionExpression("VendorInfo");
System.out.println(table.getItem(spec).toJSON());

System.out.println("A single vendor:");
spec.withProjectionExpression("VendorInfo.V03");
System.out.println(table.getItem(spec).toJSON());

System.out.println("First office location for this vendor:");
spec.withProjectionExpression("VendorInfo.V03.Offices[0]");
System.out.println(table.getItem(spec).toJSON());
```

Die Ausgabe aus dem vorherigen Beispiel sieht wie folgt aus.

```
All vendor info:
{"VendorInfo":{"V03":{"Name":"Better Buy Books","Offices":["Tokyo","Los
 Angeles","Sydney"]},"V02":{"Name":"New Publishers, Inc.","Offices":["London","New
 York"]},"V01":{"Name":"Acme Books","Offices":["Seattle"]}}}
A single vendor:
{"VendorInfo":{"V03":{"Name":"Better Buy Books","Offices":["Tokyo","Los
 Angeles","Sydney"]}}}
First office location for a single vendor:
{"VendorInfo":{"V03":{"Offices":["Tokyo"]}}}
```

Note

Mit der `toJSON`-Methode können Sie jedes beliebige Element (oder seine Attribute) in eine JSON-formatierte Zeichenfolge umwandeln. Das folgende Codeausschnitt ruft mehrere Attribute auf oberster Ebene und verschachtelte Attribute ab und druckt die Ergebnisse im JSON-Format.

```
GetItemSpec spec = new GetItemSpec()
    .withPrimaryKey("Id", 210)
    .withProjectionExpression("VendorInfo.V01, Title, Price");

Item item = table.getItem(spec);
System.out.println(item.toJSON());
```

Die Ausgabe sieht wie folgt aus.

```
{"VendorInfo":{"V01":{"Name":"Acme Books","Offices":
["Seattle"]}},"Price":30,"Title":"Book 210 Title"}
```

Batch Write: Einfügen und Löschen mehrerer Elemente

Batch Write bezieht sich auf das Einfügen und Löschen mehrerer Elemente in einem Batch. Die `batchWriteItem`-Methode ermöglicht Ihnen das Einfügen und Löschen von mehreren Elementen aus einer oder mehreren Tabellen anhand eines einzigen Aufrufs. Im Folgenden finden Sie die Schritte zum Einfügen oder Löschen mehrerer Elemente mithilfe der AWS SDK für Java Dokument-API.

1. Erstellen Sie eine Instance der `DynamoDB`-Klasse.
2. Erstellen Sie eine Instance der `TableWriteItems`-Klasse, die alle Einfügen- und Löschen-Operationen für eine Tabelle beschreibt. Wenn Sie mit einer einzigen BatchWrite-Operation in mehrere Tabellen schreiben möchten, müssen Sie eine `TableWriteItems`-Instance pro Tabelle erstellen.
3. Rufen Sie die `batchWriteItem`-Methode auf, indem Sie das bzw. die `TableWriteItems`-Objekt(e) bereitstellen, das/die Sie im vorangegangenen Schritt erstellt haben.
4. Verarbeiten Sie die Antwort. Überprüfen Sie, ob in der Antwort nicht verarbeitete Anforderungselemente zurückgegeben wurden. Dies kann der Fall sein, wenn Sie das Kontingent für den bereitgestellten Durchsatz erreichen oder ein anderer vorübergehender Fehler auftritt. Außerdem begrenzt DynamoDB die Anforderungsgröße und die Anzahl der Vorgänge, die Sie in einer Anforderung angeben können. Wenn Sie diese Limits überschreiten, wird die Anforderung von DynamoDB abgelehnt. Weitere Informationen finden Sie unter Kontingente in Amazon DynamoDB.

Im folgenden Java-Codebeispiel werden die vorherigen Schritte veranschaulicht. Das folgende Beispiel führt eine `batchWriteItem`-Operation für zwei Tabellen durch: `Forum` und `Thread`. Die entsprechenden `TableWriteItems`-Objekte definieren die folgenden Aktionen:

- Einfügen eines Elements in die `Forum`-Tabelle
- Einfügen und Löschen eines Elements in der `Thread`-Tabelle

Der Code ruft dann `batchWriteItem` auf, um die Operation durchzuführen.

```
AmazonDynamoDB client = AmazonDynamoDBClientBuilder.standard().build();
DynamoDB dynamoDB = new DynamoDB(client);

TableWriteItems forumTableWriteItems = new TableWriteItems("Forum")
    .withItemsToPut(
        new Item()
            .withPrimaryKey("Name", "Amazon RDS")
            .withNumber("Threads", 0));

TableWriteItems threadTableWriteItems = new TableWriteItems("Thread")
    .withItemsToPut(
        new Item()
            .withPrimaryKey("ForumName","Amazon RDS","Subject","Amazon RDS Thread 1")
    .withHashAndRangeKeysToDelete("ForumName","Some partition key value", "Amazon S3",
 "Some sort key value");

BatchWriteItemOutcome outcome = dynamoDB.batchWriteItem(forumTableWriteItems,
 threadTableWriteItems);

// Code for checking unprocessed items is omitted in this example
```

Ein funktionierendes Beispiel finden Sie unter Beispiel: Batch-Schreibvorgang mit der AWS SDK für Java Dokument-API.

Batch Get: Abrufen mehrerer Elemente

Die `batchGetItem`-Methode ermöglicht Ihnen das Abrufen mehrerer Elemente aus einer oder mehreren Tabellen. Um ein einzelnes Element abzurufen, können Sie die `getItem`-Methode verwenden.

Dazu gehen Sie wie folgt vor:

1. Erstellen Sie eine Instance der `DynamoDB`-Klasse.
2. Erstellen Sie eine Instance der `TableKeysAndAttributes`-Klasse, die eine Liste der Primärschlüsselwerte beschreibt, die von einer Tabelle abgerufen werden sollen. Wenn Sie mit einer einzigen BatchGet-Operation aus mehreren Tabellen lesen möchten, müssen Sie eine `TableKeysAndAttributes`-Instance pro Tabelle erstellen.
3. Rufen Sie die `batchGetItem`-Methode auf, indem Sie das bzw. die `TableKeysAndAttributes`-Objekt(e) bereitstellen, das/die Sie im vorangegangenen Schritt erstellt haben.

Im folgenden Java-Codebeispiel werden die vorherigen Schritte veranschaulicht. In dem Beispiel werden zwei Elemente aus der `Forum`-Tabelle und drei Elemente aus der `Thread`-Tabelle abgerufen.

```
AmazonDynamoDB client = AmazonDynamoDBClientBuilder.standard().build();
DynamoDB dynamoDB = new DynamoDB(client);

    TableKeysAndAttributes forumTableKeysAndAttributes = new
 TableKeysAndAttributes(forumTableName);
    forumTableKeysAndAttributes.addHashOnlyPrimaryKeys("Name",
    "Amazon S3",
    "Amazon DynamoDB");

TableKeysAndAttributes threadTableKeysAndAttributes = new
 TableKeysAndAttributes(threadTableName);
threadTableKeysAndAttributes.addHashAndRangePrimaryKeys("ForumName", "Subject",
    "Amazon DynamoDB","DynamoDB Thread 1",
    "Amazon DynamoDB","DynamoDB Thread 2",
    "Amazon S3","S3 Thread 1");

BatchGetItemOutcome outcome = dynamoDB.batchGetItem(
    forumTableKeysAndAttributes, threadTableKeysAndAttributes);

for (String tableName : outcome.getTableItems().keySet()) {
    System.out.println("Items in table " + tableName);
    List<Item> items = outcome.getTableItems().get(tableName);
    for (Item item : items) {
        System.out.println(item);
    }
}
```

Angeben eines optionalen Parameters

Neben den erforderlichen Parametern können Sie bei Verwendung von `batchGetItem` auch optionale Parameter angeben. Beispielsweise können Sie einen `ProjectionExpression` mit jedem `TableKeysAndAttributes`, das Sie definieren, bereitstellen. So können Sie die Attribute angeben, die aus der Tabelle abgerufen werden sollen.

Im folgenden C#-Codebeispiel werden zwei Elemente aus der `Forum`-Tabelle abgerufen. Der Parameter `withProjectionExpression` gibt an, dass nur das Attribut `Threads` abgerufen werden soll.

Example

```
TableKeysAndAttributes forumTableKeysAndAttributes = new
 TableKeysAndAttributes("Forum")
    .withProjectionExpression("Threads");

forumTableKeysAndAttributes.addHashOnlyPrimaryKeys("Name",
    "Amazon S3",
    "Amazon DynamoDB");

BatchGetItemOutcome outcome = dynamoDB.batchGetItem(forumTableKeysAndAttributes);
```

Aktualisieren eines Elements

Die `updateItem`-Methode eines `Table`-Objekts kann vorhandene Attributwerte aktualisieren, neue Attribute hinzuzufügen oder Attribute aus einem vorhandenen Element löschen.

Die `updateItem`-Methode verhält sich wie folgt:

- Wenn kein Element vorhanden ist (kein Element in der Tabelle mit dem angegebenen Primärschlüssel), fügt `updateItem` der Tabelle ein neues Element hinzu.
- Wenn ein Element vorhanden ist, führt `updateItem` die Aktualisierung wie vom `UpdateExpression`-Parameter angegeben aus.

> **Note**
>
> Es ist auch möglich, ein Element mit `putItem` zu "aktualisieren". Wenn Sie z. B. `putItem` aufrufen, um der Tabelle ein Element hinzuzufügen, aber bereits ein Element mit dem angegebenen Primärschlüssel vorhanden ist, wird das gesamte Element von `putItem` ersetzt. Wenn Attribute in dem vorhandenen Element nicht in der Eingabe angegeben sind, werden diese Attribute von `putItem` aus dem Element entfernt.
> Im Allgemeinen empfehlen wir, `updateItem` immer dann zu verwenden, wenn Sie Elementattribute ändern möchten. Die `updateItem`-Methode ändert nur die Elementattribute, die Sie in der Eingabe angeben. Die anderen Attribute im Element bleiben unverändert.

Dazu gehen Sie wie folgt vor:

1. Erstellen Sie eine Instance der `Table`-Klasse, um die Tabelle zu repräsentieren, mit der Sie arbeiten möchten.
2. Rufen Sie die `updateTable`-Methode für die `Table`-Instance auf. Sie müssen den Primärschlüssel des abzurufenden Elements und einen `UpdateExpression` angeben, der die zu ändernden Attribute beschreibt und mitteilt, wie diese geändert werden sollen.

Im folgenden Java-Codebeispiel werden die vorherigen Aufgaben veranschaulicht. Das Beispiel aktualisiert ein Buchelement in der `ProductCatalog`-Tabelle. Es wird ein neuer Autor zum Satz der `Authors` hinzugefügt und das vorhandene `ISBN`-Attribut wird gelöscht. Darüber hinaus wird auch der Preis um eins gesenkt.

Im `ExpressionAttributeValues` wird eine `UpdateExpression`-Zuordnung verwendet. Die Platzhalter `:val1` und `:val2` werden zur Laufzeit durch die tatsächlichen Werte für `Authors` und `Price` ersetzt.

```
AmazonDynamoDB client = AmazonDynamoDBClientBuilder.standard().build();
DynamoDB dynamoDB = new DynamoDB(client);

Table table = dynamoDB.getTable("ProductCatalog");

Map<String, String> expressionAttributeNames = new HashMap<String, String>();
expressionAttributeNames.put("#A", "Authors");
expressionAttributeNames.put("#P", "Price");
expressionAttributeNames.put("#I", "ISBN");

Map<String, Object> expressionAttributeValues = new HashMap<String, Object>();
expressionAttributeValues.put(":val1",
    new HashSet<String>(Arrays.asList("Author YY","Author ZZ")));
expressionAttributeValues.put(":val2", 1);   //Price

UpdateItemOutcome outcome =  table.updateItem(
    "Id",          // key attribute name
    101,           // key attribute value
    "add #A :val1 set #P = #P - :val2 remove #I", // UpdateExpression
    expressionAttributeNames,
    expressionAttributeValues);
```

Angeben eines optionalen Parameters

Neben den erforderlichen Parametern können Sie auch optionale Parameter für die `updateItem`-Methode angeben, z. B. eine Bedingung, die erfüllt werden muss, damit die Aktualisierung erfolgt. Wenn die von Ihnen angegebene Bedingung nicht erfüllt ist, wird AWS SDK für Java eine `ConditionalCheckFailedException` ausgelöst. Beim folgenden Java-Codebeispiel wird der Preis eines Buchelements bedingungsabhängig auf 25 erhöht. Das Beispiel zeigt einen `ConditionExpression`, der besagt, dass der Preis nur aktualisiert werden soll, wenn der vorhandene Preis 20 lautet.

Example

```
Table table = dynamoDB.getTable("ProductCatalog");

Map<String, String> expressionAttributeNames = new HashMap<String, String>();
expressionAttributeNames.put("#P", "Price");

Map<String, Object> expressionAttributeValues = new HashMap<String, Object>();
expressionAttributeValues.put(":val1", 25);  // update Price to 25...
expressionAttributeValues.put(":val2", 20);  //...but only if existing Price is 20

UpdateItemOutcome outcome = table.updateItem(
    new PrimaryKey("Id",101),
    "set #P = :val1", // UpdateExpression
    "#P = :val2",     // ConditionExpression
    expressionAttributeNames,
    expressionAttributeValues);
```

Unteilbarer Zähler

Sie können `updateItem` verwenden, um einen unteilbaren Zähler zu implementieren, mit dem der Wert eines bestehenden Attributs erhöht oder verringert wird, ohne andere Schreibanforderungen zu beeinflussen. Zum Erhöhen eines unteilbaren Zählers verwenden Sie einen `UpdateExpression` mit einer `set`-Aktion, um einem vorhandenen Attribut vom Typ `Number` einen numerischen Wert hinzuzufügen.

Das folgende Codebeispiel zeigt diesen Vorgang und erhöht das Attribut `Quantity` um eins. Es zeigt auch die Verwendung des Parameters `ExpressionAttributeNames` in einem `UpdateExpression`.

```
Table table = dynamoDB.getTable("ProductCatalog");
```

```
Map<String,String> expressionAttributeNames = new HashMap<String,String>();
expressionAttributeNames.put("#p", "PageCount");

Map<String,Object> expressionAttributeValues = new HashMap<String,Object>();
expressionAttributeValues.put(":val", 1);

UpdateItemOutcome outcome = table.updateItem(
    "Id", 121,
    "set #p = #p + :val",
    expressionAttributeNames,
    expressionAttributeValues);
```

Löschen eines Elements

Die `deleteItem`-Methode löscht ein Element aus einer Tabelle. Sie müssen den Primärschlüssel des Elements bereitstellen, das Sie löschen möchten.

Dazu gehen Sie wie folgt vor:

1. Erstellen Sie eine Instance des `DynamoDB`-Clients.
2. Rufen Sie die `deleteItem`-Methode durch Angabe des Schlüssels des Elements auf, das Sie löschen möchten.

Das folgende Java-Codebeispiel veranschaulicht diese Vorgehensweisen.

Example

```
AmazonDynamoDB client = AmazonDynamoDBClientBuilder.standard().build();
DynamoDB dynamoDB = new DynamoDB(client);

Table table = dynamoDB.getTable("ProductCatalog");

DeleteItemOutcome outcome = table.deleteItem("Id", 101);
```

Angeben eines optionalen Parameters

Sie können für `deleteItem` optionale Parameter angeben. Das folgende Java-Codebeispiel enthält einen `ConditionExpression`, der besagt, dass ein Buchelement im `InPublication` nur dann gelöscht werden darf, wenn das Buch nicht mehr veröffentlicht wird (das Attribut `ProductCatalog` ist False).

Example

```
Map<String,Object> expressionAttributeValues = new HashMap<String,Object>();
expressionAttributeValues.put(":val", false);

DeleteItemOutcome outcome = table.deleteItem("Id",103,
    "InPublication = :val",
    null, // ExpressionAttributeNames - not used in this example
    expressionAttributeValues);
```

Beispiel: CRUD-Operationen mit der AWS SDK für Java Dokument-API

Das folgende Codebeispiel veranschaulicht CRUD-Operationen für ein Amazon-DynamoDB-Element. In dem Beispiel wird ein Element erstellt, abgerufen, es werden verschiedene Aktualisierungen durchgeführt und es wird schließlich gelöscht.

> **Note**
>
> Das SDK für Java stellt auch ein Objektpersistenzmodell bereit, mit dem Sie Ihre clientseitigen Klassen DynamoDB-Tabellen zuordnen können. Mit diesem Ansatz können Sie die Codemenge, die Sie schreiben müssen, verringern. Weitere Informationen finden Sie unter Java 1.x: Dynamo DBMapper.

> **Note**
>
> In diesem Codebeispiel wird davon ausgegangen, dass Sie bereits Daten für Ihr Konto in DynamoDB geladen haben, indem Sie die Anweisungen im Erstellen von Tabellen und Laden von Daten für Codebeispiele in DynamoDB-Abschnitt befolgen.
> step-by-stepAnweisungen zur Ausführung des folgenden Beispiels finden Sie unterJava-Codebeispiele.

```
package com.amazonaws.codesamples.document;

import java.io.IOException;
import java.util.Arrays;
import java.util.HashMap;
```

```
import java.util.HashSet;
import java.util.Map;

import com.amazonaws.services.dynamodbv2.AmazonDynamoDB;
import com.amazonaws.services.dynamodbv2.AmazonDynamoDBClientBuilder;
import com.amazonaws.services.dynamodbv2.document.DeleteItemOutcome;
import com.amazonaws.services.dynamodbv2.document.DynamoDB;
import com.amazonaws.services.dynamodbv2.document.Item;
import com.amazonaws.services.dynamodbv2.document.Table;
import com.amazonaws.services.dynamodbv2.document.UpdateItemOutcome;
import com.amazonaws.services.dynamodbv2.document.spec.DeleteItemSpec;
import com.amazonaws.services.dynamodbv2.document.spec.UpdateItemSpec;
import com.amazonaws.services.dynamodbv2.document.utils.NameMap;
import com.amazonaws.services.dynamodbv2.document.utils.ValueMap;
import com.amazonaws.services.dynamodbv2.model.ReturnValue;

public class DocumentAPIItemCRUDExample {

    static AmazonDynamoDB client = AmazonDynamoDBClientBuilder.standard().build();
    static DynamoDB dynamoDB = new DynamoDB(client);

    static String tableName = "ProductCatalog";

    public static void main(String[] args) throws IOException {

        createItems();

        retrieveItem();

        // Perform various updates.
        updateMultipleAttributes();
        updateAddNewAttribute();
        updateExistingAttributeConditionally();

        // Delete the item.
        deleteItem();

    }

    private static void createItems() {

        Table table = dynamoDB.getTable(tableName);
        try {
```

```
            Item item = new Item().withPrimaryKey("Id", 120).withString("Title", "Book
120 Title")
                    .withString("ISBN", "120-1111111111")
                    .withStringSet("Authors", new
HashSet<String>(Arrays.asList("Author12", "Author22")))
                    .withNumber("Price", 20).withString("Dimensions",
"8.5x11.0x.75").withNumber("PageCount", 500)
                    .withBoolean("InPublication", false).withString("ProductCategory",
"Book");
            table.putItem(item);

            item = new Item().withPrimaryKey("Id", 121).withString("Title", "Book 121
Title")
                    .withString("ISBN", "121-1111111111")
                    .withStringSet("Authors", new
HashSet<String>(Arrays.asList("Author21", "Author 22")))
                    .withNumber("Price", 20).withString("Dimensions",
"8.5x11.0x.75").withNumber("PageCount", 500)
                    .withBoolean("InPublication", true).withString("ProductCategory",
"Book");
            table.putItem(item);

        } catch (Exception e) {
            System.err.println("Create items failed.");
            System.err.println(e.getMessage());

        }
    }

    private static void retrieveItem() {
        Table table = dynamoDB.getTable(tableName);

        try {

            Item item = table.getItem("Id", 120, "Id, ISBN, Title, Authors", null);

            System.out.println("Printing item after retrieving it....");
            System.out.println(item.toJSONPretty());

        } catch (Exception e) {
            System.err.println("GetItem failed.");
            System.err.println(e.getMessage());
        }
```

```
    }

    private static void updateAddNewAttribute() {
        Table table = dynamoDB.getTable(tableName);

        try {

            UpdateItemSpec updateItemSpec = new UpdateItemSpec().withPrimaryKey("Id",
121)
                    .withUpdateExpression("set #na = :val1").withNameMap(new
NameMap().with("#na", "NewAttribute"))
                    .withValueMap(new ValueMap().withString(":val1", "Some value"))
                    .withReturnValues(ReturnValue.ALL_NEW);

            UpdateItemOutcome outcome = table.updateItem(updateItemSpec);

            // Check the response.
            System.out.println("Printing item after adding new attribute...");
            System.out.println(outcome.getItem().toJSONPretty());

        } catch (Exception e) {
            System.err.println("Failed to add new attribute in " + tableName);
            System.err.println(e.getMessage());
        }
    }

    private static void updateMultipleAttributes() {

        Table table = dynamoDB.getTable(tableName);

        try {

            UpdateItemSpec updateItemSpec = new UpdateItemSpec().withPrimaryKey("Id",
120)
                    .withUpdateExpression("add #a :val1 set #na=:val2")
                    .withNameMap(new NameMap().with("#a", "Authors").with("#na",
"NewAttribute"))
                    .withValueMap(
                            new ValueMap().withStringSet(":val1", "Author YY", "Author
ZZ").withString(":val2",
                                    "someValue"))
                    .withReturnValues(ReturnValue.ALL_NEW);

            UpdateItemOutcome outcome = table.updateItem(updateItemSpec);
```

```
            // Check the response.
            System.out.println("Printing item after multiple attribute update...");
            System.out.println(outcome.getItem().toJSONPretty());

        } catch (Exception e) {
            System.err.println("Failed to update multiple attributes in " + tableName);
            System.err.println(e.getMessage());

        }
    }

    private static void updateExistingAttributeConditionally() {

        Table table = dynamoDB.getTable(tableName);

        try {

            // Specify the desired price (25.00) and also the condition (price =
            // 20.00)

            UpdateItemSpec updateItemSpec = new UpdateItemSpec().withPrimaryKey("Id",
120)
                    .withReturnValues(ReturnValue.ALL_NEW).withUpdateExpression("set #p
= :val1")
                    .withConditionExpression("#p = :val2").withNameMap(new
NameMap().with("#p", "Price"))
                    .withValueMap(new ValueMap().withNumber(":val1",
25).withNumber(":val2", 20));

            UpdateItemOutcome outcome = table.updateItem(updateItemSpec);

            // Check the response.
            System.out.println("Printing item after conditional update to new
attribute...");
            System.out.println(outcome.getItem().toJSONPretty());

        } catch (Exception e) {
            System.err.println("Error updating item in " + tableName);
            System.err.println(e.getMessage());
        }
    }

    private static void deleteItem() {
```

```
        Table table = dynamoDB.getTable(tableName);

        try {

            DeleteItemSpec deleteItemSpec = new DeleteItemSpec().withPrimaryKey("Id",
 120)
                    .withConditionExpression("#ip = :val").withNameMap(new
 NameMap().with("#ip", "InPublication"))
                    .withValueMap(new ValueMap().withBoolean(":val",
 false)).withReturnValues(ReturnValue.ALL_OLD);

            DeleteItemOutcome outcome = table.deleteItem(deleteItemSpec);

            // Check the response.
            System.out.println("Printing item that was deleted...");
            System.out.println(outcome.getItem().toJSONPretty());

        } catch (Exception e) {
            System.err.println("Error deleting item in " + tableName);
            System.err.println(e.getMessage());
        }
    }
}
```

Beispiel: Batch-Operationen mithilfe der AWS SDK für Java Dokument-API

Dieser Abschnitt enthält Beispiele für Batch-Write- und Batch-Abruf-Operationen in Amazon DynamoDB mithilfe der AWS SDK für Java Document API.

> **Note**
>
> Das SDK für Java stellt auch ein Objektpersistenzmodell bereit, mit dem Sie Ihre clientseitigen Klassen DynamoDB-Tabellen zuordnen können. Mit diesem Ansatz können Sie die Codemenge, die Sie schreiben müssen, verringern. Weitere Informationen finden Sie unter Java 1.x: Dynamo DBMapper.

Themen

- Beispiel: Batch-Schreibvorgang mit der AWS SDK für Java Dokument-API

- Beispiel: Batch-Abruf-Vorgang mit der AWS SDK für Java Dokument-API

Beispiel: Batch-Schreibvorgang mit der AWS SDK für Java Dokument-API

Das folgende Java-Beispielcode verwendet die `batchWriteItem`-Methode zur Durchführung der folgenden Operationen zum Einfügen und Löschen:

- Einfügen eines Elements in die `Forum`-Tabelle
- Einfügen und Löschen eines Elements aus der `Thread`-Tabelle

Bei der Erstellung Ihrer BatchWrite-Anforderung können Sie eine beliebige Anzahl von Einfüge- und Lösch-Anforderungen angeben. `batchWriteItem` begrenzt jedoch die Größe einer BatchWrite-Anforderung sowie die Anzahl der Einfüge- und Löschoperationen in einer einzelnen BatchWrite-Operation. Wenn Ihre Anforderung diese Grenzwerte überschreitet, wird sie abgelehnt. Wenn die Tabelle nicht über ausreichend bereitgestellten Durchsatz für diese Anforderung verfügt, werden nicht verarbeitete Anforderungselemente in der Antwort zurückgegeben.

Im folgenden Beispiel wird die Antwort auf unverarbeitete Anforderungselemente überprüft. Liegen unverarbeitete Elemente vor, wird der Vorgang wiederholt und die `batchWriteItem`-Anforderung mit unverarbeiteten Elementen wird erneut gesendet. Wenn Sie die Beispiele in dieser Anleitung befolgt haben, sollten Sie die Tabellen `Forum` und `Thread` bereits erstellt haben. Sie können diese Tabellen auch programmgesteuert erstellen und auf die gleiche Weise Beispieldaten hochladen. Weitere Informationen finden Sie unter Erstellen von Beispieltabellen und Hochladen von Daten mit dem AWS SDK für Java.

step-by-stepAnweisungen zum Testen des folgenden Beispiels finden Sie unterJava-Codebeispiele.

Example

```
package com.amazonaws.codesamples.document;

import java.io.IOException;
import java.util.Arrays;
import java.util.HashSet;
import java.util.List;
import java.util.Map;

import com.amazonaws.services.dynamodbv2.AmazonDynamoDB;
import com.amazonaws.services.dynamodbv2.AmazonDynamoDBClientBuilder;
```

```
import com.amazonaws.services.dynamodbv2.document.BatchWriteItemOutcome;
import com.amazonaws.services.dynamodbv2.document.DynamoDB;
import com.amazonaws.services.dynamodbv2.document.Item;
import com.amazonaws.services.dynamodbv2.document.TableWriteItems;
import com.amazonaws.services.dynamodbv2.model.WriteRequest;

public class DocumentAPIBatchWrite {

    static AmazonDynamoDB client = AmazonDynamoDBClientBuilder.standard().build();
    static DynamoDB dynamoDB = new DynamoDB(client);

    static String forumTableName = "Forum";
    static String threadTableName = "Thread";

    public static void main(String[] args) throws IOException {

        writeMultipleItemsBatchWrite();

    }

    private static void writeMultipleItemsBatchWrite() {
        try {

            // Add a new item to Forum
            TableWriteItems forumTableWriteItems = new
 TableWriteItems(forumTableName) // Forum
                    .withItemsToPut(new Item().withPrimaryKey("Name", "Amazon
 RDS").withNumber("Threads", 0));

            // Add a new item, and delete an existing item, from Thread
            // This table has a partition key and range key, so need to specify
            // both of them
            TableWriteItems threadTableWriteItems = new
 TableWriteItems(threadTableName)
                    .withItemsToPut(
                            new Item().withPrimaryKey("ForumName", "Amazon RDS",
 "Subject", "Amazon RDS Thread 1")
                                    .withString("Message", "ElastiCache Thread 1
 message")
                                    .withStringSet("Tags", new
 HashSet<String>(Arrays.asList("cache", "in-memory"))))
                    .withHashAndRangeKeysToDelete("ForumName", "Subject", "Amazon S3",
 "S3 Thread 100");
```

```
            System.out.println("Making the request.");
            BatchWriteItemOutcome outcome =
 dynamoDB.batchWriteItem(forumTableWriteItems, threadTableWriteItems);

            do {

                // Check for unprocessed keys which could happen if you exceed
                // provisioned throughput

                Map<String, List<WriteRequest>> unprocessedItems =
 outcome.getUnprocessedItems();

                if (outcome.getUnprocessedItems().size() == 0) {
                    System.out.println("No unprocessed items found");
                } else {
                    System.out.println("Retrieving the unprocessed items");
                    outcome = dynamoDB.batchWriteItemUnprocessed(unprocessedItems);
                }

            } while (outcome.getUnprocessedItems().size() > 0);

        } catch (Exception e) {
            System.err.println("Failed to retrieve items: ");
            e.printStackTrace(System.err);
        }

    }

}
```

Beispiel: Batch-Abruf-Vorgang mit der AWS SDK für Java Dokument-API

Das folgende Java-Codebeispiel verwendet die `batchGetItem`-Methode zum Abrufen mehrerer Elemente aus den Tabellen `Forum` und `Thread`. Die `BatchGetItemRequest` gibt die Tabellennamen sowie eine Liste der Schlüssel für jedes abzurufende Element an. In dem Beispiel wird die Antwort durch Drucken der abgerufenen Elemente verarbeitet.

> **Note**
>
> In diesem Codebeispiel wird davon ausgegangen, dass Sie bereits Daten für Ihr Konto in DynamoDB geladen haben, indem Sie die Anweisungen im Erstellen von Tabellen und Laden von Daten für Codebeispiele in DynamoDB-Abschnitt befolgen.
> step-by-stepAnweisungen zur Ausführung des folgenden Beispiels finden Sie unterJava-Codebeispiele.

Example

```
package com.amazonaws.codesamples.document;

import java.io.IOException;
import java.util.List;
import java.util.Map;

import com.amazonaws.services.dynamodbv2.AmazonDynamoDB;
import com.amazonaws.services.dynamodbv2.AmazonDynamoDBClientBuilder;
import com.amazonaws.services.dynamodbv2.document.BatchGetItemOutcome;
import com.amazonaws.services.dynamodbv2.document.DynamoDB;
import com.amazonaws.services.dynamodbv2.document.Item;
import com.amazonaws.services.dynamodbv2.document.TableKeysAndAttributes;
import com.amazonaws.services.dynamodbv2.model.KeysAndAttributes;

public class DocumentAPIBatchGet {
    static AmazonDynamoDB client = AmazonDynamoDBClientBuilder.standard().build();
    static DynamoDB dynamoDB = new DynamoDB(client);

    static String forumTableName = "Forum";
    static String threadTableName = "Thread";

    public static void main(String[] args) throws IOException {
        retrieveMultipleItemsBatchGet();
    }

    private static void retrieveMultipleItemsBatchGet() {

        try {
```

```
            TableKeysAndAttributes forumTableKeysAndAttributes = new
TableKeysAndAttributes(forumTableName);
            // Add a partition key
            forumTableKeysAndAttributes.addHashOnlyPrimaryKeys("Name", "Amazon S3",
"Amazon DynamoDB");

            TableKeysAndAttributes threadTableKeysAndAttributes = new
TableKeysAndAttributes(threadTableName);
            // Add a partition key and a sort key
            threadTableKeysAndAttributes.addHashAndRangePrimaryKeys("ForumName",
"Subject", "Amazon DynamoDB",
                    "DynamoDB Thread 1", "Amazon DynamoDB", "DynamoDB Thread 2",
"Amazon S3", "S3 Thread 1");

            System.out.println("Making the request.");

            BatchGetItemOutcome outcome =
dynamoDB.batchGetItem(forumTableKeysAndAttributes,
                    threadTableKeysAndAttributes);

            Map<String, KeysAndAttributes> unprocessed = null;

            do {
                for (String tableName : outcome.getTableItems().keySet()) {
                    System.out.println("Items in table " + tableName);
                    List<Item> items = outcome.getTableItems().get(tableName);
                    for (Item item : items) {
                        System.out.println(item.toJSONPretty());
                    }
                }

                // Check for unprocessed keys which could happen if you exceed
                // provisioned
                // throughput or reach the limit on response size.
                unprocessed = outcome.getUnprocessedKeys();

                if (unprocessed.isEmpty()) {
                    System.out.println("No unprocessed keys found");
                } else {
                    System.out.println("Retrieving the unprocessed keys");
                    outcome = dynamoDB.batchGetItemUnprocessed(unprocessed);
                }

            } while (!unprocessed.isEmpty());
```

```
        } catch (Exception e) {
            System.err.println("Failed to retrieve items.");
            System.err.println(e.getMessage());
        }

    }

}
```

Beispiel: Umgang mit binären Typattributen mithilfe der AWS SDK für Java Dokument-API

Das folgende Java-Codebeispiel veranschaulicht die Verarbeitung von Attributen des Typs Binärzahl. Bei dem Beispiel wird der `Reply`-Tabelle ein Element hinzugefügt. Das Element enthält ein Binärtypattribut (`ExtendedMessage`), in dem komprimierte Daten gespeichert sind. Daraufhin wird das Element abgerufen und es werden alle Attributwerte gedruckt. Zur Veranschaulichung wird in dem Beispiel die `GZIPOutputStream`-Klasse verwendet, um einen Beispiel-Stream zu komprimieren und dem `ExtendedMessage`-Attribut zuzuweisen. Wenn das Binärattribut abgerufen wird, wird es mit der `GZIPInputStream`-Klasse dekomprimiert.

> **Note**
>
> Das SDK für Java stellt auch ein Objektpersistenzmodell bereit, mit dem Sie Ihre clientseitigen Klassen DynamoDB-Tabellen zuordnen können. Mit diesem Ansatz können Sie die Codemenge, die Sie schreiben müssen, verringern. Weitere Informationen finden Sie unter Java 1.x: Dynamo DBMapper.

Wenn Sie die Schritte im Abschnitt Erstellen von Tabellen und Laden von Daten für Codebeispiele in DynamoDB befolgt haben, sollten Sie die `Reply`-Tabelle bereits erstellt haben. Sie können diese Tabelle auch programmgesteuert erstellen. Weitere Informationen finden Sie unter Erstellen von Beispieltabellen und Hochladen von Daten mit dem AWS SDK für Java.

step-by-stepAnweisungen zum Testen des folgenden Beispiels finden Sie unterJava-Codebeispiele.

Example

```
package com.amazonaws.codesamples.document;

import java.io.ByteArrayInputStream;
import java.io.ByteArrayOutputStream;
import java.io.IOException;
import java.nio.ByteBuffer;
import java.text.SimpleDateFormat;
import java.util.Date;
import java.util.TimeZone;
import java.util.zip.GZIPInputStream;
import java.util.zip.GZIPOutputStream;

import com.amazonaws.services.dynamodbv2.AmazonDynamoDB;
import com.amazonaws.services.dynamodbv2.AmazonDynamoDBClientBuilder;
import com.amazonaws.services.dynamodbv2.document.DynamoDB;
import com.amazonaws.services.dynamodbv2.document.Item;
import com.amazonaws.services.dynamodbv2.document.Table;
import com.amazonaws.services.dynamodbv2.document.spec.GetItemSpec;

public class DocumentAPIItemBinaryExample {

    static AmazonDynamoDB client = AmazonDynamoDBClientBuilder.standard().build();
    static DynamoDB dynamoDB = new DynamoDB(client);

    static String tableName = "Reply";
    static SimpleDateFormat dateFormatter = new SimpleDateFormat("yyyy-MM-
dd'T'HH:mm:ss.SSS'Z'");

    public static void main(String[] args) throws IOException {
        try {

            // Format the primary key values
            String threadId = "Amazon DynamoDB#DynamoDB Thread 2";

            dateFormatter.setTimeZone(TimeZone.getTimeZone("UTC"));
            String replyDateTime = dateFormatter.format(new Date());

            // Add a new reply with a binary attribute type
            createItem(threadId, replyDateTime);

            // Retrieve the reply with a binary attribute type
            retrieveItem(threadId, replyDateTime);

            // clean up by deleting the item
```

```
            deleteItem(threadId, replyDateTime);
        } catch (Exception e) {
            System.err.println("Error running the binary attribute type example: " +
e);
            e.printStackTrace(System.err);
        }
    }

    public static void createItem(String threadId, String replyDateTime) throws
IOException {

        Table table = dynamoDB.getTable(tableName);

        // Craft a long message
        String messageInput = "Long message to be compressed in a lengthy forum reply";

        // Compress the long message
        ByteBuffer compressedMessage = compressString(messageInput.toString());

        table.putItem(new Item().withPrimaryKey("Id",
threadId).withString("ReplyDateTime", replyDateTime)
                .withString("Message", "Long message
follows").withBinary("ExtendedMessage", compressedMessage)
                .withString("PostedBy", "User A"));
    }

    public static void retrieveItem(String threadId, String replyDateTime) throws
IOException {

        Table table = dynamoDB.getTable(tableName);

        GetItemSpec spec = new GetItemSpec().withPrimaryKey("Id", threadId,
"ReplyDateTime", replyDateTime)
                .withConsistentRead(true);

        Item item = table.getItem(spec);

        // Uncompress the reply message and print
        String uncompressed =
uncompressString(ByteBuffer.wrap(item.getBinary("ExtendedMessage")));

        System.out.println("Reply message:\n" + " Id: " + item.getString("Id") + "\n" +
" ReplyDateTime: "
```

```
                + item.getString("ReplyDateTime") + "\n" + " PostedBy: " +
item.getString("PostedBy") + "\n"
                + " Message: "
                + item.getString("Message") + "\n" + " ExtendedMessage (uncompressed):
" + uncompressed + "\n");
    }

    public static void deleteItem(String threadId, String replyDateTime) {

        Table table = dynamoDB.getTable(tableName);
        table.deleteItem("Id", threadId, "ReplyDateTime", replyDateTime);
    }

    private static ByteBuffer compressString(String input) throws IOException {
        // Compress the UTF-8 encoded String into a byte[]
        ByteArrayOutputStream baos = new ByteArrayOutputStream();
        GZIPOutputStream os = new GZIPOutputStream(baos);
        os.write(input.getBytes("UTF-8"));
        os.close();
        baos.close();
        byte[] compressedBytes = baos.toByteArray();

        // The following code writes the compressed bytes to a ByteBuffer.
        // A simpler way to do this is by simply calling
        // ByteBuffer.wrap(compressedBytes);
        // However, the longer form below shows the importance of resetting the
        // position of the buffer
        // back to the beginning of the buffer if you are writing bytes directly
        // to it, since the SDK
        // will consider only the bytes after the current position when sending
        // data to DynamoDB.
        // Using the "wrap" method automatically resets the position to zero.
        ByteBuffer buffer = ByteBuffer.allocate(compressedBytes.length);
        buffer.put(compressedBytes, 0, compressedBytes.length);
        buffer.position(0); // Important: reset the position of the ByteBuffer
                            // to the beginning
        return buffer;
    }

    private static String uncompressString(ByteBuffer input) throws IOException {
        byte[] bytes = input.array();
        ByteArrayInputStream bais = new ByteArrayInputStream(bytes);
        ByteArrayOutputStream baos = new ByteArrayOutputStream();
        GZIPInputStream is = new GZIPInputStream(bais);
```

```
        int chunkSize = 1024;
        byte[] buffer = new byte[chunkSize];
        int length = 0;
        while ((length = is.read(buffer, 0, chunkSize)) != -1) {
            baos.write(buffer, 0, length);
        }

        String result = new String(baos.toByteArray(), "UTF-8");

        is.close();
        baos.close();
        bais.close();

        return result;
    }
}
```

Arbeiten mit Elementen: .NET

Sie können die AWS SDK für .NET Low-Level-API verwenden, um typische Erstellungs-, Lese-, Aktualisierungs- und Löschvorgänge (CRUD) für ein Element in einer Tabelle durchzuführen. Nachfolgend sind die üblichen Schritte zum Ausführen von CRUD-Operationen mit der .NET-Low-Level-API ausführen:

1. Erstellen einer Instance der `AmazonDynamoDBClient`-Klasse (Client).
2. Geben Sie die für die Operation spezifischen, erforderlichen Parameter in einem entsprechenden Anforderungsobjekt an.

   Verwenden Sie z. B. das `PutItemRequest`-Anforderungsobjekt zum Hochladen eines Elements und das `GetItemRequest`-Anforderungsobjekt, wenn Sie ein vorhandenes Element abrufen.

   Mit dem Anforderungsobjekt können Sie sowohl die erforderlichen als auch die optionalen Parameter angeben.
3. Führen Sie die vom Client bereitgestellte, geeignete Methode aus, indem Sie das Anforderungsobjekt, das Sie im vorhergehenden Schritt erstellt haben, übergeben.

   Der `AmazonDynamoDBClient`-Client bietet die Methoden `PutItem`, `GetItem`, `UpdateItem` und `DeleteItem` für die CRUD-Operationen.

Themen

- Einfügen eines Elements
- Abrufen eines Elements
- Aktualisieren eines Elements
- Unteilbarer Zähler
- Löschen eines Elements
- Batch Write: Einfügen und Löschen mehrerer Elemente
- Batch Get: Abrufen mehrerer Elemente
- Beispiel: CRUD-Operationen unter Verwendung der AWS SDK für .NET Low-Level-API
- Beispiel: Batch-Operationen mit der AWS SDK für .NET Low-Level-API
- Beispiel: Umgang mit binären Attributen mithilfe der AWS SDK für .NET Low-Level-API

Einfügen eines Elements

Mit der `PutItem`-Methode wird ein Element in eine Tabelle hochgeladen. Wenn das Element bereits vorhanden ist, wird das ganze Element ersetzt.

> **Note**
>
> Anstatt das gesamte Element zu ersetzen und nur spezifische Attribute zu aktualisieren, können Sie die Methode `UpdateItem` verwenden. Weitere Informationen finden Sie unter Aktualisieren eines Elements.

Im Folgenden finden Sie die Schritte zum Hochladen eines Elements mithilfe der .NET SDK-API auf niedriger Ebene:

1. Erstellen Sie eine Instance der `AmazonDynamoDBClient`-Klasse.
2. Stellen Sie die erforderlichen Parameter bereit, in dem Sie eine Instance der `PutItemRequest`-Klasse erstellen.

   Um für ein Element eine PUT-Operation auszuführen, müssen Sie den Tabellennamen und das Element angeben.
3. Führen Sie die `PutItem`-Methode aus, indem Sie das `PutItemRequest`-Objekt bereitstellen, das Sie im vorherigen Schritt erstellt haben.

Im folgenden C#-Codebeispiel werden die vorherigen Schritte veranschaulicht. Das Beispiel lädt ein Element in die `ProductCatalog`-Tabelle hoch.

Example

```
AmazonDynamoDBClient client = new AmazonDynamoDBClient();
string tableName = "ProductCatalog";

var request = new PutItemRequest
{
   TableName = tableName,
   Item = new Dictionary<string, AttributeValue>()
      {
          { "Id", new AttributeValue { N = "201" }},
          { "Title", new AttributeValue { S = "Book 201 Title" }},
          { "ISBN", new AttributeValue { S = "11-11-11-11" }},
          { "Price", new AttributeValue { S = "20.00" }},
          {
            "Authors",
            new AttributeValue
            { SS = new List<string>{"Author1", "Author2"}   }
          }
      }
};
client.PutItem(request);
```

Im vorherigen Beispiel laden Sie ein Buchelement mit den Attributen `Id`, `Title`, `ISBN` und `Authors` hoch. Beachten Sie, dass `Id` ein Attribut vom Typ Zahl ist. Alle anderen Attribute sind vom Typ Zeichenfolge. Autoren ist eine `String`-Einstellung.

Angeben eines optionalen Parameters

Sie können mit dem `PutItemRequest`-Objekt auch optionale Parameter angeben, wie im folgenden C#-Beispiel dargestellt. Das Beispiel gibt die folgenden optionalen Parameter an:

- `ExpressionAttributeNames`, `ExpressionAttributeValues` und `ConditionExpression` geben an, dass das Element nur ersetzt werden kann, wenn das vorhandene Element das Attribut "ISBN" mit einem spezifischen Wert besitzt.
- Mit dem Parameter `ReturnValues` wird das alte Element in der Antwort angefordert.

Example

```
var request = new PutItemRequest
 {
   TableName = tableName,
   Item = new Dictionary<string, AttributeValue>()
               {
                   { "Id", new AttributeValue { N = "104" }},
                   { "Title", new AttributeValue { S = "Book 104  Title" }},
                   { "ISBN", new AttributeValue { S = "444-4444444444" }},
                   { "Authors",
                     new AttributeValue { SS = new List<string>{"Author3"}}}
               },
    // Optional parameters.
    ExpressionAttributeNames = new Dictionary<string,string>()
    {
        {"#I", "ISBN"}
    },
    ExpressionAttributeValues = new Dictionary<string, AttributeValue>()
    {
        {":isbn",new AttributeValue {S = "444-4444444444"}}
    },
    ConditionExpression = "#I = :isbn"

};
var response = client.PutItem(request);
```

Weitere Informationen finden Sie unter PutItem.

Abrufen eines Elements

Die `GetItem`-Methode ruft ein Element ab.

> **Note**
>
> Um mehrere Elemente abzurufen, können Sie die `BatchGetItem`-Methode verwenden. Weitere Informationen finden Sie unter Batch Get: Abrufen mehrerer Elemente.

Im Folgenden werden die Schritte zum Abrufen eines vorhandenen Elements mithilfe des AWS SDK für .NET -Low-Level-API beschrieben.

1. Erstellen Sie eine Instance der `AmazonDynamoDBClient`-Klasse.
2. Stellen Sie die erforderlichen Parameter bereit, in dem Sie eine Instance der `GetItemRequest`-Klasse erstellen.

   Zum Abrufen eines Elements müssen Sie den Tabellennamen und den Primärschlüssel des Elements angeben.
3. Führen Sie die `GetItem`-Methode aus, indem Sie das `GetItemRequest`-Objekt bereitstellen, das Sie im vorherigen Schritt erstellt haben.

Im folgenden C#-Codebeispiel werden die vorherigen Schritte veranschaulicht. Das Beispiel ruft ein Objekt aus der `ProductCatalog`-Tabelle ab.

```
AmazonDynamoDBClient client = new AmazonDynamoDBClient();
string tableName = "ProductCatalog";

var request = new GetItemRequest
 {
   TableName = tableName,
   Key = new Dictionary<string,AttributeValue>() { { "Id", new AttributeValue { N =
 "202" } } },
 };
 var response = client.GetItem(request);

// Check the response.
var result = response.GetItemResult;
var attributeMap = result.Item; // Attribute list in the response.
```

Angeben eines optionalen Parameters

Sie können mit dem `GetItemRequest`-Objekt auch optionale Parameter angeben, wie im folgenden C#-Beispiel dargestellt. Das Beispiel gibt die folgenden optionalen Parameter an:

- `ProjectionExpression` Parameter, um die abzurufenden Attribute anzugeben.
- `ConsistentRead` Parameter zum Ausführen eines Strongly-Consistent-Lesevorgangs. Weitere Informationen zur Lesekonsistenz finden Sie unter DynamoDB-Lesekonsistenz.

Example

```
AmazonDynamoDBClient client = new AmazonDynamoDBClient();
```

```
string tableName = "ProductCatalog";

var request = new GetItemRequest
 {
   TableName = tableName,
   Key = new Dictionary<string,AttributeValue>() { { "Id", new AttributeValue { N =
 "202" } } },
   // Optional parameters.
   ProjectionExpression = "Id, ISBN, Title, Authors",
   ConsistentRead = true
 };

 var response = client.GetItem(request);

// Check the response.
var result = response.GetItemResult;
var attributeMap = result.Item;
```

Weitere Informationen finden Sie unter GetItem.

Aktualisieren eines Elements

Mit der `UpdateItem`-Methode wird ein bestehendes Element aktualisiert, wenn es vorhanden ist. Sie können die `UpdateItem`-Operation verwenden, um vorhandene Attributwerte zu aktualisieren, der vorhandenen Sammlung neue Attribute hinzuzufügen oder Attribute aus der vorhandenen Sammlung zu löschen. Wenn das Element, das den bestimmten Primärschlüssel besitzt, nicht gefunden wird, wird ein neues Element hinzugefügt.

Die `UpdateItem`-Operation verwendet folgende Leitlinien:

- Wenn das Element nicht vorhanden ist, fügt `UpdateItem` ein neues Element hinzu, indem der in der Eingabe angegebene Primärschlüssel genutzt wird.
- Wenn das Element vorhanden ist, wendet `UpdateItem` die Aktualisierung wie folgt an:
  - Ersetzt die vorhandenen Attributwerte durch die Werte in der Aktualisierung.
  - Wenn das Attribut, das Sie in der Eingabe angegeben haben, nicht vorhanden ist, wird dem Element ein neues Attribut hinzugefügt.
  - Wenn das Eingabeattribut Null ist, wird das Attribut gelöscht, sofern es vorhanden ist.
  - Wenn Sie `ADD` für die Aktion `Action` verwenden, können Sie einem vorhandenen Satz (Zeichenfolgen- oder Zahlensatz) Werte hinzufügen oder mathematisch addieren (unter

Verwendung einer positiven Zahl) bzw. von dem vorhandenen numerischen Attributwert subtrahieren (unter Verwendung einer negativen Zahl).

> **Note**
>
> Die `PutItem`-Operation kann auch eine Aktualisierung durchführen. Weitere Informationen finden Sie unter Einfügen eines Elements. Wenn Sie z. B. `PutItem` aufrufen, um ein Element hochzuladen, und der Primärschlüssel vorhanden ist, ersetzt die `PutItem`-Operation das gesamte Element. Wenn das vorhandene Element Attribute entiert und diese Attribute nicht in der Eingabe angegeben sind, werden diese Attribute vom `PutItem`-Vorgang gelöscht. Jedoch aktualisiert `UpdateItem` lediglich die angegebenen Eingabeattribute. Alle anderen vorhandenen Attribute dieses Elements bleiben unverändert.

Im Folgenden werden die Schritte zum Aktualisieren eines vorhandenen Elements mithilfe der Low-Level-.NET.SDK-API beschrieben.

1. Erstellen Sie eine Instance der `AmazonDynamoDBClient`-Klasse.
2. Stellen Sie die erforderlichen Parameter bereit, in dem Sie eine Instance der `UpdateItemRequest`-Klasse erstellen.

   Dies ist das Anforderungsobjekt, in dem Sie alle Aktualisierungen beschreiben, z. B. Attribute hinzufügen, vorhandene Attribute aktualisieren oder Attribute löschen. Um ein vorhandenes Attribut zu löschen, geben Sie den Attributnamen mit Nullwert an.
3. Führen Sie die `UpdateItem`-Methode aus, indem Sie das `UpdateItemRequest`-Objekt bereitstellen, das Sie im vorherigen Schritt erstellt haben.

Im folgenden C#-Codebeispiel werden die vorherigen Schritte veranschaulicht. Im Beispiel wird ein Buchelement in der `ProductCatalog`-Tabelle aktualisiert. Es fügt der `Authors`-Auflistung einen neuen Autor hinzu und löscht das vorhandene `ISBN`-Attribut. Darüber hinaus wird auch der Preis um eins gesenkt.

```
AmazonDynamoDBClient client = new AmazonDynamoDBClient();
string tableName = "ProductCatalog";

var request = new UpdateItemRequest
```

```
{
    TableName = tableName,
    Key = new Dictionary<string,AttributeValue>() { { "Id", new AttributeValue { N =
 "202" } } },
    ExpressionAttributeNames = new Dictionary<string,string>()
    {
        {"#A", "Authors"},
        {"#P", "Price"},
        {"#NA", "NewAttribute"},
        {"#I", "ISBN"}
    },
    ExpressionAttributeValues = new Dictionary<string, AttributeValue>()
    {
        {":auth",new AttributeValue { SS = {"Author YY","Author ZZ"}}},
        {":p",new AttributeValue {N = "1"}},
        {":newattr",new AttributeValue {S = "someValue"}},
    },

    // This expression does the following:
    // 1) Adds two new authors to the list
    // 2) Reduces the price
    // 3) Adds a new attribute to the item
    // 4) Removes the ISBN attribute from the item
    UpdateExpression = "ADD #A :auth SET #P = #P - :p, #NA = :newattr REMOVE #I"
};
var response = client.UpdateItem(request);
```

Angeben eines optionalen Parameters

Sie können auch optionale Parameter mithilfe des `UpdateItemRequest`-Objekts bereitstellen, wie im folgenden C#-Beispiel gezeigt. Folgende optionale Parameter werden angegeben:

- `ExpressionAttributeValues` und `ConditionExpression`, um festzulegen, dass der Preis nur aktualisiert werden kann, wenn er momentan 20,00 beträgt.
- Der Parameter `ReturnValues` zum Anfordern des aktualisierten Elements in der Antwort.

Example

```
AmazonDynamoDBClient client = new AmazonDynamoDBClient();
string tableName = "ProductCatalog";

var request = new UpdateItemRequest
```

```
{
    Key = new Dictionary<string,AttributeValue>() { { "Id", new AttributeValue { N =
 "202" } } },

    // Update price only if the current price is 20.00.
    ExpressionAttributeNames = new Dictionary<string,string>()
    {
        {"#P", "Price"}
    },
    ExpressionAttributeValues = new Dictionary<string, AttributeValue>()
    {
        {":newprice",new AttributeValue {N = "22"}},
        {":currprice",new AttributeValue {N = "20"}}
    },
    UpdateExpression = "SET #P = :newprice",
    ConditionExpression = "#P = :currprice",
    TableName = tableName,
    ReturnValues = "ALL_NEW" // Return all the attributes of the updated item.
};

var response = client.UpdateItem(request);
```

Weitere Informationen finden Sie unter UpdateItem.

Unteilbarer Zähler

Sie können `updateItem` verwenden, um einen unteilbaren Zähler zu implementieren, mit dem der Wert eines bestehenden Attributs erhöht oder verringert wird, ohne andere Schreibanforderungen zu beeinflussen. Um einen Atomzähler zu aktualisieren, verwenden Sie `updateItem` mit einem Attribut vom Typ `Number` im Parameter `UpdateExpression` und `ADD` als `Action`.

Das folgende Codebeispiel zeigt diesen Vorgang und erhöht das Attribut `Quantity` um eins.

```
AmazonDynamoDBClient client = new AmazonDynamoDBClient();
string tableName = "ProductCatalog";

var request = new UpdateItemRequest
{
    Key = new Dictionary<string, AttributeValue>() { { "Id", new AttributeValue { N =
 "121" } } },
    ExpressionAttributeNames = new Dictionary<string, string>()
    {
        {"#Q", "Quantity"}
```

```
    },
    ExpressionAttributeValues = new Dictionary<string, AttributeValue>()
    {
        {":incr",new AttributeValue {N = "1"}}
    },
    UpdateExpression = "SET #Q = #Q + :incr",
    TableName = tableName
};

var response = client.UpdateItem(request);
```

Löschen eines Elements

Die `DeleteItem`-Methode löscht ein Element aus einer Tabelle.

Im Folgenden werden die Schritte zum Löschen eines Elements mithilfe der .NET SDK-Low-Level-API beschrieben.

1. Erstellen Sie eine Instance der `AmazonDynamoDBClient`-Klasse.
2. Stellen Sie die erforderlichen Parameter bereit, in dem Sie eine Instance der `DeleteItemRequest`-Klasse erstellen.

   Zum Löschen eines Elements sind der Tabellenname und der Primärschlüssel des Elements erforderlich.
3. Führen Sie die `DeleteItem`-Methode aus, indem Sie das `DeleteItemRequest`-Objekt bereitstellen, das Sie im vorherigen Schritt erstellt haben.

Example

```
AmazonDynamoDBClient client = new AmazonDynamoDBClient();
string tableName = "ProductCatalog";

var request = new DeleteItemRequest
{
    TableName = tableName,
    Key = new Dictionary<string,AttributeValue>() { { "Id", new AttributeValue { N =
 "201" } } },
};

var response = client.DeleteItem(request);
```

Angeben eines optionalen Parameters

Sie können mit dem `DeleteItemRequest`-Objekt auch optionale Parameter angeben, wie im folgenden C#-Beispiel dargestellt. Folgende optionale Parameter werden angegeben:

- `ExpressionAttributeValues`und `ConditionExpression` um anzugeben, dass das Buchelement nur gelöscht werden kann, wenn es nicht mehr veröffentlicht wird (der InPublication Attributwert ist falsch).
- Der Parameter `ReturnValues` zum Anfordern des gelöschten Elements in der Antwort.

Example

```
var request = new DeleteItemRequest
{
    TableName = tableName,
    Key = new Dictionary<string,AttributeValue>() { { "Id", new AttributeValue { N =
 "201" } } },

    // Optional parameters.
    ReturnValues = "ALL_OLD",
    ExpressionAttributeNames = new Dictionary<string, string>()
    {
        {"#IP", "InPublication"}
    },
    ExpressionAttributeValues = new Dictionary<string, AttributeValue>()
    {
        {":inpub",new AttributeValue {BOOL = false}}
    },
    ConditionExpression = "#IP = :inpub"
};

var response = client.DeleteItem(request);
```

Weitere Informationen finden Sie unter DeleteItem.

Batch Write: Einfügen und Löschen mehrerer Elemente

Batch Write bezieht sich auf das Einfügen und Löschen mehrerer Elemente in einem Batch. Die `BatchWriteItem`-Methode ermöglicht Ihnen das Einfügen und Löschen von mehreren Elementen aus einer oder mehreren Tabellen anhand eines einzigen Aufrufs. Im Folgenden werden die Schritte zum Abrufen mehrerer Elemente mithilfe der .NET SDK-Low-Level-API beschrieben.

1. Erstellen Sie eine Instance der `AmazonDynamoDBClient`-Klasse.
2. Beschreiben Sie alle Put- und Delete-Operationen, indem Sie eine Instance der `BatchWriteItemRequest`-Klasse erstellen.
3. Führen Sie die `BatchWriteItem`-Methode aus, indem Sie das `BatchWriteItemRequest`-Objekt bereitstellen, das Sie im vorherigen Schritt erstellt haben.
4. Verarbeiten Sie die Antwort. Überprüfen Sie, ob in der Antwort nicht verarbeitete Anforderungselemente zurückgegeben wurden. Dies kann der Fall sein, wenn Sie das Kontingent für den bereitgestellten Durchsatz erreichen oder ein anderer vorübergehender Fehler auftritt. Außerdem begrenzt DynamoDB die Anforderungsgröße und die Anzahl der Vorgänge, die Sie in einer Anforderung angeben können. Wenn Sie diese Limits überschreiten, wird die Anforderung von DynamoDB abgelehnt. Weitere Informationen finden Sie unter BatchWriteItem.

Im folgenden C#-Codebeispiel werden die vorherigen Schritte veranschaulicht. Das Beispiel erstellt eine `BatchWriteItemRequest` zum Ausführen der folgenden Schreibvorgänge:

- Einfügen eines Elements in die `Forum`-Tabelle
- Einfügen und Löschen eines Elements in der `Thread`-Tabelle.

Der Code führt `BatchWriteItem` aus, um einen Batchvorgang auszuführen.

```
AmazonDynamoDBClient client = new AmazonDynamoDBClient();

string table1Name = "Forum";
string table2Name = "Thread";

var request = new BatchWriteItemRequest
 {
   RequestItems = new Dictionary<string, List<WriteRequest>>
    {
      {
        table1Name, new List<WriteRequest>
        {
          new WriteRequest
          {
             PutRequest = new PutRequest
             {
                Item = new Dictionary<string,AttributeValue>
```

```
                {
                  { "Name", new AttributeValue { S = "Amazon S3 forum" } },
                  { "Threads", new AttributeValue { N = "0" }}
                }
            }
          }
        }
      } ,
      {
        table2Name, new List<WriteRequest>
        {
          new WriteRequest
          {
            PutRequest = new PutRequest
            {
               Item = new Dictionary<string,AttributeValue>
               {
                 { "ForumName", new AttributeValue { S = "Amazon S3 forum" } },
                 { "Subject", new AttributeValue { S = "My sample question" } },
                 { "Message", new AttributeValue { S = "Message Text." } },
                 { "KeywordTags", new AttributeValue { SS = new List<string> { "Amazon
 S3", "Bucket" }  } }
               }
            }
          },
          new WriteRequest
          {
             DeleteRequest = new DeleteRequest
             {
                Key = new Dictionary<string,AttributeValue>()
                {
                   { "ForumName", new AttributeValue { S = "Some forum name" } },
                   { "Subject", new AttributeValue { S = "Some subject" } }
                }
             }
          }
        }
      }
    }
 };
response = client.BatchWriteItem(request);
```

Ein funktionierendes Beispiel finden Sie unter Beispiel: Batch-Operationen mit der AWS SDK für .NET Low-Level-API.

Batch Get: Abrufen mehrerer Elemente

Die `BatchGetItem`-Methode ermöglicht Ihnen das Abrufen mehrerer Elemente aus einer oder mehreren Tabellen.

> **Note**
>
> Um ein einzelnes Element abzurufen, können Sie die `GetItem`-Methode verwenden.

Im Folgenden werden die Schritte zum Abrufen mehrerer Elemente mithilfe der AWS SDK für .NET - Low-Level-API beschrieben.

1. Erstellen Sie eine Instance der `AmazonDynamoDBClient`-Klasse.
2. Stellen Sie die erforderlichen Parameter bereit, in dem Sie eine Instance der `BatchGetItemRequest`-Klasse erstellen.

   Zum Abrufen mehrerer Elemente, sind der Tabellenname und eine Liste der Primärschlüsselwerte erforderlich.
3. Führen Sie die `BatchGetItem`-Methode aus, indem Sie das `BatchGetItemRequest`-Objekt bereitstellen, das Sie im vorherigen Schritt erstellt haben.
4. Verarbeiten Sie die Antwort. Überprüfen Sie, ob nicht verarbeitete Schlüssel vorhanden sind. Dies kann der Fall sein, wenn Sie das Kontingent für den bereitgestellten Durchsatz erreichen oder ein anderer vorübergehender Fehler auftritt.

Im folgenden C#-Codebeispiel werden die vorherigen Schritte veranschaulicht. Das Beispiel ruft Elemente aus zwei Tabellen, `Forum` und `Thread`, ab. Die Anforderung gibt zwei Elemente in der Tabelle `Forum` und drei Elemente in der Tabelle `Thread` an. Die Antwort enthält Elemente aus beiden Tabellen. Der Code zeigt, wie Sie die Antwort verarbeiten können.

```
AmazonDynamoDBClient client = new AmazonDynamoDBClient();

string table1Name = "Forum";
string table2Name = "Thread";
```

```
var request = new BatchGetItemRequest
{
  RequestItems = new Dictionary<string, KeysAndAttributes>()
  {
    { table1Name,
      new KeysAndAttributes
      {
        Keys = new List<Dictionary<string, AttributeValue>>()
        {
          new Dictionary<string, AttributeValue>()
          {
            { "Name", new AttributeValue { S = "DynamoDB" } }
          },
          new Dictionary<string, AttributeValue>()
          {
            { "Name", new AttributeValue { S = "Amazon S3" } }
          }
        }
      }
    },
    {
      table2Name,
      new KeysAndAttributes
      {
        Keys = new List<Dictionary<string, AttributeValue>>()
        {
          new Dictionary<string, AttributeValue>()
          {
            { "ForumName", new AttributeValue { S = "DynamoDB" } },
            { "Subject", new AttributeValue { S = "DynamoDB Thread 1" } }
          },
          new Dictionary<string, AttributeValue>()
          {
            { "ForumName", new AttributeValue { S = "DynamoDB" } },
            { "Subject", new AttributeValue { S = "DynamoDB Thread 2" } }
          },
          new Dictionary<string, AttributeValue>()
          {
            { "ForumName", new AttributeValue { S = "Amazon S3" } },
            { "Subject", new AttributeValue { S = "Amazon S3 Thread 1" } }
          }
        }
      }
    }
```

```
  }
};

var response = client.BatchGetItem(request);

// Check the response.
var result = response.BatchGetItemResult;
var responses = result.Responses; // The attribute list in the response.

var table1Results = responses[table1Name];
Console.WriteLine("Items in table {0}" + table1Name);
foreach (var item1 in table1Results.Items)
{
  PrintItem(item1);
}

var table2Results = responses[table2Name];
Console.WriteLine("Items in table {1}" + table2Name);
foreach (var item2 in table2Results.Items)
{
  PrintItem(item2);
}
// Any unprocessed keys? could happen if you exceed ProvisionedThroughput or some other
 error.
Dictionary<string, KeysAndAttributes> unprocessedKeys = result.UnprocessedKeys;
foreach (KeyValuePair<string, KeysAndAttributes> pair in unprocessedKeys)
{
    Console.WriteLine(pair.Key, pair.Value);
}
```

Angeben eines optionalen Parameters

Sie können mit dem `BatchGetItemRequest`-Objekt auch optionale Parameter angeben, wie im folgenden C#-Beispiel dargestellt. Im Beispiel werden zwei Elemente aus der `Forum` Tabelle abgerufen. Folgende optionale Parameter werden angegeben:

- `ProjectionExpression` Parameter, um die abzurufenden Attribute anzugeben.

Example

```
AmazonDynamoDBClient client = new AmazonDynamoDBClient();
```

```
string table1Name = "Forum";

var request = new BatchGetItemRequest
{
  RequestItems = new Dictionary<string, KeysAndAttributes>()
  {
    { table1Name,
      new KeysAndAttributes
      {
        Keys = new List<Dictionary<string, AttributeValue>>()
        {
          new Dictionary<string, AttributeValue>()
          {
            { "Name", new AttributeValue { S = "DynamoDB" } }
          },
          new Dictionary<string, AttributeValue>()
          {
            { "Name", new AttributeValue { S = "Amazon S3" } }
          }
        }
      },
      // Optional - name of an attribute to retrieve.
      ProjectionExpression = "Title"
    }
  }
};

var response = client.BatchGetItem(request);
```

Weitere Informationen finden Sie unter BatchGetItem.

Beispiel: CRUD-Operationen unter Verwendung der AWS SDK für .NET Low-Level-API

Das folgende C#-Codebeispiel veranschaulicht CRUD-Operationen für ein Amazon-DynamoDB-Element. Das Beispiel fügt der Tabelle `ProductCatalog` ein Element hinzu, ruft es ab, führt verschiedene Aktualisierungen aus und löscht schließlich das Element. Wenn Sie diese Tabelle nicht erstellt haben, können Sie sie auch programmgesteuert erstellen. Weitere Informationen finden Sie unter Erstellen von Beispieltabellen und Hochladen von Daten mit dem AWS SDK für .NET.

step-by-stepAnweisungen zum Testen des folgenden Beispiels finden Sie unter. .NET-Codebeispiele

Example

```
using System;
using System.Collections.Generic;
using Amazon.DynamoDBv2;
using Amazon.DynamoDBv2.Model;
using Amazon.Runtime;
using Amazon.SecurityToken;

namespace com.amazonaws.codesamples
{
    class LowLevelItemCRUDExample
    {
        private static string tableName = "ProductCatalog";
        private static AmazonDynamoDBClient client = new AmazonDynamoDBClient();

        static void Main(string[] args)
        {
            try
            {
                CreateItem();
                RetrieveItem();

                // Perform various updates.
                UpdateMultipleAttributes();
                UpdateExistingAttributeConditionally();

                // Delete item.
                DeleteItem();
                Console.WriteLine("To continue, press Enter");
                Console.ReadLine();
            }
            catch (Exception e)
            {
                Console.WriteLine(e.Message);
                Console.WriteLine("To continue, press Enter");
                Console.ReadLine();
            }
        }

        private static void CreateItem()
        {
            var request = new PutItemRequest
            {
```

```
            TableName = tableName,
            Item = new Dictionary<string, AttributeValue>()
        {
            { "Id", new AttributeValue {
                  N = "1000"
              }},
            { "Title", new AttributeValue {
                  S = "Book 201 Title"
              }},
            { "ISBN", new AttributeValue {
                  S = "11-11-11-11"
              }},
            { "Authors", new AttributeValue {
                  SS = new List<string>{"Author1", "Author2" }
              }},
            { "Price", new AttributeValue {
                  N = "20.00"
              }},
            { "Dimensions", new AttributeValue {
                  S = "8.5x11.0x.75"
              }},
            { "InPublication", new AttributeValue {
                  BOOL = false
              } }
        }
        };
        client.PutItem(request);
    }

    private static void RetrieveItem()
    {
        var request = new GetItemRequest
        {
            TableName = tableName,
            Key = new Dictionary<string, AttributeValue>()
        {
            { "Id", new AttributeValue {
                  N = "1000"
              } }
        },
            ProjectionExpression = "Id, ISBN, Title, Authors",
            ConsistentRead = true
        };
        var response = client.GetItem(request);
```

```
            // Check the response.
            var attributeList = response.Item; // attribute list in the response.
            Console.WriteLine("\nPrinting item after retrieving it ............");
            PrintItem(attributeList);
        }

        private static void UpdateMultipleAttributes()
        {
            var request = new UpdateItemRequest
            {
                Key = new Dictionary<string, AttributeValue>()
            {
                { "Id", new AttributeValue {
                      N = "1000"
                  } }
            },
                // Perform the following updates:
                // 1) Add two new authors to the list
                // 1) Set a new attribute
                // 2) Remove the ISBN attribute
                ExpressionAttributeNames = new Dictionary<string, string>()
            {
                {"#A","Authors"},
                {"#NA","NewAttribute"},
                {"#I","ISBN"}
            },
                ExpressionAttributeValues = new Dictionary<string, AttributeValue>()
            {
                {":auth",new AttributeValue {
                     SS = {"Author YY", "Author ZZ"}
                 }},
                {":new",new AttributeValue {
                     S = "New Value"
                 }}
            },

                UpdateExpression = "ADD #A :auth SET #NA = :new REMOVE #I",

                TableName = tableName,
                ReturnValues = "ALL_NEW" // Give me all attributes of the updated item.
            };
            var response = client.UpdateItem(request);
```

```
            // Check the response.
            var attributeList = response.Attributes; // attribute list in the response.
                                                     // print attributeList.
            Console.WriteLine("\nPrinting item after multiple attribute
update ............");
            PrintItem(attributeList);
        }

        private static void UpdateExistingAttributeConditionally()
        {
            var request = new UpdateItemRequest
            {
                Key = new Dictionary<string, AttributeValue>()
            {
                { "Id", new AttributeValue {
                      N = "1000"
                  } }
            },
                ExpressionAttributeNames = new Dictionary<string, string>()
            {
                {"#P", "Price"}
            },
                ExpressionAttributeValues = new Dictionary<string, AttributeValue>()
            {
                {":newprice",new AttributeValue {
                     N = "22.00"
                 }},
                {":currprice",new AttributeValue {
                     N = "20.00"
                 }}
            },
                // This updates price only if current price is 20.00.
                UpdateExpression = "SET #P = :newprice",
                ConditionExpression = "#P = :currprice",

                TableName = tableName,
                ReturnValues = "ALL_NEW" // Give me all attributes of the updated item.
            };
            var response = client.UpdateItem(request);

            // Check the response.
            var attributeList = response.Attributes; // attribute list in the response.
            Console.WriteLine("\nPrinting item after updating price value
conditionally ............");
```

```
            PrintItem(attributeList);
        }

        private static void DeleteItem()
        {
            var request = new DeleteItemRequest
            {
                TableName = tableName,
                Key = new Dictionary<string, AttributeValue>()
            {
                { "Id", new AttributeValue {
                      N = "1000"
                  } }
            },

                // Return the entire item as it appeared before the update.
                ReturnValues = "ALL_OLD",
                ExpressionAttributeNames = new Dictionary<string, string>()
            {
                {"#IP", "InPublication"}
            },
                ExpressionAttributeValues = new Dictionary<string, AttributeValue>()
            {
                {":inpub",new AttributeValue {
                     BOOL = false
                 }}
            },
                ConditionExpression = "#IP = :inpub"
            };

            var response = client.DeleteItem(request);

            // Check the response.
            var attributeList = response.Attributes; // Attribute list in the response.
                                                     // Print item.
            Console.WriteLine("\nPrinting item that was just deleted ............");
            PrintItem(attributeList);
        }

        private static void PrintItem(Dictionary<string, AttributeValue> attributeList)
        {
            foreach (KeyValuePair<string, AttributeValue> kvp in attributeList)
            {
                string attributeName = kvp.Key;
```

```
                AttributeValue value = kvp.Value;

                Console.WriteLine(
                    attributeName + " " +
                    (value.S == null ? "" : "S=[" + value.S + "]") +
                    (value.N == null ? "" : "N=[" + value.N + "]") +
                    (value.SS == null ? "" : "SS=[" + string.Join(",",
 value.SS.ToArray()) + "]") +
                    (value.NS == null ? "" : "NS=[" + string.Join(",",
 value.NS.ToArray()) + "]")
                    );
            }
            Console.WriteLine("*************************************************");
        }
    }
}
```

Beispiel: Batch-Operationen mit der AWS SDK für .NET Low-Level-API

Themen

- Beispiel: Batch-Schreibvorgang mit der AWS SDK für .NET Low-Level-API
- Beispiel: Batch-Abruf-Vorgang mit der AWS SDK für .NET Low-Level-API

Dieser Abschnitt enthält Beispiele für Batchvorgänge, Batch-Schreibvorgänge und Batch-Get, die Amazon DynamoDB unterstützt.

Beispiel: Batch-Schreibvorgang mit der AWS SDK für .NET Low-Level-API

Das folgende C#-Codebeispiel verwendet die `BatchWriteItem`-Methode für die folgenden Operationen zum Einfügen und Löschen:

- Einfügen eines Elements in die `Forum`-Tabelle
- Einfügen und Löschen eines Elements aus der `Thread`-Tabelle

Bei der Erstellung Ihrer BatchWrite-Anforderung können Sie eine beliebige Anzahl von Einfüge- und Lösch-Anforderungen angeben. DynamoDB `BatchWriteItem` begrenzt jedoch die Größe einer Batch-Schreibanforderung und die Anzahl der Put- und Delete-Vorgänge in einem einzelnen Batch-Schreibvorgang. Weitere Informationen finden Sie unter BatchWriteItem. Wenn Ihre Anforderung diese Grenzwerte überschreitet, wird sie abgelehnt. Wenn die Tabelle nicht über

ausreichend bereitgestellten Durchsatz für diese Anforderung verfügt, werden nicht verarbeitete Anforderungselemente in der Antwort zurückgegeben.

Im folgenden Beispiel wird die Antwort auf unverarbeitete Anforderungselemente überprüft. Liegen unverarbeitete Elemente vor, wird der Vorgang wiederholt und die `BatchWriteItem`-Anforderung mit unverarbeiteten Elementen wird erneut gesendet. Sie können diese Beispieltabellen auch erstellen und Beispieldaten programmgesteuert hochladen. Weitere Informationen finden Sie unter Erstellen von Beispieltabellen und Hochladen von Daten mit dem AWS SDK für .NET.

step-by-stepAnweisungen zum Testen des folgenden Beispiels finden Sie unter.NET-Codebeispiele.

Example

```
using System;
using System.Collections.Generic;
using Amazon.DynamoDBv2;
using Amazon.DynamoDBv2.Model;
using Amazon.Runtime;

namespace com.amazonaws.codesamples
{
    class LowLevelBatchWrite
    {
        private static string table1Name = "Forum";
        private static string table2Name = "Thread";
        private static AmazonDynamoDBClient client = new AmazonDynamoDBClient();

        static void Main(string[] args)
        {
            try
            {
                TestBatchWrite();
            }
            catch (AmazonServiceException e) { Console.WriteLine(e.Message); }
            catch (Exception e) { Console.WriteLine(e.Message); }

            Console.WriteLine("To continue, press Enter");
            Console.ReadLine();
        }

        private static void TestBatchWrite()
        {
            var request = new BatchWriteItemRequest
```

```
            {
                ReturnConsumedCapacity = "TOTAL",
                RequestItems = new Dictionary<string, List<WriteRequest>>
            {
                {
                    table1Name, new List<WriteRequest>
                    {
                        new WriteRequest
                        {
                            PutRequest = new PutRequest
                            {
                                Item = new Dictionary<string, AttributeValue>
                                {
                                    { "Name", new AttributeValue {
                                          S = "S3 forum"
                                      } },
                                    { "Threads", new AttributeValue {
                                          N = "0"
                                      }}
                                }
                            }
                        }
                    }
                },
                {
                    table2Name, new List<WriteRequest>
                    {
                        new WriteRequest
                        {
                            PutRequest = new PutRequest
                            {
                                Item = new Dictionary<string, AttributeValue>
                                {
                                    { "ForumName", new AttributeValue {
                                          S = "S3 forum"
                                      } },
                                    { "Subject", new AttributeValue {
                                          S = "My sample question"
                                      } },
                                    { "Message", new AttributeValue {
                                          S = "Message Text."
                                      } },
                                    { "KeywordTags", new AttributeValue {
                                          SS = new List<string> { "S3", "Bucket" }
```

```
                                    } }
                                }
                            }
                        },
                        new WriteRequest
                        {
                            // For the operation to delete an item, if you provide a
primary key value
                            // that does not exist in the table, there is no error, it
is just a no-op.
                            DeleteRequest = new DeleteRequest
                            {
                                Key = new Dictionary<string, AttributeValue>()
                                {
                                    { "ForumName",  new AttributeValue {
                                          S = "Some partition key value"
                                      } },
                                    { "Subject", new AttributeValue {
                                          S = "Some sort key value"
                                      } }
                                }
                            }
                        }
                    }
                }
            }
            };

            CallBatchWriteTillCompletion(request);
        }

        private static void CallBatchWriteTillCompletion(BatchWriteItemRequest request)
        {
            BatchWriteItemResponse response;

            int callCount = 0;
            do
            {
                Console.WriteLine("Making request");
                response = client.BatchWriteItem(request);
                callCount++;

                // Check the response.
```

```
                var tableConsumedCapacities = response.ConsumedCapacity;
                var unprocessed = response.UnprocessedItems;

                Console.WriteLine("Per-table consumed capacity");
                foreach (var tableConsumedCapacity in tableConsumedCapacities)
                {
                    Console.WriteLine("{0} - {1}", tableConsumedCapacity.TableName,
 tableConsumedCapacity.CapacityUnits);
                }

                Console.WriteLine("Unprocessed");
                foreach (var unp in unprocessed)
                {
                    Console.WriteLine("{0} - {1}", unp.Key, unp.Value.Count);
                }
                Console.WriteLine();

                // For the next iteration, the request will have unprocessed items.
                request.RequestItems = unprocessed;
            } while (response.UnprocessedItems.Count > 0);

            Console.WriteLine("Total # of batch write API calls made: {0}", callCount);
        }
    }
}
```

Beispiel: Batch-Abruf-Vorgang mit der AWS SDK für .NET Low-Level-API

Im folgenden C#-Codebeispiel wird die `BatchGetItem` Methode zum Abrufen mehrerer Elemente aus den `Forum`- und `Thread`-Tabellen in Amazon DynamoDB verwendet. Die `BatchGetItemRequest` gibt die Tabellennamen sowie eine Liste der Primärschlüssel für jede Tabelle an. In dem Beispiel wird die Antwort durch Drucken der abgerufenen Elemente verarbeitet.

step-by-stepAnweisungen zum Testen des folgenden Beispiels finden Sie unter.NET-Codebeispiele.

Example

```
using System;
using System.Collections.Generic;
using Amazon.DynamoDBv2;
using Amazon.DynamoDBv2.Model;
using Amazon.Runtime;
```

```
namespace com.amazonaws.codesamples
{
    class LowLevelBatchGet
    {
        private static string table1Name = "Forum";
        private static string table2Name = "Thread";
        private static AmazonDynamoDBClient client = new AmazonDynamoDBClient();

        static void Main(string[] args)
        {
            try
            {
                RetrieveMultipleItemsBatchGet();

                Console.WriteLine("To continue, press Enter");
                Console.ReadLine();
            }
            catch (AmazonServiceException e) { Console.WriteLine(e.Message); }
            catch (Exception e) { Console.WriteLine(e.Message); }
        }

        private static void RetrieveMultipleItemsBatchGet()
        {
            var request = new BatchGetItemRequest
            {
                RequestItems = new Dictionary<string, KeysAndAttributes>()
            {
                { table1Name,
                  new KeysAndAttributes
                  {
                      Keys = new List<Dictionary<string, AttributeValue> >()
                      {
                          new Dictionary<string, AttributeValue>()
                          {
                              { "Name", new AttributeValue {
                                S = "Amazon DynamoDB"
                      } }
                          },
                          new Dictionary<string, AttributeValue>()
                          {
                              { "Name", new AttributeValue {
                                S = "Amazon S3"
                      } }
                          }
```

```
                    }
                }},
            {
                table2Name,
                new KeysAndAttributes
                {
                    Keys = new List<Dictionary<string, AttributeValue> >()
                    {
                        new Dictionary<string, AttributeValue>()
                        {
                            { "ForumName", new AttributeValue {
                                  S = "Amazon DynamoDB"
                              } },
                            { "Subject", new AttributeValue {
                                  S = "DynamoDB Thread 1"
                              } }
                        },
                        new Dictionary<string, AttributeValue>()
                        {
                            { "ForumName", new AttributeValue {
                                  S = "Amazon DynamoDB"
                              } },
                            { "Subject", new AttributeValue {
                                  S = "DynamoDB Thread 2"
                              } }
                        },
                        new Dictionary<string, AttributeValue>()
                        {
                            { "ForumName", new AttributeValue {
                                  S = "Amazon S3"
                              } },
                            { "Subject", new AttributeValue {
                                  S = "S3 Thread 1"
                              } }
                        }
                    }
                }
            }
        }
        };

        BatchGetItemResponse response;
        do
        {
```

```
                Console.WriteLine("Making request");
                response = client.BatchGetItem(request);

                // Check the response.
                var responses = response.Responses; // Attribute list in the response.

                foreach (var tableResponse in responses)
                {
                    var tableResults = tableResponse.Value;
                    Console.WriteLine("Items retrieved from table {0}",
tableResponse.Key);
                    foreach (var item1 in tableResults)
                    {
                        PrintItem(item1);
                    }
                }

                // Any unprocessed keys? could happen if you exceed
ProvisionedThroughput or some other error.
                Dictionary<string, KeysAndAttributes> unprocessedKeys =
response.UnprocessedKeys;
                foreach (var unprocessedTableKeys in unprocessedKeys)
                {
                    // Print table name.
                    Console.WriteLine(unprocessedTableKeys.Key);
                    // Print unprocessed primary keys.
                    foreach (var key in unprocessedTableKeys.Value.Keys)
                    {
                        PrintItem(key);
                    }
                }

                request.RequestItems = unprocessedKeys;
            } while (response.UnprocessedKeys.Count > 0);
        }

        private static void PrintItem(Dictionary<string, AttributeValue> attributeList)
        {
            foreach (KeyValuePair<string, AttributeValue> kvp in attributeList)
            {
                string attributeName = kvp.Key;
                AttributeValue value = kvp.Value;

                Console.WriteLine(
```

```
                    attributeName + " " +
                    (value.S == null ? "" : "S=[" + value.S + "]") +
                    (value.N == null ? "" : "N=[" + value.N + "]") +
                    (value.SS == null ? "" : "SS=[" + string.Join(",",
 value.SS.ToArray()) + "]") +
                    (value.NS == null ? "" : "NS=[" + string.Join(",",
 value.NS.ToArray()) + "]")
                    );
            }
            Console.WriteLine("*************************************************");
        }
    }
}
```

Beispiel: Umgang mit binären Attributen mithilfe der AWS SDK für .NET Low-Level-API

Das folgende C#-Codebeispiel veranschaulicht den Umgang mit Attributen des Typs Binärwert. Bei dem Beispiel wird der `Reply`-Tabelle ein Element hinzugefügt. Das Element enthält ein Binärtypattribut (`ExtendedMessage`), in dem komprimierte Daten gespeichert sind. Daraufhin wird das Element abgerufen und es werden alle Attributwerte gedruckt. Zur Veranschaulichung verwendet das Beispiel die `GZipStream`-Klasse, um einen Beispiel-Stream zu komprimieren und ihn dem Attribut `ExtendedMessage` hinzuzufügen, und dekomprimiert den Stream beim Drucken des Attributwerts.

step-by-stepAnweisungen zum Testen des folgenden Beispiels finden Sie unter.NET-Codebeispiele.

Example

```
using System;
using System.Collections.Generic;
using System.IO;
using System.IO.Compression;
using Amazon.DynamoDBv2;
using Amazon.DynamoDBv2.Model;
using Amazon.Runtime;

namespace com.amazonaws.codesamples
{
    class LowLevelItemBinaryExample
    {
        private static string tableName = "Reply";
        private static AmazonDynamoDBClient client = new AmazonDynamoDBClient();
```

```
        static void Main(string[] args)
        {
            // Reply table primary key.
            string replyIdPartitionKey = "Amazon DynamoDB#DynamoDB Thread 1";
            string replyDateTimeSortKey = Convert.ToString(DateTime.UtcNow);

            try
            {
                CreateItem(replyIdPartitionKey, replyDateTimeSortKey);
                RetrieveItem(replyIdPartitionKey, replyDateTimeSortKey);
                // Delete item.
                DeleteItem(replyIdPartitionKey, replyDateTimeSortKey);
                Console.WriteLine("To continue, press Enter");
                Console.ReadLine();
            }
            catch (AmazonDynamoDBException e) { Console.WriteLine(e.Message); }
            catch (AmazonServiceException e) { Console.WriteLine(e.Message); }
            catch (Exception e) { Console.WriteLine(e.Message); }
        }

        private static void CreateItem(string partitionKey, string sortKey)
        {
            MemoryStream compressedMessage = ToGzipMemoryStream("Some long extended
message to compress.");
            var request = new PutItemRequest
            {
                TableName = tableName,
                Item = new Dictionary<string, AttributeValue>()
            {
                { "Id", new AttributeValue {
                      S = partitionKey
                  }},
                { "ReplyDateTime", new AttributeValue {
                      S = sortKey
                  }},
                { "Subject", new AttributeValue {
                      S = "Binary type "
                  }},
                { "Message", new AttributeValue {
                      S = "Some message about the binary type"
                  }},
                { "ExtendedMessage", new AttributeValue {
                      B = compressedMessage
```

```
                }}
            }
            };
            client.PutItem(request);
        }

        private static void RetrieveItem(string partitionKey, string sortKey)
        {
            var request = new GetItemRequest
            {
                TableName = tableName,
                Key = new Dictionary<string, AttributeValue>()
            {
                { "Id", new AttributeValue {
                      S = partitionKey
                  } },
                { "ReplyDateTime", new AttributeValue {
                      S = sortKey
                  } }
            },
                ConsistentRead = true
            };
            var response = client.GetItem(request);

            // Check the response.
            var attributeList = response.Item; // attribute list in the response.
            Console.WriteLine("\nPrinting item after retrieving it ............");

            PrintItem(attributeList);
        }

        private static void DeleteItem(string partitionKey, string sortKey)
        {
            var request = new DeleteItemRequest
            {
                TableName = tableName,
                Key = new Dictionary<string, AttributeValue>()
            {
                { "Id", new AttributeValue {
                      S = partitionKey
                  } },
                { "ReplyDateTime", new AttributeValue {
                      S = sortKey
                  } }
```

```
            }
            };
            var response = client.DeleteItem(request);
        }

        private static void PrintItem(Dictionary<string, AttributeValue> attributeList)
        {
            foreach (KeyValuePair<string, AttributeValue> kvp in attributeList)
            {
                string attributeName = kvp.Key;
                AttributeValue value = kvp.Value;

                Console.WriteLine(
                    attributeName + " " +
                    (value.S == null ? "" : "S=[" + value.S + "]") +
                    (value.N == null ? "" : "N=[" + value.N + "]") +
                    (value.SS == null ? "" : "SS=[" + string.Join(",",
value.SS.ToArray()) + "]") +
                    (value.NS == null ? "" : "NS=[" + string.Join(",",
value.NS.ToArray()) + "]") +
                    (value.B == null ? "" : "B=[" + FromGzipMemoryStream(value.B) +
"]")
                    );
            }
            Console.WriteLine("*********************************************");
        }

        private static MemoryStream ToGzipMemoryStream(string value)
        {
            MemoryStream output = new MemoryStream();
            using (GZipStream zipStream = new GZipStream(output,
CompressionMode.Compress, true))
            using (StreamWriter writer = new StreamWriter(zipStream))
            {
                writer.Write(value);
            }
            return output;
        }

        private static string FromGzipMemoryStream(MemoryStream stream)
        {
            using (GZipStream zipStream = new GZipStream(stream,
CompressionMode.Decompress))
            using (StreamReader reader = new StreamReader(zipStream))
```

```
            {
                return reader.ReadToEnd();
            }
        }
    }
}
```

Verbessern des Datenzugriffs mit sekundären Indizes in DynamoDB

Amazon DynamoDB bietet einen schnellen Zugriff auf Elemente in einer Tabelle durch Angeben von Primärschlüsselwerten. Viele Anwendungen können jedoch davon profitieren, wenn ein oder mehrere Sekundärschlüssel (oder alternative Schlüssel) verfügbar sind, um effizienten Zugriff auf Daten mit anderen Attributen als dem Primärschlüssel zu erlauben. Dazu können Sie einen oder mehrere sekundäre Indizes für eine Tabelle erstellen und `Query`- oder `Scan`-Anforderungen für diese Indizes ausgeben.

Ein sekundärer Index ist eine Datenstruktur, die eine Teilmenge der Attribute aus einer Tabelle zusammen mit einem alternativen Schlüssel zur Unterstützung von `Query`-Operationen enthält. Sie können Daten aus dem Index mit einer `Query`-Operation ähnlich wie `Query` für eine Tabelle abrufen. Eine Tabelle kann mehrere sekundäre Indizes enthalten, die Ihren Anwendungen Zugriff auf viele verschiedene Abfragemuster bieten.

> **Note**
>
> Sie können außerdem eine `Scan`-Operation für einen Index ähnlich wie `Scan` für eine Tabelle ausführen.
> Kontoübergreifender Zugriff für sekundäre Indexscanvorgänge wird derzeit nicht für ressourcenbasierte Richtlinien unterstützt.

Jeder sekundäre Index ist genau einer Tabelle zugeordnet, von der er seine Daten erhält. Diese Tabelle wird als Basistabelle für den Index bezeichnet. Beim Erstellen eines Indexes definieren Sie einen alternativen Schlüssel für den Index (Partitionsschlüssel und Sortierschlüssel). Außerdem definieren Sie die Attribute, die von der Basistabelle in den Index projiziert oder kopiert werden sollen. DynamoDB kopiert diese Attribute in den Index zusammen mit den Primärschlüsselattributen für die Basistabelle. Sie können den Index dann genauso abfragen oder scannen wie eine Tabelle.

Jeder sekundäre Index wird von DynamoDB verwaltet. Wenn Sie der Basistabelle Elemente hinzufügen, ändern oder löschen, werden alle Indizes dieser Tabelle ebenfalls aktualisiert, um diese Änderungen zu berücksichtigen.

DynamoDB unterstützt zwei Arten sekundärer Indizes:

- Globaler sekundärer Index – Dieser Index verfügt über einen Partitionsschlüssel und einen Sortierschlüssel, die nicht mit denen der Basistabelle übereinstimmen müssen. Ein globaler sekundärer Index wird als „global“ betrachtet, da Indexabfragen partitionsübergreifend alle Daten in der Basistabelle umfassen können. Ein globaler sekundärer Index wird in einem eigenen Partitionsbereich abseits der Basistabelle gespeichert und von der Basistabelle getrennt skaliert.
- Lokaler sekundärer Index – Dieser Index weist denselben Partitionsschlüssel wie die Basistabelle auf, hat aber einen anderen Sortierschlüssel. Ein lokaler sekundärer Index wird als „lokal“ betrachtet, da jede Partition eines lokalen sekundären Index auf die Basistabellenpartition bezogen ist, die denselben Partitionsschlüsselwert besitzt.

Einen Vergleich der globalen Sekundärindizes und der lokalen Sekundärindizes finden Sie in diesem Video.

Richtig zwischen GSI und LSI entscheiden

Themen

- Verwenden globaler sekundärer Indizes in DynamoDB
- Lokale sekundäre Indizes

Berücksichtigen Sie die Anforderungen Ihrer Anwendung, wenn Sie sich für einen Indextyp entscheiden. In der folgenden Tabelle werden die Hauptunterschiede zwischen einem globalen sekundären Index und einem lokalen sekundären Index dargelegt.

Merkmal	Globaler sekundärer Index	Lokaler sekundärer Index
Schlüsselschema	Der Primärschlüssel eines globalen sekundärer Indizes kann entweder einfach (Partitionsschlüssel) oder zusammengesetzt sein	Der Primärschlüssel eines lokalen sekundären Indizes muss zusammengesetzt sein (Partitions- und Sortiersc hlüssel).

Merkmal	Globaler sekundärer Index	Lokaler sekundärer Index
	(Partitions- und Sortiersc hlüssel).	
Schlüsselattribute	Der Indexpartitions- und Sortierschlüssel (sofern vorhanden) kann ein beliebige s Basistabellenattribut vom Typ Zeichenfolge, Zahl oder Binärwert sein.	Der Partitionsschlüssel des Indexes ist dasselbe Attribut wie der Partitionsschlüssel der Basistabelle. Der Sortiersc hlüssel kann ein beliebige s Basistabellenattribut vom Typ Zeichenfolge, Zahl oder Binärwert sein.
Größenbeschränkungen pro Partitionsschlüsselwert	Für globale sekundäre Indizes gibt es keine Größenbes chränkungen.	Für jeden Partitionsschlüsse lwert darf die Gesamtgröße aller indizierten Elemente höchstens 10 GB betragen.
Online-Indizierungsoperatio nen	Globale sekundäre Indizes können gleichzeitig mit der Tabelle erstellt werden. Sie können einer vorhanden en Tabelle auch einen neuen globalen sekundäre n Index hinzufügen oder einen vorhandenen globalen sekundären Index löschen. Weitere Informationen finden Sie unter Verwalten globaler sekundärer Indizes in DynamoDB.	Globale sekundäre Indizes werden zur selben Zeit wie die Tabelle erstellt. Sie können einer vorhandenen Tabelle keinen lokalen sekundäre n Index hinzufügen oder vorhandene lokale sekundäre Indizes löschen.
Abfragen und Partitionen	Mit einem globalen sekundäre n Index können Sie die gesamte Tabelle partition sübergreifend abfragen.	Ein lokaler sekundärer Index ermöglicht das Abfragen über eine einzelne Partition, wie vom Partitionsschlüsselwert in der Abfrage angegeben.

Merkmal	Globaler sekundärer Index	Lokaler sekundärer Index
Lesekonsistenz	Abfragen zu globalen sekundären Indizes unterstützen nur Eventual Consistency.	Wenn Sie einen lokalen sekundären Index abfragen, können Sie entweder Eventually Consistent oder Strongly Consistent auswählen.
Verbrauch der bereitgestellten Durchsatzkapazität	Jeder globaler sekundärer Index verfügt über eigene Einstellungen für den bereitgestellten Durchsatz für Lese- und Schreibaktivitäten. Abfragen oder Scans für einen globalen sekundären Index verbrauchen Kapazitätseinheiten des Indexes und nicht der Basistabelle. Dasselbe gilt für globale sekundäre Index-Aktualisierungen aufgrund von Tabellenschreibvorgängen. Ein globaler sekundärer Index, der globalen Tabellen zugeordnet ist, verbraucht Schreibkapazitätseinheiten.	Abfragen oder Scans für einen lokalen sekundären Index verbrauchen Lesekapazitätseinheiten der Basistabelle. Wenn Sie Daten in eine Tabelle schreiben, werden die zugehörigen lokalen sekundären Indizes ebenfalls aktualisiert. Diese Aktualisierungen verbrauchen Schreibkapazitätseinheiten aus der Basistabelle. Ein lokaler sekundärer Index, der globalen Tabellen zugeordnet ist, verbraucht replizierte Schreibkapazitätseinheiten.
Projizierte Attribute	Mit globalen sekundären Index-Abfragen oder -Scans können Sie nur die Attribute anfordern, die in den Index projiziert sind. DynamoDB ruft keine Attribute aus der Tabelle ab.	Wenn Sie einen lokalen sekundären Index abfragen oder scannen können Sie Attribute anfordern, die nicht in den Index projiziert sind. DynamoDB ruft diese Attribute automatisch aus der Tabelle ab.

Wenn Sie mehr als eine Tabelle mit sekundären Indizes erstellen möchten, müssen Sie diesen Vorgang sequenziell ausführen. Beispielsweise erstellen Sie die erste Tabelle und warten, bis sie `ACTIVE` wird. Dann erstellen Sie die nächste Tabelle und warten bis sie `ACTIVE` wird und so weiter. Wenn Sie versuchen, gleichzeitig mehr als eine Tabelle mit einem sekundären Index zu erstellen, gibt DynamoDB eine `LimitExceededException` zurück.

Jeder sekundäre Index verwendet dieselbe Tabellenklasse und denselben Kapazitätsmodus wie die Basistabelle, der er zugeordnet ist. Geben Sie für jeden sekundären Index Folgendes an:

- Der Typ des zu erstellenden Indexes – entweder ein globaler sekundärer Index oder ein lokaler sekundärer Index.
- Einen Namen für den Index. Die Namenskonventionen für Indizes sind mit den Konventionen für Tabellen identisch, wie in Kontingente in Amazon DynamoDB aufgelistet. Der Name muss für die Basistabelle, der er zugeordnet ist, eindeutig sein. Sie können allerdings denselben Namen für Indizes verwenden, die verschiedenen Basistabellen zugeordnet sind.
- Das Schlüsselschema für den Index. Jedes Attribut im Indexschlüsselschema muss ein Attribut auf oberster Ebene vom Typ `String`, `Number` oder `Binary` sein. Andere Datentypen, einschließlich Dokumenten und Sätzen, sind nicht zulässig. Andere Anforderungen für das Schlüsselschema hängen vom Indextyp ab:
  - Für einen globalen sekundären Index kann der Partitionsschlüssel ein beliebiges skalares Attribut der Basistabelle sein. Ein Sortierschlüssel ist optional und kann ebenfalls ein beliebiges skalares Attribut der Basistabelle sein.
  - Für einen lokalen sekundären Index muss der Partitionsschlüssel mit dem Partitionsschlüssel der Basistabelle übereinstimmen und der Sortierschlüssel muss ein Nicht-Schlüssel-Basistabellenattribut sein.
- Zusätzliche Attribute, sofern vorhanden, werden von der Basistabelle in den Index projiziert. Diese Attribute ergänzen die Schlüsselattribute der Tabelle, die automatisch in jeden Index projiziert werden. Sie können Attribute jedes Datentyps, einschließlich Skalaren, Dokumenten und Sätzen, projizieren.
- Die Einstellungen des für den Index bereitgestellten Durchsatzes, falls erforderlich:
  - Für einen globalen sekundären Index müssen Sie Einstellungen für Lese- und Schreibkapazitätseinheiten angeben. Diese Einstellungen für den bereitgestellten Durchsatz sind von den Einstellungen der Basistabelle unabhängig.
  - Für einen lokalen sekundären Index müssen Sie keine Einstellungen für Lese- und Schreibkapazitätseinheiten angeben. Alle Lese- und Schreibvorgänge für einen lokalen

sekundären Index nutzen die Einstellungen des bereitgestellten Durchsatzes der entsprechenden Basistabelle.

Für maximale Abfrageflexibilität können Sie bis zu 20 globale sekundäre Indizes (Standardkontingent) und bis zu 5 lokale sekundäre Indizes pro Tabelle erstellen.

Um eine detaillierte Liste der sekundären Indizes in einer Tabelle abzurufen, verwenden Sie die `DescribeTable`-Operation. `DescribeTable` gibt den Namen, die Speichergröße und die Elementanzahl für jeden sekundären Index in der Tabelle zurück. Diese Werte werden nicht in Echtzeit, sondern etwa alle sechs Stunden aktualisiert.

Sie können auf die Daten in einem sekundären Index entweder mit der `Query`- oder der `Scan`-Operation zugreifen. Sie müssen den Namen der Basistabelle und den Namen des Indexes, den Sie verwenden möchten, die Attribute, die in den Abfrageergebnissen zurückgegeben werden sollen, und etwaige Bedingungsausdrücke oder Filter, die Sie anwenden möchten, angeben. DynamoDB kann Ergebnisse in auf- oder absteigender Reihenfolge liefern.

Beim Löschen einer Tabelle werden alle dieser Tabelle zugeordneten Indizes ebenfalls gelöscht.

Bewährte Methoden finden Sie unter Bewährte Methoden für die Verwendung sekundärer Indexe in DynamoDB.

Verwenden globaler sekundärer Indizes in DynamoDB

Einige Anwendungen müssen u. U. viele Arten von Abfragen ausführen und verwenden dabei eine Vielzahl von verschiedenen Attributen als Abfragekriterien. Um diese Anforderungen zu unterstützen, können Sie eine oder mehrere globale sekundäre Indizes erstellen und `Query`-Anforderungen für diese Indizes in Amazon DynamoDB generieren.

Themen

- Schritt 6: Verwenden eines globalen sekundären Indexes
- Attributprojektionen
- Schlüsselschema mit mehreren Attributen
- Lesen von Daten aus einem globalen sekundären Index
- Datensynchronisierung zwischen Tabellen und globalen sekundären Indizes
- Tabellenklassen mit globalen sekundären Indizes

- Überlegungen im Hinblick auf die bereitgestellte Durchsatzkapazität für globale sekundäre Indizes
- Speicherüberlegungen für globale sekundäre Indizes
- Muster entwerfen
- Verwalten globaler sekundärer Indizes in DynamoDB
- Erkennen und Korrigieren von Indexschlüsselverstößen in DynamoDB
- Arbeiten mit globalen sekundären Indizes: Java
- Arbeiten mit globalen sekundären Indizes: .NET
- Arbeiten mit globalen sekundären Indizes in DynamoDB mithilfe der AWS CLI

Schritt 6: Verwenden eines globalen sekundären Indexes

Betrachten wir ein Beispiel zur Veranschaulichung. Eine Tabelle mit dem Namen `GameScores` erfasst die Benutzer und Punktzahlen für eine mobile Gaming-Anwendung. Jedes Element in `GameScores` wird anhand eines Partitionsschlüssels (`UserId`) und eines Sortierschlüssels (`GameTitle`) identifiziert. Das folgende Diagramm zeigt, wie die Elemente in der Tabelle organisiert wären. (Es werden nicht alle Attribute angezeigt.)

GameScores

UserId	GameTitle	TopScore	TopScoreDateTime	Wins	Losses	
"101"	"Galaxy Invaders"	5842	"2015-09-15:17:24:31"	21	72	...
"101"	"Meteor Blasters"	1000	"2015-10-22:23:18:01"	12	3	...
"101"	"Starship X"	24	"2015-08-31:13:14:21"	4	9	...
"102"	"Alien Adventure"	192	"2015-07-12:11:07:56"	32	192	...
"102"	"Galaxy Invaders"	0	"2015-09-18:07:33:42"	0	5	...
"103"	"Attack Ships"	3	"2015-10-19:01:13:24"	1	8	...
"103"	"Galaxy Invaders"	2317	"2015-09-11:06:53:00"	40	3	...
"103"	"Meteor Blasters"	723	"2015-10-19:01:13:24"	22	12	...
"103"	"Starship X"	42	"2015-07-11:06:53:00"	4	19	...
...	...	...	...	...	...	

Angenommen, Sie möchten eine Bestenlisten-Anwendung für die Anzeige der höchsten Punktzahlen für jedes Spiel entwickeln. Eine Abfrage, die die Schlüsselattribute (`UserId` und `GameTitle`) angibt, wäre äußerst effizient. Wenn die Anwendung Daten aus `GameScores` nur basierend auf `GameTitle` abrufen muss, muss sie eine `Scan`-Operation verwenden. Wenn immer mehr Elemente der Tabelle hinzugefügt werden, werden die Scans aller Daten langsam und ineffizient. Dadurch wird die Beantwortung von folgenden Fragen erschwert:

- Was ist die höchste Punktzahl, die für das Spiel Meteor Blaster jemals erfasst wurde?
- Welche Benutzer hatte die höchste Punktzahl für Galaxy Invaders?
- Wie war das höchste Verhältnis zwischen gewonnen und verlorenen Spielen?

Zum Beschleunigen von Abfragen auf der Basis von Nicht-Schlüsselattributen können Sie einen globalen sekundären Index erstellen. Ein globaler sekundärer Index enthält eine Auswahl von Attributen aus der Basistabelle. Diese sind jedoch nach einem Primärschlüssel organisiert, der sich von dem Schlüssel der Tabelle unterscheidet. Der Indexschlüssel benötigt keinen der Schlüsselattribute aus der Tabelle. Er muss nicht einmal über dasselbe Schlüsselschema verfügen wie eine Tabelle.

Sie können beispielsweise einen globalen sekundären Index mit dem Namen `GameTitleIndex` mit einem Partitionsschlüssel `GameTitle` und einem Sortierschlüssel `TopScore` erstellen. Da die Primärschlüsselattribute der Basistabelle immer in einen Index projiziert werden, ist das Attribut `UserId` ebenfalls vorhanden. Das folgende Diagramm veranschaulicht, wie der Index `GameTitleIndex` aussehen würde.

GameTitleIndex

GameTitle	TopScore	UserId
"Alien Adventure"	192	"102"
"Attack Ships"	3	"103"
"Galaxy Invaders"	0	"102"
"Galaxy Invaders"	2317	"103"
"Galaxy Invaders"	5842	"101"
"Meteor Blasters"	723	"103"
"Meteor Blasters"	1000	"101"
"Starship X"	24	"101"
"Starship X"	42	"103"
...	...	...

Jetzt können Sie `GameTitleIndex` abfragen und die Punktzahlen für Meteor Blasters einfach erhalten. Die Ergebnisse werden nach den Sortierschlüsselwerten `TopScore` sortiert. Wenn Sie den Parameter `ScanIndexForward` auf False festlegen, werden die Ergebnisse in der absteigender Reihenfolge mit der höchsten Punktzahl zuerst zurückgegeben.

Jeder globale sekundäre Index muss über einen Partitionsschlüssel verfügen und kann einen optionalen Sortierschlüssel besitzen. Das Indexschlüsselschema kann sich vom Basistabellenschema unterscheiden. Sie können eine Tabelle mit einem einfachen Primärschlüssel (Partitionsschlüssel) vorliegen haben und einen globalen sekundären Index mit einem zusammengesetzten Primärschlüssel (Partitions- und Sortierschlüssel) erstellen oder umgekehrt. Die Indexschlüsselattribute können aus beliebigen Top-Level-Attributen vom Typ `String`, `Number` oder `Binary` der Basistabelle bestehen. Andere Skalar-, Dokument- und Satztypen sind nicht zulässig.

Sie können andere Basistabellenattribute in den Index projizieren, wenn Sie möchten. Wenn Sie den Index abfragen, kann DynamoDB diese projizierten Attribute effizient abrufen. Globale sekundäre Index-Abfragen können jedoch keine Attribute aus der Basistabelle abrufen. Wenn Sie beispielsweise `GameTitleIndex` wie im Diagramm oben veranschaulicht abgefragt haben, kann die Abfrage nicht

auf Nicht-Schlüsselattribute außer `TopScore` zugreifen (die Schlüsselattribute `GameTitle` und `UserId` werden allerdings automatisch projiziert).

In einer DynamoDB-Tabelle, muss jeder Schlüsselwert eindeutig sein. Die Schlüsselwerte in einem globalen sekundären Index müssen jedoch nicht eindeutig sein. Beispiel: Angenommen, ein Spiel mit dem Namen Comet Quest ist besonders schwierig. Viele neue Benutzer probieren es aus, schaffen es aber nicht, eine Punktzahl über Null zu erreichen. Im Folgenden finden Sie einige der Daten, die dies veranschaulichen.

UserId	GameTitle	TopScore
123	Comet Quest	0
201	Comet Quest	0
301	Comet Quest	0

Wenn diese Daten der Tabelle `GameScores` hinzugefügt werden, verteilt DynamoDB sie auf `GameTitleIndex`. Wenn wir dann den Index mit Comet Quest für `GameTitle` und mit 0 für `TopScore` abfragen, werden die folgenden Daten zurückgegeben.

GameTitle	*TopScore*	*UserId*
"Comet Quest"	*0*	*"123"*
"Comet Quest"	*0*	*"201"*
"Comet Quest"	*0*	*"301"*

Es werden nur die Elemente mit den angegebenen Schlüsselwerten in der Antwort angezeigt. Innerhalb dieser Datengruppe werden die Elemente nicht in einer bestimmten Reihenfolge angezeigt.

Ein globaler sekundärer Index verfolgt nur Datenelemente, bei denen die Schlüsselattribute tatsächlich vorhanden sind. Angenommen, Sie haben der Tabelle `GameScores` ein neues Element hinzugefügt, jedoch nur die erforderlichen Primärschlüsselattribute bereitgestellt.

UserId	GameTitle
400	Comet Quest

Da Sie das Attribut `TopScore` nicht angegeben haben, verteilt DynamoDB dieses Element nicht über `GameTitleIndex`. Wenn Sie `GameScores` für alle Comet Quest-Elemente abfragen, erhalten Sie die folgenden vier Elemente:

UserId	GameTitle	TopScore
"123"	"Comet Quest"	0
"201"	"Comet Quest"	0
"301"	"Comet Quest"	0
"400"	"Comet Quest"	

Eine ähnliche Abfrage für `GameTitleIndex` gibt drei Elemente statt vier zurück. Der Grund hierfür ist, dass das Element mit dem nicht vorhandenen `TopScore` nicht an den Index verteilt wird.

GameTitle	*TopScore*	*UserId*
"Comet Quest"	*0*	*"123"*
"Comet Quest"	*0*	*"201"*
"Comet Quest"	*0*	*"301"*

Attributprojektionen

Eine Projektion ist der Satz von Attributen, die aus einer Tabelle in einen sekundären Index kopiert werden. Der Partitionsschlüssel und der Sortierschlüssel der Tabelle werden immer in den Index projiziert. Sie können andere Attribute projizieren, um die Abfrageanforderungen Ihrer Anwendung zu unterstützen. Wenn Sie einen Index abfragen, kann Amazon DynamoDB auf jedes Attribut in der Projektion zugreifen, als ob sich diese Attribute in einer eigenen Tabelle befinden.

Wenn Sie einen sekundären Index erstellen, müssen Sie die Attribute angeben, die in den Index projiziert werden. DynamoDB bietet hierfür drei verschiedene Optionen:

- KEYS_ONLY – Jeder Eintrag im Index besteht nur aus dem Tabellenpartitionsschlüssel und Sortierschlüsselwerten, sowie den Indexschlüsselwerten. Die Option `KEYS_ONLY` führt zu dem kleinstmöglichen sekundären Index.
- INCLUDE – Zusätzlich zu den in `KEYS_ONLY` beschriebenen Attributen, enthält der sekundäre Index andere Nicht-Schlüsselattribute, die Sie angeben.

- ALL – Der sekundäre Index enthält alle Attribute der Quelltabelle. Da alle Tabellendaten im Index dupliziert werden, wird ein `ALL`-Projektion führt zu dem größtmöglichen sekundären Index.

Im vorherigen Diagramm verfügt `GameTitleIndex` nur über ein projiziertes Attribut: `UserId`. Obwohl eine Anwendung die `UserId` der besten Ergebnisse für jedes Spiel mit `GameTitle` und `TopScore` effizient in Abfragen bestimmen kann, ist es nicht möglich, effizient die höchste Punktzahl für ein bestimmtes Spiel oder das höchste Verhältnis gewonnener und verlorener Spiele bei den besten Ergebnissen zu bestimmen. Dazu muss eine zusätzliche Abfrage der Basistabelle durchgeführt werden, um die gewonnen und verlorenen Spiele für jedes beste Ergebnis abzurufen. Abfragen für diese Daten lassen sich effizienter unterstützen, indem diese Attribute aus der Basistabelle in den globalen sekundären Index projiziert werden, wie in folgendem Diagramm dargestellt.

GameTitleIndex

GameTitle	*TopScore*	*UserId*	*Wins*	*Losses*
"Alien Adventure"	*192*	*"102"*	*32*	*192*
"Attack Ships"	*3*	*"103"*	*1*	*8*
"Galaxy Invaders"	*0*	*"102"*	*0*	*5*
"Galaxy Invaders"	*2317*	*"103"*	*40*	*3*
"Galaxy Invaders"	*5842*	*"101"*	*21*	*72*
"Meteor Blasters"	*723*	*"103"*	*22*	*12*
"Meteor Blasters"	*1000*	*"101"*	*12*	*3*
"Starship X"	*24*	*"101"*	*4*	*9*
"Starship X"	*42*	*"103"*	*4*	*19*
...	...	...	...	...

Da die Nicht-Schlüsselattribute `Wins` und `Losses` in den Index projiziert werden, kann eine Anwendung das Verhältnis zwischen Siegen und Niederlagen für jedes Spiel oder für eine beliebige Kombination aus Spiel und Benutzer-ID bestimmen.

Wenn Sie die Attribute zum Projizieren in einen globalen sekundären Index auswählen, müssen Sie die Differenz der Kosten des bereitgestellten Durchsatzes und der Speicherkosten berücksichtigen:

- Wenn Sie nur auf wenige Attribute mit möglichst niedriger Latenz zugreifen müssen, erwägen Sie, nur diese Attribute in einen globalen sekundären Index zu projizieren. Je kleiner der Index, desto geringer die Speicher- und somit die Schreibkosten.
- Greift Ihre Anwendung häufig auf einige Nicht-Schlüsselattribute zu, sollten Sie erwägen, diese Attribute in einen globalen sekundären Index zu projizieren. Die zusätzlichen Speicherkosten für den globalen sekundären Index wiegen die Kosten für die Durchführung häufiger Tabellen-Scans auf.
- Wenn Sie auf die meisten Nicht-Schlüsselattribute in regelmäßigen Abständen zugreifen müssen, können Sie diese Attribute – oder sogar die gesamte Basistabelle – in einen globalen sekundären Index projizieren. Dadurch erhalten Sie maximale Flexibilität. Ihre Speicherkosten würden allerdings steigen oder sich sogar verdoppeln.
- Wenn Ihre Anwendung eine Tabelle selten abfragen, jedoch viele Schreibvorgänge oder Updates für die Daten in der Tabelle durchführen muss, erwägen Sie das Projizieren von `KEYS_ONLY`. Der globale sekundäre Index wäre nur klein, stünde aber weiterhin bei Bedarf für Abfrageaktivitäten zur Verfügung.

Schlüsselschema mit mehreren Attributen

Globale Sekundärindizes unterstützen Schlüssel mit mehreren Attributen, sodass Sie Partitionsschlüssel zusammenstellen und Schlüssel aus mehreren Attributen sortieren können. Mit Schlüsseln mit mehreren Attributen können Sie einen Partitionsschlüssel aus bis zu vier Attributen und einen Sortierschlüssel aus bis zu vier Attributen erstellen, sodass insgesamt bis zu acht Attribute pro Schlüsselschema vorhanden sind.

Schlüssel mit mehreren Attributen vereinfachen Ihr Datenmodell, indem sie das manuelle Verketten von Attributen zu synthetischen Schlüsseln überflüssig machen. Anstatt zusammengesetzte Zeichenketten zu erstellen`TOURNAMENT#WINTER2024#REGION#NA-EAST`, können Sie die natürlichen Attribute aus Ihrem Domänenmodell direkt verwenden. DynamoDB verarbeitet die zusammengesetzte Schlüssellogik automatisch, indem es mehrere Partitionsschlüsselattribute für die Datenverteilung zusammenfasst und die hierarchische Sortierreihenfolge für mehrere Sortierschlüsselattribute beibehält.

Stellen Sie sich beispielsweise ein Spieleturniersystem vor, bei dem Sie Spiele nach Turnieren und Regionen organisieren möchten. Bei Schlüsseln mit mehreren Attributen können Sie Ihren

Partitionsschlüssel als zwei separate Attribute definieren: `tournamentId` und`region`. In ähnlicher Weise können Sie Ihren Sortierschlüssel mithilfe mehrerer Attribute wie`round`, und definieren`bracket`, `matchId` um eine natürliche Hierarchie zu erstellen. Dieser Ansatz sorgt dafür, dass Ihre Daten eingegeben und Ihr Code sauber bleibt, ohne dass Zeichenketten manipuliert oder analysiert werden müssen.

Wenn Sie einen globalen sekundären Index mit Schlüsseln mit mehreren Attributen abfragen, müssen Sie alle Partitionsschlüsselattribute unter Verwendung von Gleichheitsbedingungen angeben. Bei Sortierschlüsselattributen können Sie sie left-to-right in der Reihenfolge abfragen, in der sie im Schlüsselschema definiert sind. Das bedeutet, dass Sie das erste Sortierschlüsselattribut allein, die ersten beiden Attribute zusammen oder alle Attribute zusammen abfragen können, aber Sie können keine Attribute in der Mitte überspringen. Ungleichheitsbedingungen wie>, `<BETWEEN`, oder `begins_with()` müssen die letzte Bedingung in Ihrer Abfrage sein.

Schlüssel mit mehreren Attributen eignen sich besonders gut, wenn globale Sekundärindizes für bestehende Tabellen erstellt werden. Sie können Attribute verwenden, die bereits in Ihrer Tabelle vorhanden sind, ohne synthetische Schlüssel in Ihren Daten aufzufüllen. Auf diese Weise können Sie Ihrer Anwendung ganz einfach neue Abfragemuster hinzufügen, indem Sie Indizes erstellen, die Ihre Daten mithilfe verschiedener Attributkombinationen neu organisieren.

Jedes Attribut in einem Schlüssel mit mehreren Attributen kann seinen eigenen Datentyp haben: `String` (S), `Number` (N) oder `Binary` (B). Beachten Sie bei der Auswahl von Datentypen, dass `Number` Attribute numerisch sortiert werden, ohne dass Nullen aufgefüllt werden müssen, während Attribute lexikografisch sortiert werden. `String` Wenn Sie beispielsweise einen `Number` Typ für ein Bewertungsattribut verwenden, werden die Werte 5, 50, 500 und 1000 in natürlicher numerischer Reihenfolge sortiert. Dieselben Werte wie `String` Typ würden nach „1000", „5", „50", „500" sortiert, sofern Sie sie nicht mit führenden Nullen auffüllen.

Ordnen Sie beim Entwerfen von Schlüsseln mit mehreren Attributen die Attribute von den allgemeinsten bis hin zu den spezifischsten. Kombinieren Sie bei Partitionsschlüsseln Attribute, die immer zusammen abgefragt werden und für eine gute Datenverteilung sorgen. Platzieren Sie bei Sortierschlüsseln häufig abgefragte Attribute an erster Stelle in der Hierarchie, um die Flexibilität bei der Abfrage zu maximieren. Diese Reihenfolge ermöglicht es Ihnen, Abfragen auf jeder Granularitätsebene durchzuführen, die Ihren Zugriffsmustern entspricht.

Implementierungsbeispiele finden Sie in. Schlüssel mit mehreren Attributen

Lesen von Daten aus einem globalen sekundären Index

Sie können Elemente aus einem globalen sekundären Index mithilfe von `Query` und `Scan` verwenden. Die `GetItem` und `BatchGetItem`-Operationen können für einen globalen sekundären Index nicht verwendet werden.

Abfragen eines globalen sekundären Index

Sie können die `Query`-Operation für den Zugriff auf ein oder mehrere Elemente in einem globalen sekundären Index verwenden. Die Abfrage muss den Namen der Basistabelle und den Namen des Index, den Sie verwenden möchten, die Attribute, die in den Abfrageergebnissen zurückgegeben werden sollen, und etwaige Abfragebedingungen, die Sie anwenden möchten, angeben. DynamoDB kann Ergebnisse in auf- oder absteigender Reihenfolge liefern.

Sehen Sie sich die folgenden Daten an, die von einer `Query` zurückgegeben wurden, die Spieledaten für eine Bestenlisten-Anwendung abfragt.

```
{
    "TableName": "GameScores",
    "IndexName": "GameTitleIndex",
    "KeyConditionExpression": "GameTitle = :v_title",
    "ExpressionAttributeValues": {
        ":v_title": {"S": "Meteor Blasters"}
    },
    "ProjectionExpression": "UserId, TopScore",
    "ScanIndexForward": false
}
```

Vorgänge in dieser Abfrage:

- DynamoDB greift zu und verwendet den GameTitlePartitionsschlüssel GameTitleIndex, um die Indexelemente für Meteor Blasters zu finden. Alle Indexelemente mit diesem Schlüssel werden nebeneinander gespeichert, um ein schnelles Abrufen zu ermöglichen.
- In diesem Spiel verwendet DynamoDB den Index, um auf alle Benutzer IDs - und Top-Scores für dieses Spiel zuzugreifen.
- Die Ergebnisse werden in absteigender Reihenfolge sortiert zurückgegeben, da der Parameter `ScanIndexForward` auf False festgelegt ist.

Scannen eines globalen sekundären Index

Sie können die `Scan`-Operation zum Abrufen aller Daten aus einem globalen sekundären Index verwenden. Geben Sie dazu den Namen der Basistabelle sowie den Indexnamen in der Abfrage an. Mit einer `Scan`-Operation liest DynamoDB alle Daten im Index und gibt sie an die Anwendung zurück. Sie können auch anfordern, dass nur einige der Daten zurückgegeben und die verbleibenden Daten verworfen werden. Verwenden Sie dazu den Parameter `FilterExpression` der `Scan`-Operation. Weitere Informationen finden Sie unter Filterausdrücke für Scan.

Datensynchronisierung zwischen Tabellen und globalen sekundären Indizes

DynamoDB synchronisiert automatisch jeden globalen sekundären Index mit der Basistabelle. Wenn eine Anwendung Elemente in eine Tabelle schreibt oder daraus löscht, werden globale sekundäre Indizes in dieser Tabelle mit einem Eventually-Consistent-Modell asynchron aktualisiert. Anwendungen schreiben niemals direkt in einen Index. Allerdings ist es wichtig, zu wissen, welche Auswirkungen es hat, wie DynamoDB diese Indizes verwaltet.

Globale sekundäre Indizes erben den read/write Kapazitätsmodus aus der Basistabelle. Weitere Informationen finden Sie unter Überlegungen beim Umstellen der Kapazitätsmodi in DynamoDB.

Wenn Sie einen globalen sekundären Index erstellen, geben Sie ein oder mehrere Indexschlüsselattribute und deren Datentypen an. Das bedeutet, dass wenn Sie ein Element in die Basistabelle schreiben, die Datentypen für diese Attribute den Datentypen des Indexschlüsselschemas entsprechen müssen. Im Fall von `GameTitleIndex` wird der Partitionsschlüssel `GameTitle` im Index als Datentyp `String` definiert. Der Sortierschlüssel `TopScore` im Index ist vom Typ `Number`. Wenn Sie versuchen, der Tabelle `GameScores` ein Element hinzuzufügen und einen anderen Datentyp entweder für `GameTitle` oder `TopScore` anzugeben, gibt DynamoDB eine `ValidationException` aufgrund der fehlenden Übereinstimmung des Datentyps zurück.

Wenn Sie Elemente in eine Tabelle schreiben oder daraus löschen, werden die globalen sekundären Indizes in dieser Tabelle in Eventually Consistent-Form aktualisiert. Änderungen an den Tabellendaten werden unter normalen Bedingungen im Bruchteil einer Sekunde auf die globalen sekundären Indizes verteilt. In einigen seltenen Fehlerszenarien können längere Verzögerungen bei der Verteilung auftreten. Ihre Anwendungen sollten daher Situationen abhelfen können, in denen eine Abfrage für einen globalen sekundären Index Ergebnisse zurückgibt, die nicht auf dem neuesten Stand sind.

Wenn Sie ein Element in eine Tabelle schreiben, müssen Sie keine Attribute für globale sekundäre Index-Sortierschlüssel angeben. Bei `GameTitleIndex` müssen Sie z. B. keinen Wert für das Attribut `TopScore` angeben, um ein neues Element in die Tabelle `GameScores` zu schreiben. In diesem Fall schreibt DynamoDB keine Daten in den Index für dieses bestimmte Element.

Eine Tabelle mit vielen globalen sekundären Indizes verursacht höhere Kosten für Schreibaktivitäten als Tabellen mit weniger Indizes. Weitere Informationen finden Sie unter Überlegungen im Hinblick auf die bereitgestellte Durchsatzkapazität für globale sekundäre Indizes.

Tabellenklassen mit globalen sekundären Indizes

Ein globaler sekundärer Index verwendet immer dieselbe Tabellenklasse wie seine Basistabelle. Jedes Mal, wenn ein neuer globaler sekundärer Index für eine Tabelle hinzugefügt wird, verwendet der neue Index dieselbe Tabellenklasse wie seine Basistabelle. Wenn die Tabellenklasse einer Tabelle aktualisiert wird, werden auch alle zugehörigen globalen sekundären Indizes aktualisiert.

Überlegungen im Hinblick auf die bereitgestellte Durchsatzkapazität für globale sekundäre Indizes

Wenn Sie einen globalen sekundären Index für eine Tabelle mit dem Modus bereitgestellter Kapazität erstellen, müssen Sie Lese- und Schreibkapazitätseinheiten für den erwarteten Workload dieses Indexes angeben. Die Einstellungen für den bereitgestellten Durchsatz eines globalen sekundären Indizes sind getrennt von denen der Basistabelle. Eine `Query`-Operation auf einem globalen sekundären Index verbraucht Lesekapazitätseinheiten des Index und nicht der Basistabelle. Wenn Sie Elemente in eine Tabelle schreiben, sie aktualisieren oder aus der Tabelle löschen, werden die globalen sekundären Indizes in dieser Tabelle in Eventually Consistent-Form aktualisiert. Diese Indexaktualisierungen verbrauchen Kapazitätseinheiten des Indexes und nicht der Basistabelle.

Wenn Sie beispielsweise eine `Query`-Operation für einen ausführen und die bereitgestellte Lesekapazität überschreiten, wird die Anforderung gedrosselt. Wenn Sie viele Schreibaktivitäten für die Tabelle ausführen, ein globaler sekundärer Index dieser Tabelle jedoch über nicht genügend Schreibkapazität verfügt, dann wird die Schreibaktivität der Tabelle gedrosselt.

> **Important**
>
> Damit es zu keiner Ablehnung kommt, sollte die bereitgestellte Kapazität für Schreibvorgänge bei einem globalen sekundären Index gleich oder größer als die Kapazität für Schreibvorgänge der Basistabelle sein, da neue Aktualisierungen in die Basistabelle und den globalen sekundären Index geschrieben werden.

Verwenden Sie zum Anzeigen der Einstellungen des bereitgestellten Durchsatzes für einen globalen sekundären Index die `DescribeTable`-Operation. Es werden detaillierte Informationen zu globalen sekundären Indizes für die Tabelle zurückgegeben.

Lesekapazitätseinheiten

Globale sekundäre Indizes unterstützen Eventually Consistent-Lesevorgänge, die jeweils eine halbe Lesekapazitätseinheit verbrauchen. Das bedeutet, dass eine einzelne globale sekundäre Index-Abfrage bis zu 2 × 4 KB = 8 KB pro Lesekapazitätseinheit abrufen kann.

Für globale sekundäre Index-Abfragen berechnet DynamoDB die bereitgestellten Lesevorgänge auf die gleiche Weise wie für Abfragen der Tabellen. Der einzige Unterschied besteht darin, dass die Berechnung auf der Größe der Indexeinträge und nicht auf der Größe des Elements in der Basistabelle beruht. Die Anzahl der Lesekapazitätseinheiten ist die Summe aller projizierten Attributgrößen sämtlicher zurückgegebener Elemente. Das Ergebnis wird dann auf den nächsten 4 KB-Grenzwert aufgerundet. Weitere Informationen darüber, wie DynamoDB die bereitgestellte Durchsatznutzung berechnet, finden Sie unter DynamoDB – Modus mit bereitgestellter Kapazität.

Die maximale Größe der von einer `Query`-Operation zurückgegebenen Ergebnisse beträgt 1MB. Diese umfasst die Größen aller Attributnamen und Werte sämtlicher zurückgegebenen Elemente.

Nehmen wir als Beispiel einen globalen sekundären Index bei dem jedes Element 2 000 Byte Daten enthält. Angenommen, Sie führen eine `Query`-Operation für diesen Index aus, bei der `KeyConditionExpression` 8 Elemente zurückgegeben werden. Die Gesamtgröße der übereinstimmenden Elemente beträgt: 2 000 Byte x 8 Elemente = 16 000 Byte. Das Ergebnis wird dann auf den nächsten 4-KB-Grenzwert aufgerundet. Da globale sekundäre Index-Abfragen Eventually Consistent ausgeführt werden, belaufen sich die Gesamtkosten auf 0,5 (16 KB/ 4 KB) oder 2 Lesekapazitätseinheiten.

Schreibkapazitätseinheiten

Wenn ein Element in einer Tabelle hinzugefügt, aktualisiert oder gelöscht wird und ein globaler sekundärer Index davon betroffen ist, dann belegt der globale sekundäre Index bereitgestellte Schreibkapazitätseinheiten für die Operation. Die Gesamtkosten für den bereitgestellten Durchsatz ergeben sich aus der Summe der Schreibkapazitätseinheiten, die durch Schreiben in die Basistabelle verbraucht wurden, und der durch Aktualisieren der globalen sekundären Indizes belegten Einheiten. Hinweis: Wenn ein Schreibvorgang in eine Tabelle keine globale sekundäre Index-Aktualisierung erfordert, wird keine Schreibkapazität vom Index verbraucht.

Damit ein Tabellenschreibvorgang erfolgreich ausgeführt wird, muss der für die Tabelle und alle zugehörigen globalen sekundären Indizes bereitgestellte Durchsatz genügend Kapazität aufweisen. Andernfalls wird der Schreibvorgang in die Tabelle gedrosselt.

> **Important**
>
> Bei der Erstellung eines globalen sekundären Index (GSI) können Schreibvorgänge in die Basistabelle gedrosselt werden, wenn die GSI-Aktivität, die sich aus Schreibvorgängen in die Basistabelle ergibt, die bereitgestellte Schreibkapazität der GSI überschreitet. Diese Drosselung wirkt sich auf alle Schreibvorgänge aus, vom Indizierungsprozess bis hin zur möglichen Unterbrechung Ihrer Produktions-Workloads. Weitere Informationen finden Sie unter Beheben von Drosselungsereignissen in Amazon DynamoDB.

Die Kosten für das Schreiben eines Elements in einen globalen sekundären Index hängen von mehreren Faktoren ab:

- Wenn Sie ein neues Element in die Tabelle schreiben, die ein indiziertes Attribut definiert, oder ein vorhandenes Element zum Definieren eines zuvor nicht definierten indizierten Attributs aktualisieren, ist ein Schreibvorgang erforderlich, um das Element in den Index einzufügen.
- Wenn eine Aktualisierung der Tabelle den Wert eines indizierten Schlüsselattributs (von A in B) ändert, sind zwei Schreibvorgänge erforderlich, und zwar einer zum Löschen des vorherigen Elements aus dem Index und einer zum Schreiben des neuen Elements in den Index.
- Wenn ein Element im Index vorhanden war, ein Schreibvorgang in der Tabelle jedoch dazu führte, dass das indizierte Attribut gelöscht wurde, ist ein Schreibvorgang erforderlich, um die alte Elementprojektion im Index zu löschen.
- Wenn ein Element nicht im Index vorhanden ist, bevor oder nachdem das Element aktualisiert wird, fallen keine zusätzlichen Kosten für das Schreiben in den Index an.
- Wenn durch eine Aktualisierung der Tabelle nur der Wert von projizierten Attributen im Indexschlüsselschema, nicht aber der Wert von indizierten Schlüsselattributen geändert wird, ist ein Schreibvorgang erforderlich, um die Werte der projizierten Attribute im Index zu aktualisieren.

Alle diese Faktoren setzen voraus, dass die Größe der einzelnen Elemente im Index kleiner oder gleich der 1-KB- Elementgröße für das Berechnen der Schreibkapazitätseinheiten ist. Größere Indexeinträge erfordern zusätzliche Schreibkapazitätseinheiten. Sie können Ihre Kosten für

Schreibvorgänge minimieren, indem Sie überlegen, welche Attribute Ihre Abfragen zurückgeben müssen, und nur diese Attribute in den Index projizieren.

Speicherüberlegungen für globale sekundäre Indizes

Wenn eine Anwendung ein Element in eine Tabelle schreibt, kopiert DynamoDB automatisch die richtige Teilmenge der Attribute in den globalen sekundären Index, in dem diese Attribute angezeigt werden sollen. Ihrem AWS Konto werden sowohl die Speicherung des Elements in der Basistabelle als auch die Speicherung von Attributen in allen globalen Sekundärindizes dieser Tabelle in Rechnung gestellt.

Der Speicherplatz, der von einem Indexelement belegt wird, ergibt sich aus der Summe von folgenden Werten:

- Die Größe in Byte des Primärschlüssels der Basistabelle (Partitionsschlüssel und Sortierschlüssel)
- Die Größe in Byte des Indexschlüsselattributs
- Die Größe in Byte der projizierten Attribute (sofern vorhanden)
- 100 Bytes des Overheads pro Indexelement

Zur Schätzung der Speicheranforderungen für einen globalen sekundären Index können Sie die durchschnittliche Größe eines Elements im Index schätzen und diesen Wert mit der Anzahl der Elemente in der Basistabelle multiplizieren, die die globalen sekundären Index-Schlüsselattribute besitzen.

Wenn eine Tabelle ein Element enthält, für das ein oder mehrere bestimmte Attribute nicht definiert sind, dieses Attribut jedoch als Indexpartitionsschlüssel oder Sortierschlüssel definiert ist, schreibt DynamoDB keine Daten für dieses Element in den Index.

Muster entwerfen

Entwurfsmuster bieten bewährte Lösungen für häufig auftretende Herausforderungen bei der Arbeit mit globalen Sekundärindizes. Diese Muster helfen Ihnen bei der Entwicklung effizienter, skalierbarer Anwendungen, indem sie Ihnen zeigen, wie Sie Ihre Indizes für bestimmte Anwendungsfälle strukturieren.

Jedes Muster enthält einen vollständigen Implementierungsleitfaden mit Codebeispielen, bewährten Methoden und realen Anwendungsfällen, die Ihnen helfen, das Muster auf Ihre eigenen Anwendungen anzuwenden.

Themen

- Schlüsselmuster mit mehreren Attributen

Schlüsselmuster mit mehreren Attributen

Übersicht

Mit Schlüsseln mit mehreren Attributen können Sie eine GSI-Partition (Global Secondary Index) erstellen und Schlüssel sortieren, die jeweils aus bis zu vier Attributen bestehen. Dadurch wird der clientseitige Code reduziert und es ist einfacher, Daten zunächst zu modellieren und später neue Zugriffsmuster hinzuzufügen.

Stellen Sie sich ein gängiges Szenario vor: Um einen globalen Index zu erstellen, der Elemente anhand mehrerer hierarchischer Attribute abfragt, müssten Sie üblicherweise synthetische Schlüssel erstellen, indem Sie Werte verketten. Um beispielsweise in einer Gaming-App Turnierspiele nach Turnier, Region und Runde abzufragen, könnten Sie einen synthetischen GSI-Partitionsschlüssel wie TOURNAMENT# WINTER2 024 #REGION #NA -EAST und einen synthetischen Sortierschlüssel wie ROUND #SEMIFINALS #BRACKET #UPPER erstellen. Dieser Ansatz funktioniert, erfordert jedoch die Verkettung von Zeichenketten beim Schreiben von Daten, das Parsen beim Lesen und das Auffüllen synthetischer Schlüssel für alle vorhandenen Elemente, wenn Sie den GSI zu einer vorhandenen Tabelle hinzufügen. Dadurch wird der Code unübersichtlicher und es ist schwierig, die Typsicherheit einzelner Schlüsselkomponenten aufrechtzuerhalten.

Schlüssel mit mehreren Attributen lösen dieses Problem für. GSIs Sie definieren Ihren GSI-Partitionsschlüssel mithilfe mehrerer vorhandener Attribute wie TournamentID und Region. DynamoDB verarbeitet die Logik der zusammengesetzten Schlüssel automatisch und hasht sie für die Datenverteilung zusammen. Sie schreiben Elemente mit natürlichen Attributen aus Ihrem Domänenmodell, und die GSI indexiert sie automatisch. Keine Verkettung, kein Parsen, kein Hinterfüllen. Ihr Code bleibt sauber, Ihre Daten bleiben eingegeben und Ihre Abfragen bleiben einfach. Dieser Ansatz ist besonders nützlich, wenn Sie über hierarchische Daten mit natürlichen Attributgruppierungen verfügen (wie Turnier → Region → Runde oder Organisation → Abteilung → Team).

Beispiel für eine Anwendung

In diesem Leitfaden wird der Aufbau eines Systems zur Nachverfolgung von Turnierspielen für eine Esports-Plattform beschrieben. Die Plattform muss Spiele effizient über mehrere Dimensionen hinweg abfragen können: nach Turnier und Region für die Gruppenverwaltung, nach Spielern für den Spielverlauf und nach Datum für die Planung.

Datenmodell

In dieser exemplarischen Vorgehensweise unterstützt das System zur Nachverfolgung von Turnierspielen drei primäre Zugriffsmuster, für die jeweils eine andere Schlüsselstruktur erforderlich ist:

Zugriffsmuster 1: Suchen Sie ein bestimmtes Spiel anhand seiner eindeutigen ID

- Lösung: Basistabelle mit einem `matchId` Partitionsschlüssel

Zugriffsmuster 2: Alle Spiele für ein bestimmtes Turnier und eine bestimmte Region abfragen und optional nach Runde, Bracket oder Spiel filtern

- Lösung: Globaler sekundärer Index mit Partitionsschlüssel (`tournamentId+region`) mit mehreren Attributen und Sortierschlüssel mit mehreren Attributen (`round+ +bracket`) `matchId`
- Beispielabfragen: „Alle WINTER2 024 Spiele in der Region NA-OST“ oder „Alle HALBFINALSPIELE in OBERER Klammer für 024/NA-EAST“ WINTER2

Zugriffsmuster 3: Fragen Sie den Spielverlauf eines Spielers ab und filtern Sie optional nach Zeitraum oder Turnierrunde

- Lösung: Globaler sekundärer Index mit einem einzigen Partitionsschlüssel (`player1Id`) und einem Sortierschlüssel mit mehreren Attributen (`matchDate+round`)
- Beispielabfragen: „Alle Spiele für Spieler 101“ oder „Spiele von Spieler 101 im Januar 2024“

Der entscheidende Unterschied zwischen traditionellen Ansätzen und Methoden mit mehreren Attributen wird deutlich, wenn man sich die Artikelstruktur ansieht:

Traditioneller Ansatz für globale Sekundärindizes (verkettete Schlüssel):

```
// Manual concatenation required for GSI keys
const item = {
    matchId: 'match-001',                                    // Base table PK
    tournamentId: 'WINTER2024',
    region: 'NA-EAST',
    round: 'SEMIFINALS',
    bracket: 'UPPER',
    player1Id: '101',
    // Synthetic keys needed for GSI
```

```
    GSI_PK: `TOURNAMENT#${tournamentId}#REGION#${region}`,       // Must concatenate
    GSI_SK: `${round}#${bracket}#${matchId}`,                    // Must concatenate
    // ... other attributes
};
```

Globaler Sekundärindex-Ansatz mit mehreren Attributen (systemeigene Schlüssel):

```
// Use existing attributes directly - no concatenation needed
const item = {
    matchId: 'match-001',                                          // Base table PK
    tournamentId: 'WINTER2024',
    region: 'NA-EAST',
    round: 'SEMIFINALS',
    bracket: 'UPPER',
    player1Id: '101',
    matchDate: '2024-01-18',
    // No synthetic keys needed - GSI uses existing attributes directly
    // ... other attributes
};
```

Bei Schlüsseln mit mehreren Attributen schreiben Sie Elemente einmal mit natürlichen Domänenattributen. DynamoDB indexiert sie automatisch über mehrere, GSIs ohne dass synthetische verkettete Schlüssel erforderlich sind.

Schema der Basistabelle:

- Partitionsschlüssel: `matchId` (1 Attribut)

Globales sekundäres Indexschema (TournamentRegionIndex mit Schlüsseln mit mehreren Attributen):

- Partitionsschlüssel:`tournamentId`, `region` (2 Attribute)
- Sortierschlüssel:`round`,`bracket`, `matchId` (3 Attribute)

Globales sekundäres Indexschema (PlayerMatchHistoryIndex mit Schlüsseln mit mehreren Attributen):

- Partitionsschlüssel: `player1Id` (1 Attribut)
- Sortierschlüssel:`matchDate`, `round` (2 Attribute)

Basistabelle: TournamentMatches

MatchID (PK)	Turnier-ID	Region	round	Halterun	Spieler1-ID	Spieler2-ID	Datum des Spiels	Gewinne	Bewertung
Match-0(	WINTER 4	IM OSTEN	ENDSPII	MEISTEI HAFT	101	103	20.01.20 4-01	101	3-1
Spiel-00	WINTER 4	IM OSTEN	HALBFIN E	UPPER	101	105	2024-01- 8	101	3-2
Spiel-00	WINTER 4	IM OSTEN	HALBFIN E	UPPER	103	107	2024-01- 8	103	3-0
Spiel-00	WINTER 4	IM OSTEN	VIERTEL NALE	UPPER	101	109	2024-01- 5	101	3-1
Spiel-00	WINTER 4	IM WESTEN	ENDSPII	MEISTEI HAFT	102	104	20.01.20 4-01	102	3-2
Spiel-00(	WINTER 4	IM WESTEN	HALBFIN E	UPPER	102	106	2024-01- 8	102	3-1
Spiel 007	SPRING 4	IM OSTEN	VIERTEL NALE	UPPER	101	108	2024-03- 5	101	3-0
Spiel-00	SPRING 4	IM OSTEN	VIERTEL NALE	LOWER	103	110	2024-03- 5	103	3-2

GSI: TournamentRegionIndex (Schlüssel mit mehreren Attributen)

Turnier-ID (PK)	Region (PK)	rund (SK)	Klammer (SK)	Spiel-ID (SK)	Spieler1-ID	Spieler2-ID	Datum des Spiels	Gewinne	Bewertung
WINTER 4	IM OSTEN	ENDSPII	MEISTEI HAFT	Spiel-00	101	103	20.01.20 4	101	3-1

Turnier-ID (PK)	Region (PK)	rund (SK)	Klammer (SK)	Spiel-ID (SK)	Spieler1-ID	Spieler2-ID	Datum des Spiels	Gewinne	Bewertung
WINTER 4	IM OSTEN	VIERTEL NALE	UPPER	Spiel-004	101	109	15.01.20 4	101	3-1
WINTER 4	IM OSTEN	HALBFIN E	UPPER	Spiel-002	101	105	18.01.20 4	101	3-2
WINTER 4	IM OSTEN	HALBFIN E	UPPER	Spiel-003	103	107	18.01.20 4	103	3-0
WINTER 4	IM WESTEN	ENDSPIL	MEISTEL HAFT	Spiel-005	102	104	20.01.20 4	102	3-2
WINTER 4	IM WESTEN	HALBFIN E	UPPER	Spiel-006	102	106	18.01.20 4	102	3-1
SPRING 4	IM OSTEN	VIERTEL NALE	LOWER	Spiel-008	103	110	15.03.20 4	103	3-2
SPRING 4	IM OSTEN	VIERTEL NALE	UPPER	Spiel-007	101	108	15.03.20 4	101	3-0

GSI: PlayerMatchHistoryIndex (Schlüssel mit mehreren Attributen)

Spieler1-ID (PK)	Spieldatum (SK)	Runde (SK)	ID des Turniers	Region	Halterung	ID abgleichen	Spieler2-ID	Gewinne	Bewertung
101	2024-01-5	VIERTEL NALE	WINTER 4	IM OSTEN	UPPER	Spiel-004	109	101	3-1
101	18.01.20 4	HALBFIN E	WINTER 4	IM OSTEN	UPPER	Spiel-002	105	101	3-2

Spieler1-ID (PK)	Spieldatum (SK)	Runde (SK)	ID des Turniers	Region	Halterung	ID abgleichen	Spieler2-ID	Gewinner	Bewertung
101	20.01.2024	ENDSPIEL	WINTER4	IM OSTEN	MEISTERHAFT	Spiel-001	103	101	3-1
101	15.03.2024	VIERTELFINALE	SPRING4	IM OSTEN	UPPER	Spiel 007	108	101	3-0
102	18.01.2024	HALBFINALE	WINTER4	IM WESTEN	UPPER	Spiel-006	106	102	3-1
102	20.01.2024	ENDSPIEL	WINTER4	IM WESTEN	MEISTERHAFT	Spiel-005	104	102	3-2
103	18.01.2024	HALBFINALE	WINTER4	IM OSTEN	UPPER	Spiel-003	107	103	3-0
103	2024-03-05	VIERTELFINALE	SPRING4	IM OSTEN	LOWER	Spiel-008	110	103	3-2

Voraussetzungen

Stellen Sie vor Beginn sicher, dass Sie über Folgendes verfügen:

Konto und Berechtigungen

- Ein aktives AWS Konto (erstellen Sie hier bei Bedarf eines)
- IAM-Berechtigungen für DynamoDB-Operationen:
  - `dynamodb:CreateTable`
  - `dynamodb:DeleteTable`
  - `dynamodb:DescribeTable`
  - `dynamodb:PutItem`
  - `dynamodb:Query`
  - `dynamodb:BatchWriteItem`

> **Note**
>
> Sicherheitshinweis: Erstellen Sie für Produktionszwecke eine benutzerdefinierte IAM-Richtlinie mit nur den Berechtigungen, die Sie benötigen. Für dieses Tutorial können Sie die AWS verwaltete Richtlinie `AmazonDynamoDBFullAccessV2` verwenden.

Entwicklungsumgebung

- Node.js ist auf Ihrem Computer installiert
- AWS Anmeldeinformationen, die mit einer der folgenden Methoden konfiguriert wurden:

Option 1: AWS CLI

```
aws configure
```

Option 2: Umgebungsvariablen

```
export AWS_ACCESS_KEY_ID=your_access_key_here
export AWS_SECRET_ACCESS_KEY=your_secret_key_here
export AWS_DEFAULT_REGION=us-east-1
```

Installieren erforderlicher Pakete

```
npm install @aws-sdk/client-dynamodb @aws-sdk/lib-dynamodb
```

Implementierung

Schritt 1: Erstellen Sie eine Tabelle mit Schlüsseln GSIs mit mehreren Attributen

Erstellen Sie eine Tabelle mit einer einfachen Basisschlüsselstruktur GSIs , die Schlüssel mit mehreren Attributen verwendet.

Codebeispiel

```
import { DynamoDBClient, CreateTableCommand } from "@aws-sdk/client-dynamodb";

const client = new DynamoDBClient({ region: 'us-west-2' });

const response = await client.send(new CreateTableCommand({
```

```
    TableName: 'TournamentMatches',

    // Base table: Simple partition key
    KeySchema: [
        { AttributeName: 'matchId', KeyType: 'HASH' }              // Simple PK
    ],

    AttributeDefinitions: [
        { AttributeName: 'matchId', AttributeType: 'S' },
        { AttributeName: 'tournamentId', AttributeType: 'S' },
        { AttributeName: 'region', AttributeType: 'S' },
        { AttributeName: 'round', AttributeType: 'S' },
        { AttributeName: 'bracket', AttributeType: 'S' },
        { AttributeName: 'player1Id', AttributeType: 'S' },
        { AttributeName: 'matchDate', AttributeType: 'S' }
    ],

    // GSIs with multi-attribute keys
    GlobalSecondaryIndexes: [
        {
            IndexName: 'TournamentRegionIndex',
            KeySchema: [
                { AttributeName: 'tournamentId', KeyType: 'HASH' },    // GSI PK
attribute 1
                { AttributeName: 'region', KeyType: 'HASH' },          // GSI PK
attribute 2
                { AttributeName: 'round', KeyType: 'RANGE' },          // GSI SK
attribute 1
                { AttributeName: 'bracket', KeyType: 'RANGE' },        // GSI SK
attribute 2
                { AttributeName: 'matchId', KeyType: 'RANGE' }         // GSI SK
attribute 3
            ],
            Projection: { ProjectionType: 'ALL' }
        },
        {
            IndexName: 'PlayerMatchHistoryIndex',
            KeySchema: [
                { AttributeName: 'player1Id', KeyType: 'HASH' },       // GSI PK
                { AttributeName: 'matchDate', KeyType: 'RANGE' },      // GSI SK
attribute 1
                { AttributeName: 'round', KeyType: 'RANGE' }           // GSI SK
attribute 2
            ],
```

```
            Projection: { ProjectionType: 'ALL' }
        }
    ],

    BillingMode: 'PAY_PER_REQUEST'
}));

console.log("Table with multi-attribute GSI keys created successfully");
```

Wichtige Designentscheidungen:

Basistabelle: Die Basistabelle verwendet einen einfachen `matchId` Partitionsschlüssel für direkte Suchabfragen, wodurch die Struktur der Basistabelle übersichtlich bleibt und gleichzeitig die komplexen Abfragemuster GSIs bereitgestellt werden.

TournamentRegionIndex Globaler Sekundärindex: Der `TournamentRegionIndex` globale Sekundärindex verwendet `tournamentId` + `region` als Partitionsschlüssel mit mehreren Attributen, wodurch eine Isolierung der Turnierregion erreicht wird, bei der die Daten durch den Hash beider Attribute zusammen verteilt werden, was effiziente Abfragen innerhalb eines bestimmten Turnierregionskontextes ermöglicht. Der Sortierschlüssel mit mehreren Attributen (`round`+ `bracket` +`matchId`) ermöglicht eine hierarchische Sortierung, die Abfragen auf jeder Hierarchieebene mit natürlicher Reihenfolge von allgemein (rund) bis spezifisch (Match-ID) unterstützt.

PlayerMatchHistoryIndex Globaler Sekundärindex: Der `PlayerMatchHistoryIndex` globale sekundäre Index reorganisiert Daten nach Spielern, die `player1Id` als Partitionsschlüssel verwendet werden, und ermöglicht so turnierübergreifende Abfragen für einen bestimmten Spieler. Der Sortierschlüssel mit mehreren Attributen (`matchDate+round`) bietet eine chronologische Reihenfolge mit der Möglichkeit, nach Datumsbereichen oder bestimmten Turnierrunden zu filtern.

Schritt 2: Fügen Sie Daten mit systemeigenen Attributen ein

Fügen Sie Turnierspieldaten mit natürlichen Attributen hinzu. Die GSI indexiert diese Attribute automatisch, ohne dass synthetische Schlüssel erforderlich sind.

Codebeispiel

```
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";
import { DynamoDBDocumentClient, PutCommand } from "@aws-sdk/lib-dynamodb";

const client = new DynamoDBClient({ region: 'us-west-2' });
const docClient = DynamoDBDocumentClient.from(client);
```

```
// Tournament match data - no synthetic keys needed for GSIs
const matches = [
    // Winter 2024 Tournament, NA-EAST region
    {
        matchId: 'match-001',
        tournamentId: 'WINTER2024',
        region: 'NA-EAST',
        round: 'FINALS',
        bracket: 'CHAMPIONSHIP',
        player1Id: '101',
        player2Id: '103',
        matchDate: '2024-01-20',
        winner: '101',
        score: '3-1'
    },
    {
        matchId: 'match-002',
        tournamentId: 'WINTER2024',
        region: 'NA-EAST',
        round: 'SEMIFINALS',
        bracket: 'UPPER',
        player1Id: '101',
        player2Id: '105',
        matchDate: '2024-01-18',
        winner: '101',
        score: '3-2'
    },
    {
        matchId: 'match-003',
        tournamentId: 'WINTER2024',
        region: 'NA-EAST',
        round: 'SEMIFINALS',
        bracket: 'UPPER',
        player1Id: '103',
        player2Id: '107',
        matchDate: '2024-01-18',
        winner: '103',
        score: '3-0'
    },
    {
        matchId: 'match-004',
        tournamentId: 'WINTER2024',
        region: 'NA-EAST',
        round: 'QUARTERFINALS',
```

```
        bracket: 'UPPER',
        player1Id: '101',
        player2Id: '109',
        matchDate: '2024-01-15',
        winner: '101',
        score: '3-1'
    },

    // Winter 2024 Tournament, NA-WEST region
    {
        matchId: 'match-005',
        tournamentId: 'WINTER2024',
        region: 'NA-WEST',
        round: 'FINALS',
        bracket: 'CHAMPIONSHIP',
        player1Id: '102',
        player2Id: '104',
        matchDate: '2024-01-20',
        winner: '102',
        score: '3-2'
    },
    {
        matchId: 'match-006',
        tournamentId: 'WINTER2024',
        region: 'NA-WEST',
        round: 'SEMIFINALS',
        bracket: 'UPPER',
        player1Id: '102',
        player2Id: '106',
        matchDate: '2024-01-18',
        winner: '102',
        score: '3-1'
    },

    // Spring 2024 Tournament, NA-EAST region
    {
        matchId: 'match-007',
        tournamentId: 'SPRING2024',
        region: 'NA-EAST',
        round: 'QUARTERFINALS',
        bracket: 'UPPER',
        player1Id: '101',
        player2Id: '108',
        matchDate: '2024-03-15',
```

```
        winner: '101',
        score: '3-0'
    },
    {
        matchId: 'match-008',
        tournamentId: 'SPRING2024',
        region: 'NA-EAST',
        round: 'QUARTERFINALS',
        bracket: 'LOWER',
        player1Id: '103',
        player2Id: '110',
        matchDate: '2024-03-15',
        winner: '103',
        score: '3-2'
    }
];

// Insert all matches
for (const match of matches) {
    await docClient.send(new PutCommand({
        TableName: 'TournamentMatches',
        Item: match
    }));

    console.log(`Added: ${match.matchId} - ${match.tournamentId}/${match.region} -
 ${match.round} ${match.bracket}`);
}

console.log(`\nInserted ${matches.length} tournament matches`);
console.log("No synthetic keys created - GSIs use native attributes automatically");
```

Datenstruktur erklärt:

Natürliche Verwendung von Attributen: Jedes Attribut steht für ein echtes Turnierkonzept, bei dem keine Verkettung oder Analyse von Zeichenketten erforderlich ist, sodass eine direkte Zuordnung zum Domänenmodell möglich ist.

Automatische Indizierung des globalen sekundären Indexes: Die Indexierung von Elementen GSIs erfolgt automatisch unter Verwendung der vorhandenen Attribute (`tournamentId`,,,`region`, `matchId` für TournamentRegionIndex und `roundbracket`, `round` für PlayerMatchHistoryIndex), ohne dass `player1Id` synthetische `matchDate` verkettete Schlüssel erforderlich sind.

Kein Backfilling erforderlich: Wenn Sie einer vorhandenen Tabelle einen neuen globalen sekundären Index mit Schlüsseln mit mehreren Attributen hinzufügen, indexiert DynamoDB automatisch alle vorhandenen Elemente anhand ihrer natürlichen Attribute — es ist nicht erforderlich, Elemente mit synthetischen Schlüsseln zu aktualisieren.

Schritt 3: Fragen Sie den globalen sekundären Index mit allen Partitionsschlüsselattributen ab TournamentRegionIndex

In diesem Beispiel wird der TournamentRegionIndex globale sekundäre Index abgefragt, der über einen Partitionsschlüssel mit mehreren Attributen (`tournamentId+region`) verfügt. Alle Partitionsschlüsselattribute müssen in Abfragen mit Gleichheitsbedingungen angegeben werden. Es ist nicht möglich, Abfragen nur `tournamentId` allein oder Ungleichheitsoperatoren für Partitionsschlüsselattribute zu verwenden.

Codebeispiel

```
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";
import { DynamoDBDocumentClient, QueryCommand } from "@aws-sdk/lib-dynamodb";

const client = new DynamoDBClient({ region: 'us-west-2' });
const docClient = DynamoDBDocumentClient.from(client);

// Query GSI: All matches for WINTER2024 tournament in NA-EAST region
const response = await docClient.send(new QueryCommand({
    TableName: 'TournamentMatches',
    IndexName: 'TournamentRegionIndex',
    KeyConditionExpression: 'tournamentId = :tournament AND #region = :region',
    ExpressionAttributeNames: {
        '#region': 'region',  // 'region' is a reserved keyword
        '#tournament': 'tournament'
    },
    ExpressionAttributeValues: {
        ':tournament': 'WINTER2024',
        ':region': 'NA-EAST'
    }
}));

console.log(`Found ${response.Items.length} matches for WINTER2024/NA-EAST:\n`);
response.Items.forEach(match => {
    console.log(`  ${match.round} | ${match.bracket} | ${match.matchId}`);
    console.log(`    Players: ${match.player1Id} vs ${match.player2Id}`);
    console.log(`    Winner: ${match.winner}, Score: ${match.score}\n`);
```

```
});
```

Erwartete Ausgabe:

```
Found 4 matches for WINTER2024/NA-EAST:

  FINALS | CHAMPIONSHIP | match-001
    Players: 101 vs 103
    Winner: 101, Score: 3-1

  QUARTERFINALS | UPPER | match-004
    Players: 101 vs 109
    Winner: 101, Score: 3-1

  SEMIFINALS | UPPER | match-002
    Players: 101 vs 105
    Winner: 101, Score: 3-2

  SEMIFINALS | UPPER | match-003
    Players: 103 vs 107
    Winner: 103, Score: 3-0
```

Ungültige Abfragen:

```
// Missing region attribute
KeyConditionExpression: 'tournamentId = :tournament'

// Using inequality on partition key attribute
KeyConditionExpression: 'tournamentId = :tournament AND #region > :region'
```

Leistung: Partitionsschlüssel mit mehreren Attributen werden zusammen gehasht und bieten so dieselbe O (1) -Suchleistung wie Schlüssel mit einem Attribut.

Schritt 4: Sortierschlüssel für den globalen sekundären Index abfragen left-to-right

Sortierschlüsselattribute müssen left-to-right in der Reihenfolge abgefragt werden, in der sie im globalen sekundären Index definiert sind. Dieses Beispiel zeigt die Abfrage von TournamentRegionIndex auf verschiedenen Hierarchieebenen: das Filtern nur nach`round`, nach `round` + `bracket` oder nach allen drei Sortierschlüsselattributen. Sie können keine Attribute in der Mitte überspringen. Sie können beispielsweise keine Abfragen nach und nach `round` dem Überspringen durchführen. `matchId` `bracket`

Codebeispiel

```
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";
import { DynamoDBDocumentClient, QueryCommand } from "@aws-sdk/lib-dynamodb";

const client = new DynamoDBClient({ region: 'us-west-2' });
const docClient = DynamoDBDocumentClient.from(client);

// Query 1: Filter by first sort key attribute (round)
console.log("Query 1: All SEMIFINALS matches");
const query1 = await docClient.send(new QueryCommand({
    TableName: 'TournamentMatches',
    IndexName: 'TournamentRegionIndex',
    KeyConditionExpression: 'tournamentId = :tournament AND #region = :region AND round
 = :round',
    ExpressionAttributeNames: {
        '#region': 'region'  // 'region' is a reserved keyword
    },
    ExpressionAttributeValues: {
        ':tournament': 'WINTER2024',
        ':region': 'NA-EAST',
        ':round': 'SEMIFINALS'
    }
}));
console.log(`  Found ${query1.Items.length} matches\n`);

// Query 2: Filter by first two sort key attributes (round + bracket)
console.log("Query 2: SEMIFINALS UPPER bracket matches");
const query2 = await docClient.send(new QueryCommand({
    TableName: 'TournamentMatches',
    IndexName: 'TournamentRegionIndex',
    KeyConditionExpression: 'tournamentId = :tournament AND #region = :region AND round
 = :round AND bracket = :bracket',
    ExpressionAttributeNames: {
        '#region': 'region'  // 'region' is a reserved keyword
    },
    ExpressionAttributeValues: {
        ':tournament': 'WINTER2024',
        ':region': 'NA-EAST',
        ':round': 'SEMIFINALS',
        ':bracket': 'UPPER'
    }
}));
console.log(`  Found ${query2.Items.length} matches\n`);
```

```
// Query 3: Filter by all three sort key attributes (round + bracket + matchId)
console.log("Query 3: Specific match in SEMIFINALS UPPER bracket");
const query3 = await docClient.send(new QueryCommand({
    TableName: 'TournamentMatches',
    IndexName: 'TournamentRegionIndex',
    KeyConditionExpression: 'tournamentId = :tournament AND #region = :region AND round
 = :round AND bracket = :bracket AND matchId = :matchId',
    ExpressionAttributeNames: {
        '#region': 'region'  // 'region' is a reserved keyword
    },
    ExpressionAttributeValues: {
        ':tournament': 'WINTER2024',
        ':region': 'NA-EAST',
        ':round': 'SEMIFINALS',
        ':bracket': 'UPPER',
        ':matchId': 'match-002'
    }
}));
console.log(`  Found ${query3.Items.length} matches\n`);

// Query 4: INVALID - skipping round
console.log("Query 4: Attempting to skip first sort key attribute (WILL FAIL)");
try {
    const query4 = await docClient.send(new QueryCommand({
        TableName: 'TournamentMatches',
        IndexName: 'TournamentRegionIndex',
        KeyConditionExpression: 'tournamentId = :tournament AND #region = :region AND
 bracket = :bracket',
        ExpressionAttributeNames: {
            '#region': 'region'  // 'region' is a reserved keyword
        },
        ExpressionAttributeValues: {
            ':tournament': 'WINTER2024',
            ':region': 'NA-EAST',
            ':bracket': 'UPPER'
        }
    }));
} catch (error) {
    console.log(`  Error: ${error.message}`);
    console.log(`  Cannot skip sort key attributes - must query left-to-right\n`);
}
```

Erwartete Ausgabe:

```
Query 1: All SEMIFINALS matches
  Found 2 matches

Query 2: SEMIFINALS UPPER bracket matches
  Found 2 matches

Query 3: Specific match in SEMIFINALS UPPER bracket
  Found 1 matches

Query 4: Attempting to skip first sort key attribute (WILL FAIL)
  Error: Query key condition not supported
  Cannot skip sort key attributes - must query left-to-right
```

Left-to-right Abfrageregeln: Sie müssen Attribute in der Reihenfolge von links nach rechts abfragen, ohne irgendwelche zu überspringen.

Gültige Muster:

- Nur erstes Attribut: `round = 'SEMIFINALS'`
- Die ersten beiden Attribute: `round = 'SEMIFINALS' AND bracket = 'UPPER'`
- Alle drei Attribute: `round = 'SEMIFINALS' AND bracket = 'UPPER' AND matchId = 'match-002'`

Ungültige Muster:

- Das erste Attribut wird übersprungen: `bracket = 'UPPER'` (überspringt die Runde)
- Die Abfrage ist nicht in der richtigen Reihenfolge: `matchId = 'match-002' AND round = 'SEMIFINALS'`
- Lücken hinterlassen: `round = 'SEMIFINALS' AND matchId = 'match-002'` (überspringt die Klammer)

> **Note**
>
> Design-Tipp: Sortieren Sie die Schlüsselattribute von den allgemeinsten bis hin zu den spezifischsten, um die Flexibilität bei der Abfrage zu maximieren.

Schritt 5: Verwenden Sie Ungleichheitsbedingungen für Sortierschlüssel im globalen sekundären Index

Ungleichheitsbedingungen müssen die letzte Bedingung in Ihrer Abfrage sein. Dieses Beispiel zeigt die Verwendung von Vergleichsoperatoren (>=,`BETWEEN`) und Präfixabgleich (`begins_with()`) für Sortierschlüsselattribute. Sobald Sie einen Ungleichheitsoperator verwendet haben, können Sie danach keine weiteren Sortierschlüsselbedingungen hinzufügen — die Ungleichheit muss die letzte Bedingung in Ihrem Schlüsselbedingungsausdruck sein.

Codebeispiel

```
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";
import { DynamoDBDocumentClient, QueryCommand } from "@aws-sdk/lib-dynamodb";

const client = new DynamoDBClient({ region: 'us-west-2' });
const docClient = DynamoDBDocumentClient.from(client);

// Query 1: Round comparison (inequality on first sort key attribute)
console.log("Query 1: Matches from QUARTERFINALS onwards");
const query1 = await docClient.send(new QueryCommand({
    TableName: 'TournamentMatches',
    IndexName: 'TournamentRegionIndex',
    KeyConditionExpression: 'tournamentId = :tournament AND #region = :region AND round
 >= :round',
    ExpressionAttributeNames: {
        '#region': 'region'  // 'region' is a reserved keyword
    },
    ExpressionAttributeValues: {
        ':tournament': 'WINTER2024',
        ':region': 'NA-EAST',
        ':round': 'QUARTERFINALS'
    }
}));
console.log(`  Found ${query1.Items.length} matches\n`);

// Query 2: Round range with BETWEEN
console.log("Query 2: Matches between QUARTERFINALS and SEMIFINALS");
const query2 = await docClient.send(new QueryCommand({
    TableName: 'TournamentMatches',
    IndexName: 'TournamentRegionIndex',
    KeyConditionExpression: 'tournamentId = :tournament AND #region = :region AND round
 BETWEEN :start AND :end',
    ExpressionAttributeNames: {
```

```
        '#region': 'region'  // 'region' is a reserved keyword
    },
    ExpressionAttributeValues: {
        ':tournament': 'WINTER2024',
        ':region': 'NA-EAST',
        ':start': 'QUARTERFINALS',
        ':end': 'SEMIFINALS'
    }
}));
console.log(`  Found ${query2.Items.length} matches\n`);

// Query 3: Prefix matching with begins_with (treated as inequality)
console.log("Query 3: Matches in brackets starting with 'U'");
const query3 = await docClient.send(new QueryCommand({
    TableName: 'TournamentMatches',
    IndexName: 'TournamentRegionIndex',
    KeyConditionExpression: 'tournamentId = :tournament AND #region = :region AND round
 = :round AND begins_with(bracket, :prefix)',
    ExpressionAttributeNames: {
        '#region': 'region'  // 'region' is a reserved keyword
    },
    ExpressionAttributeValues: {
        ':tournament': 'WINTER2024',
        ':region': 'NA-EAST',
        ':round': 'SEMIFINALS',
        ':prefix': 'U'
    }
}));
console.log(`  Found ${query3.Items.length} matches\n`);

// Query 4: INVALID - condition after inequality
console.log("Query 4: Attempting condition after inequality (WILL FAIL)");
try {
    const query4 = await docClient.send(new QueryCommand({
        TableName: 'TournamentMatches',
        IndexName: 'TournamentRegionIndex',
        KeyConditionExpression: 'tournamentId = :tournament AND #region = :region AND
 round > :round AND bracket = :bracket',
        ExpressionAttributeNames: {
            '#region': 'region'  // 'region' is a reserved keyword
        },
        ExpressionAttributeValues: {
            ':tournament': 'WINTER2024',
            ':region': 'NA-EAST',
```

```
            ':round': 'QUARTERFINALS',
            ':bracket': 'UPPER'
        }
    }));
} catch (error) {
    console.log(`  Error: ${error.message}`);
    console.log(`  Cannot add conditions after inequality - it must be last\n`);
}
```

Regeln für Ungleichheitsoperatoren: Sie können Vergleichsoperatoren (>, >=<,<=) BETWEEN für Bereichsabfragen und `begins_with()` für den Präfixabgleich verwenden. Die Ungleichheit muss die letzte Bedingung in Ihrer Abfrage sein.

Gültige Muster:

- Gleichheitsbedingungen, gefolgt von Ungleichheit: `round = 'SEMIFINALS' AND bracket = 'UPPER' AND matchId > 'match-001'`
- Ungleichheit beim ersten Merkmal: `round BETWEEN 'QUARTERFINALS' AND 'SEMIFINALS'`
- Präfixübereinstimmung als letzte Bedingung: `round = 'SEMIFINALS' AND begins_with(bracket, 'U')`

Ungültige Muster:

- Hinzufügen von Bedingungen nach einer Ungleichung: `round > 'QUARTERFINALS' AND bracket = 'UPPER'`
- Verwendung mehrerer Ungleichungen: `round > 'QUARTERFINALS' AND bracket > 'L'`

> **Important**
>
> `begins_with()`wird als Ungleichheitsbedingung behandelt, sodass ihr keine weiteren Sortierschlüsselbedingungen folgen können.

Schritt 6: Fragen Sie den PlayerMatchHistoryIndex globalen sekundären Index mit einem Sortierschlüssel mit mehreren Attributen ab

In diesem Beispiel wird abgefragt PlayerMatchHistoryIndex , welcher einen einzelnen Partitionsschlüssel (`player1Id`) und einen Sortierschlüssel mit mehreren Attributen

(`matchDate+round`) hat. Dies ermöglicht eine turnierübergreifende Analyse, indem alle Spiele eines bestimmten Spielers abgefragt werden, ohne das Turnier IDs zu kennen. In der Basistabelle wären für jede Kombination aus Turnier und Region separate Abfragen erforderlich.

Codebeispiel

```
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";
import { DynamoDBDocumentClient, QueryCommand } from "@aws-sdk/lib-dynamodb";

const client = new DynamoDBClient({ region: 'us-west-2' });
const docClient = DynamoDBDocumentClient.from(client);

// Query 1: All matches for Player 101 across all tournaments
console.log("Query 1: All matches for Player 101");
const query1 = await docClient.send(new QueryCommand({
    TableName: 'TournamentMatches',
    IndexName: 'PlayerMatchHistoryIndex',
    KeyConditionExpression: 'player1Id = :player',
    ExpressionAttributeValues: {
        ':player': '101'
    }
}));

console.log(`  Found ${query1.Items.length} matches for Player 101:`);
query1.Items.forEach(match => {
    console.log(`    ${match.tournamentId}/${match.region} - ${match.matchDate} -
 ${match.round}`);
});
console.log();

// Query 2: Player 101 matches on specific date
console.log("Query 2: Player 101 matches on 2024-01-18");
const query2 = await docClient.send(new QueryCommand({
    TableName: 'TournamentMatches',
    IndexName: 'PlayerMatchHistoryIndex',
    KeyConditionExpression: 'player1Id = :player AND matchDate = :date',
    ExpressionAttributeValues: {
        ':player': '101',
        ':date': '2024-01-18'
    }
}));

console.log(`  Found ${query2.Items.length} matches\n`);
```

```
// Query 3: Player 101 SEMIFINALS matches on specific date
console.log("Query 3: Player 101 SEMIFINALS matches on 2024-01-18");
const query3 = await docClient.send(new QueryCommand({
    TableName: 'TournamentMatches',
    IndexName: 'PlayerMatchHistoryIndex',
    KeyConditionExpression: 'player1Id = :player AND matchDate = :date AND round
 = :round',
    ExpressionAttributeValues: {
        ':player': '101',
        ':date': '2024-01-18',
        ':round': 'SEMIFINALS'
    }
}));

console.log(`  Found ${query3.Items.length} matches\n`);

// Query 4: Player 101 matches in date range
console.log("Query 4: Player 101 matches in January 2024");
const query4 = await docClient.send(new QueryCommand({
    TableName: 'TournamentMatches',
    IndexName: 'PlayerMatchHistoryIndex',
    KeyConditionExpression: 'player1Id = :player AND matchDate BETWEEN :start
 AND :end',
    ExpressionAttributeValues: {
        ':player': '101',
        ':start': '2024-01-01',
        ':end': '2024-01-31'
    }
}));

console.log(`  Found ${query4.Items.length} matches\n`);
```

Variationen des Musters

Zeitreihendaten mit Schlüsseln mit mehreren Attributen

Optimieren Sie für Zeitreihenabfragen mit hierarchischen Zeitattributen

Codebeispiel

```
{
    TableName: 'IoTReadings',
    // Base table: Simple partition key
```

```
    KeySchema: [
        { AttributeName: 'readingId', KeyType: 'HASH' }
    ],
    AttributeDefinitions: [
        { AttributeName: 'readingId', AttributeType: 'S' },
        { AttributeName: 'deviceId', AttributeType: 'S' },
        { AttributeName: 'locationId', AttributeType: 'S' },
        { AttributeName: 'year', AttributeType: 'S' },
        { AttributeName: 'month', AttributeType: 'S' },
        { AttributeName: 'day', AttributeType: 'S' },
        { AttributeName: 'timestamp', AttributeType: 'S' }
    ],
    // GSI with multi-attribute keys for time-series queries
    GlobalSecondaryIndexes: [{
        IndexName: 'DeviceLocationTimeIndex',
        KeySchema: [
            { AttributeName: 'deviceId', KeyType: 'HASH' },
            { AttributeName: 'locationId', KeyType: 'HASH' },
            { AttributeName: 'year', KeyType: 'RANGE' },
            { AttributeName: 'month', KeyType: 'RANGE' },
            { AttributeName: 'day', KeyType: 'RANGE' },
            { AttributeName: 'timestamp', KeyType: 'RANGE' }
        ],
        Projection: { ProjectionType: 'ALL' }
    }],
    BillingMode: 'PAY_PER_REQUEST'
}

// Query patterns enabled via GSI:
// - All readings for device in location
// - Readings for specific year
// - Readings for specific month in year
// - Readings for specific day
// - Readings in time range
```

Vorteile: Die natürliche Zeithierarchie (Jahr → Monat → Tag → Zeitstempel) ermöglicht effiziente Abfragen mit beliebiger Granularität, ohne dass das Datum analysiert oder manipuliert werden muss. Der globale Sekundärindex indexiert automatisch alle Messwerte anhand ihrer natürlichen Zeitattribute.

E-Commerce-Bestellungen mit Schlüsseln mit mehreren Attributen

Verfolgen Sie Bestellungen mit mehreren Dimensionen

Codebeispiel

```
{
    TableName: 'Orders',
    // Base table: Simple partition key
    KeySchema: [
        { AttributeName: 'orderId', KeyType: 'HASH' }
    ],
    AttributeDefinitions: [
        { AttributeName: 'orderId', AttributeType: 'S' },
        { AttributeName: 'sellerId', AttributeType: 'S' },
        { AttributeName: 'region', AttributeType: 'S' },
        { AttributeName: 'orderDate', AttributeType: 'S' },
        { AttributeName: 'category', AttributeType: 'S' },
        { AttributeName: 'customerId', AttributeType: 'S' },
        { AttributeName: 'orderStatus', AttributeType: 'S' }
    ],
    GlobalSecondaryIndexes: [
        {
            IndexName: 'SellerRegionIndex',
            KeySchema: [
                { AttributeName: 'sellerId', KeyType: 'HASH' },
                { AttributeName: 'region', KeyType: 'HASH' },
                { AttributeName: 'orderDate', KeyType: 'RANGE' },
                { AttributeName: 'category', KeyType: 'RANGE' },
                { AttributeName: 'orderId', KeyType: 'RANGE' }
            ],
            Projection: { ProjectionType: 'ALL' }
        },
        {
            IndexName: 'CustomerOrdersIndex',
            KeySchema: [
                { AttributeName: 'customerId', KeyType: 'HASH' },
                { AttributeName: 'orderDate', KeyType: 'RANGE' },
                { AttributeName: 'orderStatus', KeyType: 'RANGE' }
            ],
            Projection: { ProjectionType: 'ALL' }
        }
    ],
    BillingMode: 'PAY_PER_REQUEST'
}

// SellerRegionIndex GSI queries:
// - Orders by seller and region
```

```
// - Orders by seller, region, and date
// - Orders by seller, region, date, and category

// CustomerOrdersIndex GSI queries:
// - Customer's orders
// - Customer's orders by date
// - Customer's orders by date and status
```

Hierarchische Organisationsdaten

Modellieren Sie Organisationshierarchien

Codebeispiel

```
{
    TableName: 'Employees',
    // Base table: Simple partition key
    KeySchema: [
        { AttributeName: 'employeeId', KeyType: 'HASH' }
    ],
    AttributeDefinitions: [
        { AttributeName: 'employeeId', AttributeType: 'S' },
        { AttributeName: 'companyId', AttributeType: 'S' },
        { AttributeName: 'divisionId', AttributeType: 'S' },
        { AttributeName: 'departmentId', AttributeType: 'S' },
        { AttributeName: 'teamId', AttributeType: 'S' },
        { AttributeName: 'skillCategory', AttributeType: 'S' },
        { AttributeName: 'skillLevel', AttributeType: 'S' },
        { AttributeName: 'yearsExperience', AttributeType: 'N' }
    ],
    GlobalSecondaryIndexes: [
        {
            IndexName: 'OrganizationIndex',
            KeySchema: [
                { AttributeName: 'companyId', KeyType: 'HASH' },
                { AttributeName: 'divisionId', KeyType: 'HASH' },
                { AttributeName: 'departmentId', KeyType: 'RANGE' },
                { AttributeName: 'teamId', KeyType: 'RANGE' },
                { AttributeName: 'employeeId', KeyType: 'RANGE' }
            ],
            Projection: { ProjectionType: 'ALL' }
        },
        {
            IndexName: 'SkillsIndex',
```

```
            KeySchema: [
                { AttributeName: 'skillCategory', KeyType: 'HASH' },
                { AttributeName: 'skillLevel', KeyType: 'RANGE' },
                { AttributeName: 'yearsExperience', KeyType: 'RANGE' }
            ],
            Projection: { ProjectionType: 'INCLUDE', NonKeyAttributes: ['employeeId',
 'name'] }
        }
    ],
    BillingMode: 'PAY_PER_REQUEST'
}

// OrganizationIndex GSI query patterns:
// - All employees in company/division
// - Employees in specific department
// - Employees in specific team

// SkillsIndex GSI query patterns:
// - Employees by skill and experience level
```

Wenige Schlüssel mit mehreren Attributen

Kombinieren Sie Schlüssel mit mehreren Attributen, um einen globalen Index mit geringer Dichte zu erstellen

Codebeispiel

```
{
    TableName: 'Products',
    // Base table: Simple partition key
    KeySchema: [
        { AttributeName: 'productId', KeyType: 'HASH' }
    ],
    AttributeDefinitions: [
        { AttributeName: 'productId', AttributeType: 'S' },
        { AttributeName: 'categoryId', AttributeType: 'S' },
        { AttributeName: 'subcategoryId', AttributeType: 'S' },
        { AttributeName: 'averageRating', AttributeType: 'N' },
        { AttributeName: 'reviewCount', AttributeType: 'N' }
    ],
    GlobalSecondaryIndexes: [
        {
            IndexName: 'CategoryIndex',
```

```
            KeySchema: [
                { AttributeName: 'categoryId', KeyType: 'HASH' },
                { AttributeName: 'subcategoryId', KeyType: 'HASH' },
                { AttributeName: 'productId', KeyType: 'RANGE' }
            ],
            Projection: { ProjectionType: 'ALL' }
        },
        {
            IndexName: 'ReviewedProductsIndex',
            KeySchema: [
                { AttributeName: 'categoryId', KeyType: 'HASH' },
                { AttributeName: 'averageRating', KeyType: 'RANGE' },  // Optional
 attribute
                { AttributeName: 'reviewCount', KeyType: 'RANGE' }     // Optional
 attribute
            ],
            Projection: { ProjectionType: 'ALL' }
        }
    ],
    BillingMode: 'PAY_PER_REQUEST'
}

// Only products with reviews appear in ReviewedProductsIndex GSI
// Automatic filtering without application logic
// Multi-attribute sort key enables rating and count queries
```

SaaS für mehrere Mandanten

Mehrinstanzenfähige SaaS-Plattform mit Kundenisolierung

Codebeispiel

```
// Table design
{
    TableName: 'SaasData',
    // Base table: Simple partition key
    KeySchema: [
        { AttributeName: 'resourceId', KeyType: 'HASH' }
    ],
    AttributeDefinitions: [
        { AttributeName: 'resourceId', AttributeType: 'S' },
        { AttributeName: 'tenantId', AttributeType: 'S' },
        { AttributeName: 'customerId', AttributeType: 'S' },
        { AttributeName: 'resourceType', AttributeType: 'S' }
```

```
    ],
    // GSI with multi-attribute keys for tenant-customer isolation
    GlobalSecondaryIndexes: [{
        IndexName: 'TenantCustomerIndex',
        KeySchema: [
            { AttributeName: 'tenantId', KeyType: 'HASH' },
            { AttributeName: 'customerId', KeyType: 'HASH' },
            { AttributeName: 'resourceType', KeyType: 'RANGE' },
            { AttributeName: 'resourceId', KeyType: 'RANGE' }
        ],
        Projection: { ProjectionType: 'ALL' }
    }],
    BillingMode: 'PAY_PER_REQUEST'
}

// Query GSI: All resources for tenant T001, customer C001
const resources = await docClient.send(new QueryCommand({
    TableName: 'SaasData',
    IndexName: 'TenantCustomerIndex',
    KeyConditionExpression: 'tenantId = :tenant AND customerId = :customer',
    ExpressionAttributeValues: {
        ':tenant': 'T001',
        ':customer': 'C001'
    }
}));

// Query GSI: Specific resource type for tenant/customer
const documents = await docClient.send(new QueryCommand({
    TableName: 'SaasData',
    IndexName: 'TenantCustomerIndex',
    KeyConditionExpression: 'tenantId = :tenant AND customerId = :customer AND
 resourceType = :type',
    ExpressionAttributeValues: {
        ':tenant': 'T001',
        ':customer': 'C001',
        ':type': 'document'
    }
}));
```

Vorteile: Effiziente Abfragen im Mandanten-Kunden-Kontext und natürliche Datenorganisation.

Finanztransaktionen

Banksystem zur Verfolgung von Kontotransaktionen mithilfe von GSIs

Codebeispiel

```
// Table design
{
    TableName: 'BankTransactions',
    // Base table: Simple partition key
    KeySchema: [
        { AttributeName: 'transactionId', KeyType: 'HASH' }
    ],
    AttributeDefinitions: [
        { AttributeName: 'transactionId', AttributeType: 'S' },
        { AttributeName: 'accountId', AttributeType: 'S' },
        { AttributeName: 'year', AttributeType: 'S' },
        { AttributeName: 'month', AttributeType: 'S' },
        { AttributeName: 'day', AttributeType: 'S' },
        { AttributeName: 'transactionType', AttributeType: 'S' }
    ],
    GlobalSecondaryIndexes: [
        {
            IndexName: 'AccountTimeIndex',
            KeySchema: [
                { AttributeName: 'accountId', KeyType: 'HASH' },
                { AttributeName: 'year', KeyType: 'RANGE' },
                { AttributeName: 'month', KeyType: 'RANGE' },
                { AttributeName: 'day', KeyType: 'RANGE' },
                { AttributeName: 'transactionId', KeyType: 'RANGE' }
            ],
            Projection: { ProjectionType: 'ALL' }
        },
        {
            IndexName: 'TransactionTypeIndex',
            KeySchema: [
                { AttributeName: 'accountId', KeyType: 'HASH' },
                { AttributeName: 'transactionType', KeyType: 'RANGE' },
                { AttributeName: 'year', KeyType: 'RANGE' },
                { AttributeName: 'month', KeyType: 'RANGE' }
            ],
            Projection: { ProjectionType: 'ALL' }
        }
    ],
    BillingMode: 'PAY_PER_REQUEST'
}

// Query AccountTimeIndex GSI: All transactions for account in 2023
```

```
const yearTransactions = await docClient.send(new QueryCommand({
    TableName: 'BankTransactions',
    IndexName: 'AccountTimeIndex',
    KeyConditionExpression: 'accountId = :account AND #year = :year',
    ExpressionAttributeNames: { '#year': 'year' },
    ExpressionAttributeValues: {
        ':account': 'ACC-12345',
        ':year': '2023'
    }
}));

// Query AccountTimeIndex GSI: Transactions in specific month
const monthTransactions = await docClient.send(new QueryCommand({
    TableName: 'BankTransactions',
    IndexName: 'AccountTimeIndex',
    KeyConditionExpression: 'accountId = :account AND #year = :year AND #month
 = :month',
    ExpressionAttributeNames: { '#year': 'year', '#month': 'month' },
    ExpressionAttributeValues: {
        ':account': 'ACC-12345',
        ':year': '2023',
        ':month': '11'
    }
}));

// Query TransactionTypeIndex GSI: Deposits in 2023
const deposits = await docClient.send(new QueryCommand({
    TableName: 'BankTransactions',
    IndexName: 'TransactionTypeIndex',
    KeyConditionExpression: 'accountId = :account AND transactionType = :type AND #year
 = :year',
    ExpressionAttributeNames: { '#year': 'year' },
    ExpressionAttributeValues: {
        ':account': 'ACC-12345',
        ':type': 'deposit',
        ':year': '2023'
    }
}));
```

Vollständiges Beispiel

Das folgende Beispiel zeigt Schlüssel mit mehreren Attributen von der Einrichtung bis zur Bereinigung:

Codebeispiel

```
import {
    DynamoDBClient,
    CreateTableCommand,
    DeleteTableCommand,
    waitUntilTableExists
} from "@aws-sdk/client-dynamodb";
import {
    DynamoDBDocumentClient,
    PutCommand,
    QueryCommand
} from "@aws-sdk/lib-dynamodb";

const client = new DynamoDBClient({ region: 'us-west-2' });
const docClient = DynamoDBDocumentClient.from(client);

async function multiAttributeKeysDemo() {
    console.log("Starting Multi-Attribute GSI Keys Demo\n");

    // Step 1: Create table with GSIs using multi-attribute keys
    console.log("1. Creating table with multi-attribute GSI keys...");
    await client.send(new CreateTableCommand({
        TableName: 'TournamentMatches',
        KeySchema: [
            { AttributeName: 'matchId', KeyType: 'HASH' }
        ],
        AttributeDefinitions: [
            { AttributeName: 'matchId', AttributeType: 'S' },
            { AttributeName: 'tournamentId', AttributeType: 'S' },
            { AttributeName: 'region', AttributeType: 'S' },
            { AttributeName: 'round', AttributeType: 'S' },
            { AttributeName: 'bracket', AttributeType: 'S' },
            { AttributeName: 'player1Id', AttributeType: 'S' },
            { AttributeName: 'matchDate', AttributeType: 'S' }
        ],
        GlobalSecondaryIndexes: [
            {
                IndexName: 'TournamentRegionIndex',
                KeySchema: [
                    { AttributeName: 'tournamentId', KeyType: 'HASH' },
                    { AttributeName: 'region', KeyType: 'HASH' },
                    { AttributeName: 'round', KeyType: 'RANGE' },
                    { AttributeName: 'bracket', KeyType: 'RANGE' },
```

```
                    { AttributeName: 'matchId', KeyType: 'RANGE' }
                ],
                Projection: { ProjectionType: 'ALL' }
            },
            {
                IndexName: 'PlayerMatchHistoryIndex',
                KeySchema: [
                    { AttributeName: 'player1Id', KeyType: 'HASH' },
                    { AttributeName: 'matchDate', KeyType: 'RANGE' },
                    { AttributeName: 'round', KeyType: 'RANGE' }
                ],
                Projection: { ProjectionType: 'ALL' }
            }
        ],
        BillingMode: 'PAY_PER_REQUEST'
    }));

    await waitUntilTableExists({ client, maxWaitTime: 120 }, { TableName:
'TournamentMatches' });
    console.log("Table created\n");

    // Step 2: Insert tournament matches
    console.log("2. Inserting tournament matches...");
    const matches = [
        { matchId: 'match-001', tournamentId: 'WINTER2024', region: 'NA-EAST', round:
'FINALS', bracket: 'CHAMPIONSHIP', player1Id: '101', player2Id: '103', matchDate:
'2024-01-20', winner: '101', score: '3-1' },
        { matchId: 'match-002', tournamentId: 'WINTER2024', region: 'NA-EAST', round:
'SEMIFINALS', bracket: 'UPPER', player1Id: '101', player2Id: '105', matchDate:
'2024-01-18', winner: '101', score: '3-2' },
        { matchId: 'match-003', tournamentId: 'WINTER2024', region: 'NA-WEST', round:
'FINALS', bracket: 'CHAMPIONSHIP', player1Id: '102', player2Id: '104', matchDate:
'2024-01-20', winner: '102', score: '3-2' },
        { matchId: 'match-004', tournamentId: 'SPRING2024', region: 'NA-EAST', round:
'QUARTERFINALS', bracket: 'UPPER', player1Id: '101', player2Id: '108', matchDate:
'2024-03-15', winner: '101', score: '3-0' }
    ];

    for (const match of matches) {
        await docClient.send(new PutCommand({ TableName: 'TournamentMatches', Item:
match }));
    }
    console.log(`Inserted ${matches.length} tournament matches\n`);
```

```
    // Step 3: Query GSI with multi-attribute partition key
    console.log("3. Query TournamentRegionIndex GSI: WINTER2024/NA-EAST matches");
    const gsiQuery1 = await docClient.send(new QueryCommand({
        TableName: 'TournamentMatches',
        IndexName: 'TournamentRegionIndex',
        KeyConditionExpression: 'tournamentId = :tournament AND #region = :region',
        ExpressionAttributeNames: { '#region': 'region' },
        ExpressionAttributeValues: { ':tournament': 'WINTER2024', ':region': 'NA-
EAST' }
    }));

    console.log(`  Found ${gsiQuery1.Items.length} matches:`);
    gsiQuery1.Items.forEach(match => {
        console.log(`    ${match.round} - ${match.bracket} - ${match.winner} won`);
    });

    // Step 4: Query GSI with multi-attribute sort key
    console.log("\n4. Query PlayerMatchHistoryIndex GSI: All matches for Player 101");
    const gsiQuery2 = await docClient.send(new QueryCommand({
        TableName: 'TournamentMatches',
        IndexName: 'PlayerMatchHistoryIndex',
        KeyConditionExpression: 'player1Id = :player',
        ExpressionAttributeValues: { ':player': '101' }
    }));

    console.log(`  Found ${gsiQuery2.Items.length} matches for Player 101:`);
    gsiQuery2.Items.forEach(match => {
        console.log(`    ${match.tournamentId}/${match.region} - ${match.matchDate} -
 ${match.round}`);
    });

    console.log("\nDemo complete");
    console.log("No synthetic keys needed - GSIs use native attributes automatically");
}

async function cleanup() {
    console.log("Deleting table...");
    await client.send(new DeleteTableCommand({ TableName: 'TournamentMatches' }));
    console.log("Table deleted");
}

// Run demo
multiAttributeKeysDemo().catch(console.error);
```

```
// Uncomment to cleanup:
// cleanup().catch(console.error);
```

Minimales Codegerüst

Codebeispiel

```
// 1. Create table with GSI using multi-attribute keys
await client.send(new CreateTableCommand({
    TableName: 'MyTable',
    KeySchema: [
        { AttributeName: 'id', KeyType: 'HASH' }        // Simple base table PK
    ],
    AttributeDefinitions: [
        { AttributeName: 'id', AttributeType: 'S' },
        { AttributeName: 'attr1', AttributeType: 'S' },
        { AttributeName: 'attr2', AttributeType: 'S' },
        { AttributeName: 'attr3', AttributeType: 'S' },
        { AttributeName: 'attr4', AttributeType: 'S' }
    ],
    GlobalSecondaryIndexes: [{
        IndexName: 'MyGSI',
        KeySchema: [
            { AttributeName: 'attr1', KeyType: 'HASH' },    // GSI PK attribute 1
            { AttributeName: 'attr2', KeyType: 'HASH' },    // GSI PK attribute 2
            { AttributeName: 'attr3', KeyType: 'RANGE' },   // GSI SK attribute 1
            { AttributeName: 'attr4', KeyType: 'RANGE' }    // GSI SK attribute 2
        ],
        Projection: { ProjectionType: 'ALL' }
    }],
    BillingMode: 'PAY_PER_REQUEST'
}));

// 2. Insert items with native attributes (no concatenation needed for GSI)
await docClient.send(new PutCommand({
    TableName: 'MyTable',
    Item: {
        id: 'item-001',
        attr1: 'value1',
        attr2: 'value2',
        attr3: 'value3',
        attr4: 'value4',
        // ... other attributes
    }
```

```
}));

// 3. Query GSI with all partition key attributes
await docClient.send(new QueryCommand({
    TableName: 'MyTable',
    IndexName: 'MyGSI',
    KeyConditionExpression: 'attr1 = :v1 AND attr2 = :v2',
    ExpressionAttributeValues: {
        ':v1': 'value1',
        ':v2': 'value2'
    }
}));

// 4. Query GSI with sort key attributes (left-to-right)
await docClient.send(new QueryCommand({
    TableName: 'MyTable',
    IndexName: 'MyGSI',
    KeyConditionExpression: 'attr1 = :v1 AND attr2 = :v2 AND attr3 = :v3',
    ExpressionAttributeValues: {
        ':v1': 'value1',
        ':v2': 'value2',
        ':v3': 'value3'
    }
}));

// Note: If any attribute name is a DynamoDB reserved keyword, use
 ExpressionAttributeNames:
// KeyConditionExpression: 'attr1 = :v1 AND #attr2 = :v2'
// ExpressionAttributeNames: { '#attr2': 'attr2' }
```

Weitere Ressourcen

- Bewährte Methoden für DynamoDB
- Arbeiten mit Tabellen und Daten
- Globale Sekundärindizes
- Abfrage- und Scanvorgänge

Verwalten globaler sekundärer Indizes in DynamoDB

Dieser Abschnitt beschreibt, wie Sie globale sekundäre Indizes in Amazon DynamoDB erstellen, ändern und löschen.

Themen

- Erstellen einer Tabelle mit globalen sekundären Indizes
- Beschreiben globaler sekundären Indizes in einer Tabelle
- Hinzufügen eines globalen sekundären Index zu einer vorhandenen Tabelle
- Löschen eines globalen sekundären Index
- Ändern eines globalen sekundären Indexes während der Erstellung

Erstellen einer Tabelle mit globalen sekundären Indizes

Um eine Tabelle mit einem oder mehreren globalen sekundären Indizes zu erstellen, verwenden Sie `CreateTable` mit dem Parameter `GlobalSecondaryIndexes`. Für maximale Abfrageflexibilität können Sie bis zu 20 globale sekundäre Indizes pro Tabelle (Standardkontingent) erstellen.

Sie müssen ein Attribut angeben, das als Partitionsschlüssel des Index fungieren soll. Sie können optional ein weiteres Attribut für den Sortierschlüssel des Index festlegen. Diese beiden Schlüsselattribute müssen nicht mit den Schlüsselattributen in der Tabelle übereinstimmen. Beispielsweise sind in der GameScoresTabelle (siehe Verwenden globaler sekundärer Indizes in DynamoDB) `TopScore` weder Schlüsselattribute `TopScoreDateTime` noch Schlüsselattribute enthalten. Sie könnten einen globalen sekundären Index mit einem Partitionsschlüssel von `TopScore` und einem Sortierschlüssel von `TopScoreDateTime` erstellen. Sie könnten einen solchen Index verwenden, um festzustellen, ob eine Verbindung zwischen den Highscores und der Uhrzeit, an der ein Spiel stattfindet, besteht.

Jedes Schlüsselattribut eines Indexes muss ein Skalar des `String`, `Number` oder `Binary` sein. (Es kann kein Dokument oder Satz sein.) Sie können Attribute jeden Datentyps in einen globalen sekundären Index projizieren. Dazu zählen skalare Werte, Dokumente und Sätze. Eine vollständige Liste der Datentypen finden Sie unter Datentypen.

Wenn Sie den bereitgestellten Modus verwenden, müssen Sie die `ProvisionedThroughput`-Einstellungen für den Index, die aus `ReadCapacityUnits` und `WriteCapacityUnits` bestehen, bereitstellen. Diese Einstellungen des bereitgestellten Durchsatzes sind getrennt von denen der Tabelle, aber verhalten sich auf ähnliche Weise. Weitere Informationen finden Sie unter Überlegungen im Hinblick auf die bereitgestellte Durchsatzkapazität für globale sekundäre Indizes.

Globale Sekundärindizes erben den read/write Kapazitätsmodus aus der Basistabelle. Weitere Informationen finden Sie unter Überlegungen beim Umstellen der Kapazitätsmodi in DynamoDB.

> **Note**
>
> Beim Erstellen eines neuen GSI kann es wichtig sein zu prüfen, ob Ihre Wahl des Partitionsschlüssels eine ungleichmäßige oder eingeschränkte Verteilung von Daten oder Datenverkehr über die Partitionsschlüssel-Werte des neuen Index erzeugt. In diesem Fall werden möglicherweise Backfill- und Schreibvorgänge gleichzeitig auftreten und Schreibvorgänge in die Basistabelle drosseln. Der Service ergreift Maßnahmen, um das Potenzial für dieses Szenario zu minimieren, hat jedoch keinen Einblick in die Form von Kundendaten in Bezug auf den Indexpartitionsschlüssel, die gewählte Projektion oder die Spärlichkeit des Index-Primärschlüssels.
> Wenn Sie vermuten, dass Ihr neuer globaler Sekundärindex möglicherweise enge oder verzerrte Daten oder Datenverkehrsverteilung über Partitionsschlüsselwerte hinweg aufweisen könnte, sollten Sie Folgendes berücksichtigen, bevor Sie betrieblich wichtigen Tabellen neue Indizes hinzufügen.
>
> - Es ist möglicherweise am sichersten, den Index zu einem Zeitpunkt hinzuzufügen, zu dem Ihre Anwendung den geringsten Datenverkehr steuert.
> - Erwägen Sie, CloudWatch Contributor Insights für Ihre Basistabelle und Indizes zu aktivieren. Dies gibt Ihnen wertvolle Einblicke in Ihre Verkehrsverteilung.
> - Beobachten Sie `WriteThrottleEvents` den gesamten Prozess und `OnlineIndexPercentageProgress` CloudWatch geben Sie Kennzahlen an. `ThrottledRequests` Passen Sie die bereitgestellte Schreibkapazität nach Bedarf an, um das Backfill innerhalb einer angemessenen Zeit abzuschließen, ohne dass Ihr laufender Betrieb nennenswert beeinträchtigt wird. `OnlineIndexConsumedWriteCapacity`und `OnlineThrottleEvents` es wird erwartet, dass sie beim Auffüllen des Index 0 anzeigen.
> - Seien Sie bereit, die Indexerstellung abzubrechen, falls Sie aufgrund der Schreibdrosselung betriebliche Auswirkungen haben.

Beschreiben globaler sekundären Indizes in einer Tabelle

Um den Status aller globalen sekundären Indizes in einer Tabelle anzuzeigen, verwenden Sie die `DescribeTable`-Operation. Der `GlobalSecondaryIndexes`-Teil der Antwort zeigt alle Indizes der Tabelle, zusammen mit deren jeweiligem aktuellen Status (`IndexStatus`).

`IndexStatus` für einen globalen sekundären Index ist einer der folgenden:

- `CREATING` – Der Index wird derzeit erstellt und ist noch nicht verfügbar.

- `ACTIVE` — Der Index ist betriebsbereit und Anwendungen können in diesem `Query`-Operationen durchführen
- `UPDATING` — Die Einstellungen des bereitgestellten Durchsatzes des Index werden geändert.
- `DELETING` – Der Index wird derzeit gelöscht und kann nicht länger verwendet werden.

Wenn DynamoDB einen globalen sekundären Index fertiggestellt hat, ändert sich der Indexstatus von `CREATING` in `ACTIVE`.

Hinzufügen eines globalen sekundären Index zu einer vorhandenen Tabelle

Um einen globalen sekundären Index einer vorhandenen Tabelle hinzuzufügen, verwenden Sie die `UpdateTable`-Operation mit dem Parameter `GlobalSecondaryIndexUpdates`. Geben Sie die folgenden Informationen ein:

- Ein Indexname. Der Name muss innerhalb der Indizes der Tabelle eindeutig sein.
- Das Schlüsselschema des Index. Sie müssen ein Attribut für den Partitionsschlüssel des Index angeben. Sie können optional ein weiteres Attribut für den Sortierschlüssel des Index festlegen. Diese beiden Schlüsselattribute müssen nicht mit den Schlüsselattributen in der Tabelle übereinstimmen. Die Datentypen für jedes Schemaattribut müssen skalar sein: `String`, `Number` oder `Binary`.
- Die Attribute, die von der Tabelle in den Index projiziert werden:
  - `KEYS_ONLY` — Jeder Eintrag im Index besteht nur aus dem Tabellenpartitionsschlüssel und Sortierschlüsselwerten, sowie den Indexschlüsselwerten.
  - `INCLUDE` — Zusätzlich zu den in `KEYS_ONLY` beschriebenen Attributen, enthält der sekundäre Index andere Nicht-Schlüsselattribute, die Sie angeben.
  - `ALL` — Der Index enthält alle Attribute der Quelltabelle.
- Die Einstellungen für den bereitgestellten Durchsatz für den Index bestehen aus `ReadCapacityUnits` und `WriteCapacityUnits`. Diese Einstellungen für den bereitgestellten Durchsatz sind getrennt von denen der Tabelle.

Sie können nur einen globalen sekundären Index pro `UpdateTable`-Operation verwenden.

Phasen der Index-Erstellung

Wenn Sie einen neuen globalen sekundären Index einer vorhandenen Tabelle hinzufügen, ist die Tabelle weiterhin verfügbar, während der Index erstellt wird. Jedoch ist der neue Index für Abfrageoperationen nicht verfügbar bis sein Status sich von `CREATING` in `ACTIVE` ändert.

> **Note**
>
> Bei der Erstellung des globalen sekundären Index wird Application Auto Scaling nicht verwendet. Wenn die `MIN`-Kapazität für Application Auto Scaling erhöht wird, verkürzt sich die Erstellungszeit für den globalen sekundären Index nicht.

Im Hintergrund erstellt DynamoDB den Index in zwei Phasen:

Ressourcenzuweisung

DynamoDB weist die erforderlichen Rechen- und Speicherressourcen zu, um den Index zu erstellen.

Während der Ressourcenzuordnungsphase ist das `IndexStatus`-Attribut `CREATING` und das `Backfilling`-Attribut ist False. Um den Status einer Tabelle und all ihre sekundären Indizes abzurufen, verwenden Sie die `DescribeTable`-Operation.

Während sich der Index in der Ressourcenzuweisungsphase befindet, können Sie weder den Index noch seine übergeordnete Tabelle löschen. Sie können auch den bereitgestellten Durchsatz des Index oder der Tabelle nicht ändern. Sie in der Tabelle keine anderen Indizes hinzufügen oder löschen. Sie können jedoch den bereitgestellten Durchsatz dieser anderen Indizes ändern.

Abgleichen

Für jedes Element in der Tabelle bestimmt DynamoDB, welche Reihe von Attributen auf Basis seiner Projektion (`KEYS_ONLY`, `INCLUDE`, oder `ALL`) in den Index geschrieben werden. Es schreibt dann diese Attribute in den Index. Während der Abgleichphase verfolgt DynamoDB Elemente, die in der Tabelle hinzugefügt, gelöscht oder aktualisiert werden. Die Attribute dieser Elemente werden, soweit erforderlich, auch in dem Index hinzugefügt, gelöscht oder aktualisiert.

Während der Abgleichphase ist das `IndexStatus`-Attribut auf `CREATING` gesetzt und das `Backfilling`-Attribut ist True. Um den Status einer Tabelle und all ihrer es abzurufen, verwenden Sie die `DescribeTable`-Operation.

Während der Index abgleicht, können Sie seine übergeordnete Tabelle nicht löschen. Sie können jedoch weiterhin den den Index löschen oder den bereitgestellten Durchsatz der Tabelle und dessen sekundäre Indizes ändern.

> **Note**
>
> Während der Abgleichphase können einige Schreibvorgänge regelwidriger Elemente erfolgreich sein, während andere abgelehnt werden. Nach dem Abgleich werden alle Schreibvorgänge für Elemente, die gegen das neue Schlüsselschema des Index verstoßen, abgelehnt. Wir empfehlen, dass Sie das Tool Violation Detector ausführen, nachdem die Abgleichphase abgeschlossen ist, um alle Schlüsselverstöße, die gegebenenfalls aufgetreten sind, zu erkennen und zu beheben. Weitere Informationen finden Sie unter Erkennen und Korrigieren von Indexschlüsselverstößen in DynamoDB.

Während der laufenden Ressourcenzuordnung und Abgleichphasen ist der Index im Status `CREATING`. Während dieser Zeit führt DynamoDB Leseoperationen für die Tabelle durch. Lesevorgänge aus der Basistabelle zum Auffüllen des globalen sekundären Indexes werden Ihnen nicht in Rechnung gestellt.

Sobald die Erstellung des Index abgeschlossen ist, ändert sich sein Status in `ACTIVE`. Sie können den Index nicht `Query` oder `Scan` bis er `ACTIVE` ist.

> **Note**
>
> In einigen Fällen kann DynamoDB aufgrund von Indexschlüsselverstößen keine Daten aus der Tabelle in den Index schreiben. Dies kann in folgenden Fällen passieren:
>
> - Der Datentyp eines Attributwerts stimmt nicht mit dem Datentyp eines Indexschlüssel-Schema-Datentyps überein.
> - Die Größe eines Attributs überschreitet die maximale Länge für ein Indexschlüsselattribut.
> - Ein Indexschlüsselattribut hat eine leere Zeichenfolge oder einen leeren Binärattributwert.
>
> Indexschlüsselverstöße beeinträchtigen die Erstellung des globalen sekundären Indexes nicht. Wenn der Index jedoch `ACTIVE` wird, sind die verstoßenden Schlüssel nicht im Index vorhanden.

DynamoDB bietet ein eigenständiges Tool für das Erkennen und Lösen dieser Probleme. Weitere Informationen finden Sie unter Erkennen und Korrigieren von Indexschlüsselverstößen in DynamoDB.

Hinzufügen eines globalen sekundären Index zu einer großen Tabelle

Die erforderliche Zeit für das Erstellen eines globalen sekundären Indizes hängt von mehreren Faktoren ab, z. B.:

- Die Tabellengröße
- Die Anzahl der Elemente in der Tabelle, die sich für eine Aufnahme in den Index eignen
- Die Reihe von Attributen, die in den Index projiziert werden
- Schreibaktivität der Haupttabelle, während der Index erstellt wird

Wenn Sie einen globalen sekundären Index einer sehr große Tabelle hinzufügen, kann es sehr lang dauern, bis der Erstellungsvorgang abgeschlossen ist. Um den Fortschritt zu überwachen und festzustellen, ob der Index über ausreichende Schreibkapazität verfügt, ziehen Sie die folgenden CloudWatch Amazon-Metriken zu Rate:

- `OnlineIndexPercentageProgress`

Weitere Informationen zu CloudWatch Metriken im Zusammenhang mit DynamoDB finden Sie unter. DynamoDB-Metriken

Important

Möglicherweise müssen Sie sehr große Tabellen auf eine Zulassungsliste setzen, bevor Sie einen globalen sekundären Index erstellen oder aktualisieren können. Bitte wenden Sie sich an den AWS Support, um Ihre Tische auf die Zulassungsliste zu setzen.

Während ein Index abgeglichen wird, verwendet DynamoDB interne Systemkapazität um aus der Tabelle zu lesen. Dies soll die Auswirkung der Indexerstellung minimieren und vermeiden, dass der Tabelle nicht genügend Schreibkapazität zur Verfügung steht.

Löschen eines globalen sekundären Index

Wenn Sie keinen globalen sekundären Index mehr benötigen, können Sie ihn mithilfe von `UpdateTable` verwenden.

Sie können nur einen globalen sekundären Index pro `UpdateTable`verwenden.

Während der globale sekundäre Index gelöscht wird, hat dies keine Auswirkung auf jegliche Lese- oder Schreibaktivität in der übergeordneten Tabelle. Während die Löschung ausgeführt wird, können Sie weiterhin den bereitgestellten Durchsatz anderer Indizes ändern

> **Note**
>
> - Beim Löschen einer Tabelle mithilfe der `DeleteTable`-Aktion, werden alle es in dieser Tabelle ebenfalls gelöscht.
> - Die Löschung des globalen sekundären Index wird Ihrem Konto nicht berechnet.

Ändern eines globalen sekundären Indexes während der Erstellung

Während ein Index erstellt wird, können Sie die `DescribeTable`-Operation verwenden, um zu bestimmen, in welcher Phase er sich befindet. Die Beschreibung für den Index enthält ein Boolesches Attribut, `Backfilling`, um anzugeben, ob DynamoDB derzeit den Index mit Elementen aus der Tabelle lädt. Wenn `Backfilling` True ist, dann ist die Phase der Ressourcenzuordnung abgeschlossen und der Index ist dabei, abzugleichen.

Während des Abgleichens können Sie den erstellten Index löschen. Während der Phase können Sie in der Tabelle keine anderen Indizes hinzufügen oder löschen.

> **Note**
>
> Bei Indizes, die als Teil einer `CreateTable`-Operation erstellt wurden, wird das `Backfilling`-Attribut in der `DescribeTable`-Ausgabe nicht angezeigt. Weitere Informationen finden Sie unter Phasen der Index-Erstellung.

Erkennen und Korrigieren von Indexschlüsselverstößen in DynamoDB

Während der Abgleichphase der globalen sekundären Index-Erstellung untersucht Amazon DynamoDB jedes Element in der Tabelle, um zu ermitteln, ob es in den Index aufgenommen werden

kann. Einige Elemente können möglicherweise nicht in den Index aufgenommen werden, da sie zu Indexschlüsselverstößen führen würden. In diesem Fall verbleiben die Elemente in der Tabelle, der Index verfügt dann jedoch über keinen entsprechenden Eintrag für dieses Element.

Verstoß des Indexschlüssels tritt in folgenden Situationen auf:

- Die Datentypen zwischen einem Attributwert und dem Indexschlüsselschema stimmen nicht überein. Angenommen, eines der Elemente in der Tabelle `GameScores` verfügt über den Wert `TopScore` des Typs `String`. Wenn Sie dann einen globalen sekundären Index mit dem Partitionsschlüssel `TopScore` vom Typ `Number` hinzufügen, verstößt das Element aus der Tabelle gegen den Index.
- Ein Attributwert aus der Tabelle überschreitet die maximale Länge für ein Indexschlüsselattribut. Die maximale Länge des Partitionsschlüssels beträgt 2048 bytes und die maximale Länge des Sortierschlüssels 1024 bytes. Wenn einer der entsprechenden Attributwerte in der Tabelle diese Grenzwerte überschreitet, würde das Element aus der Tabelle gegen den Index verstoßen.

> **Note**
>
> Wenn ein Zeichenfolgen- oder Binär-Attributwert für ein Attribut festgelegt ist, das als Indexschlüssel verwendet wird, muss der Attributwert eine Länge größer als Null haben. Andernfalls würde das Element aus der Tabelle gegen den Indexschlüssel verstoßen. Dieses Tool kennzeichnet diesen Indexschlüsselverstoß derzeit nicht.

Wenn ein Indexschlüsselverstoß auftritt, wird die Auffüllphase ohne Unterbrechung fortgesetzt. Verstoßende Elemente sind jedoch nicht im Index enthalten. Nach Abschluss der Abgleichphase werden alle Schreibvorgänge für Elemente, die gegen das neue Schlüsselschema des Indexes verstoßen, abgelehnt.

Um die Attributwerte in einer Tabelle, die gegen einen Indexschlüssel verstoßen, zu identifizieren und zu korrigieren, verwenden Sie das Tool Violation Detector. Um Violation Detector auszuführen, erstellen Sie eine Konfigurationsdatei, die den Namen einer zu scannenden Tabelle sowie die Datentypen des Partitions- und Sortierschlüssels des globalen sekundären Indizes enthält und angibt, welche Aktionen unternommen werden, sollten Verstöße gegen den Indexschlüssel gefunden werden. Violation Detector kann in einem der beiden folgenden Modi ausgeführt werden:

- Erkennungsmodus — Erkennt Indexschlüsselverstöße. Verwenden Sie den Erkennungsmodus, um die Elemente in der Tabelle zu ermitteln, die in einem globalen sekundären Index zu

Schlüsselverstößen führen. (Sie können optional angeben, dass die den Verstoß verursachenden Tabellenelemente sofort gelöscht werden, nachdem sie gefunden wurden.) Die Ausgabe des Erkennungsmodus wird eine Datei geschrieben, die Sie für weitere Analysen verwenden können.

- Korrekturmodus — Korrigiert Indexschlüsselverstöße. Im Korrekturmodus liest Violation Detector eine Eingabedatei im gleichen Format wie die Ausgabedatei des Erkennungsmodus. Der Korrekturmodus liest die Datensätze aus der Eingabedatei und löscht zu jedem Datensatz die entsprechenden Elemente in der Tabelle bzw. aktualisiert diese. (Wenn Sie beschließen, die Elemente zu aktualisieren, müssen Sie die Eingabedatei bearbeiten und für diese Aktualisierungen entsprechende Werte festlegen.)

Herunterladen und Ausführen von Violation Detector

Violation Detector steht als ausführbares Java-Archiv (`.jar`-Datei) zur Verfügung und kann auf Windows-, MacOS- oder Linux-Computern ausgeführt werden. Der Violation Detector erfordert Java 1.7 (oder höher) und Apache Maven.

- Laden Sie den Verletzungsdetektor herunter von GitHub

Um Violation Detector mit Maven herunterzuladen und zu installieren, befolgen Sie die Anweisungen in der `README.md`-Datei.

Um Violation Detector zu starten, gehen Sie zu dem Verzeichnis, in dem Sie `ViolationDetector.java` erstellt haben, und geben Sie den folgenden Befehl ein:

```
java -jar ViolationDetector.jar [options]
```

Die Violation-Detector-Befehlszeile akzeptiert die folgenden Optionen:

- `-h | --help` – Druckt eine Nutzungszusammenfassung und die Optionen für Violation Detector.
- `-p | --configFilePath value` – Der vollständig qualifizierte Namen einer Violation Detector Konfigurationsdatei. Weitere Informationen finden Sie unter Die Konfigurationsdatei für den Violation Detector.
- `-t | --detect value`– Erkennt Indexschlüsselverstöße in der Tabelle und schreibt sie in die Ausgabedatei von Violation Detector. Wenn der Wert für diesen Parameter auf `keep` festgelegt ist, werden Elemente mit Schlüsselverstößen nicht geändert. Wenn der Wert auf `delete` festgelegt ist, werden Elemente mit Schlüsselverstößen aus der Tabelle gelöscht.

- `-c | --correct value` — Liest Indexschlüsselverstöße aus einer Eingabedatei und nimmt entsprechende Korrekturen an den Elementen in der Tabelle vor. Wenn der Wert für diesen Parameter auf `update` festgelegt ist, werden Elemente mit Schlüsselverstößen mit neuen, konformen Werten aktualisiert. Wenn der Wert auf `delete` festgelegt ist, werden Elemente mit Schlüsselverstößen aus der Tabelle gelöscht.

Die Konfigurationsdatei für den Violation Detector

Zur Laufzeit erfordert das Tool Violation Detector eine Konfigurationsdatei. Die Parameter in dieser Datei bestimmen, auf welche DynamoDB-Ressourcen Violation Detector zugreifen kann und wie viel bereitgestellten Durchsatz es verbrauchen darf. In der Tabelle unten werden diese Parameter beschrieben.

Parametername	Description	Erforderlich?

`awsCredentialsFile`	Der vollständig qualifizierte Namen einer Datei, die Ihre AWS -Anmeldeinformationen enthält. Die Datei mit den Anmeldeinformationen muss das folgende Format aufweisen:
	`accessKey = access_key_id_goes_here`
	`secretKey = secret_key_goes_here`
		Ja

`dynamoDBRegion`	Die AWS Region, in der sich die Tabelle befindet. Beispiel: `us-west-2` .	Ja
`tableName`	Der Name der zu scannenden DynamoDB-Tabelle.	Ja
`gsiHashKeyName`	Der Name des Partitionsschlüssels des Index.	Ja

Parametername	Description	Erforderlich?
`gsiHashKeyType`	Der Datentyp des Partition sschlüssels des Index – `String`, `Number` oder `Binary`: `S | N | B`	Ja
`gsiRangeKeyName`	Der Name des Sortiersc hlüssels des Index. Geben Sie diesen Parameter nicht an, wenn der Index nur über einen einfachen Primärschlüssel (Partitionsschlüssel) verfügt.	Nein
`gsiRangeKeyType`	Der Datentyp des Sortiersc hlüssels–`String`,`Number`, oder`Binary`: `S | N | B` Geben Sie diesen Parameter nicht an, wenn der Index nur über einen einfachen Primärschlüssel (Partitio nsschlüssel) verfügt.	Nein

Parametername	Description	Erforderlich?
`recordDetails`	Gibt an, ob alle Details der Indexschlüsselverstöße in die Ausgabedatei geschrieben werden sollen. Ist dieser Parameter auf `true` (Standardwert) gesetzt, werden alle Informationen zu den Verstoß-Elementen aufgezeichnet. Wenn er auf `false` festgelegt ist, wird nur die Anzahl der Verstöße erfasst.	Nein
`recordGsiValueInViolationRecord`	Gibt an, ob die Werte der verursachenden Indexschlüssel in die Ausgabedatei geschrieben werden sollen. Ist dieser Parameter auf `true` (Standardwert) gesetzt, werden die Schlüsselwerte aufgezeichnet. Ist dieser Parameter auf `false` festgelegt, werden die Schlüsselwerte nicht aufgezeichnet.	Nein

Parametername	Description	Erforderlich?

`detectionOutputPath`	Der vollständige Pfad der Violation-Detector-Ausgabedatei. Dieser Parameter unterstützt das Schreiben in ein lokales Verzeichnis oder in Amazon Simple Storage Service (Amazon S3). Im Folgenden sind einige Beispiele aufgeführt:
	
	`detectionOutputPath = //local/path/filename.csv`
	
	`detectionOutputPath = s3://bucket/filename.csv`
	
	Die Informationen in der Ausgabedatei werden im komma-separierten Werte (CSV)-Format angezeigt. Wenn Sie `detectionOutputPath` nicht festlegen, wird die Ausgabedatei `violation_detection.csv` genannt und in das aktuelle Arbeitsverzeichnis geschrieben.
		Nein

Parametername	Description	Erforderlich?
`numOfSegments`	Die Anzahl der Segmente des parallelen Scans, die verwendet werden sollen, wenn Violation Detector die Tabelle scannt. Der Standardwert ist 1. Dies bedeutet, dass die Tabelle sequenziell gescannt wird. Wenn der Wert 2 oder höher lautet, unterteilt Violation Detector die Tabelle in diese Anzahl logischer Segmente und eine gleichmäßigen Anzahl von Scan-Threads. Die maximale Einstellung für `numOfSegments` lautet 4.096 Tage. Bei größeren Tabellen ist ein paralleler Scan in der Regel schneller als ein sequenzieller Scan. Wenn die Tabelle außerdem groß genug ist, um sich über mehrere Partitionen zu erstrecken, verteilt ein paralleler Scan die Lesevorgänge gleichmäßig auf mehrere Partitionen. Weitere Informationen zu parallelen Scans in DynamoDB finden Sie im Abschnitt Parallele Scans.	Nein

Parametername	Description	Erforderlich?
`numOfViolations`	Der obere Grenzwert für Indexschlüsselverstöße, die in die Ausgabedatei geschrieben werden sollen. Ist dieser Parameter auf `-1` (Standardwert) gesetzt, wird die gesamte Tabelle gescannt. Ist eine positive Ganzzahl festgelegt, beendet Violation Detector den Vorgang nach Erreichen dieser Anzahl von Verstößen.	Nein
`numOfRecords`	Die Anzahl der Elemente in der Tabelle, die gescannt werden sollen. Ist dieser Parameter auf -1 (Standard wert) gesetzt, wird die gesamte Tabelle gescannt. Ist eine positive Ganzzahl festgelegt, beendet Violation Detector den Vorgang, nachdem es diese Anzahl von Elementen in der Tabelle gescannt hat.	Nein

Parametername	Description	Erforderlich?
`readWriteIOPSPercent`	Regelt den Prozentsatz der bereitgestellten Lesekapaz itätseinheiten, die während des Tabellen-Scans verbrauch t werden. Der Bereich gültiger Werte lautet 1 bis `100`. Der Standardwert (25) bedeutet, dass Violation Detector nicht mehr als 25 % des bereitges tellten Lesedurchsatzes der Tabelle verbraucht.	Nein
`correctionInputPath`	Der vollständige Pfad der Violation-Detector-Korrektu reingabedatei. Wenn Sie Violation Detector im Korrektur modus ausführen, wird der Inhalt dieser Datei herangezo gen, um die Datenelemente in der Tabelle, die gegen den verstoßen, zu ändern oder zu löschen. Das Format der `correctio nInputPath` -Datei ist identisch mit der `detection OutputPath` -Datei. So können Sie die Ausgabe des Erkennungsmodus als Eingabe für den Korrektur modus verarbeiten.	Nein

Parametername	Description	Erforderlich?

`correctionOutputPath`	Der vollständige Pfad der Violation-Detector-Korrektu rausgabedatei. Diese Datei wird nur erstellt, wenn bei der Aktualisierung Fehler aufgetret en sind.
	
	Dieser Parameter unterstützt das Schreiben in ein lokales Verzeichnis oder in Amazon S3. Im Folgenden sind einige Beispiele aufgeführt:
	
	`correctionOutputPa th = //local/path/ filename.csv`
	
	`correctio nOutputPath = s3://bucket/filename. csv`
	
	Die Informationen in der Ausgabedatei werden im CSV-Format angezeigt. Wenn Sie `correctionOutputPa th` nicht festlegen, wird die Ausgabedatei `violation _update_errors.csv` genannt und in das aktuelle Arbeitsverzeichnis geschrieb en.
		Nein

Erkennung

Um Verstöße gegen den Indexschlüssel zu erkennen, verwenden Sie Violation Detector mit der Befehlszeilenoption `--detect`. Um zu verstehen, wie diese Option funktioniert, sollten Sie die Tabelle `ProductCatalog` zurate ziehen. Im Folgenden finden Sie eine Liste der Elemente in der Tabelle. Nur der Primärschlüssel (`Id`) und das `Price`-Attribut werden angezeigt.

ID (Primärschlüssel)	Preis
101	5
102	20
103	200
201	100
202	200
203	300
204	400
205	500

Alle Werte für `Price` sind vom Typ `Number`. Da DynamoDB jedoch schemalos ist, können Sie ein Element mit einem nicht numerischen `Price` hinzufügen. Angenommen, wir fügen ein anderes Element zur Tabelle `ProductCatalog` hinzu.

ID (Primärschlüssel)	Preis
999	"Hello"

Die Tabelle verfügt jetzt über insgesamt neun Elemente.

Nun fügen Sie der Tabelle einen neuen globalen sekundären Index hinzu: `PriceIndex`. Der Primärschlüssel für diesen Index ist ein Partitionsschlüssel, `Price`, vom Typ `Number`. Nachdem der Index erstellt wurde, enthält er acht Elemente—aber die `ProductCatalog`-Tabelle verfügt aber über

neun Elemente. Der Grund für diese Abweichung ist, dass der Wert `"Hello"` vom Typ `String` ist, `PriceIndex` aber über einen Primärschlüssel vom Typ `Number` verfügt. Der `String`-Wert verstößt gegen den Schlüssel des globalen sekundären Indizes, also ist er im Index nicht vorhanden.

Um Violation Detector in diesem Szenario zu verwenden, erstellen Sie zuerst eine Konfigurationsdatei, wie z. B. folgende:

```
# Properties file for violation detection tool configuration.
# Parameters that are not specified will use default values.

awsCredentialsFile = /home/alice/credentials.txt
dynamoDBRegion = us-west-2
tableName = ProductCatalog
gsiHashKeyName = Price
gsiHashKeyType = N
recordDetails = true
recordGsiValueInViolationRecord = true
detectionOutputPath = ./gsi_violation_check.csv
correctionInputPath = ./gsi_violation_check.csv
numOfSegments = 1
readWriteIOPSPercent = 40
```

Als Nächstes führen Sie den Violation Detector aus, wie im folgenden Beispiel gezeigt.

```
$  java -jar ViolationDetector.jar --configFilePath config.txt --detect keep

Violation detection started: sequential scan, Table name: ProductCatalog, GSI name:
 PriceIndex
Progress: Items scanned in total: 9,    Items scanned by this thread: 9,    Violations
 found by this thread: 1, Violations deleted by this thread: 0
Violation detection finished: Records scanned: 9, Violations found: 1, Violations
 deleted: 0, see results at: ./gsi_violation_check.csv
```

Wenn der Konfigurationsparameter `recordDetails` auf `true` gesetzt ist, schreibt Violation Detector die Details jedes Verstoßes in die Ausgabedatei, wie im folgenden Beispiel:

```
Table Hash Key,GSI Hash Key Value,GSI Hash Key Violation Type,GSI Hash Key Violation
 Description,GSI Hash Key Update Value(FOR USER),Delete Blank Attributes When Updating?
(Y/N)
```

```
999,"{""S"":""Hello""}",Type Violation,Expected: N Found: S,,
```

Die Ausgabedatei ist im CSV-Format. Die erste Zeile in der Datei ist eine Überschrift, gefolgt von einem Datensatz für jedes Element, das gegen den Indexschlüssel verstößt. Die Felder dieser die Verletzung verursachenden Datensätze sind folgende:

- Hash-Schlüssel der Tabelle — Der Partitionsschlüsselwert des Elements in der Tabelle.
- Tabellenbereichsschlüssel — Der Sortierschlüsselwert des Elements in der Tabelle.
- GSI-Hash-Schlüsselwert – Partitionsschlüsselwert des globalen sekundären Indizes.
- Art des GSI-Hash-Schlüsselverstoßes — Entweder `Type Violation` oder `Size Violation`.
- Beschreibung des GSI-Hash-Schlüsselverstoßes — Die Ursache des Verstoßes.
- Aktualisierungswert des GSI-Hash-Schlüssels (FÜR BENUTZER) — Im Korrekturmodus ein neuer, vom Benutzer angegebener Wert für das Attribut.
- Schlüsselwert des GSI-Bereichs – Der Sortierschlüsselwert des globalen sekundären Indexes.
- Art des GSI-Bereichsschlüsselverstoßes — Entweder `Type Violation` oder `Size Violation`.
- Beschreibung des GSI-Bereichsschlüsselverstoßes — Die Ursache des Verstoßes.
- Aktualisierungswert des GSI-Bereichsschlüssels (FÜR BENUTZER) — Im Korrekturmodus ein neuer, vom Benutzer angegebener Wert für das Attribut.
- Leeres Attribut bei Aktualisierung löschen (J/N) — Bestimmt im Korrekturmodus, ob das den Verstoß verursachende Element in der Tabelle gelöscht (J) oder beibehalten (N) werden soll; jedoch nur, wenn eines der folgenden Felder leer ist:
  - `GSI Hash Key Update Value(FOR USER)`
  - `GSI Range Key Update Value(FOR USER)`

  Wenn eines dieser Felder nicht leer ist, hat `Delete Blank Attribute When Updating(Y/N)` keine Auswirkungen.

> **Note**
>
> Das Ausgabeformat kann abhängig von der Konfigurationsdatei und den Befehlszeilenoptionen variieren. Wenn die Tabelle z. B. über einen einfachen Primärschlüssel (ohne Sortierschlüssel) verfügt, sind in der Ausgabe keine Sortierschlüsselfelder vorhanden. Die den Verstoß verursachenden Datensätze in der Datei sind möglicherweise nicht sortiert.

Korrektur

Um Verstöße gegen den Indexschlüssel zu korrigieren, verwenden Sie Violation Detector mit der Befehlszeilenoption `--correct`. Im Korrekturmodus liest Violation Detector die Eingabedatei, die im Parameter `correctionInputPath` angegeben ist. Diese Datei hat das gleiche Format wie die Datei `detectionOutputPath`. Sie können also die Ausgabe der Erkennung als Eingabe für die Korrektur verwenden.

Violation Detector bietet zwei verschiedene Möglichkeiten zur Korrektur von Indexschlüsselverstößen:

- Verstöße löschen – Löscht die Tabellenelemente, die über Verstoß-Attributwerte verfügen.
- Verstöße aktualisieren – Aktualisiert die Tabellenelemente durch Ersetzen der Verstoß-Attribute durch konforme Werte.

In beiden Fällen können Sie die Ausgabedatei aus dem Erkennungsmodus als Eingabe für den Korrekturmodus verwenden.

Setzen wir unser Beispiel `ProductCatalog` fort: Angenommen, wir möchten das Verstoß-Element aus der Tabelle löschen. Zu diesem Zweck verwenden Sie die folgenden Befehlszeile:

```
$  java -jar ViolationDetector.jar --configFilePath config.txt --correct delete
```

An diesem Punkt werden Sie gebeten zu bestätigen, ob die verletzenden Elemente gelöscht werden sollen.

```
Are you sure to delete all violations on the table?y/n
y
Confirmed, will delete violations on the table...
Violation correction from file started: Reading records from file: ./
gsi_violation_check.csv, will delete these records from table.
Violation correction from file finished: Violations delete: 1, Violations Update: 0
```

Nun verfügen `ProductCatalog` und `PriceIndex` über dieselbe Anzahl Elemente.

Arbeiten mit globalen sekundären Indizes: Java

Sie können die AWS SDK für Java Document API verwenden, um eine Amazon DynamoDB-Tabelle mit einem oder mehreren globalen Sekundärindizes zu erstellen, die Indizes in der Tabelle zu beschreiben und Abfragen mithilfe der Indizes durchzuführen.

Nachfolgend sind die allgemeinen Schritte für Tabellenoperationen aufgeführt.

1. Erstellen Sie eine Instance der `DynamoDB`-Klasse.
2. Stellen Sie den erforderlichen und optionalen Parameter für die Operation bereit, indem Sie die entsprechenden Anforderungsobjekte erstellen.
3. Rufen Sie die entsprechende Methode auf, die vom Client, den Sie im vorhergehenden Schritt erstellt haben, bereitgestellt wird.

Themen

- Erstellen einer Tabelle mit einem globalen sekundären Index
- Beschreiben einer Tabelle mit einem globalen sekundären Index
- Abfragen eines globalen sekundären Indexes
- Beispiel: Globale Sekundärindizes mithilfe der AWS SDK für Java Dokument-API

Erstellen einer Tabelle mit einem globalen sekundären Index

Sie können globale sekundäre Indizes gleichzeitig mit der Tabelle erstellen. Zu diesem Zweck verwenden Sie `CreateTable` und geben Ihre Spezifikationen für ein oder mehrere globale sekundäre Indizes an. Das folgende Java-Codebeispiel erstellt eine Tabelle, die Informationen über Wetterdaten enthält. Der Partitionsschlüssel ist `Location` und der Sortierschlüssel `Date`. Ein globaler sekundärer Index mit Namen `PrecipIndex` ermöglicht einen schnellen Zugriff auf Niederschlagsdaten für verschiedene Standorte.

Im Folgenden sind die Schritte zum Erstellen einer Tabelle mit einem globalen sekundären Index unter Verwendung der DynamoDB-Dokument-API aufgeführt.

1. Erstellen Sie eine Instance der `DynamoDB`-Klasse.
2. Erstellen Sie eine Instance der `CreateTableRequest`-Klasse, um die Anforderungsinformationen bereitzustellen.

   Sie müssen den Tabellennamen, seinen zugehörigen Primärschlüssel und die Werte des bereitgestellten Durchsatzes angeben. Für den globalen sekundären Index müssen Sie den Indexnamen, seine Einstellungen des bereitgestellten Durchsatzes, die Attributdefinitionen für den Index, das Schlüsselschema für den Index und die Attributprojektion angeben.
3. Rufen Sie die `createTable`-Methode auf, indem das Anforderungsobjekt als Parameter festgelegt wird.

Im folgenden Java-Codebeispiel werden die vorherigen Schritte veranschaulicht. Der Code erstellt eine Tabelle (`WeatherData`) mit einem globalen sekundären Index (`PrecipIndex`). Der Index-Partitionsschlüssel ist `Date` und der Sortierschlüssel `Precipitation`. Alle Tabellenattribute werden in den Index projiziert. Benutzer können diesen Index abfragen, um Wetterdaten für ein bestimmtes Datum abzurufen. Optional können diese nach Niederschlagsmenge sortiert werden.

Da es sich bei `Precipitation` um kein Schlüsselattribut für die Tabelle handelt, ist es nicht erforderlich. `WeatherData`-Elemente ohne `Precipitation` erscheinen jedoch nicht in `PrecipIndex`.

```
AmazonDynamoDB client = AmazonDynamoDBClientBuilder.standard().build();
DynamoDB dynamoDB = new DynamoDB(client);

// Attribute definitions
ArrayList<AttributeDefinition> attributeDefinitions = new
 ArrayList<AttributeDefinition>();

attributeDefinitions.add(new AttributeDefinition()
    .withAttributeName("Location")
    .withAttributeType("S"));
attributeDefinitions.add(new AttributeDefinition()
    .withAttributeName("Date")
    .withAttributeType("S"));
attributeDefinitions.add(new AttributeDefinition()
    .withAttributeName("Precipitation")
    .withAttributeType("N"));

// Table key schema
ArrayList<KeySchemaElement> tableKeySchema = new ArrayList<KeySchemaElement>();
tableKeySchema.add(new KeySchemaElement()
    .withAttributeName("Location")
    .withKeyType(KeyType.HASH));  //Partition key
tableKeySchema.add(new KeySchemaElement()
    .withAttributeName("Date")
    .withKeyType(KeyType.RANGE));  //Sort key

// PrecipIndex
GlobalSecondaryIndex precipIndex = new GlobalSecondaryIndex()
    .withIndexName("PrecipIndex")
    .withProvisionedThroughput(new ProvisionedThroughput()
        .withReadCapacityUnits((long) 10)
        .withWriteCapacityUnits((long) 1))
```

```
        .withProjection(new Projection().withProjectionType(ProjectionType.ALL));

ArrayList<KeySchemaElement> indexKeySchema = new ArrayList<KeySchemaElement>();

indexKeySchema.add(new KeySchemaElement()
    .withAttributeName("Date")
    .withKeyType(KeyType.HASH));  //Partition key
indexKeySchema.add(new KeySchemaElement()
    .withAttributeName("Precipitation")
    .withKeyType(KeyType.RANGE));  //Sort key

precipIndex.setKeySchema(indexKeySchema);

CreateTableRequest createTableRequest = new CreateTableRequest()
    .withTableName("WeatherData")
    .withProvisionedThroughput(new ProvisionedThroughput()
        .withReadCapacityUnits((long) 5)
        .withWriteCapacityUnits((long) 1))
    .withAttributeDefinitions(attributeDefinitions)
    .withKeySchema(tableKeySchema)
    .withGlobalSecondaryIndexes(precipIndex);

Table table = dynamoDB.createTable(createTableRequest);
System.out.println(table.getDescription());
```

Sie müssen warten bis DynamoDB die Tabelle erstellt und den Tabellenstatus auf `ACTIVE` setzt. Im Anschluss können Sie die Daten in der Tabelle ablegen.

Beschreiben einer Tabelle mit einem globalen sekundären Index

Um Informationen zu globalen sekundären Indizes in einer Tabelle zu erhalten, verwenden Sie `DescribeTable`. Sie können auf den Namen, das Schlüsselschema und die projizierten Attribute von jedem Index zugreifen.

Im Folgenden werden die Schritte zum Zugriff auf globale sekundäre Index-Informationen in einer Tabelle dargelegt.

1. Erstellen Sie eine Instance der `DynamoDB`-Klasse.
2. Erstellen Sie eine Instance der `Table`-Klasse, um den Index darzustellen, mit dem Sie arbeiten möchten.
3. Rufen Sie die `describe`-Methode für das `Table`-Objekt auf.

Im folgenden Java-Codebeispiel werden die vorherigen Schritte veranschaulicht.

Example

```
AmazonDynamoDB client = AmazonDynamoDBClientBuilder.standard().build();
DynamoDB dynamoDB = new DynamoDB(client);

Table table = dynamoDB.getTable("WeatherData");
TableDescription tableDesc = table.describe();


Iterator<GlobalSecondaryIndexDescription> gsiIter =
 tableDesc.getGlobalSecondaryIndexes().iterator();
while (gsiIter.hasNext()) {
    GlobalSecondaryIndexDescription gsiDesc = gsiIter.next();
    System.out.println("Info for index "
         + gsiDesc.getIndexName() + ":");

    Iterator<KeySchemaElement> kseIter = gsiDesc.getKeySchema().iterator();
    while (kseIter.hasNext()) {
        KeySchemaElement kse = kseIter.next();
        System.out.printf("\t%s: %s\n", kse.getAttributeName(), kse.getKeyType());
    }
    Projection projection = gsiDesc.getProjection();
    System.out.println("\tThe projection type is: "
        + projection.getProjectionType());
    if (projection.getProjectionType().toString().equals("INCLUDE")) {
        System.out.println("\t\tThe non-key projected attributes are: "
            + projection.getNonKeyAttributes());
    }
}
```

Abfragen eines globalen sekundären Indexes

Sie können `Query` für einen globalen sekundären Index genauso nutzen, wie Sie `Query` für eine Tabelle nutzen. Sie müssen den Indexnamen, die Abfragekriterien für den Indexpartitionsschlüssel und Sortierschlüssel (falls vorhanden) und die Attribute angeben, die Sie zurückgeben möchten. In diesem Beispiel ist der Index `PrecipIndex`, der über den Partitionsschlüssel `Date` und den Sortierschlüssel `Precipitation` verfügt. Die Indexabfrage gibt alle Wetterdaten für ein bestimmtes Datum zurück, in denen der Niederschlag größer als Null ist.

Im Folgenden werden die Schritte beschrieben, um einen globalen sekundären Index mithilfe der Document API abzufragen. AWS SDK für Java

1. Erstellen Sie eine Instance der `DynamoDB`-Klasse.
2. Erstellen Sie eine Instance der `Table`-Klasse, um den Index darzustellen, mit dem Sie arbeiten möchten.
3. Erstellen Sie für den Index, den Sie abfragen möchten, eine Instance der `Index`-Klasse.
4. Rufen Sie die `query`-Methode für das `Index`-Objekt auf.

Der Attributname `Date` ist ein DynamoDB-reserviertes Wort. Daher müssen Sie einen Ausdrucksattributnamen als Platzhalter in dem `KeyConditionExpression` verwenden.

Im folgenden Java-Codebeispiel werden die vorherigen Schritte veranschaulicht.

Example

```
AmazonDynamoDB client = AmazonDynamoDBClientBuilder.standard().build();
DynamoDB dynamoDB = new DynamoDB(client);

Table table = dynamoDB.getTable("WeatherData");
Index index = table.getIndex("PrecipIndex");

QuerySpec spec = new QuerySpec()
    .withKeyConditionExpression("#d = :v_date and Precipitation = :v_precip")
    .withNameMap(new NameMap()
        .with("#d", "Date"))
    .withValueMap(new ValueMap()
        .withString(":v_date","2013-08-10")
        .withNumber(":v_precip",0));

ItemCollection<QueryOutcome> items = index.query(spec);
Iterator<Item> iter = items.iterator();
while (iter.hasNext()) {
    System.out.println(iter.next().toJSONPretty());
}
```

Beispiel: Globale Sekundärindizes mithilfe der AWS SDK für Java Dokument-API

Das folgende Java-Codebeispiel zeigt, wie Sie mit globalen sekundären Indizes arbeiten. Das Beispiel erstellt eine Tabelle mit dem Namen `Issues`, die in einem einfachen

Fehlerüberwachungssystem für Softwareentwicklung verwendet werden könnte. Der Partitionsschlüssel ist `IssueId` und der Sortierschlüssel `Title`. Es gibt drei globale sekundäre Indizes in dieser Tabelle:

- `CreateDateIndex` – Der Partitionsschlüssel ist `CreateDate` und der Sortierschlüssel `IssueId`. Zusätzlich zu den Tabellenschlüsseln werden die Attribute `Description` und `Status` in den Index projiziert.
- `TitleIndex` – Der Partitionsschlüssel ist `Title` und der Sortierschlüssel `IssueId`. Keine anderen Attribute als die Tabellenschlüssel werden in den Index projiziert.
- `DueDateIndex` — Der Partitionsschlüssel ist `DueDate`. Ein Sortierschlüssel ist nicht vorhanden.) Alle Tabellenattribute werden in den Index projiziert.

Nachdem die `Issues`-Tabelle erstellt wurde, lädt das Programm die Tabelle mit Daten, die Software-Fehlerberichte darstellen. Anschließend werden die Daten mithilfe der globalen sekundären Indizes abgefragt. Schließlich löscht das Programm die `Issues`-Tabelle.

step-by-stepAnweisungen zum Testen des folgenden Beispiels finden Sie unterJava-Codebeispiele.

Example

```
package com.amazonaws.codesamples.document;

import java.util.ArrayList;
import java.util.Iterator;

import com.amazonaws.services.dynamodbv2.AmazonDynamoDB;
import com.amazonaws.services.dynamodbv2.AmazonDynamoDBClientBuilder;
import com.amazonaws.services.dynamodbv2.document.DynamoDB;
import com.amazonaws.services.dynamodbv2.document.Index;
import com.amazonaws.services.dynamodbv2.document.Item;
import com.amazonaws.services.dynamodbv2.document.ItemCollection;
import com.amazonaws.services.dynamodbv2.document.QueryOutcome;
import com.amazonaws.services.dynamodbv2.document.Table;
import com.amazonaws.services.dynamodbv2.document.spec.QuerySpec;
import com.amazonaws.services.dynamodbv2.document.utils.ValueMap;
import com.amazonaws.services.dynamodbv2.model.AttributeDefinition;
import com.amazonaws.services.dynamodbv2.model.CreateTableRequest;
import com.amazonaws.services.dynamodbv2.model.GlobalSecondaryIndex;
import com.amazonaws.services.dynamodbv2.model.KeySchemaElement;
import com.amazonaws.services.dynamodbv2.model.KeyType;
```

```
import com.amazonaws.services.dynamodbv2.model.Projection;
import com.amazonaws.services.dynamodbv2.model.ProvisionedThroughput;

public class DocumentAPIGlobalSecondaryIndexExample {

    static AmazonDynamoDB client = AmazonDynamoDBClientBuilder.standard().build();
    static DynamoDB dynamoDB = new DynamoDB(client);

    public static String tableName = "Issues";

    public static void main(String[] args) throws Exception {

        createTable();
        loadData();

        queryIndex("CreateDateIndex");
        queryIndex("TitleIndex");
        queryIndex("DueDateIndex");

        deleteTable(tableName);

    }

    public static void createTable() {

        // Attribute definitions
        ArrayList<AttributeDefinition> attributeDefinitions = new
 ArrayList<AttributeDefinition>();

        attributeDefinitions.add(new
 AttributeDefinition().withAttributeName("IssueId").withAttributeType("S"));
        attributeDefinitions.add(new
 AttributeDefinition().withAttributeName("Title").withAttributeType("S"));
        attributeDefinitions.add(new
 AttributeDefinition().withAttributeName("CreateDate").withAttributeType("S"));
        attributeDefinitions.add(new
 AttributeDefinition().withAttributeName("DueDate").withAttributeType("S"));

        // Key schema for table
        ArrayList<KeySchemaElement> tableKeySchema = new ArrayList<KeySchemaElement>();
        tableKeySchema.add(new
 KeySchemaElement().withAttributeName("IssueId").withKeyType(KeyType.HASH)); //
 Partition
```

```
                    // key
        tableKeySchema.add(new
KeySchemaElement().withAttributeName("Title").withKeyType(KeyType.RANGE)); // Sort

                   // key

        // Initial provisioned throughput settings for the indexes
        ProvisionedThroughput ptIndex = new
ProvisionedThroughput().withReadCapacityUnits(1L)
                .withWriteCapacityUnits(1L);

        // CreateDateIndex
        GlobalSecondaryIndex createDateIndex = new
GlobalSecondaryIndex().withIndexName("CreateDateIndex")
                .withProvisionedThroughput(ptIndex)
                .withKeySchema(new
KeySchemaElement().withAttributeName("CreateDate").withKeyType(KeyType.HASH), //
Partition

                        // key
                       new
KeySchemaElement().withAttributeName("IssueId").withKeyType(KeyType.RANGE)) // Sort

                // key
                .withProjection(
                        new
Projection().withProjectionType("INCLUDE").withNonKeyAttributes("Description",
"Status"));

        // TitleIndex
        GlobalSecondaryIndex titleIndex = new
GlobalSecondaryIndex().withIndexName("TitleIndex")
                .withProvisionedThroughput(ptIndex)
                .withKeySchema(new
KeySchemaElement().withAttributeName("Title").withKeyType(KeyType.HASH), // Partition

                     // key
                        new
KeySchemaElement().withAttributeName("IssueId").withKeyType(KeyType.RANGE)) // Sort

                // key
                .withProjection(new Projection().withProjectionType("KEYS_ONLY"));
```

```
        // DueDateIndex
        GlobalSecondaryIndex dueDateIndex = new
GlobalSecondaryIndex().withIndexName("DueDateIndex")
                .withProvisionedThroughput(ptIndex)
                .withKeySchema(new
KeySchemaElement().withAttributeName("DueDate").withKeyType(KeyType.HASH)) //
Partition

                        // key
                .withProjection(new Projection().withProjectionType("ALL"));

        CreateTableRequest createTableRequest = new
CreateTableRequest().withTableName(tableName)
                .withProvisionedThroughput(
                        new ProvisionedThroughput().withReadCapacityUnits((long)
1).withWriteCapacityUnits((long) 1))

.withAttributeDefinitions(attributeDefinitions).withKeySchema(tableKeySchema)
                .withGlobalSecondaryIndexes(createDateIndex, titleIndex, dueDateIndex);

        System.out.println("Creating table " + tableName + "...");
        dynamoDB.createTable(createTableRequest);

        // Wait for table to become active
        System.out.println("Waiting for " + tableName + " to become ACTIVE...");
        try {
            Table table = dynamoDB.getTable(tableName);
            table.waitForActive();
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }

    public static void queryIndex(String indexName) {

        Table table = dynamoDB.getTable(tableName);


System.out.println("\n***********************************************************\n");
        System.out.print("Querying index " + indexName + "...");

        Index index = table.getIndex(indexName);

        ItemCollection<QueryOutcome> items = null;
```

```
        QuerySpec querySpec = new QuerySpec();

        if (indexName == "CreateDateIndex") {
            System.out.println("Issues filed on 2013-11-01");
            querySpec.withKeyConditionExpression("CreateDate = :v_date and
begins_with(IssueId, :v_issue)")
                    .withValueMap(new ValueMap().withString(":v_date",
"2013-11-01").withString(":v_issue", "A-"));
            items = index.query(querySpec);
        } else if (indexName == "TitleIndex") {
            System.out.println("Compilation errors");
            querySpec.withKeyConditionExpression("Title = :v_title and
begins_with(IssueId, :v_issue)")
                    .withValueMap(
                            new ValueMap().withString(":v_title", "Compilation
error").withString(":v_issue", "A-"));
            items = index.query(querySpec);
        } else if (indexName == "DueDateIndex") {
            System.out.println("Items that are due on 2013-11-30");
            querySpec.withKeyConditionExpression("DueDate = :v_date")
                    .withValueMap(new ValueMap().withString(":v_date", "2013-11-30"));
            items = index.query(querySpec);
        } else {
            System.out.println("\nNo valid index name provided");
            return;
        }

        Iterator<Item> iterator = items.iterator();

        System.out.println("Query: printing results...");

        while (iterator.hasNext()) {
            System.out.println(iterator.next().toJSONPretty());
        }

    }

    public static void deleteTable(String tableName) {

        System.out.println("Deleting table " + tableName + "...");

        Table table = dynamoDB.getTable(tableName);
        table.delete();
```

```
        // Wait for table to be deleted
        System.out.println("Waiting for " + tableName + " to be deleted...");
        try {
            table.waitForDelete();
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }

    public static void loadData() {

        System.out.println("Loading data into table " + tableName + "...");

        // IssueId, Title,
        // Description,
        // CreateDate, LastUpdateDate, DueDate,
        // Priority, Status

        putItem("A-101", "Compilation error", "Can't compile Project X - bad version
number. What does this mean?",
                "2013-11-01", "2013-11-02", "2013-11-10", 1, "Assigned");

        putItem("A-102", "Can't read data file", "The main data file is missing, or the
permissions are incorrect",
                "2013-11-01", "2013-11-04", "2013-11-30", 2, "In progress");

        putItem("A-103", "Test failure", "Functional test of Project X produces
errors", "2013-11-01", "2013-11-02",
                "2013-11-10", 1, "In progress");

        putItem("A-104", "Compilation error", "Variable 'messageCount' was not
initialized.", "2013-11-15",
                "2013-11-16", "2013-11-30", 3, "Assigned");

        putItem("A-105", "Network issue", "Can't ping IP address 127.0.0.1. Please fix
this.", "2013-11-15",
                "2013-11-16", "2013-11-19", 5, "Assigned");

    }

    public static void putItem(
```

```
            String issueId, String title, String description, String createDate, String
 lastUpdateDate, String dueDate,
            Integer priority, String status) {

        Table table = dynamoDB.getTable(tableName);

        Item item = new Item().withPrimaryKey("IssueId", issueId).withString("Title",
 title)
                .withString("Description", description).withString("CreateDate",
 createDate)
                .withString("LastUpdateDate", lastUpdateDate).withString("DueDate",
 dueDate)
                .withNumber("Priority", priority).withString("Status", status);

        table.putItem(item);
    }

}
```

Arbeiten mit globalen sekundären Indizes: .NET

Sie können die AWS SDK für .NET Low-Level-API verwenden, um eine Amazon DynamoDB-Tabelle mit einem oder mehreren globalen Sekundärindizes zu erstellen, die Indizes in der Tabelle zu beschreiben und Abfragen mithilfe der Indizes durchzuführen. Diese Operationen entsprechen den entsprechenden DynamoDB-Operationen. Weitere Informationen finden Sie in der Amazon-DynamoDB-API-Referenz.

Folgende sind die allgemeinen Schritte für Tabellenoperationen mithilfe der .NET-Low-Level-API.

1. Erstellen Sie eine Instance der `AmazonDynamoDBClient`-Klasse.
2. Stellen Sie den erforderlichen und optionalen Parameter für die Operation bereit, indem Sie die entsprechenden Anforderungsobjekte erstellen.

   Erstellen Sie beispielsweise ein `CreateTableRequest`-Objekt, um eine Tabelle zu erstellen und ein `QueryRequest`-Objekt, um eine Tabelle oder einen Index abzufragen.
3. Rufen Sie die entsprechende Methode auf, die vom Client, den Sie im vorhergehenden Schritt erstellt haben, bereitgestellt wird.

Themen

- Erstellen einer Tabelle mit einem globalen sekundären Index
- Beschreiben einer Tabelle mit einem globalen sekundären Index
- Abfragen eines globalen sekundären Indexes
- Beispiel: Globale Sekundärindizes unter Verwendung der Low-Level-API AWS SDK für .NET

Erstellen einer Tabelle mit einem globalen sekundären Index

Sie können globale sekundäre Indizes gleichzeitig mit der Tabelle erstellen. Zu diesem Zweck verwenden Sie `CreateTable` und geben Ihre Spezifikationen für ein oder mehrere globale sekundäre Indizes an. Das folgende C#-Codebeispiel erstellt eine Tabelle, die Informationen über Wetterdaten enthält. Der Partitionsschlüssel ist `Location` und der Sortierschlüssel `Date`. Ein globaler sekundärer Index mit Namen `PrecipIndex` ermöglicht einen schnellen Zugriff auf Niederschlagsdaten für verschiedene Standorte.

Im Folgenden werden die Schritte zum Erstellen einer Tabelle mit einem globalen sekundären Index mithilfe der .NET-Low-Level-API dargelegt.

1. Erstellen Sie eine Instance der `AmazonDynamoDBClient`-Klasse.
2. Erstellen Sie eine Instance der `CreateTableRequest`-Klasse, um die Anforderungsinformationen bereitzustellen.

   Sie müssen den Tabellennamen, seinen zugehörigen Primärschlüssel und die Werte des bereitgestellten Durchsatzes angeben. Für den globalen sekundären Index müssen Sie den Indexnamen, seine Einstellungen des bereitgestellten Durchsatzes, die Attributdefinitionen für den Index, das Schlüsselschema für den Index und die Attributprojektion angeben.
3. Führen Sie die `CreateTable`-Methode aus, indem das Anforderungsobjekt als Parameter festgelegt wird.

Im folgenden C#-Codebeispiel werden die vorherigen Schritte veranschaulicht. Erstellt eine Tabelle (`WeatherData`) mit einem globalen und lokalen sekundären Index (`PrecipIndex`). Der Index-Partitionsschlüssel ist `Date` und der Sortierschlüssel `Precipitation`. Alle Tabellenattribute werden in den Index projiziert. Benutzer können diesen Index abfragen, um Wetterdaten für ein bestimmtes Datum abzurufen. Optional können diese nach Niederschlagsmenge sortiert werden.

Da es sich bei `Precipitation` um kein Schlüsselattribut für die Tabelle handelt, ist es nicht erforderlich. `WeatherData`-Elemente ohne `Precipitation` erscheinen jedoch nicht in `PrecipIndex`.

```
client = new AmazonDynamoDBClient();
string tableName = "WeatherData";

// Attribute definitions
var attributeDefinitions = new List<AttributeDefinition>()
{
    {new AttributeDefinition{
        AttributeName = "Location",
        AttributeType = "S"}},
    {new AttributeDefinition{
        AttributeName = "Date",
        AttributeType = "S"}},
    {new AttributeDefinition(){
        AttributeName = "Precipitation",
        AttributeType = "N"}
    }
};

// Table key schema
var tableKeySchema = new List<KeySchemaElement>()
{
    {new KeySchemaElement {
        AttributeName = "Location",
        KeyType = "HASH"}},  //Partition key
    {new KeySchemaElement {
        AttributeName = "Date",
        KeyType = "RANGE"}  //Sort key
    }
};

// PrecipIndex
var precipIndex = new GlobalSecondaryIndex
{
    IndexName = "PrecipIndex",
    ProvisionedThroughput = new ProvisionedThroughput
    {
        ReadCapacityUnits = (long)10,
        WriteCapacityUnits = (long)1
    },
    Projection = new Projection { ProjectionType = "ALL" }
};

var indexKeySchema = new List<KeySchemaElement> {
```

```
    {new KeySchemaElement { AttributeName = "Date", KeyType = "HASH"}},  //Partition
 key
    {new KeySchemaElement{AttributeName = "Precipitation",KeyType = "RANGE"}}  //Sort
 key
};

precipIndex.KeySchema = indexKeySchema;

CreateTableRequest createTableRequest = new CreateTableRequest
{
    TableName = tableName,
    ProvisionedThroughput = new ProvisionedThroughput
    {
        ReadCapacityUnits = (long)5,
        WriteCapacityUnits = (long)1
    },
    AttributeDefinitions = attributeDefinitions,
    KeySchema = tableKeySchema,
    GlobalSecondaryIndexes = { precipIndex }
};

CreateTableResponse response = client.CreateTable(createTableRequest);
Console.WriteLine(response.CreateTableResult.TableDescription.TableName);
Console.WriteLine(response.CreateTableResult.TableDescription.TableStatus);
```

Sie müssen warten bis DynamoDB die Tabelle erstellt und den Tabellenstatus auf `ACTIVE` setzt. Im Anschluss können Sie die Daten in der Tabelle ablegen.

Beschreiben einer Tabelle mit einem globalen sekundären Index

Um Informationen zu globalen sekundären Indizes in einer Tabelle zu erhalten, verwenden Sie `DescribeTable`. Sie können auf den Namen, das Schlüsselschema und die projizierten Attribute von jedem Index zugreifen.

Im Folgenden werden die Schritte zum Zugriff auf globale sekundäre Index-Informationen in einer Tabelle mithilfe der .NET-Low-Level-API dargelegt.

1. Erstellen Sie eine Instance der `AmazonDynamoDBClient`-Klasse.
2. Führen Sie die `describeTable`-Methode aus, indem das Anforderungsobjekt als Parameter festgelegt wird.

Erstellen Sie eine Instance der `DescribeTableRequest`-Klasse, um die Anforderungsinformationen bereitzustellen. Sie müssen den Tabellennamen angeben.

3.

Im folgenden C#-Codebeispiel werden die vorherigen Schritte veranschaulicht.

Example

```
client = new AmazonDynamoDBClient();
string tableName = "WeatherData";

DescribeTableResponse response = client.DescribeTable(new DescribeTableRequest
 { TableName = tableName});

List<GlobalSecondaryIndexDescription> globalSecondaryIndexes =
response.DescribeTableResult.Table.GlobalSecondaryIndexes;

// This code snippet will work for multiple indexes, even though
// there is only one index in this example.

foreach (GlobalSecondaryIndexDescription gsiDescription in globalSecondaryIndexes) {
     Console.WriteLine("Info for index " + gsiDescription.IndexName + ":");

     foreach (KeySchemaElement kse in gsiDescription.KeySchema) {
          Console.WriteLine("\t" + kse.AttributeName + ": key type is " + kse.KeyType);
     }

      Projection projection = gsiDescription.Projection;
      Console.WriteLine("\tThe projection type is: " + projection.ProjectionType);

      if (projection.ProjectionType.ToString().Equals("INCLUDE")) {
           Console.WriteLine("\t\tThe non-key projected attributes are: "
                + projection.NonKeyAttributes);
      }
}
```

Abfragen eines globalen sekundären Indexes

Sie können `Query` für einen globalen sekundären Index genauso nutzen, wie Sie `Query` für eine Tabelle nutzen. Sie müssen den Indexnamen, die Abfragekriterien für den Indexpartitionsschlüssel und Sortierschlüssel (falls vorhanden) und die Attribute angeben, die Sie zurückgeben möchten.

In diesem Beispiel ist der Index `PrecipIndex`, der über den Partitionsschlüssel `Date` und den Sortierschlüssel `Precipitation` verfügt. Die Indexabfrage gibt alle Wetterdaten für ein bestimmtes Datum zurück, in denen der Niederschlag größer als Null ist.

Im Folgenden werden die Schritte zur Abfrage eines globalen sekundären Indizes mithilfe der .NET-Low-Level-API dargelegt.

1. Erstellen Sie eine Instance der `AmazonDynamoDBClient`-Klasse.
2. Erstellen Sie eine Instance der `QueryRequest`-Klasse, um die Anforderungsinformationen bereitzustellen.
3. Führen Sie die `query`-Methode aus, indem das Anforderungsobjekt als Parameter festgelegt wird.

Der Attributname `Date` ist ein DynamoDB-reserviertes Wort. Daher müssen Sie einen Ausdrucksattributnamen als Platzhalter in dem `KeyConditionExpression` verwenden.

Im folgenden C#-Codebeispiel werden die vorherigen Schritte veranschaulicht.

Example

```
client = new AmazonDynamoDBClient();

QueryRequest queryRequest = new QueryRequest
{
    TableName = "WeatherData",
    IndexName = "PrecipIndex",
    KeyConditionExpression = "#dt = :v_date and Precipitation > :v_precip",
    ExpressionAttributeNames = new Dictionary<String, String> {
        {"#dt", "Date"}
    },
    ExpressionAttributeValues = new Dictionary<string, AttributeValue> {
        {":v_date", new AttributeValue { S =  "2013-08-01" }},
        {":v_precip", new AttributeValue { N =  "0" }}
    },
    ScanIndexForward = true
};

var result = client.Query(queryRequest);

var items = result.Items;
foreach (var currentItem in items)
```

```
{
    foreach (string attr in currentItem.Keys)
    {
        Console.Write(attr + "---> ");
        if (attr == "Precipitation")
        {
            Console.WriteLine(currentItem[attr].N);
    }
    else
    {
        Console.WriteLine(currentItem[attr].S);
    }

         }
     Console.WriteLine();
}
```

Beispiel: Globale Sekundärindizes unter Verwendung der Low-Level-API AWS SDK für .NET

Das folgende C#-Codebeispiel zeigt, wie Sie mit globalen sekundären Indizes arbeiten. Das Beispiel erstellt eine Tabelle mit dem Namen `Issues`, die in einem einfachen Fehlerüberwachungssystem für Softwareentwicklung verwendet werden könnte. Der Partitionsschlüssel ist `IssueId` und der Sortierschlüssel `Title`. Es gibt drei globale sekundäre Indizes in dieser Tabelle:

- `CreateDateIndex` – Der Partitionsschlüssel ist `CreateDate` und der Sortierschlüssel `IssueId`. Zusätzlich zu den Tabellenschlüsseln werden die Attribute `Description` und `Status` in den Index projiziert.
- `TitleIndex` – Der Partitionsschlüssel ist `Title` und der Sortierschlüssel `IssueId`. Keine anderen Attribute als die Tabellenschlüssel werden in den Index projiziert.
- `DueDateIndex` — Der Partitionsschlüssel ist `DueDate`. Ein Sortierschlüssel ist nicht vorhanden.) Alle Tabellenattribute werden in den Index projiziert.

Nachdem die `Issues`-Tabelle erstellt wurde, lädt das Programm die Tabelle mit Daten, die Software-Fehlerberichte darstellen. Anschließend werden die Daten mithilfe der globalen sekundären Indizes abgefragt. Schließlich löscht das Programm die `Issues`-Tabelle.

step-by-stepAnweisungen zum Testen des folgenden Beispiels finden Sie unter. .NET-Codebeispiele

Example

```
using System;
using System.Collections.Generic;
using System.Linq;
using Amazon.DynamoDBv2;
using Amazon.DynamoDBv2.DataModel;
using Amazon.DynamoDBv2.DocumentModel;
using Amazon.DynamoDBv2.Model;
using Amazon.Runtime;
using Amazon.SecurityToken;

namespace com.amazonaws.codesamples
{
    class LowLevelGlobalSecondaryIndexExample
    {
        private static AmazonDynamoDBClient client = new AmazonDynamoDBClient();
        public static String tableName = "Issues";

        public static void Main(string[] args)
        {
            CreateTable();
            LoadData();

            QueryIndex("CreateDateIndex");
            QueryIndex("TitleIndex");
            QueryIndex("DueDateIndex");

            DeleteTable(tableName);

            Console.WriteLine("To continue, press enter");
            Console.Read();
        }

        private static void CreateTable()
        {
            // Attribute definitions
            var attributeDefinitions = new List<AttributeDefinition>()
        {
            {new AttributeDefinition {
                 AttributeName = "IssueId", AttributeType = "S"
             }},
            {new AttributeDefinition {
                 AttributeName = "Title", AttributeType = "S"
```

```
             }},
            {new AttributeDefinition {
                 AttributeName = "CreateDate", AttributeType = "S"
             }},
            {new AttributeDefinition {
                 AttributeName = "DueDate", AttributeType = "S"
             }}
        };

            // Key schema for table
            var tableKeySchema = new List<KeySchemaElement>() {
            {
                new KeySchemaElement {
                    AttributeName= "IssueId",
                    KeyType = "HASH" //Partition key
                }
            },
            {
                new KeySchemaElement {
                    AttributeName = "Title",
                    KeyType = "RANGE" //Sort key
                }
            }
        };

            // Initial provisioned throughput settings for the indexes
            var ptIndex = new ProvisionedThroughput
            {
                ReadCapacityUnits = 1L,
                WriteCapacityUnits = 1L
            };

            // CreateDateIndex
            var createDateIndex = new GlobalSecondaryIndex()
            {
                IndexName = "CreateDateIndex",
                ProvisionedThroughput = ptIndex,
                KeySchema = {
                new KeySchemaElement {
                    AttributeName = "CreateDate", KeyType = "HASH" //Partition key
                },
                new KeySchemaElement {
                    AttributeName = "IssueId", KeyType = "RANGE" //Sort key
                }
```

```
            },
                Projection = new Projection
                {
                    ProjectionType = "INCLUDE",
                    NonKeyAttributes = {
                    "Description", "Status"
                }
                }
            };

            // TitleIndex
            var titleIndex = new GlobalSecondaryIndex()
            {
                IndexName = "TitleIndex",
                ProvisionedThroughput = ptIndex,
                KeySchema = {
                new KeySchemaElement {
                    AttributeName = "Title", KeyType = "HASH" //Partition key
                },
                new KeySchemaElement {
                    AttributeName = "IssueId", KeyType = "RANGE" //Sort key
                }
            },
                Projection = new Projection
                {
                    ProjectionType = "KEYS_ONLY"
                }
            };

            // DueDateIndex
            var dueDateIndex = new GlobalSecondaryIndex()
            {
                IndexName = "DueDateIndex",
                ProvisionedThroughput = ptIndex,
                KeySchema = {
                new KeySchemaElement {
                    AttributeName = "DueDate",
                    KeyType = "HASH" //Partition key
                }
            },
                Projection = new Projection
                {
                    ProjectionType = "ALL"
                }
```

```
            };



            var createTableRequest = new CreateTableRequest
            {
                TableName = tableName,
                ProvisionedThroughput = new ProvisionedThroughput
                {
                    ReadCapacityUnits = (long)1,
                    WriteCapacityUnits = (long)1
                },
                AttributeDefinitions = attributeDefinitions,
                KeySchema = tableKeySchema,
                GlobalSecondaryIndexes = {
                createDateIndex, titleIndex, dueDateIndex
            }
            };

            Console.WriteLine("Creating table " + tableName + "...");
            client.CreateTable(createTableRequest);

            WaitUntilTableReady(tableName);
        }

        private static void LoadData()
        {
            Console.WriteLine("Loading data into table " + tableName + "...");

            // IssueId, Title,
            // Description,
            // CreateDate, LastUpdateDate, DueDate,
            // Priority, Status

            putItem("A-101", "Compilation error",
                "Can't compile Project X - bad version number. What does this mean?",
                "2013-11-01", "2013-11-02", "2013-11-10",
                1, "Assigned");

            putItem("A-102", "Can't read data file",
                "The main data file is missing, or the permissions are incorrect",
                "2013-11-01", "2013-11-04", "2013-11-30",
                2, "In progress");
```

```
            putItem("A-103", "Test failure",
                "Functional test of Project X produces errors",
                "2013-11-01", "2013-11-02", "2013-11-10",
                1, "In progress");

            putItem("A-104", "Compilation error",
                "Variable 'messageCount' was not initialized.",
                "2013-11-15", "2013-11-16", "2013-11-30",
                3, "Assigned");

            putItem("A-105", "Network issue",
                "Can't ping IP address 127.0.0.1. Please fix this.",
                "2013-11-15", "2013-11-16", "2013-11-19",
                5, "Assigned");
        }

        private static void putItem(
            String issueId, String title,
            String description,
            String createDate, String lastUpdateDate, String dueDate,
            Int32 priority, String status)
        {
            Dictionary<String, AttributeValue> item = new Dictionary<string,
AttributeValue>();

            item.Add("IssueId", new AttributeValue
            {
                S = issueId
            });
            item.Add("Title", new AttributeValue
            {
                S = title
            });
            item.Add("Description", new AttributeValue
            {
                S = description
            });
            item.Add("CreateDate", new AttributeValue
            {
                S = createDate
            });
            item.Add("LastUpdateDate", new AttributeValue
            {
                S = lastUpdateDate
```

```
            });
            item.Add("DueDate", new AttributeValue
            {
                S = dueDate
            });
            item.Add("Priority", new AttributeValue
            {
                N = priority.ToString()
            });
            item.Add("Status", new AttributeValue
            {
                S = status
            });

            try
            {
                client.PutItem(new PutItemRequest
                {
                    TableName = tableName,
                    Item = item
                });
            }
            catch (Exception e)
            {
                Console.WriteLine(e.ToString());
            }
        }

        private static void QueryIndex(string indexName)
        {
            Console.WriteLine
                ("\n*****************************************************\n");
            Console.WriteLine("Querying index " + indexName + "...");

            QueryRequest queryRequest = new QueryRequest
            {
                TableName = tableName,
                IndexName = indexName,
                ScanIndexForward = true
            };


            String keyConditionExpression;
```

```
            Dictionary<string, AttributeValue> expressionAttributeValues = new
Dictionary<string, AttributeValue>();

            if (indexName == "CreateDateIndex")
            {
                Console.WriteLine("Issues filed on 2013-11-01\n");

                keyConditionExpression = "CreateDate = :v_date and
begins_with(IssueId, :v_issue)";
                expressionAttributeValues.Add(":v_date", new AttributeValue
                {
                    S = "2013-11-01"
                });
                expressionAttributeValues.Add(":v_issue", new AttributeValue
                {
                    S = "A-"
                });
            }
            else if (indexName == "TitleIndex")
            {
                Console.WriteLine("Compilation errors\n");

                keyConditionExpression = "Title = :v_title and
begins_with(IssueId, :v_issue)";
                expressionAttributeValues.Add(":v_title", new AttributeValue
                {
                    S = "Compilation error"
                });
                expressionAttributeValues.Add(":v_issue", new AttributeValue
                {
                    S = "A-"
                });

                // Select
                queryRequest.Select = "ALL_PROJECTED_ATTRIBUTES";
            }
            else if (indexName == "DueDateIndex")
            {
                Console.WriteLine("Items that are due on 2013-11-30\n");

                keyConditionExpression = "DueDate = :v_date";
                expressionAttributeValues.Add(":v_date", new AttributeValue
                {
                    S = "2013-11-30"
```

```
            });

            // Select
            queryRequest.Select = "ALL_PROJECTED_ATTRIBUTES";
        }
        else
        {
            Console.WriteLine("\nNo valid index name provided");
            return;
        }

        queryRequest.KeyConditionExpression = keyConditionExpression;
        queryRequest.ExpressionAttributeValues = expressionAttributeValues;

        var result = client.Query(queryRequest);
        var items = result.Items;
        foreach (var currentItem in items)
        {
            foreach (string attr in currentItem.Keys)
            {
                if (attr == "Priority")
                {
                    Console.WriteLine(attr + "---> " + currentItem[attr].N);
                }
                else
                {
                    Console.WriteLine(attr + "---> " + currentItem[attr].S);
                }
            }
            Console.WriteLine();
        }
    }

    private static void DeleteTable(string tableName)
    {
        Console.WriteLine("Deleting table " + tableName + "...");
        client.DeleteTable(new DeleteTableRequest
        {
            TableName = tableName
        });
        WaitForTableToBeDeleted(tableName);
    }

    private static void WaitUntilTableReady(string tableName)
```

```
        {
            string status = null;
            // Let us wait until table is created. Call DescribeTable.
            do
            {
                System.Threading.Thread.Sleep(5000); // Wait 5 seconds.
                try
                {
                    var res = client.DescribeTable(new DescribeTableRequest
                    {
                        TableName = tableName
                    });

                    Console.WriteLine("Table name: {0}, status: {1}",
                              res.Table.TableName,
                              res.Table.TableStatus);
                    status = res.Table.TableStatus;
                }
                catch (ResourceNotFoundException)
                {
                    // DescribeTable is eventually consistent. So you might
                    // get resource not found. So we handle the potential exception.
                }
            } while (status != "ACTIVE");
        }

        private static void WaitForTableToBeDeleted(string tableName)
        {
            bool tablePresent = true;

            while (tablePresent)
            {
                System.Threading.Thread.Sleep(5000); // Wait 5 seconds.
                try
                {
                    var res = client.DescribeTable(new DescribeTableRequest
                    {
                        TableName = tableName
                    });

                    Console.WriteLine("Table name: {0}, status: {1}",
                              res.Table.TableName,
                              res.Table.TableStatus);
                }
```

```
                catch (ResourceNotFoundException)
                {
                    tablePresent = false;
                }
            }
        }
    }
}
```

Arbeiten mit globalen sekundären Indizes in DynamoDB mithilfe der AWS CLI

Sie können die AWS CLI zum Erstellen einer Amazon-DynamoDB-Tabelle mit einem oder mehreren globalen sekundären Indizes, zum Beschreiben der Indizes in der Tabelle und zur Ausführung von Abfragen mithilfe des Indizes, verwenden.

Themen

- Erstellen einer Tabelle mit einem globalen sekundären Index
- Hinzufügen eines globalen sekundären Indexes zu einer vorhandenen Tabelle
- Beschreiben einer Tabelle mit einem globalen sekundären Index
- Abfragen eines globalen sekundären Indexes

Erstellen einer Tabelle mit einem globalen sekundären Index

Globale sekundäre Indizes können gleichzeitig mit der Tabelle erstellt werden. Zu diesem Zweck verwenden Sie den `create-table`-Parameter und geben Ihre Spezifikationen für ein oder mehrere globale sekundäre Indizes an. Im folgenden Beispiel wird eine Tabelle mit dem Namen `GameScores` mit einem globalen sekundären Index `GameTitleIndex` erstellt. Die Basistabelle hat einen Partitionsschlüssel von `UserId` und einen Sortierschlüssel von `GameTitle`, mit dem Sie effizient die beste Punktzahl eines einzelnen Benutzers für ein bestimmtes Spiel finden können, während die GSI einen Partitionsschlüssel von `GameTitle` und einen Sortierschlüssel von `TopScore` hat, mit dem Sie finden Sie schnell die höchste Gesamtpunktzahl für ein bestimmtes Spiel.

```
aws dynamodb create-table \
    --table-name GameScores \
    --attribute-definitions AttributeName=UserId,AttributeType=S \
                            AttributeName=GameTitle,AttributeType=S \
                            AttributeName=TopScore,AttributeType=N  \
    --key-schema AttributeName=UserId,KeyType=HASH \
```

```
                AttributeName=GameTitle,KeyType=RANGE \
    --provisioned-throughput ReadCapacityUnits=10,WriteCapacityUnits=5 \
    --global-secondary-indexes \
        "[
            {
                \"IndexName\": \"GameTitleIndex\",
                \"KeySchema\": [{\"AttributeName\":\"GameTitle\",\"KeyType\":\"HASH\"},
                               {\"AttributeName\":\"TopScore\",\"KeyType\":\"RANGE
\"}],
                \"Projection\":{
                    \"ProjectionType\":\"INCLUDE\",
                    \"NonKeyAttributes\":[\"UserId\"]
                },
                \"ProvisionedThroughput\": {
                    \"ReadCapacityUnits\": 10,
                    \"WriteCapacityUnits\": 5
                }
            }
        ]"
```

Sie müssen warten bis DynamoDB die Tabelle erstellt und den Tabellenstatus auf `ACTIVE` setzt. Im Anschluss können Sie die Daten in der Tabelle ablegen. Mit describe-table können Sie den Status der Tabellenerstellung ermitteln.

Hinzufügen eines globalen sekundären Indexes zu einer vorhandenen Tabelle

Globale sekundäre Indizes können auch nach der Tabellenerstellung hinzugefügt oder geändert werden. Zu diesem Zweck verwenden Sie den `update-table`-Parameter und geben Ihre Spezifikationen für ein oder mehrere globale sekundäre Indizes an. Im folgenden Beispiel wird dasselbe Schema wie im vorherigen Beispiel verwendet, es wird jedoch davon ausgegangen, dass die Tabelle bereits erstellt wurde und die GSI später hinzugefügt wird.

```
aws dynamodb update-table \
    --table-name GameScores \
    --attribute-definitions AttributeName=TopScore,AttributeType=N  \
    --global-secondary-index-updates \
        "[
            {
                \"Create\": {
                    \"IndexName\": \"GameTitleIndex\",
                    \"KeySchema\": [{\"AttributeName\":\"GameTitle\",\"KeyType\":\"HASH
\"},
```

```
                                    {\"AttributeName\":\"TopScore\",\"KeyType\":\"RANGE
\"}],
                    \"Projection\":{
                        \"ProjectionType\":\"INCLUDE\",
                        \"NonKeyAttributes\":[\"UserId\"]
                    }
                }
            }
        ]"
```

Beschreiben einer Tabelle mit einem globalen sekundären Index

Um Informationen zu globalen sekundären Indizes in einer Tabelle zu erhalten, verwenden Sie den Parameter `describe-table`. Sie können auf den Namen, das Schlüsselschema und die projizierten Attribute von jedem Index zugreifen.

```
aws dynamodb describe-table --table-name GameScores
```

Abfragen eines globalen sekundären Indexes

Sie können die Operation `query` für einen globalen sekundären Index genauso nutzen, wie Sie `query` für eine Tabelle nutzen. Sie müssen den Indexnamen, die Abfragekriterien für den Indexsortierschlüssel und die Attribute angeben, die Sie zurückgeben möchten. In diesem Beispiel ist der Index `GameTitleIndex` und der Indexsortierschlüssel `GameTitle`.

Die einzigen zurückgegebenen Attribute sind die, die in den Index projiziert wurden. Sie könnten diese Abfrage ändern, um auch Nicht-Schlüsselattribute auszuwählen, aber dies würde eine Tabellenabrufaktivität erfordern, die relativ teuer ist. Weitere Informationen zum Abrufen von Tabellen finden Sie unter Attributprojektionen.

```
aws dynamodb query --table-name GameScores\
    --index-name GameTitleIndex \
    --key-condition-expression "GameTitle = :v_game" \
    --expression-attribute-values '{":v_game":{"S":"Alien Adventure"} }'
```

Lokale sekundäre Indizes

Einige Anwendungen müssen Daten nur mithilfe der Basistabelle des Primärschlüssels abfragen. Es kann jedoch Situationen geben, in denen ein alternativer Sortierschlüssel hilfreich wäre. Um Ihrer Anwendung verschiedene Sortierschlüssel zur Auswahl anzubieten, können Sie einen oder mehrere

lokale sekundäre Indizes in einer Amazon-DynamoDB-Tabelle erstellen und `Query`- oder `Scan`-Anforderungen für diese Indizes generieren.

Themen

- Schritt 6: Verwenden eines lokalenx sekundären Indexes
- Attributprojektionen
- Erstellen eines lokalen sekundären Index
- Lesen von Daten aus einem lokalen sekundären Index
- Schreibvorgänge und lokale sekundäre Indizes
- Überlegungen im Hinblick auf die bereitgestellte Durchsatzkapazität für lokale sekundäre Indizes
- Speicherüberlegungen für lokale sekundäre Indizes
- Elementauflistungen in lokalen sekundären Indizes
- Arbeiten mit lokalen sekundären Indizes: Java
- Arbeiten mit lokalen sekundären Indizes: .NET
- Arbeiten mit lokalen sekundären Indizes in der DynamoDB-AWS CLI

Schritt 6: Verwenden eines lokalenx sekundären Indexes

Betrachten Sie als Beispiel die `Thread`-Tabelle. Diese Tabelle ist nützlich für eine Anwendung wie AWS -Diskussionsforen. Das folgende Diagramm zeigt, wie die Elemente in der Tabelle organisiert wären. (Es werden nicht alle Attribute angezeigt.)

Thread

ForumName	Subject	LastPostDateTime	Replies	
"S3"	"aaa"	"2015-03-15:17:24:31"	12	...
"S3"	"bbb"	"2015-01-22:23:18:01"	3	...
"S3"	"ccc"	"2015-02-31:13:14:21"	4	...
"S3"	"ddd"	"2015-01-03:09:21:11"	9	...
"EC2"	"yyy"	"2015-02-12:11:07:56"	18	...
"EC2"	"zzz"	"2015-01-18:07:33:42"	0	...
"RDS"	"rrr"	"2015-01-19:01:13:24"	3	...
"RDS"	"sss"	"2015-03-11:06:53:00"	11	...
"RDS"	"ttt"	"2015-10-22:12:19:44"	5	...
...	...	...	...	

DynamoDB speichert alle Elemente mit demselben Partitionsschlüsselwert fortlaufend. In diesem Beispiel könnte mit einem bestimmten `ForumName` eine `Query`-Operation sofort alle Threads für dieses Forum ermitteln. Innerhalb einer Gruppe von Elementen mit demselben Partitionsschlüsselwert werden die Elemente nach Sortierschlüsselwert sortiert. Wenn der Sortierschlüssel (`Subject`) in der Abfrage auch angegeben ist, kann DynamoDB die Ergebnisse, die zurückgegeben werden, einschränken und z. B. alle Threads im Forum „S3“ mit einem `Subject`, der mit „A“ beginnt, zurückgeben.

Einige Anforderungen erfordern komplexere Datenzugriffsmuster. Zum Beispiel:

- Welche Forum-Threads erhalten die meisten Ansichten und Antworten?
- Welcher Thread in einem bestimmten Forum enthält die meisten Nachrichten?
- Wie viele Threads wurden in einem bestimmten Forum in einem angegebenen Zeitraum gepostet?

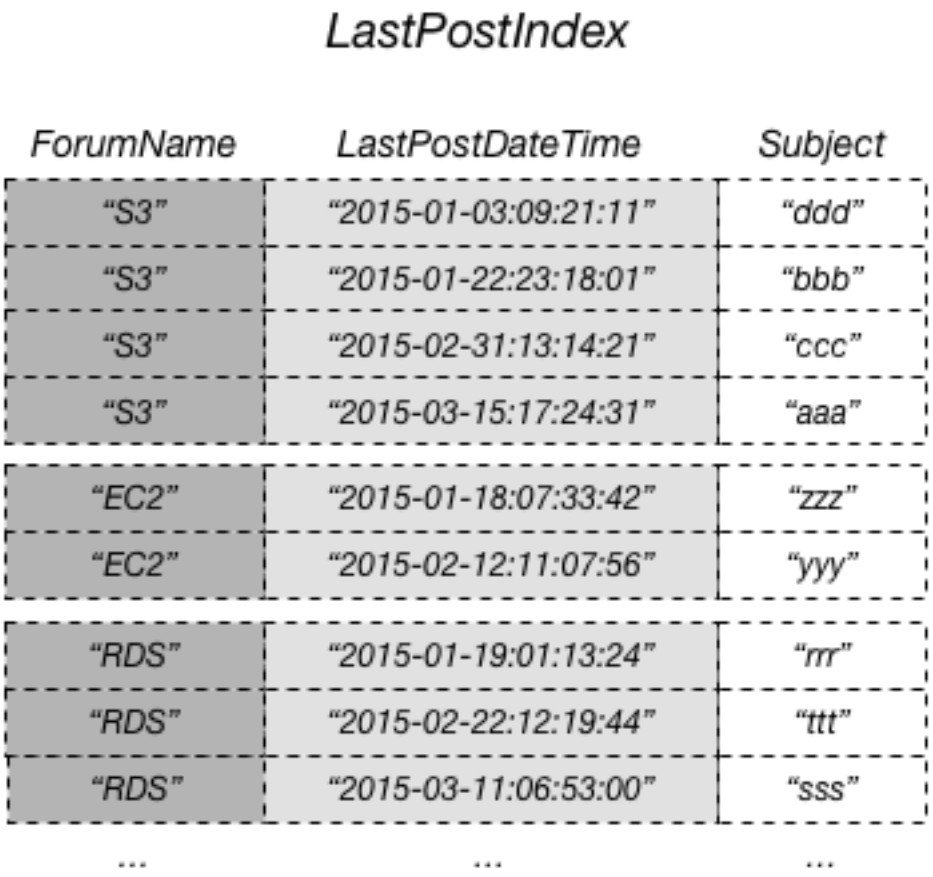
Um diese Fragen zu beantworten, würde die `Query`-Aktion nicht ausreichen. Stattdessen müssen Sie die gesamte Tabelle `Scan`. Bei einer Tabelle mit Millionen von Elementen würde dabei der bereitgestellte Lesedurchsatz größtenteils aufgebraucht und der Vorgang würde sehr lange dauern.

Sie können jedoch einen oder mehrere lokale sekundäre Indizes für Nicht-Schlüsselattribute angeben, z. B. `Replies` oder `LastPostDateTime`.

Ein lokaler sekundärer Index verwaltet einen alternativen Sortierschlüssel für einen bestimmten Partitionsschlüsselwert. Ein lokaler sekundärer Index enthält auch eine Kopie einiger oder aller Attribute aus seiner Basistabelle. Sie geben beim Erstellen der Tabelle an, welche Attribute in den lokalen sekundären Index projiziert werden. Die Daten in einem lokalen sekundären Index werden durch denselben Partitionsschlüssel strukturiert wie die Basistabelle, jedoch mit einem anderen Sortierschlüssel. Auf diese Weise können Sie in dieser anderen Dimension effizient auf Datenelemente zugreifen. Wenn Sie eine größere Abfrage- oder Scanflexibilität benötigen, können Sie bis zu fünf lokale sekundäre Indizes pro Tabelle erstellen.

Angenommen, eine Anwendung muss alle Threads ermitteln, die in den letzten drei Monaten in einem bestimmten Forum veröffentlicht wurden. Ohne lokalen sekundären Index müsste die Anwendung eine `Scan`-Aktion für die gesamte `Thread`-Tabelle ausführen und alle Beiträge, die nicht innerhalb des Zeitraums gepostet wurden, verwerfen. Mit einem lokalen sekundären Index könnte eine `Query`-Operation `LastPostDateTime` verwenden, um Daten schnell zu finden.

Das folgende Diagramm zeigt einen lokalen sekundären Index mit dem Namen `LastPostIndex`. Beachten Sie, dass der Partitionsschlüssel mit dem Schlüssel der `Thread`-Tabelle übereinstimmt, der Sortierschlüssel jedoch `LastPostDateTime` ist.



Jeder lokale sekundäre Index muss die folgenden Bedingungen erfüllen:

- Der Partitionsschlüssel ist mit dem Schlüssel der Basistabelle identisch.
- Der Sortierschlüssel besteht aus genau einem skalaren Attribut.
- Der Sortierschlüssel der Basistabelle wird in den Index projiziert, wo er als Nicht-Schlüsselattribut fungiert.

In diesem Beispiel lautet der Partitionsschlüssel `ForumName` und der Sortierschlüssel des lokalen sekundären Indexes lautet `LastPostDateTime`. Darüber hinaus wird der Sortierschlüsselwert der Basistabelle (in diesem Beispiel `Subject`) in den Index projiziert, ist jedoch kein Bestandteil des Indexschlüssels. Wenn eine Anwendung eine Liste basierend auf `ForumName` und `LastPostDateTime` benötigt, kann sie eine `Query`-Anforderung für `LastPostIndex` erstellen. Die Abfrageergebnisse sind nach `LastPostDateTime` sortiert und können in auf- oder absteigender

Reihenfolge zurückgegeben werden. Die Abfrage kann auch Schlüsselbedingungen anwenden, z. B. nur Elemente zurückgeben mit einem `LastPostDateTime` innerhalb einer bestimmten Zeitspanne.

Jeder lokale sekundäre Index enthält automatisch die Partitions- und Sortierschlüssel der Basistabelle. Nicht-Schlüsselattribute können Sie optional in den Index projizieren. Wenn Sie den Index abfragen, kann DynamoDB diese projizierten Attribute effizient abrufen. Beim Abfragen eines lokalen sekundären Indizes können auch Attribute abgerufen werden, die nicht in den Index projiziert sind. DynamoDB ruft diese Attribute automatisch aus der Basistabelle ab, jedoch mit größerer Latenz und höheren Kosten für den bereitgestellten Durchsatz.

Für alle lokale sekundäre Indizes können Sie bis zu 10 GB Daten pro eindeutigem Partitionsschlüsselwert speichern. Diese Abbildung enthält alle Elemente in der Basistabelle sowie alle Elemente in den Indizes, die denselben Partitionsschlüsselwert haben. Weitere Informationen finden Sie unter Elementauflistungen in lokalen sekundären Indizes.

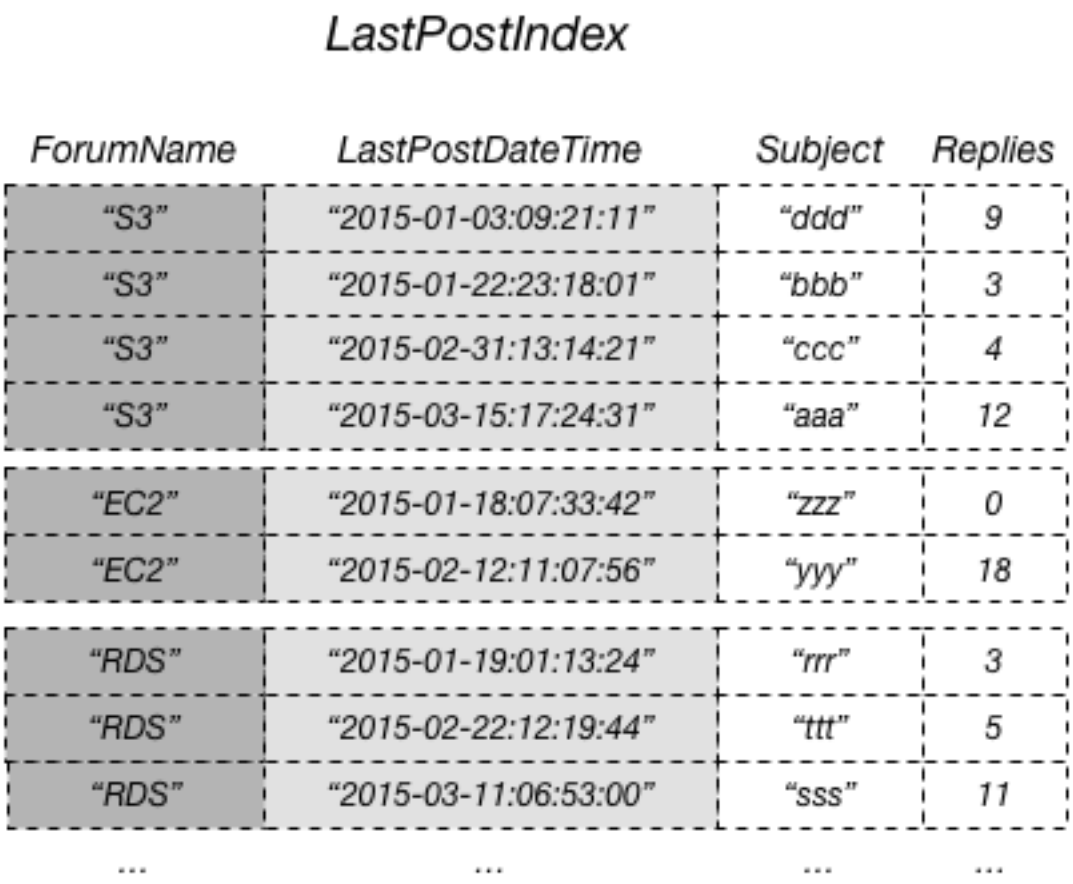
Attributprojektionen

Mit `LastPostIndex` könnte eine Anwendung `ForumName` und `LastPostDateTime` als Abfragekriterien verwenden. Um jedoch zusätzliche Attribute abzurufen, muss DynamoDB zusätzliche Lesevorgänge für die `Thread`-Tabelle ausführen. Diese zusätzlichen Lesevorgänge werden als Abrufe bezeichnet. Diese können die Gesamtgröße des bereitgestellten Durchsatzes, der für eine Abfrage erforderlich ist, erhöhen.

Angenommen, Sie möchten eine Webseite mit einer Liste aller Threads in „S3“ und der Anzahl der Antworten für jeden Thread füllen, sortiert nach der letzten Antwort, date/time beginnend mit der neuesten Antwort. Zum Auffüllen dieser Liste benötigen Sie die folgenden Attribute:

- `Subject`
- `Replies`
- `LastPostDateTime`

Die beste Möglichkeit, diese Daten abzufragen und Abrufoperationen zu vermeiden, ist, das Attribut `Replies` aus der Tabelle in den lokalen sekundären Index zu projizieren, wie in diesem Diagramm dargestellt:



Eine Projektion ist der Satz von Attributen, die aus einer Tabelle in einen sekundären Index kopiert werden. Der Partitionsschlüssel und der Sortierschlüssel der Tabelle werden immer in den Index projiziert. Sie können andere Attribute projizieren, um die Abfrageanforderungen Ihrer Anwendung zu unterstützen. Wenn Sie einen Index abfragen, kann Amazon DynamoDB auf jedes Attribut in der Projektion zugreifen, als ob sich diese Attribute in einer eigenen Tabelle befinden.

Wenn Sie einen sekundären Index erstellen, müssen Sie die Attribute angeben, die in den Index projiziert werden. DynamoDB bietet hierfür drei verschiedene Optionen:

- KEYS_ONLY – Jeder Eintrag im Index besteht nur aus dem Tabellenpartitionsschlüssel und Sortierschlüsselwerten, sowie den Indexschlüsselwerten. Die Option `KEYS_ONLY` führt zu dem kleinstmöglichen sekundären Index.
- INCLUDE – Zusätzlich zu den in `KEYS_ONLY` beschriebenen Attributen, enthält der sekundäre Index andere Nicht-Schlüsselattribute, die Sie angeben.
- ALL – Der sekundäre Index enthält alle Attribute der Quelltabelle. Da alle Tabellendaten im Index dupliziert werden, wird führt eine `ALL`-Projektion zu dem größtmöglichen sekundären Index.

Im vorherigen Diagramm wird das Nicht-Schlüsselattribut in `Replies` in `LastPostIndex` projiziert. Eine Anwendung kann `LastPostIndex` anstelle der vollständigen `Thread`-Tabelle abfragen, um eine Webseite mi t`Subject`, `Replies` und `LastPostDateTime` auszufüllen. Wenn alle anderen Nicht-Schlüsselattribute abgefragt werden, muss DynamoDB diese Attribute aus der `Thread`-Tabelle abrufen.

Von der Anwendungsseite aus betrachtet, ist das Abrufen zusätzlicher Attribute aus der Basistabelle automatisch und transparent. Es muss keine Anwendungslogik umgeschrieben werden. Beachten Sie jedoch, dass dieses Abrufen den Leistungsvorteil, den ein lokaler sekundärer Index bietet, erheblich einschränken kann.

Wenn Sie die Attribute zum Projizieren in einen lokalen sekundären Index auswählen, müssen Sie die Differenz der Kosten des bereitgestellten Durchsatzes und der Speicherkosten berücksichtigen:

- Wenn Sie nur auf wenige Attribute mit möglichst niedriger Latenz zugreifen müssen, erwägen Sie, nur diese Attribute in einen lokalen sekundären Index zu projizieren. Je kleiner der Index, desto geringer die Speicher- und somit die Schreibkosten. Wenn Sie Attribute gelegentlich abrufen müssen, können die Kosten für den bereitgestellten Durchsatz die längerfristigen Kosten für die Speicherung dieser Attribute aufwiegen.
- Greift Ihre Anwendung häufig auf einige Nicht-Schlüsselattribute zu, sollten Sie erwägen, diese Attribute in einen lokalen sekundären Index zu projizieren. Die zusätzlichen Speicherkosten für den lokalen sekundären Index wiegen die Kosten für die Durchführung häufiger Tabellen-Scans auf.
- Wenn Sie auf die meisten Nicht-Schlüsselattribute in regelmäßigen Abständen zugreifen müssen, können Sie diese Attribute – oder sogar die gesamte Basistabelle – in einen lokalen sekundären Index projizieren. Auf diese Weise erhalten Sie maximale Flexibilität und den niedrigsten Verbrauch des bereitgestellten Durchsatzes, da kein Abrufen erforderlich ist. Die Speicherkosten erhöhen oder verdoppeln sich allerdings sogar, wenn Sie alle Attribute projizieren.
- Wenn Ihre Anwendung eine Tabelle selten abfragen, jedoch viele Schreibvorgänge oder Updates für die Daten in der Tabelle durchführen muss, erwägen Sie das Projizieren von KEYS_ONLY. Der wäre nur klein, stünde aber weiterhin bei Bedarf für Abfrageaktivitäten zur Verfügung.

Erstellen eines lokalen sekundären Index

Um einen oder mehrere lokale sekundäre Indizes für eine Tabelle zu erstellen, verwenden Sie den `LocalSecondaryIndexes`-Parameter der Operation `CreateTable`. Lokale sekundäre Indizes für eine Tabelle werden erstellt, wenn die Tabelle erstellt wird. Wenn Sie eine Tabelle löschen, werden alle lokalen sekundären Indizes in dieser Tabelle ebenfalls gelöscht.

Sie müssen ein Nicht-Schlüsselattribut angeben, das als Sortierschlüssel des lokalen sekundären Indizes dient. Das Attribut, das Sie auswählen, muss ein skalarer `String`, `Number` oder `Binary` sein. Andere Skalar-, Dokument- und Satztypen sind nicht zulässig. Eine vollständige Liste der Datentypen finden Sie unter Datentypen.

> **Important**
>
> Für Tabellen mit lokalen sekundären Indizes besteht eine Größenbeschränkung von 10 GB pro Partitionsschlüsselwert. Eine Tabelle mit lokalem sekundärem Index kann eine beliebige Anzahl von Elementen speichern, solange die Gesamtgröße eines Partitionsschlüsselwerts 10 GB nicht überschreitet. Weitere Informationen finden Sie unter Größenlimit der Elementauflistung.

Sie können Attribute jeden Datentyps in einen lokalen sekundären Index projizieren. Dazu zählen skalare Werte, Dokumente und Sätze. Eine vollständige Liste der Datentypen finden Sie unter Datentypen.

Lesen von Daten aus einem lokalen sekundären Index

Sie können Elemente aus einem lokalen sekundären Index mit dem Befehl `Query` und `Scan` verwenden. Die `GetItem`- und `BatchGetItem`-Operationen können für einen lokalen sekundären Index nicht verwendet werden.

Abfragen eines lokalen sekundären Indexes

In einer DynamoDB-Tabelle müssen der kombinierte Partitionsschlüsselwert und der Sortierschlüsselwert für jedes Element eindeutig sein. In einem lokalen sekundären Index müssen die Sortierschlüsselwerte für einen bestimmten Partitionsschlüsselwert jedoch nicht eindeutig sein. Wenn mehrere Elemente mit demselben Sortierschlüsselwert im lokalen sekundären Index vorhanden sind, gibt eine `Query`-Operation alle Elemente mit demselben Partitionsschlüsselwert zurück. In der Antwort werden die übereinstimmenden Elemente nicht in einer bestimmten Reihenfolge zurückgegeben.

Sie können einen lokalen sekundären Index mit Eventually-Consistent- oder Strongly-Consistent-Lesevorgängen abfragen. Um die Art der Konsistenz anzugeben, verwenden Sie den Parameter `ConsistentRead` der `Query`-Operation. Bei einem Strongly-Consistent-Lesevorgang von einem lokalen sekundären Index werden immer die zuletzt aktualisierten Werte zurückgegeben. Wenn die

Abfrage weitere Attribute aus der Basistabelle abrufen muss, sind diese Attribute in Bezug auf den Index konsistent.

Example

Betrachten Sie die folgenden Daten, die von einer `Query`-Operation zum Abfragen von Daten aus den Diskussionsbeiträgen in einem bestimmten Forum zurückgegeben werden:

```
{
    "TableName": "Thread",
    "IndexName": "LastPostIndex",
    "ConsistentRead": false,
    "ProjectionExpression": "Subject, LastPostDateTime, Replies, Tags",
    "KeyConditionExpression":
        "ForumName = :v_forum and LastPostDateTime between :v_start and :v_end",
    "ExpressionAttributeValues": {
        ":v_start": {"S": "2015-08-31T00:00:00.000Z"},
        ":v_end": {"S": "2015-11-31T00:00:00.000Z"},
        ":v_forum": {"S": "EC2"}
    }
}
```

Vorgänge in dieser Abfrage:

- DynamoDB greift auf `LastPostIndex` mit dem Partitionsschlüssel `ForumName` zu, um die Indexelemente für „EC2“ zu ermitteln. Alle Indexelemente mit diesem Schlüssel werden nebeneinander gespeichert, um ein schnelles Abrufen zu ermöglichen.
- In diesem Forum verwendet DynamoDB den Index, um die Schlüssel, die der angegebenen `LastPostDateTime`-Bedingung entsprechen, zu suchen.
- Da das Attribut `Replies` in den Index projiziert wird, kann DynamoDB dieses Attribut abrufen, ohne zusätzlichen bereitgestellten Durchsatz zu verbrauchen.
- Das Attribut `Tags` wird nicht in den Index projiziert, sodass DynamoDB auf die `Thread`-Tabelle zugreifen muss, um dieses Attribut abzurufen.
- Die Ergebnisse werden sortiert nach `LastPostDateTime`. Die Indexeinträge sind nach Partitionsschlüsselwert und dann nach Sortierschlüsselwert sortiert und `Query` gibt sie in der Reihenfolge, in der sie gespeichert werden, zurück. (Sie können den Parameter `ScanIndexForward` verwenden, um die Ergebnisse in absteigender Reihenfolge anzuzeigen.)

Da das Attribut `Tags` nicht in den lokalen sekundären Index projiziert wird, muss DynamoDB zusätzliche Lesekapazitätseinheiten verbrauchen, um dieses Attribut aus der Basistabelle abzurufen. Wenn Sie diese Abfrage häufig ausführen müssen, sollten Sie `Tags` in `LastPostIndex` verwenden, um das Abrufen von der Basistabelle zu vermeiden. Wenn Sie jedoch nur gelegentlich auf `Tags` zugreifen müssen, lohnen sich die zusätzlichen Speicherkosten für das Projizieren von `Tags` in den Index möglicherweise nicht.

Scannen eines lokalen sekundären Indexes

Sie können `Scan` verwenden, um alle Daten aus einem lokalen sekundären Index abzurufen. Geben Sie dazu den Namen der Basistabelle sowie den Indexnamen in der Abfrage an. Mit einer `Scan`-Operation liest DynamoDB alle Daten im Index und gibt sie an die Anwendung zurück. Sie können auch anfordern, dass nur einige der Daten zurückgegeben und die verbleibenden Daten verworfen werden. Verwenden Sie dazu den Parameter `FilterExpression` der `Scan`-API. Weitere Informationen finden Sie unter Filterausdrücke für Scan.

Schreibvorgänge und lokale sekundäre Indizes

DynamoDB synchronisiert automatisch alle lokalen sekundären Indizes mit ihren jeweiligen Basistabellen. Anwendungen schreiben niemals direkt in einen Index. Allerdings ist es wichtig, zu wissen, welche Auswirkungen es hat, wie DynamoDB diese Indizes verwaltet.

Beim Erstellen eines lokalen sekundären Indizes geben Sie ein Attribut als Sortierschlüssel für den Index an. Außerdem bestimmen Sie einen Datentyp für dieses Attribut. Das bedeutet, dass wenn Sie ein Element in die Basistabelle schreiben und das Element ein Indexschlüsselattribut definiert, der entsprechende Typ dem Datentyp des Indexschlüsselschemas entsprechen muss. Im Fall von `LastPostIndex` wird der Sortierschlüssel `LastPostDateTime` im Index als Datentyp `String` definiert. Wenn Sie versuchen, der Tabelle `Thread` ein Element hinzuzufügen und einen anderen Datentyp für `LastPostDateTime` (wie z. B. `Number`) anzugeben, gibt DynamoDB eine `ValidationException` aufgrund der fehlenden Übereinstimmung des Datentyps zurück.

Es ist keine one-to-one Beziehung zwischen den Elementen in einer Basistabelle und den Elementen in einem lokalen sekundären Index erforderlich. In der Tat kann dieses Verhalten für viele Anwendungen vorteilhaft sein.

Eine Tabelle mit vielen lokalen sekundären Indizes verursacht höhere Kosten für Schreibaktivitäten als Tabellen mit weniger Indizes. Weitere Informationen finden Sie unter Überlegungen im Hinblick auf die bereitgestellte Durchsatzkapazität für lokale sekundäre Indizes.

> **Important**
>
> Für Tabellen mit lokalen sekundären Indizes besteht eine Größenbeschränkung von 10 GB pro Partitionsschlüsselwert. Eine Tabelle mit lokalem sekundärem Index kann eine beliebige Anzahl von Elementen speichern, solange die Gesamtgröße eines Partitionsschlüsselwerts 10 GB nicht überschreitet. Weitere Informationen finden Sie unter Größenlimit der Elementauflistung.

Überlegungen im Hinblick auf die bereitgestellte Durchsatzkapazität für lokale sekundäre Indizes

Wenn Sie eine Tabelle in DynamoDB erstellen, stellen Sie Lese- und Schreibkapazitätseinheiten für den erwarteten Workload der Tabelle bereit. Dieser Workload umfasst Lese- und Schreibaktivitäten in den lokalen sekundären Indizes der Tabelle.

Die aktuellen Preise für die bereitgestellte Durchsatzkapazität finden Sie unter Amazon-DynamoDB-Preise.

Lesekapazitätseinheiten

Wenn Sie einen lokalen sekundären Index abfragen, hängt die Anzahl der verbrauchten Lesekapazitätseinheiten davon ab, wie auf die Daten zugegriffen wird.

Ähnlich wie Tabellenabfragen kann eine Indexabfrage auch entweder Eventually-Consistent- oder Strongly-Consistent-Lesevorgänge abhängig vom Wert `ConsistentRead` verwenden. Ein Strongly-Consistent-Lesevorgang belegt eine Lesekapazitätseinheit, ein Eventually-Consistent-Lesevorgang verbraucht nur die Hälfte. Daher können Sie Ihre Kosten für Lesekapazitätseinheiten durch Auswählen von Eventually Consistent-Lesevorgängen reduzieren.

Für Indexabfragen, die nur Indexschlüssel und projizierte Attribute anfordern, berechnet DynamoDB die bereitgestellten Leseaktivitäten auf die gleiche Weise wie für Tabellenabfragen. Der einzige Unterschied besteht darin, dass die Berechnung auf der Größe der Indexeinträge und nicht auf der Größe des Elements in der Basistabelle beruht. Die Anzahl der Lesekapazitätseinheiten ist die Summe aller projizierten Attributgrößen sämtlicher zurückgegebener Elemente. Das Ergebnis wird dann auf den nächsten 4 KB-Grenzwert aufgerundet. Weitere Informationen darüber, wie DynamoDB die bereitgestellte Durchsatznutzung berechnet, finden Sie unter DynamoDB – Modus mit bereitgestellter Kapazität.

Bei Indexabfragen, die Attribute lesen, die nicht in den lokalen sekundären Index projiziert werden, muss DynamoDB diese Attribute zusätzlich zum Lesen der projizierten Attribute aus dem Index aus der Basistabelle abrufen. Diese Abrufe treten auf, wenn Sie nicht projizierte Attribute in die Parameter `Select` oder `ProjectionExpression` der `Query`-Operation einbeziehen. Das Abrufen verursacht zusätzliche Latenz bei Abfrageantworten und verursacht auch höhere Kosten für den bereitgestellten Durchsatz: Zusätzlich zu den zuvor beschriebenen Lesevorgängen aus dem lokalen sekundären Index werden Ihnen Lesekapazitätseinheiten für jedes abgerufene Basistabellenelement in Rechnung gestellt. Diese Kosten fallen nicht nur für die angeforderten Attribute an, sondern für jedes Element, das aus der Tabelle gelesen wird.

Die maximale Größe der von einer `Query`-Operation zurückgegebenen Ergebnisse beträgt 1 MB. Diese umfasst die Größen aller Attributnamen und Werte sämtlicher zurückgegebenen Elemente. Wenn jedoch eine Abfrage für einen lokalen sekundären Index veranlasst, dass DynamoDB Elementattribute aus der Basistabelle abruft, kann die maximale Größe der Daten in den Ergebnissen niedriger sein. In diesem Fall setzt sich die Ergebnisgröße aus folgender Summe zusammen:

- Die Größe der übereinstimmenden Elemente im Index wird auf die nächste 4 KB aufgerundet.
- Die Größe der einzelnen übereinstimmenden Elemente in der Basistabelle wird für jedes einzelne Element auf die nächste 4 KB aufgerundet.

Mit dieser Formel beträgt die maximale Größe der von einer Abfrageoperation zurückgegebenen Ergebnisse 1 MB.

Nehmen wir als Beispiel eine Tabelle, in der die Größe jedes Element 300 Byte beträgt. Es gibt einen lokalen sekundären Index in dieser Tabelle, aber nur 200 Byte pro Element werden in den Index projiziert. Angenommen, Sie führen eine `Query`-Operation für diesen Index aus, die Abfrage erfordert Tabellenabrufe für jedes Element und die Abfrage gibt vier Elemente zurück. DynamoDB fasst Folgendes zusammen:

- Die Größe der übereinstimmenden Elemente im Index: 200 Byte × 4 Elemente = 800 Byte. Dieser Wert wird dann auf 4 KB aufgerundet.
- Die Größe der einzelnen übereinstimmenden Elemente in der Basistabelle: (300 Byte, auf 4 KB aufgerundet) × 4 Elemente = 16 KB.

Die Gesamtgröße der Daten im Ergebnis beträgt somit 20 KB.

Schreibkapazitätseinheiten

Wenn ein Element in einer Tabelle hinzugefügt, aktualisiert oder gelöscht wird, verbraucht die Aktualisierung der lokalen sekundären Indizes bereitgestellte Schreibkapazitätseinheiten für die Tabelle. Die gesamten bereitgestellten Durchsatzkosten für einen Schreibvorgang sind die Summe der Schreibkapazitätseinheiten, die durch das Schreiben in die Tabelle verbraucht werden, und denjenigen, die durch die Aktualisierung der lokalen sekundären Indizes verbraucht werden.

Die Kosten für das Schreiben eines Elements in einen lokalen Sekundärindex hängen von mehreren Faktoren ab:

- Wenn Sie ein neues Element in die Tabelle schreiben, die ein indiziertes Attribut definiert, oder ein vorhandenes Element zum Definieren eines zuvor nicht definierten indizierten Attributs aktualisieren, ist ein Schreibvorgang erforderlich, um das Element in den Index einzufügen.
- Wenn eine Aktualisierung der Tabelle den Wert eines indizierten Schlüsselattributs (von A in B) ändert, sind zwei Schreibvorgänge erforderlich, und zwar einer zum Löschen des vorherigen Elements aus dem Index und einer zum Schreiben des neuen Elements in den Index.
- Wenn ein Element im Index vorhanden war, ein Schreibvorgang in der Tabelle jedoch dazu führte, dass das indizierte Attribut gelöscht wurde, ist ein Schreibvorgang erforderlich, um die alte Elementprojektion im Index zu löschen.
- Wenn ein Element nicht im Index vorhanden ist, bevor oder nachdem das Element aktualisiert wird, fallen keine zusätzlichen Kosten für das Schreiben in den Index an.

Alle diese Faktoren setzen voraus, dass die Größe der einzelnen Elemente im Index kleiner oder gleich der 1-KB- Elementgröße für das Berechnen der Schreibkapazitätseinheiten ist. Größere Indexeinträge erfordern zusätzliche Schreibkapazitätseinheiten. Sie können Ihre Kosten für Schreibvorgänge minimieren, indem Sie überlegen, welche Attribute Ihre Abfragen zurückgeben müssen, und nur diese Attribute in den Index projizieren.

Speicherüberlegungen für lokale sekundäre Indizes

Wenn eine Anwendung ein Element in eine Tabelle schreibt, kopiert DynamoDB automatisch die richtige Teilmenge der Attribute in den lokalen sekundären Index, in dem diese Attribute angezeigt werden sollen. Ihrem AWS Konto werden sowohl die Speicherung des Elements in der Basistabelle als auch die Speicherung von Attributen in allen lokalen Sekundärindizes dieser Tabelle in Rechnung gestellt.

Der Speicherplatz, der von einem Indexelement belegt wird, ergibt sich aus der Summe von folgenden Werten:

- Die Größe in Byte des Primärschlüssels der Basistabelle (Partitionsschlüssel und Sortierschlüssel)
- Die Größe in Byte des Indexschlüsselattributs
- Die Größe in Byte der projizierten Attribute (sofern vorhanden)
- 100 Bytes des Overheads pro Indexelement

Um den Speicherbedarf für einen lokalen sekundären Index zu schätzen, können Sie die durchschnittliche Größe eines Elements im Index schätzen und dann mit der Anzahl der Elemente im Index multiplizieren.

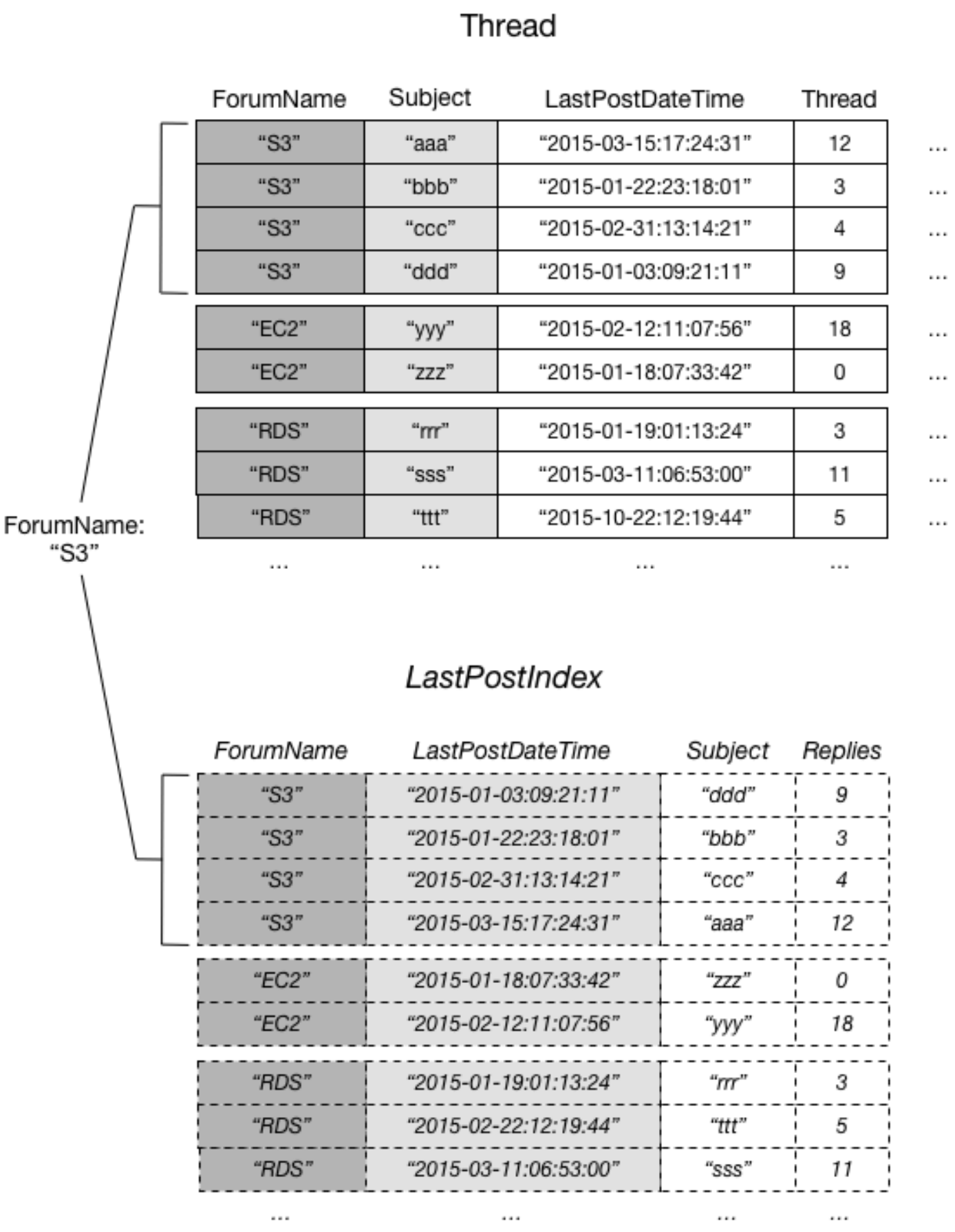
Wenn eine Tabelle ein Element enthält, in dem ein bestimmtes Attribut nicht definiert ist, dieses Attribut aber als Indexsortierschlüssel festgelegt ist, schreibt DynamoDB keine Daten für dieses Element in den Index.

Elementauflistungen in lokalen sekundären Indizes

> **Note**
>
> Dieser Abschnitt betrifft nur Tabellen mit lokalen sekundären Indizes.

In DynamoDB ist eine Elementsammlung eine beliebige Gruppe von Elementen, die denselben Partitionsschlüsselwert in einer Tabelle und allen ihren lokalen sekundären Indizes aufweisen. In den Beispielen in diesem Abschnitt lautet der Partitionsschlüssel für die `Thread`-Tabelle `ForumName` und der Partitionsschlüssel für `LastPostIndex` ist ebenfalls `ForumName`. Alle Tabellen- und Indexelemente mit demselben `ForumName` gehören zur selben Elementauflistung. In der `Thread`-Tabelle und dem `LastPostIndex`- ist beispielsweise eine Elementauflistung für das `EC2`-Forum und eine andere Elementauflistung für das `RDS`-Forum vorhanden.

Das folgende Diagramm zeigt die Elementauflistung für das `S3`-Forum:


Thread
ForumName
Subject
LastPostDateTime
Thread
"S3"
"aaa"
"2015-03-15:17:24:31"
12
"S3"
"bbb"
"2015-01-22:23:18:01"
3
"S3"
"ccc"
"2015-02-31:13:14:21"
4
"S3"
"ddd"
"2015-01-03:09:21:11"
9
"EC2"
"yyy"
"2015-02-12:11:07:56"
18
"EC2"
"zzz"
"2015-01-18:07:33:42"
0
"RDS"
"rrr"
"2015-01-19:01:13:24"
3
"RDS"
"sss"
"2015-03-11:06:53:00"
11
"RDS"
"ttt"
"2015-10-22:12:19:44"
5
ForumName:
"S3"
LastPostIndex
ForumName
LastPostDateTime
Subject
Replies
"S3"
"2015-01-03:09:21:11"
"ddd"
9
"S3"
"2015-01-22:23:18:01"
"bbb"
3
"S3"
"2015-02-31:13:14:21"
"ccc"
4
"S3"
"2015-03-15:17:24:31"
"aaa"
12
"EC2"
"2015-01-18:07:33:42"
"zzz"
0
"EC2"
"2015-02-12:11:07:56"
"yyy"
18
"RDS"
"2015-01-19:01:13:24"
"rrr"
3
"RDS"
"2015-02-22:12:19:44"
"ttt"
5
"RDS"
"2015-03-11:06:53:00"
"sss"
11

In diesem Diagramm enthält die Elementauflistung alle Elemente von `Thread` und `LastPostIndex`, wobei der Partitionsschlüsselwert `ForumName` „S3“ lautet. Wenn die Tabelle andere lokale sekundäre Indizes enthält, wären alle Elemente in diesen Indizes mit `ForumName` gleich „S3“ ebenfalls Teil der Elementsammlung.

Sie können eine der folgenden Operationen in DynamoDB verwenden, um Informationen zu Elementauflistungen zu erhalten:

- `BatchWriteItem`
- `DeleteItem`
- `PutItem`
- `UpdateItem`
- `TransactWriteItems`

Jede dieser Operationen unterstützt den Parameter `ReturnItemCollectionMetrics`. Wenn Sie diesen Parameter auf `SIZE` festlegen, können Sie Informationen über die Größe der einzelnen Elementauflistung im Index anzeigen.

Example

Nachfolgend finden Sie ein Beispiel aus der Ausgabe einer `UpdateItem`-Operation für die `Thread`-Tabelle, mit `ReturnItemCollectionMetrics` auf `SIZE`. Das Element, das aktualisiert wurde, hatte den `ForumName` „EC2“, sodass die Ausgabe Informationen über diese Elementauflistung enthält.

```
{
    ItemCollectionMetrics: {
        ItemCollectionKey: {
            ForumName: "EC2"
        },
        SizeEstimateRangeGB: [0.0, 1.0]
    }
}
```

Das `SizeEstimateRangeGB`-Objekt zeigt, dass die Größe dieser Elementauflistung zwischen 0 und 1 GB beträgt. DynamoDB aktualisiert diese Größenschätzung in regelmäßigen Abständen, sodass die Zahlen bei der nächsten Änderung des Elements anders lauten können.

Größenlimit der Elementauflistung

Die maximale Größe einer Elementauflistung für eine Tabelle mit einem oder mehreren lokalen sekundären Indizes beträgt 10 GB. Dies gilt nicht für Elementauflistungen in Tabellen ohne lokale sekundäre Indizes und auch nicht für Elementauflistungen in globalen sekundären Indizes. Es sind nur Tabellen mit einem oder mehreren lokalen sekundären Indizes betroffen.

Wenn eine Elementsammlung das Limit von 10 GB überschreitet, gibt DynamoDB ein `ItemCollectionSizeLimitExceededException` zurück, und Sie können der Elementsammlung keine weiteren Elemente hinzufügen oder die Größe der Elemente in der Elementsammlung erhöhen. (Lese- und Schreibvorgänge, die die Größe der Elementauflistung reduzieren, sind nach wie vor zulässig.) Sie können weiterhin Elemente zu anderen Elementauflistungen hinzufügen.

Um die Größe einer Elementauflistung zu reduzieren, können Sie einen der folgenden Schritte ausführen:

- Löschen Sie alle unnötigen Elemente mit dem betreffenden Partitionsschlüsselwert. Wenn Sie diese Elemente aus der Basistabelle löschen, entfernt DynamoDB auch alle Indexeinträge, die denselben Partitionsschlüsselwert aufweisen.
- Aktualisieren Sie die Elemente durch Entfernen von Attributen oder durch die Reduzierung der Größe der Attribute. Wenn diese Attribute in lokale sekundäre Indizes projiziert werden, reduziert DynamoDB auch die Größe der entsprechenden Indexeinträge.
- Erstellen Sie eine neue Tabelle mit demselben Partitions- und Sortierschlüssel und verschieben Sie die Elemente von der alten in die neue Tabelle. Dies ist ein guter Ansatz, wenn eine Tabelle über historische Daten verfügt, auf die selten zugegriffen wird. Sie können auch in Betracht ziehen, diese historischen Daten in Amazon Simple Storage Service (Amazon S3) zu archivieren.

Wenn die Gesamtgröße der Elementsammlung unter 10 GB sinkt, können Sie erneut Elemente mit demselben Partitionsschlüsselwert hinzufügen.

Wir empfehlen als bewährte Methode, Ihre Anwendung zu instrumentieren, um die Größe Ihrer Elementauflistungen zu überwachen. Eine Möglichkeit dazu ist, den Parameter `ReturnItemCollectionMetrics` auf `SIZE` festzulegen, sobald Sie `BatchWriteItem`, `DeleteItem`, `PutItem` oder `UpdateItem`. verwenden. Ihre Anwendung sollte das `ReturnItemCollectionMetrics`-Objekt in der Ausgabe untersuchen und eine Fehlermeldung protokollieren, wenn eine Elementauflistung einen benutzerdefinierten Grenzwert (z. B. 8 GB) überschreitet. Wenn Sie einen geringeren Wert als 10 GB als Limit festlegen, ist dies ein

Frühwarnsystem, da Sie so rechtzeitig erfahren, dass sich eine Elementauflistung dem Limit nähert, und entsprechende Maßnahmen ergreifen können.

Elementauflistungen und Partitionen

In einer Tabelle mit einem oder mehreren lokalen sekundären Indizes wird jede Elementauflistung in einer Partition gespeichert. Die Gesamtgröße einer solchen Elementauflistung ist auf die Kapazität dieser Partition beschränkt: 10 GB. Für eine Anwendung, bei der das Datenmodell Elementauflistungen enthält, deren Größe unbegrenzt ist oder bei denen Sie vernünftigerweise davon ausgehen können, dass einige Elementauflistungen in Zukunft auf mehr als 10 GB anwachsen werden, sollten Sie stattdessen einen globalen sekundären Index verwenden.

Sie sollten Ihre Anwendungen so konzipieren, dass Tabellendaten gleichmäßig auf unterschiedliche Partitionsschlüsselwerte verteilt werden. Für Tabellen mit lokalen sekundären Indizes sollten Ihre Anwendungen keine „Hotspots“ für Lese- und Schreibaktivitäten innerhalb einer einzigen Elementauflistung in einer einzelnen Partition erstellen.

Arbeiten mit lokalen sekundären Indizes: Java

Sie können die AWS SDK für Java Document-API verwenden, um eine Amazon DynamoDB-Tabelle mit einem oder mehreren lokalen Sekundärindizes zu erstellen, die Indizes in der Tabelle zu beschreiben und Abfragen mithilfe der Indizes durchzuführen.

Im Folgenden werden die üblichen Schritte für Tabellenoperationen mit der Document-API beschrieben. AWS SDK für Java

1. Erstellen Sie eine Instance der `DynamoDB`-Klasse.
2. Stellen Sie den erforderlichen und optionalen Parameter für die Operation bereit, indem Sie die entsprechenden Anforderungsobjekte erstellen.
3. Rufen Sie die entsprechende Methode auf, die vom Client, den Sie im vorhergehenden Schritt erstellt haben, bereitgestellt wird.

Themen

- Erstellen einer Tabelle mit einem lokalen sekundären Index
- Beschreiben einer Tabelle mit einem lokalen sekundären Index
- Abfragen eines lokalen sekundären Indexes
- Beispiel: Lokale sekundäre Indizes mit der Java-Dokument-API

Erstellen einer Tabelle mit einem lokalen sekundären Index

Lokale sekundäre Indizes müssen gleichzeitig mit dem Erstellen einer Tabelle erstellt werden. Verwenden Sie dazu die Methode `createTable` und geben Sie Ihre Angaben für einen oder mehrere lokale Sekundärindizes an. Das folgende Java-Codebeispiel erstellt eine Tabelle, die Informationen über Songs in einer Musiksammlung enthält. Der Partitionsschlüssel ist `Artist` und der Sortierschlüssel `SongTitle`. Der sekundäre Index, `AlbumTitleIndex`, vereinfacht Abfragen von Albumtiteln.

Im Folgenden werden die Schritte zum Erstellen einer Tabelle mit einem lokalen sekundären Index mithilfe der DynamoDB-Dokument-API dargelegt.

1. Erstellen Sie eine Instance der `DynamoDB`-Klasse.
2. Erstellen Sie eine Instance der `CreateTableRequest`-Klasse, um die Anforderungsinformationen bereitzustellen.

   Sie müssen den Tabellennamen, seinen zugehörigen Primärschlüssel und die Werte des bereitgestellten Durchsatzes angeben. Für den lokalen sekundären Index müssen Sie den Indexnamen, den Namen und den Datentyp des Indexsortierschlüssels, des Schlüsselschemas für den Index und der Attributprojektion angeben.
3. Rufen Sie die `createTable`-Methode auf, indem das Anforderungsobjekt als Parameter festgelegt wird.

Im folgenden Java-Codebeispiel werden die vorherigen Schritte veranschaulicht. Der Codeerstellt eine Tabelle (`Music`) mit einem sekundären Index auf dem `AlbumTitle`-Attribut. Der Tabellenpartitionsschlüssel und der Sortierschlüssel, sowie der Indexsortierschlüssel sind die einzigen in den Index projizierten Attribute.

```
AmazonDynamoDB client = AmazonDynamoDBClientBuilder.standard().build();
DynamoDB dynamoDB = new DynamoDB(client);

String tableName = "Music";

CreateTableRequest createTableRequest = new
 CreateTableRequest().withTableName(tableName);

//ProvisionedThroughput
createTableRequest.setProvisionedThroughput(new
 ProvisionedThroughput().withReadCapacityUnits((long)5).withWriteCapacityUnits((long)5));
```

```
//AttributeDefinitions
ArrayList<AttributeDefinition> attributeDefinitions= new
 ArrayList<AttributeDefinition>();
attributeDefinitions.add(new
 AttributeDefinition().withAttributeName("Artist").withAttributeType("S"));
attributeDefinitions.add(new
 AttributeDefinition().withAttributeName("SongTitle").withAttributeType("S"));
attributeDefinitions.add(new
 AttributeDefinition().withAttributeName("AlbumTitle").withAttributeType("S"));

createTableRequest.setAttributeDefinitions(attributeDefinitions);

//KeySchema
ArrayList<KeySchemaElement> tableKeySchema = new ArrayList<KeySchemaElement>();
tableKeySchema.add(new
 KeySchemaElement().withAttributeName("Artist").withKeyType(KeyType.HASH));  //
Partition key
tableKeySchema.add(new
 KeySchemaElement().withAttributeName("SongTitle").withKeyType(KeyType.RANGE));  //Sort
 key

createTableRequest.setKeySchema(tableKeySchema);

ArrayList<KeySchemaElement> indexKeySchema = new ArrayList<KeySchemaElement>();
indexKeySchema.add(new
 KeySchemaElement().withAttributeName("Artist").withKeyType(KeyType.HASH));  //
Partition key
indexKeySchema.add(new
 KeySchemaElement().withAttributeName("AlbumTitle").withKeyType(KeyType.RANGE));  //
Sort key

Projection projection = new Projection().withProjectionType(ProjectionType.INCLUDE);
ArrayList<String> nonKeyAttributes = new ArrayList<String>();
nonKeyAttributes.add("Genre");
nonKeyAttributes.add("Year");
projection.setNonKeyAttributes(nonKeyAttributes);

LocalSecondaryIndex localSecondaryIndex = new LocalSecondaryIndex()

 .withIndexName("AlbumTitleIndex").withKeySchema(indexKeySchema).withProjection(projection);

ArrayList<LocalSecondaryIndex> localSecondaryIndexes = new
 ArrayList<LocalSecondaryIndex>();
```

```
localSecondaryIndexes.add(localSecondaryIndex);
createTableRequest.setLocalSecondaryIndexes(localSecondaryIndexes);

Table table = dynamoDB.createTable(createTableRequest);
System.out.println(table.getDescription());
```

Sie müssen warten bis DynamoDB die Tabelle erstellt und den Tabellenstatus auf `ACTIVE` setzt. Im Anschluss können Sie die Daten in der Tabelle ablegen.

Beschreiben einer Tabelle mit einem lokalen sekundären Index

Um Informationen zu lokalen sekundären Indizes in einer Tabelle zu erhalten, verwenden Sie die Methode `describeTable`. Sie können auf den Namen, das Schlüsselschema und die projizierten Attribute von jedem Index zugreifen.

Im Folgenden sind die Schritte für den Zugriff auf lokale sekundäre Indexinformationen einer Tabelle mithilfe der AWS SDK für Java -Dokument-API aufgeführt.

1. Erstellen Sie eine Instance der `DynamoDB`-Klasse.
2. Erstellen Sie eine Instance der `Table`-Klasse. Sie müssen den Tabellennamen angeben.
3. Rufen Sie die `describeTable`-Methode für das `Table`-Objekt auf.

Im folgenden Java-Codebeispiel werden die vorherigen Schritte veranschaulicht.

Example

```
AmazonDynamoDB client = AmazonDynamoDBClientBuilder.standard().build();
DynamoDB dynamoDB = new DynamoDB(client);

String tableName = "Music";

Table table = dynamoDB.getTable(tableName);

TableDescription tableDescription = table.describe();

List<LocalSecondaryIndexDescription> localSecondaryIndexes
    = tableDescription.getLocalSecondaryIndexes();

// This code snippet will work for multiple indexes, even though
// there is only one index in this example.
```

```
Iterator<LocalSecondaryIndexDescription> lsiIter = localSecondaryIndexes.iterator();
while (lsiIter.hasNext()) {

    LocalSecondaryIndexDescription lsiDescription = lsiIter.next();
    System.out.println("Info for index " + lsiDescription.getIndexName() + ":");
    Iterator<KeySchemaElement> kseIter = lsiDescription.getKeySchema().iterator();
    while (kseIter.hasNext()) {
        KeySchemaElement kse = kseIter.next();
        System.out.printf("\t%s: %s\n", kse.getAttributeName(), kse.getKeyType());
    }
    Projection projection = lsiDescription.getProjection();
    System.out.println("\tThe projection type is: " + projection.getProjectionType());
    if (projection.getProjectionType().toString().equals("INCLUDE")) {
        System.out.println("\t\tThe non-key projected attributes are: " +
 projection.getNonKeyAttributes());
    }
}
```

Abfragen eines lokalen sekundären Indexes

Sie können die Operation `Query` für einen lokalen sekundären Index genauso nutzen, wie Sie `Query` für eine Tabelle nutzen. Sie müssen den Indexnamen, die Abfragekriterien für den Indexsortierschlüssel und die Attribute angeben, die Sie zurückgeben möchten. In diesem Beispiel ist der Index `AlbumTitleIndex` und der Indexsortierschlüssel `AlbumTitle`.

Die einzigen zurückgegebenen Attribute sind die, die in den Index projiziert wurden. Sie könnten diese Abfrage ändern, um auch Nicht-Schlüsselattribute auszuwählen, aber dies würde eine Tabellenabrufaktivität erfordern, die relativ teuer ist. Weitere Informationen zum Abrufen von Tabellen finden Sie unter Attributprojektionen.

Im Folgenden werden die Schritte zum Abfragen eines lokalen sekundären Indexes mithilfe der AWS SDK für Java Dokument-API beschrieben.

1. Erstellen Sie eine Instance der `DynamoDB`-Klasse.
2. Erstellen Sie eine Instance der `Table`-Klasse. Sie müssen den Tabellennamen angeben.
3. Erstellen Sie eine Instance der `Index`-Klasse. Sie müssen den Indexnamen angeben.
4. Rufen Sie die `query`-Methode für die `Index`-Klasse auf.

Im folgenden Java-Codebeispiel werden die vorherigen Schritte veranschaulicht.

Example

```
AmazonDynamoDB client = AmazonDynamoDBClientBuilder.standard().build();
DynamoDB dynamoDB = new DynamoDB(client);

String tableName = "Music";

Table table = dynamoDB.getTable(tableName);
Index index = table.getIndex("AlbumTitleIndex");

QuerySpec spec = new QuerySpec()
    .withKeyConditionExpression("Artist = :v_artist and AlbumTitle = :v_title")
    .withValueMap(new ValueMap()
        .withString(":v_artist", "Acme Band")
        .withString(":v_title", "Songs About Life"));

ItemCollection<QueryOutcome> items = index.query(spec);

Iterator<Item> itemsIter = items.iterator();

while (itemsIter.hasNext()) {
    Item item = itemsIter.next();
    System.out.println(item.toJSONPretty());
}
```

Konsistente Lesevorgänge auf einem lokalen sekundären Index

Im Gegensatz zu globalen Sekundärindizes, die nur eventuell konsistente Lesevorgänge unterstützen, unterstützt ein lokaler sekundärer Index sowohl letztlich konsistente als auch stark konsistente Lesevorgänge. Bei einem Strongly-Consistent-Lesevorgang von einem lokalen sekundären Index werden immer die zuletzt aktualisierten Werte zurückgegeben. Wenn die Abfrage zusätzliche Attribute aus der Basistabelle abrufen muss, sind diese abgerufenen Attribute auch in Bezug auf den Index konsistent.

`Query`Verwendet standardmäßig eventuell konsistente Lesevorgänge. Um einen stark konsistenten Lesevorgang anzufordern, setzen Sie `true` in der `ConsistentRead` auf`QuerySpec`. Im folgenden Beispiel wird für `AlbumTitleIndex` Abfragen ein stark konsistenter Lesevorgang verwendet:

Example

```
QuerySpec spec = new QuerySpec()
    .withKeyConditionExpression("Artist = :v_artist and AlbumTitle = :v_title")
```

```
    .withValueMap(new ValueMap()
        .withString(":v_artist", "Acme Band")
        .withString(":v_title", "Songs About Life"))
    .withConsistentRead(true);
```

> **Note**
>
> Ein stark konsistenter Lesevorgang verbraucht eine Lesekapazitätseinheit pro 4 KB zurückgegebener Daten (aufgerundet), wohingegen ein eventuell konsistenter Lesevorgang die Hälfte davon verbraucht. Beispielsweise verbraucht ein stark konsistenter Lesevorgang, der 9 KB an Daten zurückgibt, 3 Lesekapazitätseinheiten (9 KB/4 KB = 2,25, aufgerundet auf 3), während dieselbe Abfrage, die einen eventuell konsistenten Lesevorgang verwendet, 1,5 Lesekapazitätseinheiten verbraucht. Wenn Ihre Anwendung das Lesen von Daten verträgt, die möglicherweise etwas veraltet sind, verwenden Sie eventuell konsistente Lesevorgänge, um Ihre Lesekapazitätsnutzung zu reduzieren. Weitere Informationen finden Sie unter Lesekapazitätseinheiten.

Beispiel: Lokale sekundäre Indizes mit der Java-Dokument-API

Das folgende Java-Codebeispiel zeigt, wie Sie mit lokalen sekundären Indizes in Amazon DynamoDB arbeiten. Das Beispiel erstellt eine Tabelle mit dem Namen `CustomerOrders` mit `CustomerId` als Partitionsschlüssel und `OrderId` als Sortierschlüssel. Es gibt zwei lokale sekundäre Indizes in dieser Tabelle:

- `OrderCreationDateIndex` – der Sortierschlüssel ist `OrderCreationDate` und die folgenden Attribute werden in den Index projiziert:
  - `ProductCategory`
  - `ProductName`
  - `OrderStatus`
  - `ShipmentTrackingId`
- `IsOpenIndex` – der Sortierschlüssel ist `IsOpen` und alle Tabellenattribute werden in den Index projiziert.

Nachdem die `CustomerOrders`-Tabelle erstellt wurde, lädt das Programm die Tabelle mit Daten, die Kundenaufträge darstellen. Mit If Then werden die Daten unter Verwendung der lokalen sekundären Indizes abgefragt. Schließlich löscht das Programm die `CustomerOrders`-Tabelle.

step-by-stepAnweisungen zum Testen des folgenden Beispiels finden Sie unterJava-Codebeispiele.

Example

```
package com.example.dynamodb;

import software.amazon.awssdk.core.waiters.WaiterResponse;
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.*;
import software.amazon.awssdk.services.dynamodb.waiters.DynamoDbWaiter;

import java.util.HashMap;
import java.util.Map;

public class DocumentAPILocalSecondaryIndexExample {

    static DynamoDbClient client = DynamoDbClient.create();
    public static String tableName = "CustomerOrders";

    public static void main(String[] args) {
        createTable();
        loadData();
        query(null);
        query("IsOpenIndex");
        query("OrderCreationDateIndex");
        deleteTable(tableName);
    }

    public static void createTable() {
        CreateTableRequest request = CreateTableRequest.builder()
            .tableName(tableName)
            .provisionedThroughput(ProvisionedThroughput.builder()
                .readCapacityUnits(1L)
                .writeCapacityUnits(1L)
                .build())
            .attributeDefinitions(

 AttributeDefinition.builder().attributeName("CustomerId").attributeType(ScalarAttributeType.S)

 AttributeDefinition.builder().attributeName("OrderId").attributeType(ScalarAttributeType.N).bu

 AttributeDefinition.builder().attributeName("OrderCreationDate").attributeType(ScalarAttribute
```

```
AttributeDefinition.builder().attributeName("IsOpen").attributeType(ScalarAttributeType.N).bui
            .keySchema(

KeySchemaElement.builder().attributeName("CustomerId").keyType(KeyType.HASH).build(),

KeySchemaElement.builder().attributeName("OrderId").keyType(KeyType.RANGE).build())
            .localSecondaryIndexes(
                LocalSecondaryIndex.builder()
                    .indexName("OrderCreationDateIndex")
                    .keySchema(

KeySchemaElement.builder().attributeName("CustomerId").keyType(KeyType.HASH).build(),

KeySchemaElement.builder().attributeName("OrderCreationDate").keyType(KeyType.RANGE).build())
                    .projection(Projection.builder()
                        .projectionType(ProjectionType.INCLUDE)
                        .nonKeyAttributes("ProductCategory", "ProductName")
                        .build())
                    .build(),
                LocalSecondaryIndex.builder()
                    .indexName("IsOpenIndex")
                    .keySchema(

KeySchemaElement.builder().attributeName("CustomerId").keyType(KeyType.HASH).build(),

KeySchemaElement.builder().attributeName("IsOpen").keyType(KeyType.RANGE).build())
                    .projection(Projection.builder()
                        .projectionType(ProjectionType.ALL)
                        .build())
                    .build())
            .build();

        System.out.println("Creating table " + tableName + "...");
        client.createTable(request);

        try (DynamoDbWaiter waiter = client.waiter()) {
            WaiterResponse<DescribeTableResponse> response =
waiter.waitUntilTableExists(r -> r.tableName(tableName));
            response.matched().response().ifPresent(System.out::println);
        }
    }

    public static void query(String indexName) {
```

```
 System.out.println("\n***********************************************************\n");
        System.out.println("Querying table " + tableName + "...");

        if ("IsOpenIndex".equals(indexName)) {
            System.out.println("\nUsing index: '" + indexName + "': Bob's orders that
 are open.");
            System.out.println("Only a user-specified list of attributes are returned
\n");

            Map<String, AttributeValue> values = new HashMap<>();
            values.put(":v_custid",
 AttributeValue.builder().s("bob@example.com").build());
            values.put(":v_isopen", AttributeValue.builder().n("1").build());

            QueryRequest request = QueryRequest.builder()
                .tableName(tableName)
                .indexName(indexName)
                .keyConditionExpression("CustomerId = :v_custid and IsOpen
 = :v_isopen")
                .expressionAttributeValues(values)
                .projectionExpression("OrderCreationDate, ProductCategory, ProductName,
 OrderStatus")
                .build();

            System.out.println("Query: printing results...");
            client.query(request).items().forEach(System.out::println);

        } else if ("OrderCreationDateIndex".equals(indexName)) {
            System.out.println("\nUsing index: '" + indexName + "': Bob's orders that
 were placed after 01/31/2015.");
            System.out.println("Only the projected attributes are returned\n");

            Map<String, AttributeValue> values = new HashMap<>();
            values.put(":v_custid",
 AttributeValue.builder().s("bob@example.com").build());
            values.put(":v_orddate", AttributeValue.builder().n("20150131").build());

            QueryRequest request = QueryRequest.builder()
                .tableName(tableName)
                .indexName(indexName)
                .keyConditionExpression("CustomerId = :v_custid and OrderCreationDate
 >= :v_orddate")
                .expressionAttributeValues(values)
```

```
                .select(Select.ALL_PROJECTED_ATTRIBUTES)
                .build();

            System.out.println("Query: printing results...");
            client.query(request).items().forEach(System.out::println);

        } else {
            System.out.println("\nNo index: All of Bob's orders, by OrderId:\n");

            Map<String, AttributeValue> values = new HashMap<>();
            values.put(":v_custid",
AttributeValue.builder().s("bob@example.com").build());

            QueryRequest request = QueryRequest.builder()
                .tableName(tableName)
                .keyConditionExpression("CustomerId = :v_custid")
                .expressionAttributeValues(values)
                .build();

            System.out.println("Query: printing results...");
            client.query(request).items().forEach(System.out::println);
        }
    }

    public static void deleteTable(String tableName) {
        System.out.println("Deleting table " + tableName + "...");
        client.deleteTable(DeleteTableRequest.builder().tableName(tableName).build());

        try (DynamoDbWaiter waiter = client.waiter()) {
            waiter.waitUntilTableNotExists(r -> r.tableName(tableName));
        }
    }

    public static void loadData() {
        System.out.println("Loading data into table " + tableName + "...");

        putItem(Map.of(
            "CustomerId", AttributeValue.builder().s("alice@example.com").build(),
            "OrderId", AttributeValue.builder().n("1").build(),
            "IsOpen", AttributeValue.builder().n("1").build(),
            "OrderCreationDate", AttributeValue.builder().n("20150101").build(),
            "ProductCategory", AttributeValue.builder().s("Book").build(),
            "ProductName", AttributeValue.builder().s("The Great Outdoors").build(),
            "OrderStatus", AttributeValue.builder().s("PACKING ITEMS").build()));
```

```
        putItem(Map.of(
            "CustomerId", AttributeValue.builder().s("alice@example.com").build(),
            "OrderId", AttributeValue.builder().n("2").build(),
            "IsOpen", AttributeValue.builder().n("1").build(),
            "OrderCreationDate", AttributeValue.builder().n("20150221").build(),
            "ProductCategory", AttributeValue.builder().s("Bike").build(),
            "ProductName", AttributeValue.builder().s("Super Mountain").build(),
            "OrderStatus", AttributeValue.builder().s("ORDER RECEIVED").build()));

        putItem(Map.of(
            "CustomerId", AttributeValue.builder().s("alice@example.com").build(),
            "OrderId", AttributeValue.builder().n("3").build(),
            "OrderCreationDate", AttributeValue.builder().n("20150304").build(),
            "ProductCategory", AttributeValue.builder().s("Music").build(),
            "ProductName", AttributeValue.builder().s("A Quiet Interlude").build(),
            "OrderStatus", AttributeValue.builder().s("IN TRANSIT").build(),
            "ShipmentTrackingId", AttributeValue.builder().s("176493").build()));

        putItem(Map.of(
            "CustomerId", AttributeValue.builder().s("bob@example.com").build(),
            "OrderId", AttributeValue.builder().n("1").build(),
            "OrderCreationDate", AttributeValue.builder().n("20150111").build(),
            "ProductCategory", AttributeValue.builder().s("Movie").build(),
            "ProductName", AttributeValue.builder().s("Calm Before The Storm").build(),
            "OrderStatus", AttributeValue.builder().s("SHIPPING DELAY").build(),
            "ShipmentTrackingId", AttributeValue.builder().s("859323").build()));

        putItem(Map.of(
            "CustomerId", AttributeValue.builder().s("bob@example.com").build(),
            "OrderId", AttributeValue.builder().n("2").build(),
            "OrderCreationDate", AttributeValue.builder().n("20150124").build(),
            "ProductCategory", AttributeValue.builder().s("Music").build(),
            "ProductName", AttributeValue.builder().s("E-Z Listening").build(),
            "OrderStatus", AttributeValue.builder().s("DELIVERED").build(),
            "ShipmentTrackingId", AttributeValue.builder().s("756943").build()));

        putItem(Map.of(
            "CustomerId", AttributeValue.builder().s("bob@example.com").build(),
            "OrderId", AttributeValue.builder().n("3").build(),
            "OrderCreationDate", AttributeValue.builder().n("20150221").build(),
            "ProductCategory", AttributeValue.builder().s("Music").build(),
            "ProductName", AttributeValue.builder().s("Symphony 9").build(),
            "OrderStatus", AttributeValue.builder().s("DELIVERED").build(),
```

```
            "ShipmentTrackingId", AttributeValue.builder().s("645193").build()));

        putItem(Map.of(
            "CustomerId", AttributeValue.builder().s("bob@example.com").build(),
            "OrderId", AttributeValue.builder().n("4").build(),
            "IsOpen", AttributeValue.builder().n("1").build(),
            "OrderCreationDate", AttributeValue.builder().n("20150222").build(),
            "ProductCategory", AttributeValue.builder().s("Hardware").build(),
            "ProductName", AttributeValue.builder().s("Extra Heavy Hammer").build(),
            "OrderStatus", AttributeValue.builder().s("PACKING ITEMS").build()));

        putItem(Map.of(
            "CustomerId", AttributeValue.builder().s("bob@example.com").build(),
            "OrderId", AttributeValue.builder().n("5").build(),
            "OrderCreationDate", AttributeValue.builder().n("20150309").build(),
            "ProductCategory", AttributeValue.builder().s("Book").build(),
            "ProductName", AttributeValue.builder().s("How To Cook").build(),
            "OrderStatus", AttributeValue.builder().s("IN TRANSIT").build(),
            "ShipmentTrackingId", AttributeValue.builder().s("440185").build()));

        putItem(Map.of(
            "CustomerId", AttributeValue.builder().s("bob@example.com").build(),
            "OrderId", AttributeValue.builder().n("6").build(),
            "OrderCreationDate", AttributeValue.builder().n("20150318").build(),
            "ProductCategory", AttributeValue.builder().s("Luggage").build(),
            "ProductName", AttributeValue.builder().s("Really Big Suitcase").build(),
            "OrderStatus", AttributeValue.builder().s("DELIVERED").build(),
            "ShipmentTrackingId", AttributeValue.builder().s("893927").build()));

        putItem(Map.of(
            "CustomerId", AttributeValue.builder().s("bob@example.com").build(),
            "OrderId", AttributeValue.builder().n("7").build(),
            "OrderCreationDate", AttributeValue.builder().n("20150324").build(),
            "ProductCategory", AttributeValue.builder().s("Golf").build(),
            "ProductName", AttributeValue.builder().s("PGA Pro II").build(),
            "OrderStatus", AttributeValue.builder().s("OUT FOR DELIVERY").build(),
            "ShipmentTrackingId", AttributeValue.builder().s("383283").build()));
    }

    private static void putItem(Map<String, AttributeValue> item) {

 client.putItem(PutItemRequest.builder().tableName(tableName).item(item).build());
    }
}
```

Arbeiten mit lokalen sekundären Indizes: .NET

Themen

- Erstellen einer Tabelle mit einem lokalen sekundären Index
- Beschreiben einer Tabelle mit einem lokalen sekundären Index
- Abfragen eines lokalen sekundären Indexes
- Beispiel: Lokale sekundäre Indizes mithilfe der Low-Level-API AWS SDK für .NET

Sie können die AWS SDK für .NET Low-Level-API verwenden, um eine Amazon DynamoDB-Tabelle mit einem oder mehreren lokalen Sekundärindizes zu erstellen, die Indizes in der Tabelle zu beschreiben und Abfragen mithilfe der Indizes durchzuführen. Diese Operationen entsprechen den Low-Level-DynamoDB-API-Aktionen. Weitere Informationen finden Sie unter .NET-Codebeispiele.

Folgende sind die allgemeinen Schritte für Tabellenoperationen mithilfe der .NET-Low-Level-API.

1. Erstellen Sie eine Instance der `AmazonDynamoDBClient`-Klasse.
2. Stellen Sie den erforderlichen und optionalen Parameter für die Operation bereit, indem Sie die entsprechenden Anforderungsobjekte erstellen.

   Erstellen Sie beispielsweise ein `CreateTableRequest`-Objekt, um eine Tabelle zu erstellen und ein `QueryRequest`-Objekt, um eine Tabelle oder einen Index abzufragen.
3. Rufen Sie die entsprechende Methode auf, die vom Client, den Sie im vorhergehenden Schritt erstellt haben, bereitgestellt wird.

Erstellen einer Tabelle mit einem lokalen sekundären Index

Lokale sekundäre Indizes müssen gleichzeitig beim Erstellen einer Tabelle erstellt werden. Zu diesem Zweck verwenden Sie `CreateTable` und geben Ihre Spezifikationen für ein oder mehrere lokale sekundäre Indizes an. Das folgende C#-Codebeispiel erstellt eine Tabelle, die Informationen über Songs in einer Musiksammlung enthält. Der Partitionsschlüssel ist `Artist` und der Sortierschlüssel `SongTitle`. Der sekundäre Index, `AlbumTitleIndex`, vereinfacht Abfragen von Albumtiteln.

Im Folgenden sind die Schritte zum Erstellen einer Tabelle mit einem lokalen sekundären Index unter Verwendung der .NET-API auf niedriger Ebene aufgeführt.

1. Erstellen Sie eine Instance der `AmazonDynamoDBClient`-Klasse.
2. Erstellen Sie eine Instance der `CreateTableRequest`-Klasse, um die Anforderungsinformationen bereitzustellen.

   Sie müssen den Tabellennamen, seinen zugehörigen Primärschlüssel und die Werte des bereitgestellten Durchsatzes angeben. Für den lokalen sekundären Index müssen Sie den Indexnamen, den Namen und den Datentyp des Indexsortierschlüssels, das Schlüsselschema für den Index und die Attributprojektion angeben.
3. Führen Sie die `CreateTable`-Methode aus, indem das Anforderungsobjekt als Parameter festgelegt wird.

Im folgenden C#-Codebeispiel werden die vorherigen Schritte veranschaulicht. Der Codeerstellt eine Tabelle (`Music`) mit einem sekundären Index auf dem `AlbumTitle`-Attribut. Der Tabellenpartitionsschlüssel und der Sortierschlüssel, sowie der Indexsortierschlüssel sind die einzigen in den Index projizierten Attribute.

```
AmazonDynamoDBClient client = new AmazonDynamoDBClient();
string tableName = "Music";

CreateTableRequest createTableRequest = new CreateTableRequest()
{
    TableName = tableName
};

//ProvisionedThroughput
createTableRequest.ProvisionedThroughput = new ProvisionedThroughput()
{
    ReadCapacityUnits = (long)5,
    WriteCapacityUnits = (long)5
};

//AttributeDefinitions
List<AttributeDefinition> attributeDefinitions = new List<AttributeDefinition>();

attributeDefinitions.Add(new AttributeDefinition()
{
    AttributeName = "Artist",
    AttributeType = "S"
});
```

```
attributeDefinitions.Add(new AttributeDefinition()
 {
     AttributeName = "SongTitle",
     AttributeType = "S"
 });

attributeDefinitions.Add(new AttributeDefinition()
 {
     AttributeName = "AlbumTitle",
     AttributeType = "S"
 });

createTableRequest.AttributeDefinitions = attributeDefinitions;

//KeySchema
List<KeySchemaElement> tableKeySchema = new List<KeySchemaElement>();

tableKeySchema.Add(new KeySchemaElement() { AttributeName = "Artist", KeyType =
 "HASH" });  //Partition key
tableKeySchema.Add(new KeySchemaElement() { AttributeName = "SongTitle", KeyType =
 "RANGE" });  //Sort key

createTableRequest.KeySchema = tableKeySchema;

List<KeySchemaElement> indexKeySchema = new List<KeySchemaElement>();
indexKeySchema.Add(new KeySchemaElement() { AttributeName = "Artist", KeyType =
 "HASH" });  //Partition key
indexKeySchema.Add(new KeySchemaElement() { AttributeName = "AlbumTitle", KeyType =
 "RANGE" });  //Sort key

Projection projection = new Projection() { ProjectionType = "INCLUDE" };

List<string> nonKeyAttributes = new List<string>();
nonKeyAttributes.Add("Genre");
nonKeyAttributes.Add("Year");
projection.NonKeyAttributes = nonKeyAttributes;

LocalSecondaryIndex localSecondaryIndex = new LocalSecondaryIndex()
{
    IndexName = "AlbumTitleIndex",
    KeySchema = indexKeySchema,
    Projection = projection
};
```

```
List<LocalSecondaryIndex> localSecondaryIndexes = new List<LocalSecondaryIndex>();
localSecondaryIndexes.Add(localSecondaryIndex);
createTableRequest.LocalSecondaryIndexes = localSecondaryIndexes;

CreateTableResponse result = client.CreateTable(createTableRequest);
Console.WriteLine(result.CreateTableResult.TableDescription.TableName);
Console.WriteLine(result.CreateTableResult.TableDescription.TableStatus);
```

Sie müssen warten bis DynamoDB die Tabelle erstellt und den Tabellenstatus auf `ACTIVE` setzt. Im Anschluss können Sie die Daten in der Tabelle ablegen.

Beschreiben einer Tabelle mit einem lokalen sekundären Index

Um Informationen zu lokalen sekundären Indizes in einer Tabelle zu erhalten, verwenden Sie die `DescribeTable`-API. Sie können auf den Namen, das Schlüsselschema und die projizierten Attribute von jedem Index zugreifen.

Im Folgenden finden Sie die Schritte zum Zugreifen auf lokale sekundäre Indexinformationen einer Tabelle mithilfe der .NET-Low-Level-API.

1. Erstellen Sie eine Instance der `AmazonDynamoDBClient`-Klasse.
2. Erstellen Sie eine Instance der `DescribeTableRequest`-Klasse, um die Anforderungsinformationen bereitzustellen. Sie müssen den Tabellennamen angeben.
3. Führen Sie die `describeTable`-Methode aus, indem das Anforderungsobjekt als Parameter festgelegt wird.
4.

Im folgenden C#-Codebeispiel werden die vorherigen Schritte veranschaulicht.

Example

```
AmazonDynamoDBClient client = new AmazonDynamoDBClient();
string tableName = "Music";

DescribeTableResponse response = client.DescribeTable(new DescribeTableRequest()
 { TableName = tableName });
List<LocalSecondaryIndexDescription> localSecondaryIndexes =
    response.DescribeTableResult.Table.LocalSecondaryIndexes;

// This code snippet will work for multiple indexes, even though
```

```
// there is only one index in this example.
foreach (LocalSecondaryIndexDescription lsiDescription in localSecondaryIndexes)
{
    Console.WriteLine("Info for index " + lsiDescription.IndexName + ":");

    foreach (KeySchemaElement kse in lsiDescription.KeySchema)
    {
        Console.WriteLine("\t" + kse.AttributeName + ": key type is " + kse.KeyType);
    }

    Projection projection = lsiDescription.Projection;

    Console.WriteLine("\tThe projection type is: " + projection.ProjectionType);

    if (projection.ProjectionType.ToString().Equals("INCLUDE"))
    {
        Console.WriteLine("\t\tThe non-key projected attributes are:");

        foreach (String s in projection.NonKeyAttributes)
        {
            Console.WriteLine("\t\t" + s);
        }

    }
}
```

Abfragen eines lokalen sekundären Indexes

Sie können `Query` für einen lokalen sekundären Index genauso nutzen, wie Sie `Query` für eine Tabelle nutzen. Sie müssen den Indexnamen, die Abfragekriterien für den Indexsortierschlüssel und die Attribute angeben, die Sie zurückgeben möchten. In diesem Beispiel ist der Index `AlbumTitleIndex` und der Indexsortierschlüssel `AlbumTitle`.

Die einzigen zurückgegebenen Attribute sind die, die in den Index projiziert wurden. Sie könnten diese Abfrage ändern, um auch Nicht-Schlüsselattribute auszuwählen, aber dies würde eine Tabellenabrufaktivität erfordern, die relativ teuer ist. Weitere Informationen zum Abrufen von Tabellen finden Sie unter Attributprojektionen

Im Folgenden werden die Schritte zur Abfrage eines lokalen sekundären Indizes mithilfe der .NET-Low-Level-API dargelegt.

1. Erstellen Sie eine Instance der `AmazonDynamoDBClient`-Klasse.

2. Erstellen Sie eine Instance der `QueryRequest`-Klasse, um die Anforderungsinformationen bereitzustellen.
3. Führen Sie die `query`-Methode aus, indem das Anforderungsobjekt als Parameter festgelegt wird.

Im folgenden C#-Codebeispiel werden die vorherigen Schritte veranschaulicht.

Example

```
QueryRequest queryRequest = new QueryRequest
{
    TableName = "Music",
    IndexName = "AlbumTitleIndex",
    Select = "ALL_ATTRIBUTES",
    ScanIndexForward = true,
    KeyConditionExpression = "Artist = :v_artist and AlbumTitle = :v_title",
    ExpressionAttributeValues = new Dictionary<string, AttributeValue>()
    {
        {":v_artist",new AttributeValue {S = "Acme Band"}},
        {":v_title",new AttributeValue {S = "Songs About Life"}}
    },
};

QueryResponse response = client.Query(queryRequest);

foreach (var attribs in response.Items)
{
    foreach (var attrib in attribs)
    {
        Console.WriteLine(attrib.Key + " ---> " + attrib.Value.S);
    }
    Console.WriteLine();
}
```

Beispiel: Lokale sekundäre Indizes mithilfe der Low-Level-API AWS SDK für .NET

Das folgende C#-Codebeispiel zeigt, wie Sie mit lokalen sekundären Indizes in Amazon DynamoDB arbeiten. Das Beispiel erstellt eine Tabelle mit dem Namen `CustomerOrders` mit `CustomerId` als Partitionsschlüssel und `OrderId` als Sortierschlüssel. Es gibt zwei lokale sekundäre Indizes in dieser Tabelle:

- `OrderCreationDateIndex` – der Sortierschlüssel ist `OrderCreationDate` und die folgenden Attribute werden in den Index projiziert:
  - `ProductCategory`
  - `ProductName`
  - `OrderStatus`
  - `ShipmentTrackingId`
- `IsOpenIndex` – der Sortierschlüssel ist `IsOpen` und alle Tabellenattribute werden in den Index projiziert.

Nachdem die `CustomerOrders`-Tabelle erstellt wurde, lädt das Programm die Tabelle mit Daten, die Kundenaufträge darstellen. Mit If Then werden die Daten unter Verwendung der lokalen sekundären Indizes abgefragt. Schließlich löscht das Programm die `CustomerOrders`-Tabelle.

step-by-stepAnweisungen zum Testen des folgenden Beispiels finden Sie unter. .NET-Codebeispiele

Example

```
using System;
using System.Collections.Generic;
using System.Linq;
using Amazon.DynamoDBv2;
using Amazon.DynamoDBv2.DataModel;
using Amazon.DynamoDBv2.DocumentModel;
using Amazon.DynamoDBv2.Model;
using Amazon.Runtime;
using Amazon.SecurityToken;

namespace com.amazonaws.codesamples
{
    class LowLevelLocalSecondaryIndexExample
    {
        private static AmazonDynamoDBClient client = new AmazonDynamoDBClient();
        private static string tableName = "CustomerOrders";

        static void Main(string[] args)
        {
            try
            {
                CreateTable();
                LoadData();
```

```
            Query(null);
            Query("IsOpenIndex");
            Query("OrderCreationDateIndex");

            DeleteTable(tableName);

            Console.WriteLine("To continue, press Enter");
            Console.ReadLine();
        }
        catch (AmazonDynamoDBException e) { Console.WriteLine(e.Message); }
        catch (AmazonServiceException e) { Console.WriteLine(e.Message); }
        catch (Exception e) { Console.WriteLine(e.Message); }
    }

    private static void CreateTable()
    {
        var createTableRequest =
            new CreateTableRequest()
            {
                TableName = tableName,
                ProvisionedThroughput =
                new ProvisionedThroughput()
                {
                    ReadCapacityUnits = (long)1,
                    WriteCapacityUnits = (long)1
                }
            };

        var attributeDefinitions = new List<AttributeDefinition>()
    {
        // Attribute definitions for table primary key
        { new AttributeDefinition() {
              AttributeName = "CustomerId", AttributeType = "S"
          } },
        { new AttributeDefinition() {
              AttributeName = "OrderId", AttributeType = "N"
          } },
        // Attribute definitions for index primary key
        { new AttributeDefinition() {
              AttributeName = "OrderCreationDate", AttributeType = "N"
          } },
        { new AttributeDefinition() {
              AttributeName = "IsOpen", AttributeType = "N"
```

```
            }}
        };

            createTableRequest.AttributeDefinitions = attributeDefinitions;

            // Key schema for table
            var tableKeySchema = new List<KeySchemaElement>()
        {
            { new KeySchemaElement() {
                  AttributeName = "CustomerId", KeyType = "HASH"
              } },                                                  //Partition key
            { new KeySchemaElement() {
                  AttributeName = "OrderId", KeyType = "RANGE"
              } }                                                 //Sort key
        };

            createTableRequest.KeySchema = tableKeySchema;

            var localSecondaryIndexes = new List<LocalSecondaryIndex>();

            // OrderCreationDateIndex
            LocalSecondaryIndex orderCreationDateIndex = new LocalSecondaryIndex()
            {
                IndexName = "OrderCreationDateIndex"
            };

            // Key schema for OrderCreationDateIndex
            var indexKeySchema = new List<KeySchemaElement>()
        {
            { new KeySchemaElement() {
                  AttributeName = "CustomerId", KeyType = "HASH"
              } },                                                    //Partition key
            { new KeySchemaElement() {
                  AttributeName = "OrderCreationDate", KeyType = "RANGE"
              } }                                                              //Sort key
        };

            orderCreationDateIndex.KeySchema = indexKeySchema;

            // Projection (with list of projected attributes) for
            // OrderCreationDateIndex
            var projection = new Projection()
            {
                ProjectionType = "INCLUDE"
```

```
            };

            var nonKeyAttributes = new List<string>()
        {
            "ProductCategory",
            "ProductName"
        };
            projection.NonKeyAttributes = nonKeyAttributes;

            orderCreationDateIndex.Projection = projection;

            localSecondaryIndexes.Add(orderCreationDateIndex);

            // IsOpenIndex
            LocalSecondaryIndex isOpenIndex
                = new LocalSecondaryIndex()
                {
                    IndexName = "IsOpenIndex"
                };

            // Key schema for IsOpenIndex
            indexKeySchema = new List<KeySchemaElement>()
        {
            { new KeySchemaElement() {
                  AttributeName = "CustomerId", KeyType = "HASH"
              }},                                                    //Partition key
            { new KeySchemaElement() {
                  AttributeName = "IsOpen", KeyType = "RANGE"
              }}                                                  //Sort key
        };

            // Projection (all attributes) for IsOpenIndex
            projection = new Projection()
            {
                ProjectionType = "ALL"
            };

            isOpenIndex.KeySchema = indexKeySchema;
            isOpenIndex.Projection = projection;

            localSecondaryIndexes.Add(isOpenIndex);

            // Add index definitions to CreateTable request
            createTableRequest.LocalSecondaryIndexes = localSecondaryIndexes;
```

```
            Console.WriteLine("Creating table " + tableName + "...");
            client.CreateTable(createTableRequest);
            WaitUntilTableReady(tableName);
        }

        public static void Query(string indexName)
        {

Console.WriteLine("\n*****************************************************************\n");
            Console.WriteLine("Querying table " + tableName + "...");

            QueryRequest queryRequest = new QueryRequest()
            {
                TableName = tableName,
                ConsistentRead = true,
                ScanIndexForward = true,
                ReturnConsumedCapacity = "TOTAL"
            };


            String keyConditionExpression = "CustomerId = :v_customerId";
            Dictionary<string, AttributeValue> expressionAttributeValues = new
Dictionary<string, AttributeValue> {
            {":v_customerId", new AttributeValue {
                 S = "bob@example.com"
             }}
        };


            if (indexName == "IsOpenIndex")
            {
                Console.WriteLine("\nUsing index: '" + indexName
                          + "': Bob's orders that are open.");
                Console.WriteLine("Only a user-specified list of attributes are
returned\n");
                queryRequest.IndexName = indexName;

                keyConditionExpression += " and IsOpen = :v_isOpen";
                expressionAttributeValues.Add(":v_isOpen", new AttributeValue
                {
                    N = "1"
                });
```

```
                // ProjectionExpression
                queryRequest.ProjectionExpression = "OrderCreationDate,
ProductCategory, ProductName, OrderStatus";
            }
            else if (indexName == "OrderCreationDateIndex")
            {
                Console.WriteLine("\nUsing index: '" + indexName
                          + "': Bob's orders that were placed after 01/31/2013.");
                Console.WriteLine("Only the projected attributes are returned\n");
                queryRequest.IndexName = indexName;

                keyConditionExpression += " and OrderCreationDate > :v_Date";
                expressionAttributeValues.Add(":v_Date", new AttributeValue
                {
                    N = "20130131"
                });

                // Select
                queryRequest.Select = "ALL_PROJECTED_ATTRIBUTES";
            }
            else
            {
                Console.WriteLine("\nNo index: All of Bob's orders, by OrderId:\n");
            }
            queryRequest.KeyConditionExpression = keyConditionExpression;
            queryRequest.ExpressionAttributeValues = expressionAttributeValues;

            var result = client.Query(queryRequest);
            var items = result.Items;
            foreach (var currentItem in items)
            {
                foreach (string attr in currentItem.Keys)
                {
                    if (attr == "OrderId" || attr == "IsOpen"
                        || attr == "OrderCreationDate")
                    {
                        Console.WriteLine(attr + "---> " + currentItem[attr].N);
                    }
                    else
                    {
                        Console.WriteLine(attr + "---> " + currentItem[attr].S);
                    }
                }
                Console.WriteLine();
```

```
            }
            Console.WriteLine("\nConsumed capacity: " +
result.ConsumedCapacity.CapacityUnits + "\n");
        }

        private static void DeleteTable(string tableName)
        {
            Console.WriteLine("Deleting table " + tableName + "...");
            client.DeleteTable(new DeleteTableRequest()
            {
                TableName = tableName
            });
            WaitForTableToBeDeleted(tableName);
        }

        public static void LoadData()
        {
            Console.WriteLine("Loading data into table " + tableName + "...");

            Dictionary<string, AttributeValue> item = new Dictionary<string,
AttributeValue>();

            item["CustomerId"] = new AttributeValue
            {
                S = "alice@example.com"
            };
            item["OrderId"] = new AttributeValue
            {
                N = "1"
            };
            item["IsOpen"] = new AttributeValue
            {
                N = "1"
            };
            item["OrderCreationDate"] = new AttributeValue
            {
                N = "20130101"
            };
            item["ProductCategory"] = new AttributeValue
            {
                S = "Book"
            };
            item["ProductName"] = new AttributeValue
            {
```

```
            S = "The Great Outdoors"
        };
        item["OrderStatus"] = new AttributeValue
        {
            S = "PACKING ITEMS"
        };
        /* no ShipmentTrackingId attribute */
        PutItemRequest putItemRequest = new PutItemRequest
        {
            TableName = tableName,
            Item = item,
            ReturnItemCollectionMetrics = "SIZE"
        };
        client.PutItem(putItemRequest);

        item = new Dictionary<string, AttributeValue>();
        item["CustomerId"] = new AttributeValue
        {
            S = "alice@example.com"
        };
        item["OrderId"] = new AttributeValue
        {
            N = "2"
        };
        item["IsOpen"] = new AttributeValue
        {
            N = "1"
        };
        item["OrderCreationDate"] = new AttributeValue
        {
            N = "20130221"
        };
        item["ProductCategory"] = new AttributeValue
        {
            S = "Bike"
        };
        item["ProductName"] = new AttributeValue
        {
            S = "Super Mountain"
        };
        item["OrderStatus"] = new AttributeValue
        {
            S = "ORDER RECEIVED"
        };
```

```
            /* no ShipmentTrackingId attribute */
            putItemRequest = new PutItemRequest
            {
                TableName = tableName,
                Item = item,
                ReturnItemCollectionMetrics = "SIZE"
            };
            client.PutItem(putItemRequest);

            item = new Dictionary<string, AttributeValue>();
            item["CustomerId"] = new AttributeValue
            {
                S = "alice@example.com"
            };
            item["OrderId"] = new AttributeValue
            {
                N = "3"
            };
            /* no IsOpen attribute */
            item["OrderCreationDate"] = new AttributeValue
            {
                N = "20130304"
            };
            item["ProductCategory"] = new AttributeValue
            {
                S = "Music"
            };
            item["ProductName"] = new AttributeValue
            {
                S = "A Quiet Interlude"
            };
            item["OrderStatus"] = new AttributeValue
            {
                S = "IN TRANSIT"
            };
            item["ShipmentTrackingId"] = new AttributeValue
            {
                S = "176493"
            };
            putItemRequest = new PutItemRequest
            {
                TableName = tableName,
                Item = item,
                ReturnItemCollectionMetrics = "SIZE"
```

```
            };
            client.PutItem(putItemRequest);

            item = new Dictionary<string, AttributeValue>();
            item["CustomerId"] = new AttributeValue
            {
                S = "bob@example.com"
            };
            item["OrderId"] = new AttributeValue
            {
                N = "1"
            };
            /* no IsOpen attribute */
            item["OrderCreationDate"] = new AttributeValue
            {
                N = "20130111"
            };
            item["ProductCategory"] = new AttributeValue
            {
                S = "Movie"
            };
            item["ProductName"] = new AttributeValue
            {
                S = "Calm Before The Storm"
            };
            item["OrderStatus"] = new AttributeValue
            {
                S = "SHIPPING DELAY"
            };
            item["ShipmentTrackingId"] = new AttributeValue
            {
                S = "859323"
            };
            putItemRequest = new PutItemRequest
            {
                TableName = tableName,
                Item = item,
                ReturnItemCollectionMetrics = "SIZE"
            };
            client.PutItem(putItemRequest);

            item = new Dictionary<string, AttributeValue>();
            item["CustomerId"] = new AttributeValue
            {
```

```
            S = "bob@example.com"
        };
        item["OrderId"] = new AttributeValue
        {
            N = "2"
        };
        /* no IsOpen attribute */
        item["OrderCreationDate"] = new AttributeValue
        {
            N = "20130124"
        };
        item["ProductCategory"] = new AttributeValue
        {
            S = "Music"
        };
        item["ProductName"] = new AttributeValue
        {
            S = "E-Z Listening"
        };
        item["OrderStatus"] = new AttributeValue
        {
            S = "DELIVERED"
        };
        item["ShipmentTrackingId"] = new AttributeValue
        {
            S = "756943"
        };
        putItemRequest = new PutItemRequest
        {
            TableName = tableName,
            Item = item,
            ReturnItemCollectionMetrics = "SIZE"
        };
        client.PutItem(putItemRequest);

        item = new Dictionary<string, AttributeValue>();
        item["CustomerId"] = new AttributeValue
        {
            S = "bob@example.com"
        };
        item["OrderId"] = new AttributeValue
        {
            N = "3"
        };
```

```
            /* no IsOpen attribute */
            item["OrderCreationDate"] = new AttributeValue
            {
                N = "20130221"
            };
            item["ProductCategory"] = new AttributeValue
            {
                S = "Music"
            };
            item["ProductName"] = new AttributeValue
            {
                S = "Symphony 9"
            };
            item["OrderStatus"] = new AttributeValue
            {
                S = "DELIVERED"
            };
            item["ShipmentTrackingId"] = new AttributeValue
            {
                S = "645193"
            };
            putItemRequest = new PutItemRequest
            {
                TableName = tableName,
                Item = item,
                ReturnItemCollectionMetrics = "SIZE"
            };
            client.PutItem(putItemRequest);

            item = new Dictionary<string, AttributeValue>();
            item["CustomerId"] = new AttributeValue
            {
                S = "bob@example.com"
            };
            item["OrderId"] = new AttributeValue
            {
                N = "4"
            };
            item["IsOpen"] = new AttributeValue
            {
                N = "1"
            };
            item["OrderCreationDate"] = new AttributeValue
            {
```

```
            N = "20130222"
        };
        item["ProductCategory"] = new AttributeValue
        {
            S = "Hardware"
        };
        item["ProductName"] = new AttributeValue
        {
            S = "Extra Heavy Hammer"
        };
        item["OrderStatus"] = new AttributeValue
        {
            S = "PACKING ITEMS"
        };
        /* no ShipmentTrackingId attribute */
        putItemRequest = new PutItemRequest
        {
            TableName = tableName,
            Item = item,
            ReturnItemCollectionMetrics = "SIZE"
        };
        client.PutItem(putItemRequest);

        item = new Dictionary<string, AttributeValue>();
        item["CustomerId"] = new AttributeValue
        {
            S = "bob@example.com"
        };
        item["OrderId"] = new AttributeValue
        {
            N = "5"
        };
        /* no IsOpen attribute */
        item["OrderCreationDate"] = new AttributeValue
        {
            N = "20130309"
        };
        item["ProductCategory"] = new AttributeValue
        {
            S = "Book"
        };
        item["ProductName"] = new AttributeValue
        {
            S = "How To Cook"
```

```
            };
            item["OrderStatus"] = new AttributeValue
            {
                S = "IN TRANSIT"
            };
            item["ShipmentTrackingId"] = new AttributeValue
            {
                S = "440185"
            };
            putItemRequest = new PutItemRequest
            {
                TableName = tableName,
                Item = item,
                ReturnItemCollectionMetrics = "SIZE"
            };
            client.PutItem(putItemRequest);

            item = new Dictionary<string, AttributeValue>();
            item["CustomerId"] = new AttributeValue
            {
                S = "bob@example.com"
            };
            item["OrderId"] = new AttributeValue
            {
                N = "6"
            };
            /* no IsOpen attribute */
            item["OrderCreationDate"] = new AttributeValue
            {
                N = "20130318"
            };
            item["ProductCategory"] = new AttributeValue
            {
                S = "Luggage"
            };
            item["ProductName"] = new AttributeValue
            {
                S = "Really Big Suitcase"
            };
            item["OrderStatus"] = new AttributeValue
            {
                S = "DELIVERED"
            };
            item["ShipmentTrackingId"] = new AttributeValue
```

```
            {
                S = "893927"
            };
            putItemRequest = new PutItemRequest
            {
                TableName = tableName,
                Item = item,
                ReturnItemCollectionMetrics = "SIZE"
            };
            client.PutItem(putItemRequest);

            item = new Dictionary<string, AttributeValue>();
            item["CustomerId"] = new AttributeValue
            {
                S = "bob@example.com"
            };
            item["OrderId"] = new AttributeValue
            {
                N = "7"
            };
            /* no IsOpen attribute */
            item["OrderCreationDate"] = new AttributeValue
            {
                N = "20130324"
            };
            item["ProductCategory"] = new AttributeValue
            {
                S = "Golf"
            };
            item["ProductName"] = new AttributeValue
            {
                S = "PGA Pro II"
            };
            item["OrderStatus"] = new AttributeValue
            {
                S = "OUT FOR DELIVERY"
            };
            item["ShipmentTrackingId"] = new AttributeValue
            {
                S = "383283"
            };
            putItemRequest = new PutItemRequest
            {
                TableName = tableName,
```

```
            Item = item,
            ReturnItemCollectionMetrics = "SIZE"
        };
        client.PutItem(putItemRequest);
    }

    private static void WaitUntilTableReady(string tableName)
    {
        string status = null;
        // Let us wait until table is created. Call DescribeTable.
        do
        {
            System.Threading.Thread.Sleep(5000); // Wait 5 seconds.
            try
            {
                var res = client.DescribeTable(new DescribeTableRequest
                {
                    TableName = tableName
                });

                Console.WriteLine("Table name: {0}, status: {1}",
                          res.Table.TableName,
                          res.Table.TableStatus);
                status = res.Table.TableStatus;
            }
            catch (ResourceNotFoundException)
            {
                // DescribeTable is eventually consistent. So you might
                // get resource not found. So we handle the potential exception.
            }
        } while (status != "ACTIVE");
    }

    private static void WaitForTableToBeDeleted(string tableName)
    {
        bool tablePresent = true;

        while (tablePresent)
        {
            System.Threading.Thread.Sleep(5000); // Wait 5 seconds.
            try
            {
                var res = client.DescribeTable(new DescribeTableRequest
                {
```

```
                        TableName = tableName
                    });

                    Console.WriteLine("Table name: {0}, status: {1}",
                              res.Table.TableName,
                              res.Table.TableStatus);
                }
                catch (ResourceNotFoundException)
                {
                    tablePresent = false;
                }
            }
        }
    }
}
```

Arbeiten mit lokalen sekundären Indizes in der DynamoDB-AWS CLI

Sie können die AWS CLI zum Erstellen einer Amazon-DynamoDB-Tabelle mit einem oder mehreren lokalen sekundären Indizes, zum Beschreiben der Indizes in der Tabelle und zur Ausführung von Abfragen mithilfe des Indizes, verwenden.

Themen

- Erstellen einer Tabelle mit einem lokalen sekundären Index
- Beschreiben einer Tabelle mit einem lokalen sekundären Index
- Abfragen eines lokalen sekundären Indexes

Erstellen einer Tabelle mit einem lokalen sekundären Index

Lokale sekundäre Indizes müssen gleichzeitig mit dem Erstellen einer Tabelle erstellt werden. Zu diesem Zweck verwenden Sie den `create-table`-Parameter und geben Ihre Spezifikationen für ein oder mehrere lokale sekundäre Indizes an. Das folgende Beispiel erstellt eine Tabelle (`Music`), die Informationen über Songs in einer Musiksammlung enthält. Der Partitionsschlüssel ist `Artist` und der Sortierschlüssel `SongTitle`. Ein sekundärer Index, `AlbumTitleIndex` auf dem `AlbumTitle`-Attribut, erleichtert Abfragen nach Albumtitel.

```
aws dynamodb create-table \
    --table-name Music \
    --attribute-definitions AttributeName=Artist,AttributeType=S
 AttributeName=SongTitle,AttributeType=S \
```

```
        AttributeName=AlbumTitle,AttributeType=S  \
    --key-schema AttributeName=Artist,KeyType=HASH
 AttributeName=SongTitle,KeyType=RANGE \
    --provisioned-throughput \
        ReadCapacityUnits=10,WriteCapacityUnits=5 \
    --local-secondary-indexes \
        "[{\"IndexName\": \"AlbumTitleIndex\",
        \"KeySchema\":[{\"AttributeName\":\"Artist\",\"KeyType\":\"HASH\"},
                       {\"AttributeName\":\"AlbumTitle\",\"KeyType\":\"RANGE\"}],
        \"Projection\":{\"ProjectionType\":\"INCLUDE\",  \"NonKeyAttributes\":[\"Genre
\", \"Year\"]}}]"
```

Sie müssen warten bis DynamoDB die Tabelle erstellt und den Tabellenstatus auf `ACTIVE` setzt. Im Anschluss können Sie die Daten in der Tabelle ablegen. Mit describe-table können Sie den Status der Tabellenerstellung ermitteln.

Beschreiben einer Tabelle mit einem lokalen sekundären Index

Um Informationen zu lokalen sekundären Indizes in einer Tabelle zu erhalten, verwenden Sie den Parameter `describe-table`. Sie können auf den Namen, das Schlüsselschema und die projizierten Attribute von jedem Index zugreifen.

```
aws dynamodb describe-table --table-name Music
```

Abfragen eines lokalen sekundären Indexes

Sie können die Operation `query` für einen lokalen sekundären Index genauso nutzen, wie Sie `query` für eine Tabelle nutzen. Sie müssen den Indexnamen, die Abfragekriterien für den Indexsortierschlüssel und die Attribute angeben, die Sie zurückgeben möchten. In diesem Beispiel ist der Index `AlbumTitleIndex` und der Indexsortierschlüssel `AlbumTitle`.

Die einzigen zurückgegebenen Attribute sind die, die in den Index projiziert wurden. Sie könnten diese Abfrage ändern, um auch Nicht-Schlüsselattribute auszuwählen, aber dies würde eine Tabellenabrufaktivität erfordern, die relativ teuer ist. Weitere Informationen zum Abrufen von Tabellen finden Sie unter Attributprojektionen.

```
aws dynamodb query \
    --table-name Music \
    --index-name AlbumTitleIndex \
    --key-condition-expression "Artist = :v_artist and AlbumTitle = :v_title" \
```

```
    --expression-attribute-values  '{":v_artist":{"S":"Acme Band"},":v_title":
{"S":"Songs About Life"} }'
```

Verwalten komplexer Workflows mit DynamoDB-Transaktionen

Amazon DynamoDB-Transaktionen vereinfachen die Entwicklererfahrung, wenn sie koordinierte all-or-nothing Änderungen an mehreren Elementen sowohl innerhalb als auch tabellenübergreifend vornehmen müssen. Transaktionen ermöglichen in DynamoDB Atomarität, Konsistenz, Isolation und Haltbarkeit (ACID), wodurch Sie die Richtigkeit der Daten in Ihren Anwendungen einfacher verwalten können.

Sie können das transaktionale Lesen und Schreiben von DynamoDB verwenden, APIs um komplexe Geschäftsabläufe zu verwalten, bei denen mehrere Elemente in einem einzigen Vorgang hinzugefügt, aktualisiert oder gelöscht werden müssen. all-or-nothing Ein Entwickler von Videospielen kann so beispielsweise gewährleisten, dass die Profile der Spieler korrekt aktualisiert werden, wenn sie Elemente in einem Spiel austauschen oder Käufe aus einem Spiel heraus tätigen.

Mit der transaktionalen Schreib-API können Sie mehrere `Put`, `Update`-, `Delete`- und `ConditionCheck`-Aktionen gruppieren. Anschließend können Sie sie als eine einzige `TransactWriteItems`-Operation übermitteln, die entweder als Ganzes erfolgreich ist oder fehlschlägt. Dies gilt auch für mehrere `Get`-Aktionen, die Sie als einzige `TransactGetItems`-Operation gruppieren und übermitteln können.

Es fallen keine zusätzlichen Kosten für das Aktivieren von Transaktionen für Ihre DynamoDB-Tabellen an. Sie zahlen nur für Lese- oder Schreibvorgänge, die Teil Ihrer Transaktion sind. DynamoDB führt zwei zugrunde liegende Lese- oder Schreibvorgänge für jedes Element in der Transaktion aus: einen zum Vorbereiten der Transaktion und einen zum Festschreiben der Transaktion. Diese beiden zugrunde liegenden read/write Operationen sind in Ihren CloudWatch Amazon-Metriken sichtbar.

Um mit DynamoDB-Transaktionen zu beginnen, laden Sie das neueste AWS SDK oder das AWS Command Line Interface ()AWS CLI herunter. Befolgen Sie dann das Verfahren unter DynamoDB-Transaktionen-Beispiel.

Die folgenden Abschnitte bieten einen detaillierten Überblick über die Transaktion APIs und darüber, wie Sie sie in DynamoDB verwenden können.

Themen

- Amazon DynamoDB Transactions: Funktionsweise

- Verwenden von IAM mit DynamoDB-Transaktionen
- DynamoDB-Transaktionen-Beispiel

Amazon DynamoDB Transactions: Funktionsweise

Mit Amazon DynamoDB-Transaktionen können Sie mehrere Aktionen gruppieren und sie als einzelne all-or-nothing `TransactWriteItems` oder `TransactGetItems` Operation einreichen. Die folgenden Abschnitte beschreiben API-Produktionen, Kapazitätsverwaltung, bewährte Methoden und andere Details zur Verwendung von Transaktionsoperationen in DynamoDB.

Themen

- TransactWriteItems API
- TransactGetItems API
- Isolationsstufen für DynamoDB-Transaktionen
- Handhabung von Transaktionskonflikten in DynamoDB
- Verwenden von Transactional APIs in DynamoDB Accelerator (DAX)
- Kapazitätsverwaltung für Transaktionen
- Bewährte Methoden für Transaktionen
- Verwenden Sie transaktionale Tabellen APIs mit globalen Tabellen
- DynamoDB-Transaktionen im Vergleich zur AWSLabs Transactions-Clientbibliothek

TransactWriteItems API

`TransactWriteItems`ist eine synchrone und idempotente Schreiboperation, die bis zu 100 Schreibaktionen in einer einzigen Operation gruppiert. all-or-nothing Diese Aktionen können auf bis zu 100 verschiedene Elemente in einer oder mehreren DynamoDB-Tabellen innerhalb desselben AWS Kontos und in derselben Region abzielen. Die aggregierte Größe der Elemente in der Transaktion darf 4 MB nicht übersteigen. Die Aktionen werden atomarisch ausgeführt, d. h. entweder sind alle von ihnen oder keine von ihnen erfolgreich.

> **Note**
>
> - Eine `TransactWriteItems`-Operation unterscheidet sich darin von einer `BatchWriteItem`-Operation, dass alle darin enthaltenen Aktionen erfolgreich ausgeführt

werden müssen, damit irgendwelche Änderungen vorgenommen werden. Bei einer `BatchWriteItem`-Operation ist es dagegen möglich, dass nur einige der Aktionen im Stapel erfolgreich sind und andere fehlschlagen.

- Transaktionen können nicht mit Indizes ausgeführt werden.

Sie können nicht in derselben Transaktion mit mehreren Operationen auf das gleiche Element abzielen. Beispiel: In derselben Transaktion ist es nicht möglich eine `ConditionCheck`- sowie eine `Update`-Aktion für dasselbe Element auszuführen.

Sie können die folgenden Aktionstypen zu einer Transaktion hinzufügen:

- `Put` – Initiiert eine `PutItem`-Produktion, um bedingungsabhängig oder bedingungslos ein neues Element zu erstellen oder ein altes Element durch ein neues Element zu ersetzen.
- `Update` – Initiiert eine `UpdateItem`-Produktion, um die Attribute eines vorhandenen Elements zu bearbeiten oder ein neues Element zur Tabelle hinzuzufügen, sofern noch nicht vorhanden. Mit dieser Aktion können Sie Attribute für ein vorhandenes Element bedingungsabhängig oder bedingungslos hinzufügen, löschen oder aktualisieren.
- `Delete` – Initiiert eine `DeleteItem`-Produktion,, um ein einzelnes Element über seinen Primärschlüssel in einer Tabelle zu löschen.
- `ConditionCheck` – Überprüft, ob ein Element vorhanden ist, oder überprüft die Bedingung bestimmter Attribute des Elements.

Wenn eine Transaktion in DynamoDB abgeschlossen ist, werden ihre Änderungen an globale Sekundärindizes (GSIs), Streams und Backups weitergegeben. Diese Übertragung erfolgt schrittweise: Stream-Datensätze aus derselben Transaktion können zu unterschiedlichen Zeiten erscheinen und sich mit Datensätzen aus anderen Transaktionen überlappen. Stream-Konsumenten sollten nicht davon ausgehen, dass Transaktionen vollständig sind oder dass es sich um Bestellgarantien handelt.

Um eine atomare Momentaufnahme der in einer Transaktion geänderten Elemente sicherzustellen, verwenden Sie den TransactGetItems Vorgang, um alle relevanten Elemente zusammen zu lesen. Dieser Vorgang bietet eine konsistente Ansicht der Daten, sodass Sie entweder alle Änderungen einer abgeschlossenen Transaktion oder gar keine sehen.

Da die Weitergabe nicht unmittelbar erfolgt, kann eine Tabelle, wenn sie aus Backup (RestoreTableFromBackup) wiederhergestellt oder zu einem Zeitpunkt (ExportTableToPointInTime)

während der Weitergabe exportiert wird, nur einige der Änderungen enthalten, die während einer kürzlich durchgeführten Transaktion vorgenommen wurden.

Idempotenz

Sie können optional ein Client-Token einschließen, wenn Sie einen `TransactWriteItems`-Aufruf machen, um sicherzustellen, dass die Anforderung idempotent ist. Durch idempotente Transaktionen lassen sich Anwendungsfehler vermeiden, falls dieselbe Operation aufgrund einer Verbindungszeitüberschreitung oder sonstiger Konnektivitätsprobleme mehrmals übermittelt wird.

Wenn der ursprüngliche `TransactWriteItems`-Aufruf erfolgreich war, werden nachfolgende `TransactWriteItems`-Aufrufe mit demselben Client-Token als erfolgreich zurückgegeben, ohne Änderungen vorzunehmen. Wenn der Parameter `ReturnConsumedCapacity` eingestellt ist, gibt der erstmalige `TransactWriteItems`-Aufruf die Anzahl an Schreibkapazitätseinheiten zurück, die beim Vornehmen der Änderungen verbraucht wurden. Nachfolgende `TransactWriteItems`-Aufrufe mit demselben Client-Token geben die Anzahl der Lesekapazitätseinheiten zurück, die beim Lesen des Elements verbraucht wurden.

Wichtige Punkte bezüglich Idempotenz

- Ein Client-Token ist weitere 10 Minuten lang gültig, nachdem die Anforderung, die davon Gebrauch gemacht hat, beendet wurde. Nach 10 Minuten werden alle Anforderungen, die dasselbe Client-Token nutzen, als neue Anforderung angesehen. Deshalb sollten Sie dasselbe Client-Token nach 10 Minuten nicht erneut für dieselbe Anwendung verwenden.
- Wenn Sie eine Anforderung mit demselben Client-Token innerhalb des 10-minütigen Idempotenzfenster wiederholen, aber einen anderen Anforderungsparameter ändern, gibt DynamoDB die Ausnahme `IdempotentParameterMismatch` zurück.

Fehlerbehandlung beim Schreiben

Schreibtransaktionen schlagen unter den folgenden Umständen fehl:

- Wenn eine Bedingung in einem der Bedingungsausdrücke nicht erfüllt wird.
- Wenn ein Transaktionsvalidierungsfehler auftritt, da mehr als eine Aktion in derselben `TransactWriteItems`-Operation auf dasselbe Element abzielt.
- Wenn eine `TransactWriteItems`-Anforderung mit einer andauernden `TransactWriteItems`-Operation an mindestens einem Element in der `TransactWriteItems`-Anforderung in Konflikt

steht. In diesem Fall schlägt die Anfrage mit der Ausnahme `TransactionCanceledException` fehl.
- Wenn für die durchzuführende Transaktion nicht genügend Kapazität bereitgestellt wird.
- Wenn ein Element zu groß wird (größer als 400 KB) oder wenn ein lokaler sekundärer Index (LSI) zu groß wird oder wenn aufgrund der durch die Transaktion vorgenommenen Änderungen ein ähnlicher Validierungsfehler auftritt.
- Wenn ein Benutzerfehler, wie z. B. ein ungültiges Datenformat, auftritt.

Weitere Informationen zum Umgang mit Konflikten mit `TransactWriteItems`-Operationen finden Sie unter Handhabung von Transaktionskonflikten in DynamoDB.

TransactGetItems API

`TransactGetItems` ist ein synchroner Lesevorgang, der bis zu 100 `Get`-Aktionen zusammengruppiert. Diese Aktionen können auf bis zu 100 verschiedene Elemente in einer oder mehreren DynamoDB-Tabellen innerhalb desselben AWS Kontos und derselben Region abzielen. Die aggregierte Größe der Elemente in der Transaktion darf 4 MB nicht überschreiten.

Die `Get`-Aktionen werden atomarisch durchgeführt, d. h. entweder sind alle von ihnen oder keine von ihnen erfolgreich:

- `Get` – Initiiert eine `GetItem`-Operation, um einen Satz von Attributen für das Elemente mit dem angegebenen Primärschlüssel abzurufen. Wenn kein passendes Element gefunden wird, gibt `Get` keine Daten zurück.

Fehlerbehandlung beim Lesen

Lesetransaktionen schlagen unter den folgenden Umständen fehl:

- Wenn eine `TransactGetItems`-Anforderung mit einer andauernden `TransactWriteItems`-Operation an mindestens einem Element in der `TransactGetItems`-Anforderung in Konflikt steht. In diesem Fall schlägt die Anfrage mit der Ausnahme `TransactionCanceledException` fehl.
- Wenn für die durchzuführende Transaktion nicht genügend Kapazität bereitgestellt wird.
- Wenn ein Benutzerfehler, wie z. B. ein ungültiges Datenformat, auftritt.

Weitere Informationen zum Umgang mit Konflikten mit `TransactGetItems`-Operationen finden Sie unter Handhabung von Transaktionskonflikten in DynamoDB.

Isolationsstufen für DynamoDB-Transaktionen

Die Isolationsstufen von Transaktionsoperationen (`TransactWriteItems` oder `TransactGetItems`) und anderen Operationen sind folgende:

SERIALIZABLE

Durch die Isolationsstufe serializable wird sichergestellt, dass die Ergebnisse mehrerer gleichzeitiger Operationen die gleichen sind, als würde eine Operation erst beginnen, nachdem die vorherige Operation beendet wurde.

Die serialisierbare Isolation findet zwischen den folgenden Arten von Operationen statt:

- Zwischen Transaktionsoperationen und Standardschreibvorgängen (`PutItem`, `UpdateItem` oder `DeleteItem`).
- Zwischen Transaktionsoperationen und Standardlesevorgängen (`GetItem`).
- Zwischen einer `TransactWriteItems`-Operation und einer `TransactGetItems`-Operation.

Obwohl eine serialisierbare Isolation zwischen Transaktionsoperationen und einzelnen Standardschreibvorgängen in einer `BatchWriteItem`-Operation auftritt, findet keine serialisierbare Isolation zwischen der Transaktion und der `BatchWriteItem`-Operation als Einheit statt.

Dementsprechend ist die Isolationsstufe zwischen einer Transaktionsoperation und einzelnen `GetItems` in einer `BatchGetItem`-Operation serialisierbar. Die Isoloationsstufe zwischen der Transaktion und der `BatchGetItem`-Operation als Einheit ist aber read-committed.

Eine einzelne `GetItem`-Anforderung kann in Bezug auf eine `TransactWriteItems`-Anforderung auf eine von zwei Arten serialisiert werden, entweder vor oder nach der `TransactWriteItems`-Anforderung. Mehrere `GetItem`-Anforderungen, gegen Schlüssel in einem gleichzeitigen `TransactWriteItems`-Anfragen können in beliebiger Reihenfolge ausgeführt werden, und daher sind die Ergebnisse read-committed.

Wenn beispielsweise `GetItem`-Anforderungen für Element A und Element B gleichzeitig mit einer `TransactWriteItems`-Anforderung ausgeführt werden, die sowohl Element A als auch Element B ändert, gibt es vier Möglichkeiten:

- Beide `GetItem`-Anforderungen werden vor der `TransactWriteItems`-Anforderung ausgeführt.
- Beide `GetItem`-Anforderungen werden nach der `TransactWriteItems`-Anforderung ausgeführt.

- `GetItem`-Anforderung für Element A wird vor der `TransactWriteItems`-Anforderung ausgeführt. Für Element B wird die `GetItem` nach `TransactWriteItems` ausgeführt.
- `GetItem`-Anforderung für Element B wird vor der `TransactWriteItems`-Anforderung ausgeführt. Für Element A wird die `GetItem` nach `TransactWriteItems` ausgeführt.

Wenn die serialisierbare Isolationsstufe für mehrere `GetItem`-Anfragen bevorzugt wird, verwenden Sie `TransactGetItems`.

Wenn mehrere Elemente, die während der Übertragung Teil derselben Transaktionsschreibanfrage waren, nicht transaktionell gelesen werden, ist es möglich, dass Sie den neuen Status einiger Elemente und den alten Status der anderen Elemente lesen können. Sie können den neuen Status aller Elemente, die Teil der Transaktionsschreibanfrage waren, nur lesen, wenn eine erfolgreiche Antwort für den transaktionalen Schreibvorgang eingegangen ist, was darauf hinweist, dass die Transaktion abgeschlossen wurde.

Sobald die Transaktion erfolgreich abgeschlossen wurde und eine Antwort eingegangen ist, können nachfolgende, letztendlich konsistente Lesevorgänge aufgrund des Konsistenzmodells von DynamoDB für kurze Zeit immer noch den alten Status zurückgeben. Um sicherzustellen, dass unmittelbar nach einer Transaktion möglichst up-to-date viele Daten gelesen werden, sollten Sie Strongly Consistent Reads verwenden, indem Sie den Wert `ConsistentRead` auf true setzen.

READ-COMMITTED

Durch die Isolation Read-committed wird sichergestellt, dass Lesevorgänge immer festgeschriebene Werte für ein Element zurückgeben – der Lesevorgang wird niemals eine Sicht auf das Element präsentieren, die einen Zustand aus einem letztlich nicht erfolgreichen transaktionalen Schreibvorgang darstellt. Die Isolation "Read-committed" verhindert keine Änderungen am Element direkt nach dem Lesevorgang.

Die Isolationsstufe ist read-committed zwischen allen Transaktionsoperationen und allen Lesevorgängen, die mehrere Standard-Lesevorgänge (`BatchGetItem`, `Query` oder `Scan`) umfassen. Wenn bei einem transaktionalen Schreibvorgang ein Element mitten in einer `BatchGetItem`-, `Query`- oder `Scan`-Operation aktualisiert wird, gibt der nachfolgende Teil des Lesevorgangs den neu festgeschriebenen Wert zurück (mit `ConsistentRead`) oder möglicherweise einem zuvor festgeschriebenem Wert (letztendlich konsistente Lesevorgänge).

Zusammenfassung des Vorgangs

Die folgende Tabelle gibt einen Überblick über die Isolationsstufen zwischen einer Transaktionsoperation (`TransactWriteItems` oder `TransactGetItems`) und anderen Operationen.

Operation	Isolationsstufe
`DeleteItem`	Serializable
`PutItem`	Serializable
`UpdateItem`	Serializable
`GetItem`	Serializable
`BatchGetItem`	Read-committed*
`BatchWriteItem`	NICHT Serializable*
`Query`	Read-committed
`Scan`	Read-committed
Andere Transaktionsoperationen	Serializable

Mit einem Sternchen (*) markierte Stufen gelten für die Operation als Einheit. Einzelne Aktionen innerhalb dieser Operationen besitzen jedoch die Isolationsstufe serializable.

Handhabung von Transaktionskonflikten in DynamoDB

Bei Anforderungen auf Elementebene kann für ein Element in einer Transaktion ein Transaktionskonflikt auftreten. Transaktionskonflikte können in den folgenden Szenarien auftreten:

- Eine `PutItem`-, `UpdateItem`- oder `DeleteItem`-Anforderung für ein Element konfligiert mit einer laufenden `TransactWriteItems`-Anforderung, die dasselbe Element enthält.
- Ein Element in einer `TransactWriteItems`-Anforderung ist Teil einer anderen laufenden `TransactWriteItems`-Anforderung.

- Ein Element in einer `TransactGetItems`-Anforderung ist Teil einer laufenden `TransactWriteItems`-, `BatchWriteItem`-, `PutItem`-, `UpdateItem`- oder `DeleteItem`-Anforderung.

> **Note**
>
> - Wenn eine `PutItem`-, `UpdateItem`- oder `DeleteItem`-Anforderung abgelehnt wird, schlägt die Anforderung mit einer `TransactionConflictException` fehl.
> - Wenn irgendeine Anforderung auf Elementebene in `TransactWriteItems` oder `TransactGetItems` abgelehnt wird, schlägt die Anforderung mit einer `TransactionCanceledException` fehl. Wenn diese Anfrage fehlschlägt, AWS SDKs wiederholen Sie die Anfrage nicht.
>
>   Wenn Sie den verwenden AWS SDK für Java, enthält die Ausnahme die Liste der CancellationReasons, sortiert nach der Liste der Elemente im `TransactItems` Anforderungsparameter. Bei anderen Sprachen ist in der Fehlermeldung der Ausnahme eine Zeichenfolgendarstellung der Liste enthalten.
> - Wenn eine laufende `TransactWriteItems`- oder `TransactGetItems`-Operation mit einer gleichzeitigen `GetItem`-Anforderung im Konflikt steht, können beide Operationen erfolgreich durchgeführt werden.

Die TransactionConflict CloudWatch Metrik wird für jede fehlgeschlagene Anfrage auf Elementebene inkrementiert.

Verwenden von Transactional APIs in DynamoDB Accelerator (DAX)

`TransactWriteItems` und `TransactGetItems` werden in DynamoDB Accelerator (DAX) mit den gleichen Isolierungsstufen wie in DynamoDB unterstützt.

`TransactWriteItems` schreibt über DAX. DAX übergibt eineN `TransactWriteItems`-Aufruf an DynamoDB und gibt die Antwort zurück. Um den Cache nach dem Schreiben aufzufüllen, ruft DAX `TransactGetItems` im Hintergrund für jedes Element im `TransactWriteItems`-Vorgang auf, wodurch zusätzliche Lesekapazitätseinheiten verbraucht werden. (Weitere Informationen finden Sie unter Kapazitätsverwaltung für Transaktionen.) Diese Funktion ermöglicht es, dass Ihre Anwendungslogik einfach bleibt. Außerdem können Sie so DAX für Transaktionsoperationen sowie für nicht transaktionale Operationen verwenden.

`TransactGetItems`-Aufrufe werden über DAX übergeben, ohne dass Elemente lokal zwischengespeichert werden. Dies ist dasselbe Verhalten wie beim stark konsistenten Lesen in DAX. APIs

Kapazitätsverwaltung für Transaktionen

Es fallen keine zusätzlichen Kosten für das Aktivieren von Transaktionen für Ihre DynamoDB-Tabellen an. Sie zahlen nur für Lese- oder Schreibvorgänge, die Teil Ihrer Transaktion sind. DynamoDB führt zwei zugrunde liegende Lese- oder Schreibvorgänge für jedes Element in der Transaktion aus: einen zum Vorbereiten der Transaktion und einen zum Festschreiben der Transaktion. Die beiden zugrunde liegenden read/write Operationen sind in Ihren CloudWatch Amazon-Metriken sichtbar.

Planen Sie die zusätzlichen Lese- und Schreibvorgänge ein, die für Transaktionen erforderlich sind APIs , wenn Sie Kapazität für Ihre Tabellen bereitstellen. Angenommen, Ihre Anwendung führt eine Transaktion pro Sekunde aus und jede Transaktion schreibt drei 500-Byte-Elemente in Ihre Tabelle. Für jedes Element sind zwei Schreibkapazitätseinheiten (WCUs) erforderlich: eine für die Vorbereitung der Transaktion und eine für den Commit der Transaktion. Daher müssten Sie sechs für WCUs die Tabelle bereitstellen.

Wenn Sie im vorherigen Beispiel DynamoDB Accelerator (DAX) verwenden würden, würden Sie auch zwei Lesekapazitätseinheiten (RCUs) für jedes Element im `TransactWriteItems` Aufruf verwenden. Sie müssten also sechs weitere RCUs für die Tabelle bereitstellen.

Wenn Ihre Anwendung eine Lesetransaktion pro Sekunde ausführt und jede Transaktion drei 500-Byte-Elemente in Ihrer Tabelle liest, müssten Sie der Tabelle entsprechend sechs Lesekapazitätseinheiten (RCUs) bereitstellen. Zum Lesen jedes Elements sind zwei erforderlich RCUs: eine für die Vorbereitung der Transaktion und eine für die Ausführung der Transaktion.

Außerdem ist das SDK-Standardverhalten, Transaktionen im Falle der Ausnahme `TransactionInProgressException` wiederholt zu versuchen. Planen Sie die zusätzlichen Lesekapazitätseinheiten (RCUs) ein, die diese Wiederholungen verbrauchen. Dies gilt auch, wenn Sie Transaktionen in Ihrem eigenen Code mit einem `ClientRequestToken` wiederholt versuchen.

Bewährte Methoden für Transaktionen

Erwägen Sie bei der Verwendung von DynamoDB-Transaktionen die folgenden empfohlenen Methoden.

- Aktivieren Sie für Ihre Tabellen das Auto Scaling oder stellen Sie sicher, dass Sie die von Ihnen bereitgestellte Durchsatzkapazität zum Ausführen der beiden Lese- oder Schreibvorgänge für jedes Element in der Transaktion ausreicht.
- Wenn Sie kein AWS bereitgestelltes SDK verwenden, fügen Sie bei einem `TransactWriteItems` Aufruf ein `ClientRequestToken` Attribut hinzu, um sicherzustellen, dass die Anfrage idempotent ist.
- Gruppieren Sie Operationen nur dann als eine Transaktion zusammen, wenn dies wirklich erforderlich ist. Beispiel: Wenn eine einzelne Transaktion mit 10 Operationen in mehrere Transaktionen aufgeteilt werden kann, ohne die korrekte Funktionsweise der Anwendung zu gefährden, wird zur Aufteilung der Transaktion geraten. Weniger komplexe Transaktionen verbessern den Durchsatz und sind mit größerer Wahrscheinlichkeit erfolgreich.
- Wenn mehrere Transaktionen dieselben Elemente gleichzeitig aktualisieren, können Konflikte entstehen, die zum Abbruch der Transaktionen führen. Wir empfehlen die folgenden bewährten DynamoDB-Methoden für die Datenmodellierung, um solche Konflikte zu minimieren.
- Wenn ein Satz von Attributen häufig im Rahmen einer einzelnen Transaktion über mehrere Elemente hinweg aktualisiert wird, empfiehlt es sich, die Attribute in einem einzigen Element zusammenzugruppieren, um den Umfang der Transaktion zu reduzieren.
- Vermeiden Sie es, Transaktionen zur massenweisen Aufnahme von Daten zu verwenden. `BatchWriteItem` eignet sich besser für Massenschreibvorgänge.

Verwenden Sie transaktionale Tabellen APIs mit globalen Tabellen

Transaktionsoperationen bieten Garantien für Atomizität, Konsistenz, Isolation und Haltbarkeit (ACID) nur innerhalb der AWS Region, in der die Schreib-API aufgerufen wurde. Regionsübergreifende Transaktionen werden in globalen Tabellen nicht unterstützt. Angenommen, Sie haben eine globale Tabelle mit Replikaten in den Regionen USA Ost (Ohio) und USA West (Oregon) und Sie führen einen `TransactWriteItems`-Vorgang in der Region USA Ost (Nord-Virginia) aus. Dann sind möglicherweise teilweise abgeschlossene Transaktionen in der Region USA West (Oregon) zu beobachten, während Änderungen repliziert werden. Die Änderungen werden erst dann in die anderen Regionen repliziert, nachdem sie in der Quellregion in die Datenbank eingetragen wurden.

DynamoDB-Transaktionen im Vergleich zur AWSLabs Transactions-Clientbibliothek

DynamoDB-Transaktionen bieten einen kostengünstigeren, robusteren und leistungsfähigeren Ersatz für die AWSLabsTransactions-Clientbibliothek. Wir empfehlen Ihnen, Ihre Anwendungen so zu aktualisieren, dass sie die native, serverseitige Transaktion verwenden. APIs

Verwenden von IAM mit DynamoDB-Transaktionen

Sie können AWS Identity and Access Management (IAM) verwenden, um die Aktionen einzuschränken, die Transaktionsvorgänge in Amazon DynamoDB ausführen können. Weitere Informationen zur Verwendung der IAM-Richtlinien in DynamoDB finden Sie unter Identitätsbasierte Richtlinien für DynamoDB.

Die Berechtigungen für die Aktionen `Put`, `Update`, `Delete` und `Get` werden durch die Berechtigungen geregelt, die für die zugrundeliegenden Operationen `PutItem`, `UpdateItem`, `DeleteItem` und `GetItem` verwendet werden. Für die Aktion `ConditionCheck` können Sie die Berechtigung `dynamodb:ConditionCheckItem` in den IAM-Richtlinien verwenden.

Es folgen Beispiele für die IAM-Richtlinien, die Sie zum Konfigurieren der DynamoDB-Transaktionen verwenden können.

Beispiel 1: Zulassen von Transaktionsoperationen

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "dynamodb:ConditionCheckItem",
                "dynamodb:PutItem",
                "dynamodb:UpdateItem",
                "dynamodb:DeleteItem",
                "dynamodb:GetItem"
            ],
            "Resource": [
                "arn:aws:dynamodb:*:*:table/table04"
            ]
        }
    ]
}
```

Beispiel 2: Zulassen nur von Transaktionsoperationen

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "dynamodb:ConditionCheckItem",
                "dynamodb:PutItem",
                "dynamodb:UpdateItem",
                "dynamodb:DeleteItem",
                "dynamodb:GetItem"
            ],
            "Resource": [
                "arn:aws:dynamodb:*:*:table/table04"
            ],
            "Condition": {
                "ForAnyValue:StringEquals": {
                    "dynamodb:EnclosingOperation": [
                        "TransactWriteItems",
                        "TransactGetItems"
                    ]
                }
            }
        }
    ]
}
```

Beispiel 3: Zulassen von nicht-transaktionalen Lese- und Schreibvorgängen und Blockieren von transaktionalen Lese- und Schreibvorgängen

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
```

```
            "Effect": "Deny",
            "Action": [
                "dynamodb:ConditionCheckItem",
                "dynamodb:PutItem",
                "dynamodb:UpdateItem",
                "dynamodb:DeleteItem",
                "dynamodb:GetItem"
            ],
            "Resource": [
                "arn:aws:dynamodb:*:*:table/table04"
            ],
            "Condition": {
                "ForAnyValue:StringEquals": {
                    "dynamodb:EnclosingOperation": [
                        "TransactWriteItems",
                        "TransactGetItems"
                    ]
                }
            }
        },
        {
            "Effect": "Allow",
             "Action": [
                 "dynamodb:PutItem",
                 "dynamodb:DeleteItem",
                 "dynamodb:GetItem",
                 "dynamodb:UpdateItem"
             ],
             "Resource": [
                 "arn:aws:dynamodb:*:*:table/table04"
             ]
         }
    ]
}
```

Beispiel 4: Verhindern Sie, dass Informationen bei einem Fehler zurückgegeben werden ConditionCheck

JSON

```
{
```

```
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "dynamodb:ConditionCheckItem",
                "dynamodb:PutItem",
                "dynamodb:UpdateItem",
                "dynamodb:DeleteItem",
                "dynamodb:GetItem"
            ],
            "Resource": "arn:aws:dynamodb:*:*:table/table01",
            "Condition": {
                "StringEqualsIfExists": {
                    "dynamodb:ReturnValues": "NONE"
                }
            }
        }
    ]
}
```

DynamoDB-Transaktionen-Beispiel

Als Beispiel für eine Situation, in der Amazon DynamoDB Transactions nützlich sein können, betrachten Sie diese Java-Beispielanwendung für eine Online-Marketplace-Site.

Die Anwendung verfügt über drei DynamoDB-Tabellen im Backend:

- `Customers` – In dieser Tabelle werden Details zu den Marketplace-Kunden gespeichert. Der Primärschlüssel ist ein `CustomerId` eine eindeutige ID.
- `ProductCatalog` – In dieser Tabelle werden Details wie Preis und Verfügbarkeit zu den Produkten gespeichert, die auf der Marketplace-Site zum Verkauf angeboten werden. Der Primärschlüssel ist ein `ProductId` eine eindeutige ID.
- `Orders` – In dieser Tabelle werden Details zu Bestellungen von der Marketplace-Site gespeichert. Der Primärschlüssel ist ein `OrderId` eine eindeutige ID.

Bestellung

Die folgenden Codeausschnitte veranschaulichen, wie DynamoDB-Transaktionen verwendet werden, um die verschiedenen Schritte zu koordinieren, die zum Erstellen und Verarbeiten eines Auftrags erforderlich sind. Durch die Verwendung eines einzigen all-or-nothing Vorgangs wird sichergestellt, dass, falls ein Teil der Transaktion fehlschlägt, keine Aktionen in der Transaktion ausgeführt und keine Änderungen vorgenommen werden.

In diesem Beispiel richten Sie einen Auftrag eines Kunden ein, dessen `customerId` `09e8e9c8-ec48` ist. Anschließend führen Sie sie als einzelne Transaktion aus, indem Sie den folgenden einfachen Workflow für die Auftragsverarbeitung verwenden:

1. Stellen Sie sicher, dass die Kundennummer gültig ist.
2. Stellen Sie sicher, dass das Produkt `IN_STOCK` ist und aktualisieren Sie den Produktstatus auf `SOLD`.
3. Stellen Sie sicher, dass die Bestellung noch nicht vorhanden ist, und erstellen Sie den Auftrag.

Kunden validieren

Definieren Sie zunächst eine Aktion, um zu überprüfen, ob ein Kunde mit `customerId` gleich `09e8e9c8-ec48` in der Kundentabelle vorhanden ist.

```
final String CUSTOMER_TABLE_NAME = "Customers";
final String CUSTOMER_PARTITION_KEY = "CustomerId";
final String customerId = "09e8e9c8-ec48";
final HashMap<String, AttributeValue> customerItemKey = new HashMap<>();
customerItemKey.put(CUSTOMER_PARTITION_KEY, new AttributeValue(customerId));

ConditionCheck checkCustomerValid = new ConditionCheck()
    .withTableName(CUSTOMER_TABLE_NAME)
    .withKey(customerItemKey)
    .withConditionExpression("attribute_exists(" + CUSTOMER_PARTITION_KEY + ")");
```

Aktualisieren des Produktstatus

Definieren Sie als Nächstes eine Aktion, um den Produktstatus auf `SOLD` zu aktualisieren, wenn die Bedingung, auf die der Produktstatus als `IN_STOCK`derzeit festgelegt ist, `true` ist. Das Festlegen des `ReturnValuesOnConditionCheckFailure`-Parameters gibt den Artikel zurück, wenn das Produktstatusattribut des Artikels nicht gleich `IN_STOCK` war.

```
final String PRODUCT_TABLE_NAME = "ProductCatalog";
final String PRODUCT_PARTITION_KEY = "ProductId";
HashMap<String, AttributeValue> productItemKey = new HashMap<>();
productItemKey.put(PRODUCT_PARTITION_KEY, new AttributeValue(productKey));

Map<String, AttributeValue> expressionAttributeValues = new HashMap<>();
expressionAttributeValues.put(":new_status", new AttributeValue("SOLD"));
expressionAttributeValues.put(":expected_status", new AttributeValue("IN_STOCK"));

Update markItemSold = new Update()
    .withTableName(PRODUCT_TABLE_NAME)
    .withKey(productItemKey)
    .withUpdateExpression("SET ProductStatus = :new_status")
    .withExpressionAttributeValues(expressionAttributeValues)
    .withConditionExpression("ProductStatus = :expected_status")

 .withReturnValuesOnConditionCheckFailure(ReturnValuesOnConditionCheckFailure.ALL_OLD);
```

Bestellung erstellen

Schließlich erstellen Sie die Bestellung, solange eine Bestellung mit diesem `OrderId` nicht bereits existiert.

```
final String ORDER_PARTITION_KEY = "OrderId";
final String ORDER_TABLE_NAME = "Orders";

HashMap<String, AttributeValue> orderItem = new HashMap<>();
orderItem.put(ORDER_PARTITION_KEY, new AttributeValue(orderId));
orderItem.put(PRODUCT_PARTITION_KEY, new AttributeValue(productKey));
orderItem.put(CUSTOMER_PARTITION_KEY, new AttributeValue(customerId));
orderItem.put("OrderStatus", new AttributeValue("CONFIRMED"));
orderItem.put("OrderTotal", new AttributeValue("100"));

Put createOrder = new Put()
    .withTableName(ORDER_TABLE_NAME)
    .withItem(orderItem)

 .withReturnValuesOnConditionCheckFailure(ReturnValuesOnConditionCheckFailure.ALL_OLD)
    .withConditionExpression("attribute_not_exists(" + ORDER_PARTITION_KEY + ")");
```

Ausführen der Transaktion

Das folgende Beispiel zeigt, wie die zuvor als einzelne all-or-nothing Operation definierten Aktionen ausgeführt werden.

```
    Collection<TransactWriteItem> actions = Arrays.asList(
        new TransactWriteItem().withConditionCheck(checkCustomerValid),
        new TransactWriteItem().withUpdate(markItemSold),
        new TransactWriteItem().withPut(createOrder));

    TransactWriteItemsRequest placeOrderTransaction = new TransactWriteItemsRequest()
        .withTransactItems(actions)
        .withReturnConsumedCapacity(ReturnConsumedCapacity.TOTAL);

    // Run the transaction and process the result.
    try {
        client.transactWriteItems(placeOrderTransaction);
        System.out.println("Transaction Successful");

    } catch (ResourceNotFoundException rnf) {
        System.err.println("One of the table involved in the transaction is not found"
 + rnf.getMessage());
    } catch (InternalServerErrorException ise) {
        System.err.println("Internal Server Error" + ise.getMessage());
    } catch (TransactionCanceledException tce) {
        System.out.println("Transaction Canceled " + tce.getMessage());
    }
```

Lesen der Bestelldetails

Das folgende Beispiel zeigt, wie der abgeschlossene Auftrag transaktional über die `Orders`- und `ProductCatalog`-Tabellen gelesen wird.

```
HashMap<String, AttributeValue> productItemKey = new HashMap<>();
productItemKey.put(PRODUCT_PARTITION_KEY, new AttributeValue(productKey));

HashMap<String, AttributeValue> orderKey = new HashMap<>();
orderKey.put(ORDER_PARTITION_KEY, new AttributeValue(orderId));

Get readProductSold = new Get()
    .withTableName(PRODUCT_TABLE_NAME)
    .withKey(productItemKey);
```

```
Get readCreatedOrder = new Get()
    .withTableName(ORDER_TABLE_NAME)
    .withKey(orderKey);

Collection<TransactGetItem> getActions = Arrays.asList(
    new TransactGetItem().withGet(readProductSold),
    new TransactGetItem().withGet(readCreatedOrder));

TransactGetItemsRequest readCompletedOrder = new TransactGetItemsRequest()
    .withTransactItems(getActions)
    .withReturnConsumedCapacity(ReturnConsumedCapacity.TOTAL);

// Run the transaction and process the result.
try {
    TransactGetItemsResult result = client.transactGetItems(readCompletedOrder);
    System.out.println(result.getResponses());
} catch (ResourceNotFoundException rnf) {
    System.err.println("One of the table involved in the transaction is not found" +
 rnf.getMessage());
} catch (InternalServerErrorException ise) {
    System.err.println("Internal Server Error" + ise.getMessage());
} catch (TransactionCanceledException tce) {
    System.err.println("Transaction Canceled" + tce.getMessage());
}
```

Ändern der Datenerfassung mit Amazon DynamoDB

Viele Anwendungen können zum Zeitpunkt einer Änderung von der Funktion zum Erfassen der Änderungen an den in einer DynamoDB-Tabelle gespeicherten Elementen profitieren. Im Folgenden sehen Sie einige Beispielanwendungsfälle.

- Eine beliebte mobile App modifiziert Daten in einer DynamoDB-Tabelle mit einer Geschwindigkeit von Tausenden von Aktualisierungen pro Sekunde. Eine andere Anwendung erfasst und speichert Daten über diese Updates und stellt near-real-time Nutzungsmetriken für die mobile App bereit.
- Eine Finanzanwendung ändert Börsendaten in einer DynamoDB-Tabelle. Verschiedene Anwendungen, die parallel laufen, verfolgen diese Änderungen in Echtzeit value-at-risk, berechnen Portfolios und gleichen sie automatisch auf der Grundlage von Aktienkursbewegungen neu aus.
- Sensoren in Transportfahrzeugen und Industrieanlagen senden Daten an eine DynamoDB-Tabelle. Verschiedene Anwendungen überwachen die Leistung und senden Messaging-Warnungen, wenn ein Problem erkannt wird, prognostizieren potenzielle Fehler durch Anwendung von Algorithmen

für Machine Learning und komprimieren und archivieren Daten in Amazon Simple Storage Service (Amazon S3).

- Eine Anwendung sendet Benachrichtigungen automatisch an die Mobilgeräte aller Freunde in einer Gruppe, sobald ein Freund ein neues Bild hochlädt.
- Ein neuer Kunde fügt einer DynamoDB-Tabelle Daten hinzu. Dieses Ereignis ruft eine andere Anwendung auf, die eine Begrüßungs-E-Mail an den neuen Kunden sendet.

DynamoDB unterstützt das Streaming von Datensätzen für Änderungsdaten auf Elementebene in nahezu Echtzeit. Sie können Anwendungen erstellen, die diese Streams nutzen, und basierend auf dem Inhalt Maßnahmen ergreifen.

> **Note**
>
> Das Hinzufügen von Tags zu DynamoDB Streams und das Verwenden der attributbasierte Zugriffskontrolle (ABAC) bei DynamoDB Streams werden nicht unterstützt.

Das folgende Video gibt Ihnen einen einführenden Einblick in das „Change Data Capture“-Konzept.

Tabellen-Kapazitätsmodi

Themen

- Streaming-Optionen für die Erfassung der Änderung von Daten
- Verwenden von Kinesis Data Streams zum Erfassen von Änderungen an DynamoDB
- Ändern Sie die Datenerfassung für DynamoDB Streams

Streaming-Optionen für die Erfassung der Änderung von Daten

DynamoDB bietet zwei Streaming-Modelle für die Erfassung der Änderung von Daten: Kinesis Data Streams für DynamoDB und DynamoDB Streams.

Um Ihnen bei der Auswahl der richtigen Lösung für Ihre Anwendung zu helfen, werden die Funktionen der einzelnen Streaming-Modelle in der folgenden Tabelle zusammengefasst.

Eigenschaften	Kinesis Data Streams für DynamoDB	DynamoDB Streams
Datenaufbewahrung	Bis zu 1 Jahr.	24 Stunden.
Unterstützung für Kinesis Client Library (KCL)	Unterstützt KCL Versionen 1.X, 2.X und 3.X.	Unterstützt KCL Versionen 1.X und 2.X.
Anzahl der Konsumenten	Bis zu 5 gleichzeitige Verbraucher pro Shard oder bis zu 20 gleichzeitige Verbraucher pro Shard mit erweitertem Fan-Out.	Bis zu 2 gleichzeitige Verbraucher pro Shard.
Durchsatzkontingente	Unbegrenzt.	Vorbehaltlich der Durchsatz quoten nach DynamoDB-Tabelle und AWS Region.
Datensatzbereitstellungsmodell	Pull-Modell über HTTP GetRecordsund mit erweitertem Fan-Out überträgt Kinesis Data Streams die Datensätze über HTTP/2 mithilfe von. SubscribeToShard	Pull-Modell über HTTP mithilfe von. GetRecords
Bestellung von Datensätzen	Das Zeitstempel-Attribut für jeden Streamdatensatz kann verwendet werden, um die tatsächliche Reihenfolge zu identifizieren, in der Änderungen in der DynamoDB-Tabelle aufgetreten sind.	Für jedes Element, das in einer DynamoDB-Tabelle geändert wird, erscheinen die Stream-Datensätze in der gleichen Reihenfolge wie die tatsächlichen Änderungen des Elements.
Doppelte Datensätze	Gelegentlich werden doppelte Datensätze im Stream angezeigt.	Es werden keine doppelten Datensätze im Stream angezeigt.

Eigenschaften	Kinesis Data Streams für DynamoDB	DynamoDB Streams
Stream-Verarbeitungsoptionen	Verarbeiten Sie Stream-Datensätze mit AWS Lambda, Amazon Managed Service für Apache Flink, Kinesis Data Firehose oder AWS Glue Streaming-ETL.	Stream-Datensätze mithilfe von AWS Lambda oder Kinesis-Adapter für DynamoDB Streams verarbeiten.
Zuverlässigkeitsstufe	Availability Zones für automatisches Failover ohne Unterbrechung.	Availability Zones für automatisches Failover ohne Unterbrechung.

Sie können beide Streaming-Modelle in derselben DynamoDB-Tabelle aktivieren.

Im folgenden Video wird mehr über die Unterschiede zwischen den beiden Optionen gesprochen.

DynamoDB Streams verglichen mit Kinesis Data Streams

Verwenden von Kinesis Data Streams zum Erfassen von Änderungen an DynamoDB

Sie können Amazon Kinesis Data Streams verwenden, um Änderungen an Amazon DynamoDB zu erfassen.

Kinesis Data Streams erfasst Änderungen auf Elementebene in jeder DynamoDB-Tabelle und repliziert sie in einen Kinesis Data Stream. Ihre Anwendungen können auf diesen Stream zugreifen und die Änderungen auf Elementebene nahezu in Echtzeit anzeigen. Sie können kontinuierlich Terabyte an Daten pro Stunde erfassen und speichern. Sie können eine längere Datenaufbewahrungszeit nutzen, und mit der Funktion für erweiterte Rundsendungen können Sie gleichzeitig zwei oder mehr nachgelagerte Anwendungen erreichen. Weitere Vorteile sind zusätzliche Prüfungen und Sicherheitstransparenz.

Mit Kinesis Data Streams haben Sie auch Zugriff auf Amazon Data Firehose und Amazon Managed Service für Apache Flink. Diese Services können Sie bei der Erstellung von Anwendungen unterstützen, die Echtzeit-Dashboards betreiben, Warnungen generieren, dynamische

Preisgestaltung und Werbung implementieren und ausgefeilte Datenanalysen sowie Algorithmen für maschinelles Lernen umsetzen.

> **Note**
>
> Die Verwendung von Kinesis-Datenströmen für DynamoDB unterliegt sowohl den Kinesis Data Streams-Preisen für den Datenstream als auch den DynamoDB-Preisen für die Quelltabelle.

Informationen zum Aktivieren von Kinesis-Streaming in einer DynamoDB-Tabelle mithilfe der Konsole oder des Java-SDK finden Sie unter. AWS CLIErste Schritte mit Kinesis Data Streams für Amazon DynamoDB

Themen

- Wie Kinesis Data Streams mit DynamoDB funktioniert
- Erste Schritte mit Kinesis Data Streams für Amazon DynamoDB
- Verwenden von Shards und Metriken mit DynamoDB Streams und Kinesis Data Streams
- Verwendung von IAM-Richtlinien für Amazon Kinesis Data Streams und Amazon DynamoDB

Wie Kinesis Data Streams mit DynamoDB funktioniert

Wenn ein Kinesis Data Stream für eine DynamoDB-Tabelle aktiviert ist, sendet die Tabelle einen Datensatz, der alle Änderungen an den Daten dieser Tabelle erfasst. Dieser Datensatz beinhaltet:

- Die spezifische Uhrzeit, zu der ein Element kürzlich erstellt, aktualisiert oder gelöscht wurde
- den Primärschlüssel dieses Elements
- einen Snapshot des Datensatzes vor der Änderung
- einen Snapshot des Datensatzes nach der Änderung

Diese Datensätze werden nahezu in Echtzeit erfasst und veröffentlicht. Nachdem sie in den Kinesis Data Stream geschrieben wurden, können sie wie jeder andere Datensatz gelesen werden. Sie können die Kinesis Client Library verwenden AWS Lambda, die Kinesis Data Streams API verwenden, aufrufen und andere verbundene Dienste verwenden. Weitere Informationen finden Sie unter Lesen von Daten aus Amazon Kinesis Data Streams im Amazon-Kinesis-Data-Streams-Entwicklerhandbuch.

Diese Änderungen an Daten werden ebenfalls asynchron erfasst. Kinesis hat keine Auswirkungen auf die Leistung einer Tabelle, von der es streamt. Die in Ihrem Kinesis Data Stream gespeicherten Streamdatensätze werden auch im Ruhezustand verschlüsselt. Weitere Informationen finden Sie unter Datenschutz in Amazon Kinesis Data Streams.

Die Kinesis-Data-Stream-Datensätze werden möglicherweise in einer anderen Reihenfolge angezeigt als vor den Elementänderungen. Dieselben Elementbenachrichtigungen werden möglicherweise auch mehrmals im Stream angezeigt. Sie können das Attribut `ApproximateCreationDateTime` überprüfen, um die Reihenfolge zu ermitteln, in der die Elementänderungen aufgetreten sind, und um doppelte Datensätze zu identifizieren.

Wenn Sie einen Kinesis Data Stream als Streaming-Ziel einer DynamoDB-Tabelle aktivieren, können Sie die Genauigkeit der `ApproximateCreationDateTime`-Werte entweder in Millisekunden oder Mikrosekunden konfigurieren. Standardmäßig gibt `ApproximateCreationDateTime` den Zeitpunkt der Änderung in Millisekunden an. Darüber hinaus können Sie diesen Wert für ein aktives Streaming-Ziel ändern. Nach einer solchen Aktualisierung weisen in Kinesis geschriebene Stream-Datensätze `ApproximateCreationDateTime`-Werte mit der gewünschten Genauigkeit auf.

In DynamoDB geschriebene Binärwerte müssen im Base64-codierten Format codiert sein. Wenn Datensätze jedoch in einen Kinesis-Datenstrom geschrieben werden, erfolgt eine zweite Codierung dieser codierten Binärwerte mit der Base64-Codierung. Beim Lesen dieser Datensätze aus einem Kinesis-Datenstrom müssen Anwendungen diese Werte zweimal dekodieren, um die rohen Binärwerte abzurufen.

DynamoDB erhebt Gebühren für die Verwendung von Kinesis Data Streams in Änderungsdatenerfassungseinheiten. 1 KB Änderung pro Einzelelement gilt als eine Änderungsdatenerfassungseinheit. Die KB der Änderung in jedem Element wird durch das Größere der „Vorher“- und „Nachher“-Images des in den Stream geschriebenen Elements berechnet, wobei dieselbe Logik wie capacity unit consumption for write operations (Verbrauch der Kapazitätseinheit für Schreibvorgänge) verwendet wird. Sie müssen keinen Kapazitätsdurchsatz für Änderungsdatenerfassungseinheiten bereitstellen, ähnlich wie der DynamoDB on-demand-Modus funktioniert.

Aktivieren eines Kinesis Data Streams für Ihre DynamoDB-Tabelle

Sie können das Streaming zu Kinesis von Ihrer vorhandenen DynamoDB-Tabelle aus aktivieren oder deaktivieren, indem Sie das AWS-Managementkonsole, das AWS SDK oder das AWS Command Line Interface () verwenden.AWS CLI

- Sie können nur Daten von DynamoDB zu Kinesis Data Streams in demselben AWS Konto und derselben AWS Region streamen wie Ihre Tabelle.
- Sie können nur Daten von einer DynamoDB-Tabelle in einen Kinesis Data Stream streamen.

Vornehmen von Änderungen an einem Kinesis-Data-Streams-Ziel in Ihrer DynamoDB-Tabelle

Standardmäßig enthalten alle Kinesis-Data-Stream-Datensätze ein `ApproximateCreationDateTime`-Attribut. Dieses Attribut stellt einen Zeitstempel in Millisekunden der ungefähren Zeit dar, zu der jeder Datensatz erstellt wurde. Sie können die Genauigkeit dieser Werte ändern, indem Sie https://console.aws.amazon.com/kinesis, das SDK oder das AWS CLI

Erste Schritte mit Kinesis Data Streams für Amazon DynamoDB

In diesem Abschnitt wird beschrieben, wie Kinesis Data Streams für Amazon DynamoDB-Tabellen mit der Amazon DynamoDB DynamoDB-Konsole, der AWS Command Line Interface (AWS CLI) und der API verwendet wird.

Erstellen eines aktiven Amazon-Kinesis-Daten-Streams

Alle diese Beispiele verwenden die `Music`-DynamoDB-Tabelle, die als Teil des Erste Schritte mit DynamoDB-Tutorials erstellt wurde.

Weitere Informationen zum Erstellen von Konsumenten und zum Verbinden Ihres Kinesis Data Streams mit anderen AWS -Services finden Sie unter Lesen von Daten aus Amazon Kinesis Data Streams im Amazon-Kinesis-Data-Streams-Entwicklerhandbuch.

> **Note**
>
> Wenn Sie zum ersten Mal KDS-Shards verwenden, empfehlen wir, Ihre Shards so einzustellen, dass sie den Nutzungsmustern entsprechend hoch- und herunterskaliert werden. Nachdem Sie mehr Daten zu den Nutzungsmustern gesammelt haben, können Sie die Shards in Ihrem Datenstrom entsprechend anpassen.

Console

1. Melden Sie sich bei der an AWS-Managementkonsole und öffnen Sie die Kinesis-Konsole unter https://console.aws.amazon.com/kinesis/.

2. Klicken Sie auf Create data stream (Datenstrom erstellen) und befolgen Sie die Anweisungen, um einen Stream mit dem Namen `samplestream` zu erstellen.
3. Öffnen Sie die DynamoDB-Konsole unter. https://console.aws.amazon.com/dynamodb/
4. Klicken Sie im Navigationsbereich auf der linken Seite der Konsole auf Tables (Tabellen).
5. Wählen Sie die Tabelle Music (Musik).
6. Wählen Sie die Registerkarte Exports and streams (Exporte und Streams).
7. (Optional) Unter Details zu Amazon-Kinesis-Daten-Streams können Sie die Zeitstempelgenauigkeit des Datensatzes von Mikrosekunde (Standard) auf Millisekunde ändern.
8. Klicken Sie auf samplestream aus der Dropdown-Liste.
9. Klicken Sie auf die Schaltfläche Einschalten.

AWS CLI

1. Erstellen Sie einen Kinesis Data Stream mit dem Namen `samplestream`, indem Sie den Befehl create-stream wählen.

   ```
   aws kinesis create-stream --stream-name samplestream --shard-count 3
   ```

   Sehen Sie sich Überlegungen zur Shard-Verwaltung für Kinesis Data Streams an, bevor Sie die Anzahl der Shards für den Kinesis Data Stream festlegen.
2. Überprüfen Sie, ob der Kinesis Stream aktiv und einsatzbereit ist, indem Sie den Befehl Stream beschreiben auswählen.

   ```
   aws kinesis describe-stream --stream-name samplestream
   ```
3. Aktivieren Sie Kinesis-Streaming für die DynamoDB-Tabelle mithilfe des DynamoDB-Befehls `enable-kinesis-streaming-destination`. Ersetzen Sie den `stream-arn`-Wert durch den Wert, der im vorherigen Schritt von `describe-stream` zurückgegeben wurde. Aktivieren Sie optional das Streaming mit einer höheren Genauigkeit (Mikrosekunden) der für jeden Datensatz zurückgegebenen Zeitstempelwerte.

   So aktivieren Sie das Streaming mit einer Zeitstempelgenauigkeit im Mikrosekundenbereich:

   ```
   aws dynamodb enable-kinesis-streaming-destination \
     --table-name Music \
   ```

```
  --stream-arn arn:aws:kinesis:us-west-2:12345678901:stream/samplestream
  --enable-kinesis-streaming-configuration
 ApproximateCreationDateTimePrecision=MICROSECOND
```

So aktivieren Sie das Streaming mit der standardmäßigen Zeitstempelgenauigkeit (Millisekunde):

```
aws dynamodb enable-kinesis-streaming-destination \
  --table-name Music \
  --stream-arn arn:aws:kinesis:us-west-2:12345678901:stream/samplestream
```

4. Überprüfen Sie, ob Kinesis-Streaming in der Tabelle aktiv ist, mithilfe des DynamoDB-`describe-kinesis-streaming-destination`-Befehls.

```
aws dynamodb describe-kinesis-streaming-destination --table-name Music
```

5. Schreiben Sie Daten in die DynamoDB-Tabelle mithilfe der `put-item`, wie in dem Entwicklerhandbuch von DynamoDB angegeben ist.

```
aws dynamodb put-item \
    --table-name Music  \
    --item \
        '{"Artist": {"S": "No One You Know"}, "SongTitle": {"S": "Call Me
 Today"}, "AlbumTitle": {"S": "Somewhat Famous"}, "Awards": {"N": "1"}}'

aws dynamodb put-item \
    --table-name Music \
    --item \
        '{"Artist": {"S": "Acme Band"}, "SongTitle": {"S": "Happy Day"},
 "AlbumTitle": {"S": "Songs About Life"}, "Awards": {"N": "10"} }'
```

6. Verwenden Sie den Kinesis-CLI-Befehl get-records, um den Inhalt des Kinesis Streams abzurufen. Verwenden Sie dann den folgenden Codeausschnitt, um den Streaminhalt zu deserialisieren.

```
/**
 * Takes as input a Record fetched from Kinesis and does arbitrary processing as
 an example.
 */
public void processRecord(Record kinesisRecord) throws IOException {
    ByteBuffer kdsRecordByteBuffer = kinesisRecord.getData();
```

```
    JsonNode rootNode = OBJECT_MAPPER.readTree(kdsRecordByteBuffer.array());
    JsonNode dynamoDBRecord = rootNode.get("dynamodb");
    JsonNode oldItemImage = dynamoDBRecord.get("OldImage");
    JsonNode newItemImage = dynamoDBRecord.get("NewImage");
    Instant recordTimestamp = fetchTimestamp(dynamoDBRecord);

    /**
     * Say for example our record contains a String attribute named "stringName"
 and we want to fetch the value
     * of this attribute from the new item image. The following code fetches
 this value.
     */
    JsonNode attributeNode = newItemImage.get("stringName");
    JsonNode attributeValueNode = attributeNode.get("S"); // Using DynamoDB "S"
 type attribute
    String attributeValue = attributeValueNode.textValue();
    System.out.println(attributeValue);
}

private Instant fetchTimestamp(JsonNode dynamoDBRecord) {
    JsonNode timestampJson = dynamoDBRecord.get("ApproximateCreationDateTime");
    JsonNode timestampPrecisionJson =
 dynamoDBRecord.get("ApproximateCreationDateTimePrecision");
    if (timestampPrecisionJson != null &&
 timestampPrecisionJson.equals("MICROSECOND")) {
        return Instant.EPOCH.plus(timestampJson.longValue(), ChronoUnit.MICROS);
    }
    return Instant.ofEpochMilli(timestampJson.longValue());
}
```

Java

1. Befolgen Sie die Anweisungen im Kinesis-Data-Streams-Entwicklerhandbuch, um einen Kinesis Data Stream mit `samplestream` Java-Namen zu erstellen.

   Sehen Sie sich Überlegungen zur Shard-Verwaltung für Kinesis Data Streams an, bevor Sie die Anzahl der Shards für den Kinesis Data Stream festlegen.

2. Verwenden Sie den folgenden Codeausschnitt, um Kinesis-Streaming für die DynamoDB-Tabelle zu aktivieren. Aktivieren Sie optional das Streaming mit einer höheren Genauigkeit (Mikrosekunden) der für jeden Datensatz zurückgegebenen Zeitstempelwerte.

So aktivieren Sie das Streaming mit einer Zeitstempelgenauigkeit im Mikrosekundenbereich:

```
EnableKinesisStreamingConfiguration enableKdsConfig =
 EnableKinesisStreamingConfiguration.builder()

 .approximateCreationDateTimePrecision(ApproximateCreationDateTimePrecision.MICROSECOND)
  .build();

EnableKinesisStreamingDestinationRequest enableKdsRequest =
 EnableKinesisStreamingDestinationRequest.builder()
  .tableName(tableName)
  .streamArn(kdsArn)
  .enableKinesisStreamingConfiguration(enableKdsConfig)
  .build();

EnableKinesisStreamingDestinationResponse enableKdsResponse =
 ddbClient.enableKinesisStreamingDestination(enableKdsRequest);
```

So aktivieren Sie das Streaming mit der standardmäßigen Zeitstempelgenauigkeit (Millisekunde):

```
EnableKinesisStreamingDestinationRequest enableKdsRequest =
 EnableKinesisStreamingDestinationRequest.builder()
  .tableName(tableName)
  .streamArn(kdsArn)
  .build();

EnableKinesisStreamingDestinationResponse enableKdsResponse =
 ddbClient.enableKinesisStreamingDestination(enableKdsRequest);
```

3. Befolgen Sie die Anweisungen im Kinesis-Data-Streams-Entwicklerhandbuch, um aus dem erstellten Datenstrom zu lesen.
4. Verwenden Sie dann den folgenden Codeausschnitt, um den Streaminhalt zu deserialisieren

```
/**
 * Takes as input a Record fetched from Kinesis and does arbitrary processing as
 an example.
 */
public void processRecord(Record kinesisRecord) throws IOException {
    ByteBuffer kdsRecordByteBuffer = kinesisRecord.getData();
    JsonNode rootNode = OBJECT_MAPPER.readTree(kdsRecordByteBuffer.array());
```

```
    JsonNode dynamoDBRecord = rootNode.get("dynamodb");
    JsonNode oldItemImage = dynamoDBRecord.get("OldImage");
    JsonNode newItemImage = dynamoDBRecord.get("NewImage");
    Instant recordTimestamp = fetchTimestamp(dynamoDBRecord);

    /**
     * Say for example our record contains a String attribute named "stringName"
 and we wanted to fetch the value
     * of this attribute from the new item image, the below code would fetch
 this.
     */
    JsonNode attributeNode = newItemImage.get("stringName");
    JsonNode attributeValueNode = attributeNode.get("S"); // Using DynamoDB "S"
 type attribute
    String attributeValue = attributeValueNode.textValue();
    System.out.println(attributeValue);
}

private Instant fetchTimestamp(JsonNode dynamoDBRecord) {
    JsonNode timestampJson = dynamoDBRecord.get("ApproximateCreationDateTime");
    JsonNode timestampPrecisionJson =
 dynamoDBRecord.get("ApproximateCreationDateTimePrecision");
    if (timestampPrecisionJson != null &&
 timestampPrecisionJson.equals("MICROSECOND")) {
        return Instant.EPOCH.plus(timestampJson.longValue(), ChronoUnit.MICROS);
    }
    return Instant.ofEpochMilli(timestampJson.longValue());
}
```

Vornehmen von Änderungen an einem aktiven Amazon-Kinesis-Daten-Stream

In diesem Abschnitt wird beschrieben, wie Sie mithilfe der Konsole und der API Änderungen an einem aktiven Kinesis Data Streams for DynamoDB-Setup vornehmen. AWS CLI

AWS-Managementkonsole

1. Öffnen Sie die DynamoDB-Konsole unter https://console.aws.amazon.com/dynamodb/
2. Öffnen Sie die gewünschte Tabelle.
3. Wählen Sie die Option Exporte und Streams.

AWS CLI

1. Rufen Sie `describe-kinesis-streaming-destination` auf, um zu bestätigen, dass der Stream `ACTIVE` ist.
2. Rufen Sie `UpdateKinesisStreamingDestination` auf, wie in diesem Beispiel:

```
aws dynamodb update-kinesis-streaming-destination --table-name
 enable_test_table --stream-arn arn:aws:kinesis:us-east-1:12345678901:stream/
enable_test_stream --update-kinesis-streaming-configuration
 ApproximateCreationDateTimePrecision=MICROSECOND
```

3. Rufen Sie `describe-kinesis-streaming-destination` auf, um zu bestätigen, dass der Stream `UPDATING` ist.
4. Rufen Sie `describe-kinesis-streaming-destination` regelmäßig auf, bis der Streaming-Status wieder `ACTIVE` ist. Normalerweise dauert es bis zu 5 Minuten, bis die Aktualisierungen der Zeitstempelgenauigkeit wirksam werden. Sobald der Status aktualisiert wurde, ist die Änderung abgeschlossen, und der neue Genauigkeitswert wird auf künftige Datensätze angewendet.
5. Schreiben Sie mit `putItem` in die Tabelle.
6. Verwenden Sie den Kinesis-Befehl `get-records`, um den Inhalt des Streams abzurufen.
7. Vergewissern Sie sich, dass die `ApproximateCreationDateTime` der Schreibvorgänge die gewünschte Genauigkeit haben.

Java-API

1. Stellen Sie einen Codeausschnitt bereit, der eine `UpdateKinesisStreamingDestination`-Anfrage und eine `UpdateKinesisStreamingDestination`-Antwort erstellt.
2. Stellen Sie einen Codeausschnitt bereit, der eine `DescribeKinesisStreamingDestination`-Anfrage und eine `DescribeKinesisStreamingDestination response` erstellt.
3. Rufen Sie `describe-kinesis-streaming-destination` regelmäßig auf, bis der Streaming-Status wieder `ACTIVE` ist. Dies zeigt an, dass die Änderung abgeschlossen ist und der neue Genauigkeitswert auf künftige Datensätze angewendet wird.
4. Führen Sie Schreibvorgänge in der Tabelle durch.
5. Lesen Sie Daten aus dem Stream und deserialisieren Sie den Streaminhalt.
6. Vergewissern Sie sich, dass die `ApproximateCreationDateTime` der Schreibvorgänge die gewünschte Genauigkeit haben.

Verwenden von Shards und Metriken mit DynamoDB Streams und Kinesis Data Streams

Überlegungen zur Shard-Verwaltung für Kinesis Data Streams

Ein Kinesis-Daten-Stream zählt seinen Durchsatz in Shards. In Amazon Kinesis Data Streams können Sie zwischen einem On-Demand-Modus und einem bereitgestellten Modus für Ihre Datenströme wählen.

Wir empfehlen, den On-Demand-Modus für Ihren Kinesis-Daten-Stream zu verwenden, wenn Ihre DynamoDB-Schreib-Workload sehr variabel und unvorhersehbar ist. Im On-Demand-Modus ist keine Kapazitätsplanung erforderlich, da Kinesis Data Streams die Shards automatisch verwaltet, um den erforderlichen Durchsatz bereitzustellen.

Bei vorhersehbaren Workloads können Sie den bereitgestellten Modus für Ihren Kinesis-Daten-Stream verwenden. Im bereitgestellten Modus müssen Sie die Anzahl der Shards für den Datenstrom angeben, um die Datenerfassungsdatensätze der Änderungen aus DynamoDB aufzunehmen. Um die Anzahl der Shards zu ermitteln, die der Kinesis-Daten-Stream für die DynamoDB-Tabelle benötigt, sind folgende Eingabewerte erforderlich:

- Die durchschnittliche Größe des Datensatzes Ihrer DynamoDB-Tabelle in Byte (`average_record_size_in_bytes`).
- Die maximale Anzahl von Schreibvorgängen, die Ihre DynamoDB-Tabelle pro Sekunde ausführen wird. Dazu gehören das Erstellen, Löschen und Aktualisieren von Vorgängen, die von Ihren Anwendungen ausgeführt werden, sowie automatisch generierte Vorgänge, z. B. von Time to Live generierte Löschvorgänge (`write_throughput`).
- Der Prozentsatz der Aktualisierungs- und Überschreibvorgänge, die Sie für Ihre Tabelle ausführen, im Vergleich zu Erstellungs- oder Löschvorgängen (`percentage_of_updates`). Aktualisierungs- und Überschreibvorgänge replizieren sowohl die alten als auch die neuen Images des geänderten Elements in den Stream. Dies erzeugt die doppelte DynamoDB-Elementgröße.

Sie können die Anzahl der Shards (`number_of_shards`), die der Kinesis-Daten-Stream benötigt, mit den Eingabewerte in der folgenden Formel berechnen:

```
number_of_shards = ceiling( max( ((write_throughput * (4+percentage_of_updates) *
 average_record_size_in_bytes) / 1024 / 1024), (write_throughput/1000)), 1)
```

Nehmen wir an, Sie haben einen maximalen Durchsatz von 1 040 Schreibvorgängen pro Sekunde (`write_throughput`) bei einer durchschnittlichen Datensatzgröße von 800 Byte (`average_record_size_in_bytes`). Wenn 25 Prozent dieser Schreibvorgänge Aktualisierungsvorgänge sind (`percentage_of_updates`), benötigen Sie zwei Shards (`number_of_shards`) für Ihren DynamoDB-Streaming-Durchsatz:

```
ceiling( max( ((1040 * (4+25/100) * 800)/ 1024 / 1024), (1040/1000)), 1).
```

Beachten Sie Folgendes, bevor Sie die Formel verwenden, um die Anzahl der Shards zu berechnen, die im bereitgestellten Modus für Kinesis-Daten-Streams erforderlich sind:

- Mit dieser Formel können Sie die Anzahl der Shards abschätzen, die für die Aufnahme Ihrer DynamoDB-Änderungsdatensätze erforderlich sind. Sie stellt nicht die Gesamtzahl der Shards dar, die in Ihrem Kinesis-Daten-Stream benötigt werden, z. B. Shards, die zur Unterstützung zusätzlicher Nutzer von Kinesis-Daten-Streams erforderlich sind.
- Im Bereitstellungsmodus kann es immer noch zu Ausnahmen beim Lese- und Schreibdurchsatz kommen, wenn Sie Ihren Datenstrom nicht für den Spitzendurchsatz konfigurieren. In diesem Fall müssen Sie Ihren Datenstrom manuell skalieren, um den Datenverkehr zu bewältigen.
- Diese Formel berücksichtigt die zusätzliche Überlastung, die von DynamoDB generiert wird, bevor die Änderungsprotokoll-Datensätze an Kinesis Data Streams gestreamt werden.

Weitere Informationen zu den Kapazitätsmodi in Kinesis Data Streams finden Sie unter Auswahl des Datenstrom-Kapazitätsmodus. Weitere Informationen zu den Preisunterschieden zwischen den verschiedenen Kapazitätsmodi finden Sie unter Amazon Kinesis Data Streams – Preise.

Änderung der Datenerfassung für Kinesis Data Streams überwachen

DynamoDB bietet mehrere CloudWatch Amazon-Metriken, mit denen Sie die Replikation von Change Data Capture nach Kinesis überwachen können. Eine vollständige Liste der CloudWatch Metriken finden Sie unter. DynamoDB-Metriken und -Dimensionen

Es wird empfohlen, die folgenden Elemente sowohl während der Stream-Aktivierung als auch in der Produktion zu überwachen, um festzustellen, ob der Stream über ausreichende Kapazität verfügt:

- `ThrottledPutRecordCount`: Die Anzahl der Datensätze, die von Ihrem Kinesis-Daten-Stream aufgrund unzureichender Kinesis-Daten-Stream-Kapazität gedrosselt wurden. Der `ThrottledPutRecordCount` sollte so niedrig wie möglich bleiben, obwohl sie bei

außergewöhnlichen Nutzungsspitzen eine gewisse Drosselung erfahren könnten. DynamoDB versucht erneut, gedrosselte Datensätze in das Kinesis Data Stream zu senden. Dies kann jedoch zu einer höheren Replikationslatenz führen.

Wenn eine übermäßige und regelmäßige Drosselung auftritt, müssen Sie möglicherweise die Anzahl der Kinesis-Stream-Shards proportional zum beobachteten Schreibdurchsatz Ihrer Tabelle erhöhen. Weitere Informationen zur Bestimmung der Größe eines Kinesis Data Streams finden Sie unter Bestimmen der anfänglichen Größe eines Kinesis Data Streams.

- `AgeOfOldestUnreplicatedRecord`: Die verstrichene Zeit seit der ältesten Änderung auf Elementebene, die noch in den Kinesis Data Stream repliziert wurde, wurde in der DynamoDB-Tabelle angezeigt. Im Normalbetrieb sollte `AgeOfOldestUnreplicatedRecord` in der Reihenfolge von Millisekunden liegen. Diese Zahl wächst aufgrund erfolgloser Replikationsversuche, wenn diese durch kundengesteuerte Konfigurationsoptionen verursacht werden.

  Wenn die Metrik `AgeOfOldestUnreplicatedRecord` 168 Stunden überschreitet, wird die Replikation von Änderungen auf Elementebene aus der DynamoDB-Tabelle in den Kinesis-Daten-Stream automatisch deaktiviert.

  Kundengesteuerte Konfigurationsbeispiele, die zu erfolglosen Replikationsversuchen führen, sind eine zu gering bereitgestellte Kinesis-Daten-Stream-Kapazität, die zu übermäßiger Drosselung führt, oder eine manuelle Aktualisierung der Zugriffsrichtlinien Ihres Kinesis-Daten-Streams, die DynamoDB das Hinzufügen von Daten zu Ihrem Datenstrom verweigern. Um diese Metrik so niedrig wie möglich zu halten, müssen Sie möglicherweise die richtige Bereitstellung Ihrer Kinesis-Daten-Stream-Kapazität sicherstellen und sicherstellen, dass die Berechtigungen von DynamoDB unverändert bleiben.

- `FailedToReplicateRecordCount`: Die Anzahl der Datensätze, die DynamoDB nicht in Ihren Kinesis-Datenstrom repliziert hat. Bestimmte Elemente, die größer als 34 KB sind, können sich vergrößern, um Datensätze zu ändern, die größer als die Elementgrößengrenze von 1 MB von Kinesis Data Streams sind. Diese Größenerweiterung tritt auf, wenn diese Elemente, die größer als 34 KB sind, eine große Anzahl von booleschen oder leeren Attributwerten enthalten. Boolesche und leere Attributwerte werden in DynamoDB als 1 Byte gespeichert, erweitern sich jedoch auf bis zu 5 Byte, wenn sie mit Standard-JSON für die Kinesis-Data-Streams-Replikation serialisiert werden. DynamoDB kann solche Änderungsdatensätze nicht in Ihren Kinesis Dats Stream replizieren. DynamoDB überspringt diese Änderungsdatensätze und repliziert automatisch nachfolgende Datensätze.

Sie können CloudWatch Amazon-Alarme erstellen, die eine Amazon Simple Notification Service (Amazon SNS) -Nachricht zur Benachrichtigung senden, wenn eine der oben genannten Metriken einen bestimmten Schwellenwert überschreitet.

Verwendung von IAM-Richtlinien für Amazon Kinesis Data Streams und Amazon DynamoDB

Wenn Sie Amazon Kinesis Data Streams für Amazon DynamoDB zum ersten Mal aktivieren, erstellt DynamoDB automatisch eine AWS Identity and Access Management (IAM) -Serviceverknüpfte Rolle für Sie. Diese Rolle `AWSServiceRoleForDynamoDBKinesisDataStreamsReplication` ermöglicht DynamoDB, die Replikation von Änderungen auf Elementebene in Kinesis Data Streams in Ihrem Auftrag zu verwalten. Löschen Sie diese serviceverknüpfte Rolle nicht.

Weitere Informationen zu serviceverknüpften Rollen finden Sie unter Verwenden serviceverknüpfter Rollen im IAM-Benutzerhandbuch.

> **Note**
>
> DynamoDB unterstützt keine Tag-basierten Bedingungen für IAM-Richtlinien.

Damit Sie Amazon Kinesis Data Streams für Amazon DynamoDB aktivieren können, benötigen Sie die folgenden Berechtigungen für die Tabelle:

- `dynamodb:EnableKinesisStreamingDestination`
- `kinesis:ListStreams`
- `kinesis:PutRecords`
- `kinesis:DescribeStream`

Wenn Sie Amazon Kinesis Data Streams für Amazon DynamoDB für eine bestimmte DynamoDB-Tabelle beschreiben möchten, benötigen Sie die folgenden Berechtigungen für die Tabelle.

- `dynamodb:DescribeKinesisStreamingDestination`
- `kinesis:DescribeStreamSummary`
- `kinesis:DescribeStream`

Damit Sie Amazon Kinesis Data Streams für Amazon DynamoDB deaktivieren können, benötigen Sie die folgenden Berechtigungen für die Tabelle.

- `dynamodb:DisableKinesisStreamingDestination`

Damit Sie Amazon Kinesis Data Streams für Amazon DynamoDB aktualisieren können, benötigen Sie die folgenden Berechtigungen für die Tabelle.

- `dynamodb:UpdateKinesisStreamingDestination`

Die folgenden Beispiele zeigen, wie IAM-Richtlinien verwendet werden, um Berechtigungen für Amazon Kinesis Data Streams für Amazon DynamoDB zu erteilen.

Beispiel: Aktivieren von Amazon Kinesis Data Streams für Amazon DynamoDB

Über die folgende IAM-Richtlinie werden die nötigen Berechtigungen eingeräumt, um Amazon Kinesis Data Streams für Amazon DynamoDB für die `Music`-Tabelle zu aktivieren. Sie gewährt keine Berechtigungen zum Deaktivieren, Aktualisieren oder Beschreiben von Kinesis Data Streams for DynamoDB für die `Music`-Tabelle.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "iam:CreateServiceLinkedRole",
            "Resource": "arn:aws:iam::*:role/aws-
service-role/kinesisreplication.dynamodb.amazonaws.com/
AWSServiceRoleForDynamoDBKinesisDataStreamsReplication",
            "Condition": {
                "StringLike": {
                    "iam:AWSServiceName":
 "kinesisreplication.dynamodb.amazonaws.com"
                }
            }
        },
        {
            "Effect": "Allow",
```

```
            "Action": [
                "dynamodb:EnableKinesisStreamingDestination"
            ],
            "Resource": "arn:aws:dynamodb:us-west-2:111122223333:table/Music"
        }
    ]
}
```

Beispiel: Aktualisieren von Amazon Kinesis Data Streams für Amazon DynamoDB

Über die folgende IAM-Richtlinie werden die nötigen Berechtigungen eingeräumt, um Amazon Kinesis Data Streams für Amazon DynamoDB für die `Music`-Tabelle zu aktualisieren. Sie gewährt keine Berechtigungen zum Aktivieren, Deaktivieren oder Beschreiben von Amazon Kinesis Data Streams für Amazon DynamoDB für die `Music`-Tabelle.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "dynamodb:UpdateKinesisStreamingDestination"
            ],
            "Resource": "arn:aws:dynamodb:us-west-2:111122223333:table/Music"
        }
    ]
}
```

Beispiel: Deaktivieren von Amazon Kinesis Data Streams für Amazon DynamoDB

Über die folgende IAM-Richtlinie werden die nötigen Berechtigungen eingeräumt, um Amazon Kinesis Data Streams für Amazon DynamoDB für die `Music`-Tabelle zu deaktivieren. Sie gewährt keine Berechtigungen zum Aktivieren, Aktualisieren oder Beschreiben von Amazon Kinesis Data Streams für Amazon DynamoDB für die `Music`-Tabelle.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "dynamodb:DisableKinesisStreamingDestination"
            ],
            "Resource": "arn:aws:dynamodb:us-west-2:111122223333:table/Music"
        }
    ]
}
```

Beispiel: Selektives Anwenden von Berechtigungen für Amazon Kinesis Data Streams für Amazon DynamoDB basierend auf Ressource

Über die folgende IAM-Richtlinie werden Berechtigungen zum Aktivieren und Beschreiben von Amazon Kinesis Data Streams für Amazon DynamoDB für die `Music`-Tabelle eingeräumt und Berechtigungen zum Deaktivieren von Amazon Kinesis Data Streams für Amazon DynamoDB für die Tabelle `Orders` verweigert.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "dynamodb:EnableKinesisStreamingDestination",
                "dynamodb:DescribeKinesisStreamingDestination"
            ],
            "Resource": "arn:aws:dynamodb:us-west-2:111122223333:table/Music"
        },
        {
            "Effect": "Deny",
            "Action": [
                "dynamodb:DisableKinesisStreamingDestination"
```

```
            ],
            "Resource": "arn:aws:dynamodb:us-west-2:111122223333:table/Orders"
        }
    ]
}
```

Verwenden von serviceverknüpften Rollen für Kinesis Data Streams für DynamoDB

Amazon Kinesis Data Streams für Amazon DynamoDB verwendet AWS Identity and Access Management (IAM) service-verknüpfte Rollen. Eine serviceverknüpfte Rolle ist ein spezieller Typ einer IAM-Rolle, die direkt mit Kinesis Data Streams für DynamoDB verknüpft ist. Dienstbezogene Rollen sind von Kinesis Data Streams for DynamoDB vordefiniert und beinhalten alle Berechtigungen, die der Dienst benötigt, um andere AWS Dienste in Ihrem Namen aufzurufen.

Eine serviceverknüpfte Rolle vereinfacht das Einrichten von Kinesis Data Streams für DynamoDB, da Sie die erforderlichen Berechtigungen nicht manuell hinzufügen müssen. Kinesis Data Streams für DynamoDB definiert die Berechtigungen seiner serviceverknüpften Rollen. Sofern keine andere Einstellung festgelegt wurde, können nur Kinesis Data Streams für DynamoDB die Rollen übernehmen. Die definierten Berechtigungen umfassen die Vertrauens- und Berechtigungsrichtlinie. Diese Berechtigungsrichtlinie kann keinen anderen IAM-Entitäten zugewiesen werden.

Informationen zu anderen Services, die serviceverknüpften Rollen unterstützen, finden Sie unter AWS -Services, die mit IAM funktionieren. Suchen Sie nach den Services, für die Ja in der Spalte Serviceverknüpfte Rolle angegeben ist. Wählen Sie über einen Link Yes (Ja) aus, um die Dokumentation zu einer servicegebundenen Rolle für diesen Service anzuzeigen.

Berechtigungen von serviceverknüpften Rollen für Kinesis Data Streams für DynamoDB

Kinesis Data Streams for DynamoDB verwendet die mit dem Service verknüpfte Rolle namens. AWSServiceRoleForDynamoDBKinesisDataStreamsReplication Die serviceverknüpfte Rolle ermöglicht es Amazon DynamoDB, in Ihrem Namen die Replikation von Änderungen auf Elementebene in Kinesis Data Streams zu verwalten.

Die serviceverknüpfte Rolle `AWSServiceRoleForDynamoDBKinesisDataStreamsReplication` vertraut darauf, dass die folgenden Services die Rolle annehmen:

- `kinesisreplication.dynamodb.amazonaws.com`

Die Richtlinie für Rollenberechtigungen erlaubt Kinesis Data Streams für DynamoDB die folgenden Aktionen auf den angegebenen Ressourcen durchzuführen:

- Aktion: `Put records and describe` für `Kinesis stream`
- Aktion: `Generate data keys` aktiviert, um Daten `AWS KMS` in Kinesis-Streams zu speichern, die mit benutzergenerierten AWS KMS Schlüsseln verschlüsselt sind.

Den genauen Inhalt des Richtliniendokuments finden Sie unter Dynamo. DBKinesis ReplicationServiceRolePolicy

Sie müssen Berechtigungen konfigurieren, damit eine juristische Stelle von IAM (z. B. Benutzer, Gruppe oder Rolle) eine serviceverknüpfte Rolle erstellen, bearbeiten oder löschen kann. Weitere Informationen finden Sie unter serviceverknüpfte Rollenberechtigungen im IAM-Benutzerhandbuch.

Erstellen einer serviceverknüpften Rolle für Kinesis Data Streams für DynamoDB

Sie müssen eine serviceverknüpfte Rolle nicht manuell erstellen. Wenn Sie Kinesis Data Streams for DynamoDB in der AWS-Managementkonsole, der oder der AWS API aktivieren, erstellt Kinesis Data Streams for DynamoDB die serviceverknüpfte Rolle für Sie. AWS CLI

Wenn Sie diese serviceverknüpfte Rolle löschen und sie dann erneut erstellen müssen, können Sie dasselbe Verfahren anwenden, um die Rolle in Ihrem Konto neu anzulegen. Wenn Sie Kinesis Data Streams für DynamoDB aktivieren, erstellt Kinesis Data Streams für DynamoDB die serviceverknüpfte Rolle erneut für Sie.

Bearbeiten einer serviceverknüpften Rolle für Kinesis Data Streams für DynamoDB

Kinesis Data Streams für DynamoDB ermöglicht es Ihnen nicht, die mit der `AWSServiceRoleForDynamoDBKinesisDataStreamsReplication` serviceverknüpfte Rolle zu bearbeiten. Da möglicherweise verschiedene Entitäten auf die Rolle verweisen, kann der Rollenname nach dem Erstellen einer serviceverknüpften Rolle nicht mehr geändert werden. Sie können jedoch die Beschreibung der Rolle mit IAM bearbeiten. Weitere Informationen finden Sie unter Bearbeiten einer serviceverknüpften Rolle im IAM-Benutzerhandbuch.

Löschen einer serviceverknüpften Rolle für Kinesis Data Streams für DynamoDB

Sie können auch die IAM-Konsole, die AWS CLI oder die API verwenden, um die serviceverknüpfte Rolle manuell zu löschen AWS . Sie müssen jedoch die Ressourcen für Ihre serviceverknüpfte Rolle zuerst manuell bereinigen, bevor Sie diese manuell löschen können.

> **Note**
>
> Wenn der Kinesis Data Streams für DynamoDB-Service die Rolle verwendet, wenn Sie versuchen, die Ressourcen zu löschen, schlägt das Löschen möglicherweise fehl. Wenn dies passiert, warten Sie einige Minuten und versuchen Sie es erneut.

So löschen Sie die serviceverknüpfte Rolle mit IAM

Verwenden Sie die IAM-Konsole, die oder die AWS API AWS CLI, um die serviceverknüpfte Rolle zu löschen. `AWSServiceRoleForDynamoDBKinesisDataStreamsReplication` Weitere Informationen finden Sie unter Löschen einer serviceverknüpften Rolle im IAM-Leitfaden.

Ändern Sie die Datenerfassung für DynamoDB Streams

DynamoDB Streams erfasst eine zeitlich geordnete Abfolge von Änderungen auf Elementebene in jeder beliebigen DynamoDB-Tabelle und speichert diese Informationen bis zu 24 Stunden. Anwendungen können auf dieses Protokoll zugreifen und die Datenelemente vor und nach der Änderung nahezu in Echtzeit aufrufen.

Die Verschlüsselung ruhender Daten verschlüsselt die Daten in DynamoDB Streams. Weitere Informationen finden Sie unter Ruhende DynamoDB-Verschlüsselung.

Ein DynamoDB-Stream ist ein strukturierter Informationsfluss zu Elementänderungen in einer DynamoDB-Tabelle. Wenn Sie den Stream für eine Tabelle aktivieren, werden von DynamoDB Informationen über jede Änderung an den Datenelementen in der Tabelle erfasst.

Wenn eine Anwendung Elemente in der Tabelle erstellt, aktualisiert oder löscht, schreibt DynamoDB Streams einen Stream-Datensatz mit dem bzw. den Primärschlüsselattributen der Elemente, die geändert wurden. Ein Stream-Datensatz enthält Informationen über eine Datenänderung an einem einzelnen Element einer DynamoDB-Tabelle. Sie können den Stream konfigurieren, sodass die Stream-Datensätze zusätzliche Informationen erfassen, z. B. Images der geänderten Elemente vor und nach der Änderung.

Mit DynamoDB Streams wird Folgendes sichergestellt:

- Jeder Stream-Datensatz erscheint genau einmal im Stream.
- Für jedes Element, das in einer DynamoDB-Tabelle geändert wird, erscheinen die Stream-Datensätze in der gleichen Reihenfolge wie die tatsächlichen Änderungen des Elements.

DynamoDB Streams schreibt Stream-Datensätze nahezu in Echtzeit, sodass Sie Anwendungen erstellen können, die diese Streams verbrauchen und basierend auf den Inhalten Aktionen einleiten.

Themen

- Endpunkte für DynamoDB Streams
- Aktivieren eines Streams
- Lesen und Verarbeiten eines Streams
- DynamoDB Streams und Time to Live (TTL)
- Verwenden des DynamoDB-Streams-Kinesis-Adapters zum Verarbeiten von Stream-Datensätzen
- DynamoDB-Streams-Low-Level-API: Java-Beispiel
- DynamoDB Streams und -Trigger AWS Lambda
- DynamoDB Streams und Apache Flink

Endpunkte für DynamoDB Streams

AWS verwaltet separate Endpunkte für DynamoDB- und DynamoDB Streams. Für die Arbeit mit Datenbanktabellen und Indexen muss Ihre Anwendung auf einen DynamoDB-Endpunkt zugreifen. Um DynamoDB-Streams-Datensätze zu lesen und zu verarbeiten, muss die Anwendung auf einen DynamoDB-Streams-Endpunkt in derselben Region zugreifen.

DynamoDB Streams bietet zwei Gruppen von Endpunkten. Diese sind:

- IPv4-only Endpoints: Endpoints mit der Benennungskonvention. `streams.dynamodb.<region>.amazonaws.com`
- Dual-Stack-Endpunkte: Neue Endpunkte, die sowohl mit als auch kompatibel sind IPv4 und IPv6 der Namenskonvention folgen. `streams-dynamodb.<region>.api.aws`

> **Note**
>
> Eine vollständige Liste der DynamoDB- und DynamoDB-Streams-Regionen und Endpunkte finden Sie unter Regionen und Endpunkte in der Allgemeine AWS-Referenz.

AWS SDKs Sie bieten separate Clients für DynamoDB- und DynamoDB Streams. Je nach Anforderung kann die Anwendung auf einen DynamoDB-Endpunkt, einen DynamoDB-Streams-Endpunkt oder beide gleichzeitig zugreifen. Für die Verbindung mit beiden Endpunkten muss Ihre

Anwendung zwei Clients instanziieren, und zwar einen für DynamoDB und einen für DynamoDB Streams.

Aktivieren eines Streams

Sie können einen Stream in einer neuen Tabelle aktivieren, wenn Sie ihn mit der AWS CLI oder einer der folgenden Optionen erstellen. AWS SDKs Außerdem können Sie einen Stream in einer vorhandenen Tabelle aktivieren oder deaktivieren oder die Einstellungen eines Streams ändern. DynamoDB Streams wird asynchron betrieben. Daher wirkt es sich nicht auf die Leistung der Tabelle aus, wenn Sie einen Stream aktivieren.

Die einfachste Möglichkeit zum Verwalten von DynamoDB Streams bietet die AWS-Managementkonsole.

1. Melden Sie sich bei der an AWS-Managementkonsole und öffnen Sie die DynamoDB-Konsole unter. https://console.aws.amazon.com/dynamodb/
2. Wählen Sie im Dashboard der DynamoDB-Konsole die Option Tables (Tabellen) aus und wählen Sie eine vorhandene Tabelle aus.
3. Wählen Sie die Registerkarte Exports and streams (Exporte und Streams).
4. Wählen Sie im Abschnitt Details zum DynamoDB-Stream die Option Einschalten aus.
5. Wählen Sie auf der Seite DynamoDB-Stream einschalten die Informationen aus, die in den Stream geschrieben werden sollen, sobald Daten in der Tabelle geändert werden:

   - Nur Schlüsselattribute – nur die Schlüsselattribute des geänderten Elements.
   - Neues Image – das gesamte Element wie es nach der Änderung erscheint.
   - Altes Image – das gesamte Element wie es vor der Änderung erscheint.
   - Neues und altes Image – sowohl das neue als auch das alte Image des Elements.

   Wenn Sie die gewünschten Einstellungen vorgenommen haben, wählen Sie Stream einschalten aus.
6. (Optional) Um einen vorhandenen Stream zu deaktivieren, wählen Sie unter Details zum DynamoDB-Stream die Option Deaktivieren aus.

Sie können auch die `CreateTable`- oder `UpdateTable`-API-Operationen zum Aktivieren oder Ändern eines Streams verwenden. Der Parameter `StreamSpecification` bestimmt, wie der Stream konfiguriert wird:

- `StreamEnabled` – gibt an, ob ein Stream für die Tabelle aktiviert (`true`) oder deaktiviert (`false`) ist.
- `StreamViewType` – legt die Informationen fest, die in den Stream geschrieben werden, sobald Daten in der Tabelle geändert werden:
  - `KEYS_ONLY` – nur die Schlüsselattribute des geänderten Elements.
  - `NEW_IMAGE` – das gesamte Element, wie es nach der Änderung erscheint.
  - `OLD_IMAGE` – das gesamte Element, wie es vor der Änderung erscheint.
  - `NEW_AND_OLD_IMAGES` – sowohl das neue als auch das alte Image des Elements.

Sie können einen Stream jederzeit aktivieren oder deaktivieren. Wenn Sie versuchen, einen Stream für eine Tabelle zu aktivieren, die bereits über einen Stream verfügt, erhalten Sie jedoch eine `ValidationException`. Wenn Sie versuchen, einen Stream für eine Tabelle zu deaktivieren, die nicht über einen Stream verfügt, erhalten Sie auch eine `ValidationException`.

Wenn Sie `StreamEnabled` auf `true` festlegen, erstellt DynamoDB einen neuen Stream mit einem zugewiesenen, eindeutigen Stream-Deskriptor. Wenn Sie einen Stream in der Tabelle deaktivieren und anschließend erneut aktivieren, wird ein neuer Stream mit einem anderen Stream-Deskriptor erstellt.

Jeder Stream wird anhand eines Amazon-Ressourcennamens (ARN) eindeutig identifiziert. Nachfolgend ist ein Beispiel-ARN für einen Stream in einer DynamoDB-Tabelle mit dem Namen `TestTable` aufgeführt:

```
arn:aws:dynamodb:us-west-2:111122223333:table/TestTable/stream/2015-05-11T21:21:33.291
```

Um den aktuellen Stream-Deskriptor für eine Tabelle zu bestimmen, erstellen Sie eine DynamoDB-Anforderung `DescribeTable` und suchen das Element `LatestStreamArn` in der Antwort.

> **Note**
>
> Es ist nicht möglich, ein `StreamViewType` zu bearbeiten, sobald ein Stream eingerichtet wurde. Wenn Sie nach der Einrichtung Änderungen an einem Stream vornehmen möchten, müssen Sie den aktuellen Stream deaktivieren und einen neuen erstellen.

Lesen und Verarbeiten eines Streams

Zum Lesen und Verarbeiten eines Streams muss Ihre Anwendung eine Verbindung mit einem DynamoDB-Streams-Endpunkt herstellen und API-Anforderungen ausgeben.

Ein Stream besteht aus Stream-Datensätzen. Jeder Stream-Datensatz stellt eine einzelne Datenänderung in der DynamoDB-Tabelle dar, zu der der Stream gehört. Jedem Stream-Datensatz ist eine Sequenznummer zugewiesen, wodurch die Reihenfolge dargestellt wird, in der der Datensatz im Stream veröffentlicht wurde.

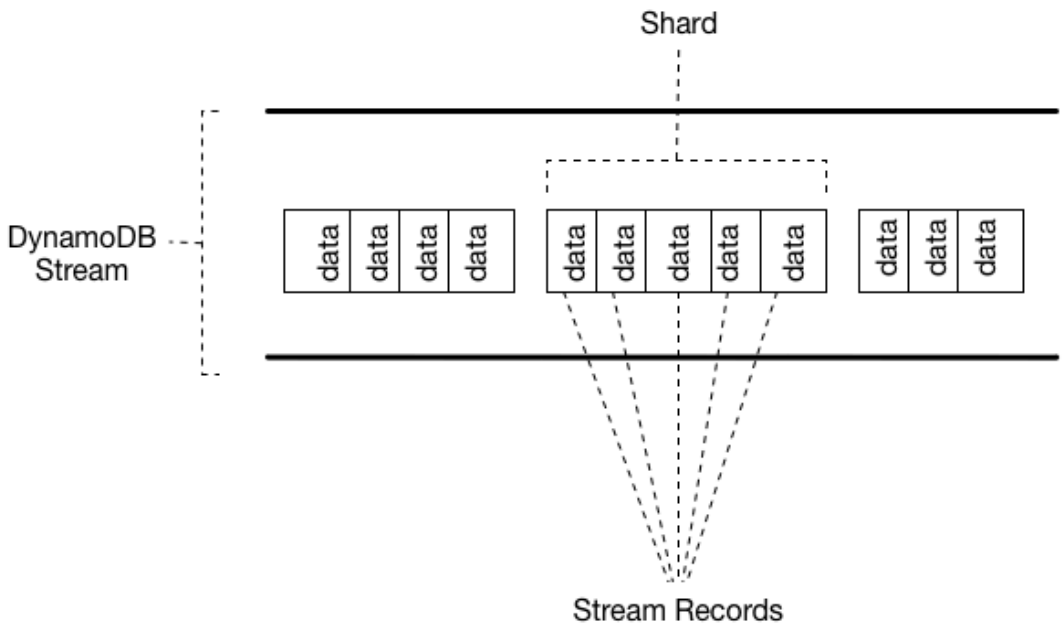
Stream-Datensätze werden in Gruppen oder Shards verwaltet. Jeder Shard fungiert als Container für mehrere Stream-Datensätze und enthält Informationen, die zum Abrufen und Durchlaufen dieser Datensätze erforderlich sind. Die Stream-Datensätze in einem Shard werden nach 24 Stunden automatisch entfernt.

Shards sind flüchtig, d. h., sie werden nach Bedarf automatisch erstellt und gelöscht. Jeder Shard kann in mehrere neue Shards unterteilt werden. Dieser Vorgang erfolgt automatisch. (Es ist auch möglich, dass ein übergeordneter Shard nur einen untergeordneten Shard besitzt.) Ein Shard kann aufgrund hoher Schreibaktivitäten in der übergeordneten Tabelle aufgeteilt werden, sodass Anwendungen Datensätze aus mehreren Shards parallel verarbeiten können.

Wenn Sie einen Stream deaktivieren, werden alle offenen Shards geschlossen. Die Daten im Stream bleiben 24 Stunden lang lesbar.

Da Shards hierarchisch (über- und untergeordnet) aufgebaut sind, müssen Anwendungen einen übergeordneten Shard immer vor einem untergeordneten Shard verarbeiten. So wird sichergestellt, dass die Stream-Datensätze ebenfalls in der richtigen Reihenfolge verarbeitet werden. (Wenn Sie den DynamoDB-Streams-Kinesis-Adapter verwenden, wird dies für Sie erledigt. Ihre Anwendung verarbeitet die Shards und Streamdatensätze in der richtigen Reihenfolge. Es verarbeitet automatisch neue oder abgelaufene Shards sowie Shards, die sich während der Ausführung der Anwendung teilen. Weitere Informationen finden Sie unter Verwenden des DynamoDB-Streams-Kinesis-Adapters zum Verarbeiten von Stream-Datensätzen.)

Das folgende Diagramm zeigt die Beziehung zwischen einem Stream, Shards im Stream und Stream-Datensätzen in den Shards.



> **Note**
>
> Wenn Sie eine `PutItem` oder `UpdateItem` Operation ausführen, mit der keine Daten in einem Element geändert werden, schreibt DynamoDB Streams keinen Stream-Datensatz für diese Operation.

Um auf einen Stream zuzugreifen und die darin enthaltenen Stream-Datensätze zu verarbeiten, führen Sie die folgenden Schritte aus:

- Ermitteln Sie den eindeutigen ARN des Streams, auf den Sie zugreifen möchten.
- Bestimmen Sie, welcher bzw. welche Shards im Stream die gewünschten Stream-Datensätze enthalten.
- Greifen Sie auf den bzw. die Shards zu und rufen Sie die gewünschten Stream-Datensätze ab.

> **Note**
>
> Es sollten nicht mehr als zwei Prozesse gleichzeitig aus dem Shard desselben Streams lesen. Wenn mehr als zwei Leser pro Shard vorhanden sind, kann eine Drosselung die Folge sein.

Die DynamoDB Streams-API bietet die folgenden Aktionen zur Verwendung durch Anwendungsprogramme:

- `ListStreams` – Gibt eine Liste der Stream-Deskriptoren für das aktuelle Konto und den Endpunkt zurück. Sie können optional nur die Stream-Deskriptoren für einen bestimmten Tabellennamen anfordern.
- `DescribeStream` – Gibt Informationen über einen Stream zurück, z. B. den aktuellen Status des Streams, seinen Amazon-Ressourcennamen (ARN), die Zusammenstellung der Shards und die zugehörige DynamoDB-Tabelle. Sie können das Feld `ShardFilter` optional verwenden, um den vorhandenen untergeordneten Shard abzurufen, der dem übergeordneten Shard zugeordnet ist.
- `GetShardIterator` – Gibt einen Shard Iterator zurück, der eine Position innerhalb eines Shards beschreibt. Sie können anfordern, dass der Iterator Zugriff auf den ältesten Punkt, den neuesten Punkt oder einen bestimmten Punkt im Stream bereitstellt.
- `GetRecords` – Gibt die Stream-Datensätze innerhalb eines bestimmten Shards zurück. Sie müssen den von einer `GetShardIterator`-Anforderung zurückgegebenen Shard Iterator angeben.

Eine detaillierte Beschreibung dieser API-Aktionen, einschließlich Anforderungs- und Antwortbeispielen, finden Sie unter Amazon-DynamoDB-Streams-API-Referenz.

Shard-Erkennung

Mit zwei leistungsstarken Methoden können Sie neue Shards in Ihrem DynamoDB-Stream erkennen. Als Nutzer von Amazon DynamoDB Streams haben Sie zwei effektive Möglichkeiten, neue Shards zu verfolgen und zu identifizieren:

Abfragen der gesamten Stream-Topologie

Verwenden Sie die `DescribeStream`-API, um den Stream regelmäßig abzufragen. Dadurch werden alle Shards im Stream zurückgegeben, einschließlich aller neu erstellten Shards. Indem Sie die Ergebnisse im Zeitverlauf vergleichen, können Sie neu hinzugefügte Shards erkennen.

Erkennen untergeordneter Shards

Verwenden Sie die `DescribeStream`-API mit dem Parameter `ShardFilter`, um eine Teilmenge von Shards zu finden. Wenn Sie in der Anfrage einen übergeordneten Shard benennen, gibt DynamoDB Streams seine unmittelbar untergeordneten Shards zurück. Dieser

Ansatz ist nützlich, wenn Sie nur die Herkunft der Shards verfolgen müssen, ohne den gesamten Stream zu scannen.

Anwendungen, die Daten aus DynamoDB Streams aufnehmen, können mithilfe dieses `ShardFilter`-Parameters effizient vom Lesen eines geschlossenen Shards zum zugehörigen untergeordneten Shard übergehen. Dadurch werden wiederholte Aufrufe der `DescribeStream`-API vermieden, um die Shard-Zuordnung für alle geschlossenen und offenen Shards abzurufen und zu durchlaufen. So können untergeordnete Shards schnell erkannt werden, nachdem ein übergeordneter Shard geschlossen wurde. Auf diese Weise werden Stream-Verarbeitungsanwendungen reaktionsschneller und kostengünstiger.

Mit beiden Methoden können Sie den Überblick über die sich entwickelnde Struktur Ihrer DynamoDB Streams behalten, um keine wichtigen Datenaktualisierungen oder Shard-Änderungen zu verpassen.

Datenaufbewahrungsfrist für DynamoDB Streams

Alle Daten in DynamoDB Streams unterliegen einer 24-Stunden-Nutzungsdauer. Sie können die Aktivitäten der letzten 24 Stunden für eine bestimmte Tabelle abrufen und analysieren. Daten, die älter als 24 Stunden sind, können jedoch jederzeit entfernt werden.

Wenn Sie einen Stream in einer Tabelle deaktivieren, bleiben die Daten im Stream 24 Stunden lang lesbar. Nach Ablauf dieses Zeitraums verfallen die Daten und die Stream-Datensätze werden automatisch gelöscht. Es gibt keinen Mechanismus zum manuellen Löschen eines vorhandenen Streams. Sie müssen nur warten, bis die Aufbewahrungsfrist abgelaufen ist (24 Stunden) und alle Stream-Datensätze gelöscht werden.

DynamoDB Streams und Time to Live (TTL)

Sie können Elemente, die durch die Gültigkeitsdauer (TTL) gelöscht wurden, sichern oder anderweitig verarbeiten, indem Sie Amazon DynamoDB Streams in der Tabelle aktivieren und die Streams-Datensätze der abgelaufenen Elemente verarbeiten. Weitere Informationen finden Sie unter Lesen und Verarbeiten eines Streams.

Der Stream-Datensatz enthält ein Benutzeridentitätsfeld `Records[<index>].userIdentity`.

Die Elemente, die nach Ablauf durch den Gültigkeitsdauer-Prozess gelöscht wurden, haben die folgenden Felder:

- `Records[<index>].userIdentity.type`

```
  "Service"
```

- `Records[<index>].userIdentity.principalId`

```
  "dynamodb.amazonaws.com"
```

> **Note**
>
> Wenn Sie TTL in einer globalen Tabelle verwenden, wird das Feld `userIdentity` für die Region, in der die TTL ausgeführt wurde, festgelegt. Dieses Feld wird in anderen Regionen nicht festgelegt, wenn der Löschvorgang repliziert wird.

Der folgende JSON-Ausdruck zeigt den relevanten Teil eines einzelnen Stream-Datensatzes.

```
"Records": [
    {
        ...

        "userIdentity": {
            "type": "Service",
            "principalId": "dynamodb.amazonaws.com"
        }

        ...

    }
]
```

Verwenden von DynamoDB Streams und Lambda zum Archivieren von TTL-gelöschten Elementen

Die Kombination von DynamoDB Time to Live (TTL), DynamoDB Streams und AWS -Lambda kann dazu beitragen, die Archivierung von Daten zu vereinfachen, die DynamoDB-Speicherkosten zu senken und die Codekomplexität zu reduzieren. Die Verwendung von Lambda als Stream-Konsument bietet viele Vorteile, insbesondere geringere Kosten im Vergleich zu anderen Konsumenten wie der Kinesis Client Library (KCL). `GetRecords`-API-Aufrufe in Ihrem DynamoDB-Stream werden Ihnen nicht in Rechnung gestellt, wenn Sie Lambda zum Verarbeiten von Ereignissen verwenden, und Lambda kann eine Ereignisfilterung bieten, indem es JSON-Muster in einem Stream-Ereignis identifiziert. Bei der Inhaltsfilterung nach Ereignismustern können Sie bis zu fünf

verschiedene Filter definieren, um zu steuern, welche Ereignisse zur Verarbeitung an Lambda gesendet werden. Dies hilft, die Zahl der Aufrufe Ihrer Lambda-Funktionen zu reduzieren, den Code zu vereinfachen und die Gesamtkosten zu senken.

Zwar enthält DynamoDB Streams alle Datenänderungen, beispielsweise `Create`-, `Modify`- und `Remove`-Aktionen, dies kann jedoch zu unerwünschten Aufrufen Ihrer Lambda-Archivfunktion führen. Angenommen, Sie verfügen über eine Tabelle mit 2 Millionen Datenänderungen pro Stunde, die in den Stream fließen. Allerdings handelt es sich bei weniger als 5 Prozent dieser Änderungen um Elementlöschungen, die während des TTL-Prozesses ablaufen und archiviert werden müssen. Bei Verwendung von Lambda-Ereignisquellenfiltern wird die Lambda-Funktion nur 100 000-mal pro Stunde aufgerufen. Infolge der Ereignisfilterung werden Ihnen anstelle der 2 Millionen Aufrufe, die Sie ohne Ereignisfilterung hätten, nur die erforderlichen Aufrufe in Rechnung gestellt.

Die Ereignisfilterung wird auf die Lambda-Ereignisquellenzuweisung, angewendet. Hierbei handelt es sich um eine Ressource, die aus einem ausgewählten Ereignis – dem DynamoDB-Stream – Daten liest und eine Lambda-Funktion aufruft. Im folgenden Diagramm ist zu sehen, wie ein durch Time to Live gelöschtes Element von einer Lambda-Funktion unter Verwendung von Streams und Ereignisfiltern verarbeitet wird.

Ereignisfiltermuster für DynamoDB Time to Live

Durch Hinzufügen der folgenden JSON zu den Filterkriterien für Ihre Ereignisquellenzuweisung kann Ihre Lambda-Funktion nur für TTL-gelöschte Elemente aufgerufen werden:

```
{
    "Filters": [
        {
            "Pattern": { "userIdentity": { "type": ["Service"], "principalId":
 ["dynamodb.amazonaws.com"] } }
        }
    ]
}
```

AWS Lambda Erstellen Sie eine Zuordnung der Ereignisquellen

Verwenden Sie die folgenden Codeausschnitte, um eine gefilterte Ereignisquellenzuweisung zu erstellen, die Sie mit dem DynamoDB-Stream einer Tabelle verbinden können. Jeder Codeblock enthält das Ereignisfiltermuster.

AWS CLI

```
aws lambda create-event-source-mapping \
--event-source-arn 'arn:aws:dynamodb:eu-west-1:012345678910:table/test/
stream/2021-12-10T00:00:00.000' \
--batch-size 10 \
--enabled \
--function-name test_func \
--starting-position LATEST \
--filter-criteria '{"Filters": [{"Pattern": "{\"userIdentity\":{\"type\":[\"Service
\"],\"principalId\":[\"dynamodb.amazonaws.com\"]}}"}]}'
```

Java

```
LambdaClient client = LambdaClient.builder()
        .region(Region.EU_WEST_1)
        .build();

Filter userIdentity = Filter.builder()
        .pattern("{\"userIdentity\":{\"type\":[\"Service\"],\"principalId\":
[\"dynamodb.amazonaws.com\"]}}")
        .build();

FilterCriteria filterCriteria = FilterCriteria.builder()
        .filters(userIdentity)
        .build();

CreateEventSourceMappingRequest mappingRequest =
 CreateEventSourceMappingRequest.builder()
        .eventSourceArn("arn:aws:dynamodb:eu-west-1:012345678910:table/test/
stream/2021-12-10T00:00:00.000")
        .batchSize(10)
        .enabled(Boolean.TRUE)
        .functionName("test_func")
        .startingPosition("LATEST")
        .filterCriteria(filterCriteria)
        .build();
```

```
try{
    CreateEventSourceMappingResponse eventSourceMappingResponse =
 client.createEventSourceMapping(mappingRequest);
    System.out.println("The mapping ARN is
 "+eventSourceMappingResponse.eventSourceArn());

}catch (ServiceException e){
    System.out.println(e.getMessage());
}
```

Node

```
const client = new LambdaClient({ region: "eu-west-1" });

const input = {
    EventSourceArn: "arn:aws:dynamodb:eu-west-1:012345678910:table/test/
stream/2021-12-10T00:00:00.000",
    BatchSize: 10,
    Enabled: true,
    FunctionName: "test_func",
    StartingPosition: "LATEST",
    FilterCriteria: { "Filters": [{ "Pattern": "{\"userIdentity\":{\"type\":
[\"Service\"],\"principalId\":[\"dynamodb.amazonaws.com\"]}}" }] }
}

const command = new CreateEventSourceMappingCommand(input);

try {
    const results = await client.send(command);
    console.log(results);
} catch (err) {
    console.error(err);
}
```

Python

```
session = boto3.session.Session(region_name = 'eu-west-1')
client = session.client('lambda')

try:
    response = client.create_event_source_mapping(
```

```
        EventSourceArn='arn:aws:dynamodb:eu-west-1:012345678910:table/test/
stream/2021-12-10T00:00:00.000',
        BatchSize=10,
        Enabled=True,
        FunctionName='test_func',
        StartingPosition='LATEST',
        FilterCriteria={
            'Filters': [
                {
                    'Pattern': "{\"userIdentity\":{\"type\":[\"Service\"],
\"principalId\":[\"dynamodb.amazonaws.com\"]}}"
                },
            ]
        }
    )
    print(response)
except Exception as e:
    print(e)
```

JSON

```
{
  "userIdentity": {
     "type": ["Service"],
     "principalId": ["dynamodb.amazonaws.com"]
   }
}
```

Verwenden des DynamoDB-Streams-Kinesis-Adapters zum Verarbeiten von Stream-Datensätzen

Die Verwendung des Amazon-Kinesis-Adapters ist die empfohlene Methode zum Verwenden von Streams aus Amazon DynamoDB. Die DynamoDB-Streams-API ist absichtlich ähnlich gestaltet wie die von Kinesis Data Streams. In beiden Services bestehen die Daten-Streams aus Shards. Dieses sind Container für Stream-Datensätze. Beide Dienste APIs enthalten`ListStreams`, `DescribeStreamGetShards`, und `GetShardIterator` Operationen. (Diese DynamoDB-Streams-Aktionen ähneln ihren Gegenstücken in Kinesis Data Streams, sind jedoch nicht vollkommen identisch.)

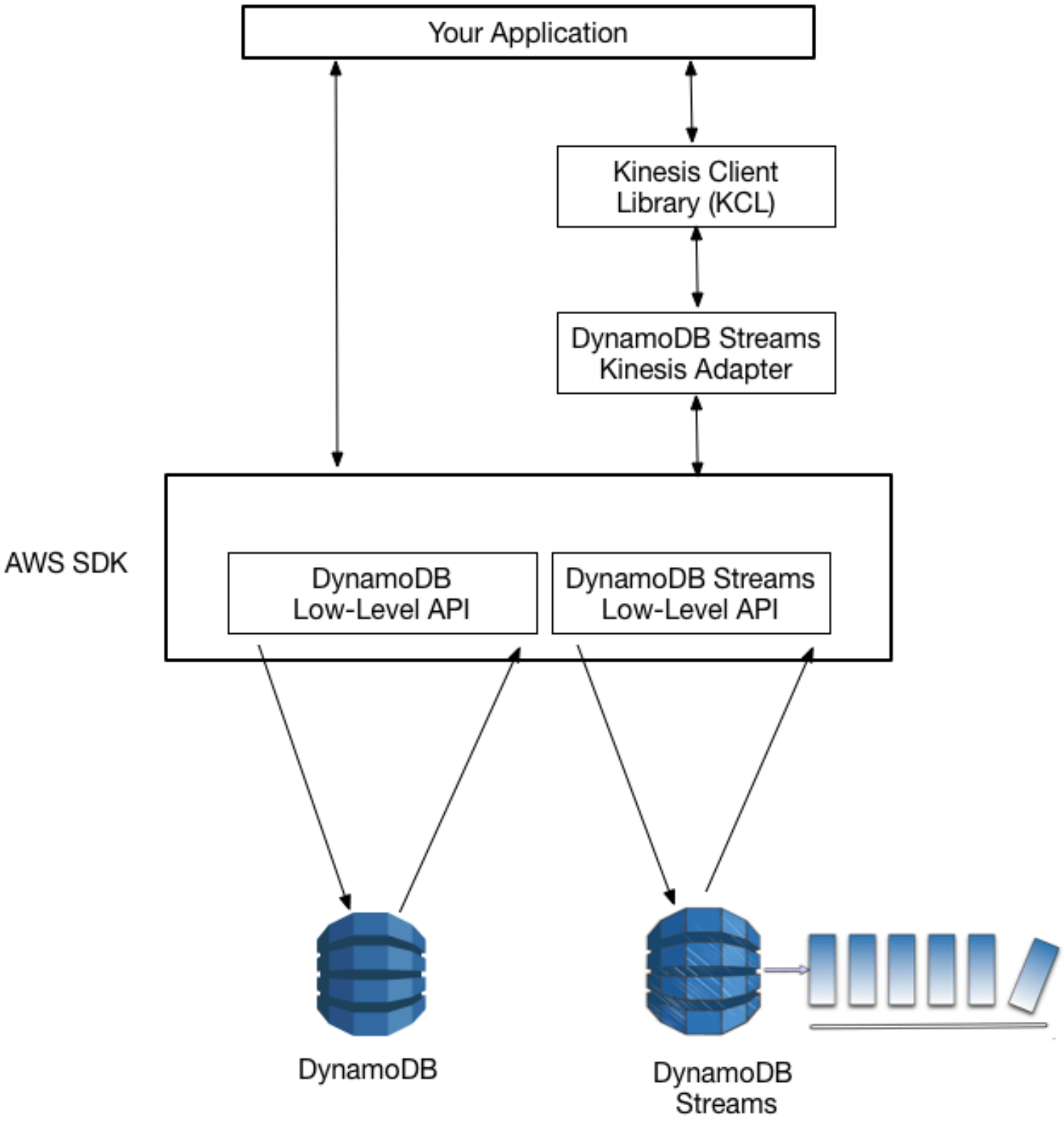
Als DynamoDB-Streams-Benutzer können Sie das Entwurfsmuster in der KCL zum Verarbeiten von DynamoDB-Streams-Shards und Stream-Datensätzen verwenden. Verwenden Sie dazu den DynamoDB-Streams-Kinesis-Adapter. Der Kinesis-Adapter implementiert die Kinesis-Data-Streams-Schnittstelle so, dass die KCL zum Verwenden und Verarbeiten von Datensätzen aus DynamoDB Streams eingesetzt werden kann. Anweisungen zur Einrichtung und Installation des DynamoDB Streams Kinesis Adapters finden Sie im Repository. GitHub

Sie können Anwendungen für Kinesis Data Streams mit der Kinesis Client Library (KCL) schreiben. Die KCL vereinfacht die Codierung durch Bereitstellen nützlicher Abstraktionen oberhalb der Low-Level-Kinesis-Data-Streams-API. Weitere Informationen zur KCL finden Sie im Entwickeln von Konsumenten mit der Kinesis-Client-Library im Entwicklerhandbuch zu Amazon Kinesis Data Streams.

DynamoDB empfiehlt die Verwendung von KCL Version 3.x mit AWS SDK for Java v2.x. Die aktuelle Version 1.x des DynamoDB Streams Kinesis Adapters mit AWS SDK für AWS SDK für Java v1.x wird während des gesamten Lebenszyklus weiterhin vollständig unterstützt, wie dies während der Übergangsphase gemäß den Wartungsrichtlinien für Tools vorgesehen war.AWS SDKs

> **Note**
>
> Die Versionen 1.x und 2.x der Amazon Kinesis Client Library (KCL) sind veraltet. KCL 1.x wird am 30. Januar 2026 verfügbar sein. end-of-support Wir empfehlen dringend, KCL-Anwendungen, die Version 1.x verwenden, vor dem 30. Januar 2026 zur neuesten KCL-Version zu migrieren. Die neueste KCL-Version finden Sie auf der Seite Amazon Kinesis Client Library unter. GitHub Weitere Informationen zur aktuellen Versionen finden Sie unter Use Kinesis Client Library. Weitere Informationen zur Migration von KCL 1.x zu KCL 3.x finden Sie unter „Migrating from KCL 1.x to KCL 3.x“.

Das folgende Diagramm zeigt, wie diese Bibliotheken miteinander interagieren.



Wenn Sie den DynamoDB-Streams-Kinesis-Adapter eingerichtet haben, können Sie die Entwicklung mit der KCL-Schnittstelle starten, wobei die API-Aufrufe nahtlos an den DynamoDB-Streams-Endpunkt gerichtet werden.

Beim Start der Anwendung wird die KCL aufgerufen, einen Worker zu instanziieren. Sie müssen dem Worker Konfigurationsinformationen für die Anwendung, wie z. B. den Stream-Deskriptor und die AWS Anmeldeinformationen, sowie den Namen einer von Ihnen angegebenen Datensatzprozessorklasse zur Verfügung stellen. Da der Code im Datensatzprozessor ausgeführt wird, erledigt der Worker die folgenden Aufgaben:

- Stellt eine Verbindung mit dem Stream her
- Listet die Shards innerhalb des Streams auf
- Überprüft und listet untergeordnete Shards eines geschlossenen übergeordneten Shards innerhalb des Streams auf
- Koordiniert Shard-Zuordnungen mit anderen Auftragnehmern (wenn vorhanden)
- Instanziiert einen Datensatzverarbeiter für jeden Shard, der verwaltet wird
- Ruft Datensätze aus dem Stream per Pull ab
- Skaliert die GetRecords API-Aufrufrate bei hohem Durchsatz (wenn der Nachholmodus konfiguriert ist)
- Überträgt per Push Datensätze an den entsprechenden Datensatzverarbeiter
- Verwendet Checkpoints für verarbeitete Datensätze
- Gleicht Shard-Auftragnehmer-Zuordnungen aus, wenn die Auftragnehmer-Instance Änderungen zählt
- Gleicht Shard-Worker-Zuordnungen aus, wenn Shards aufgeteilt werden

Der KCL-Adapter unterstützt den Catch-up-Modus, eine Funktion zur automatischen Anpassung der Anrufrate zur Bewältigung vorübergehender Durchsatzerhöhungen. Wenn die Verzögerung bei der Stream-Verarbeitung einen konfigurierbaren Schwellenwert (standardmäßig eine Minute) überschreitet, skaliert der Aufholmodus die GetRecords API-Aufruffrequenz um einen konfigurierbaren Wert (Standard 3x), um Datensätze schneller abzurufen, und kehrt dann zum Normalzustand zurück, sobald die Verzögerung nachlässt. Dies ist in Zeiten mit hohem Durchsatz nützlich, in denen DynamoDB-Schreibaktivitäten die Verbraucher bei Verwendung der Standardabfrageraten überfordern können. Der Nachholmodus kann über den Konfigurationsparameter aktiviert werden (Standardeinstellung falsch). `catchupEnabled`

> **Note**
>
> Eine Beschreibung der hier aufgeführten KCL-Konzepte finden Sie unter Entwickeln von Konsumenten mithilfe der Kinesis-Clientbibliothek im Amazon-Kinesis-Data-Streams-Entwicklerhandbuch.
> Weitere Informationen zur Verwendung von Streams mit finden Sie unter AWS Lambda DynamoDB Streams und -Trigger AWS Lambda

Migrieren von KCL 1.x zu KCL 3.x

-Übersicht

In dieser Anleitung wird erläutert, wie Sie Ihre Verbraucheranwendung von KCL 1.x zu KCL 3.x migrieren. Aufgrund der unterschiedlichen Architektur von KCL 1.x und KCL 3.x müssen für die Migration mehrere Komponenten aktualisiert werden, um die Kompatibilität sicherzustellen.

KCL 1.x verwendet im Vergleich zu KCL 3.x andere Klassen und Schnittstellen. Sie müssen zuerst die Klassen „Datensatzprozessor“, „Datensatzprozessor-Factory“ und „Worker“ in das KCL 3.x-kompatible Format migrieren und dann die Schritte für die Migration von KCL 1.x zu KCL 3.x ausführen.

Schritte zur Migration

Themen

- Schritt 1: Migrieren des Datensatzprozessors
- Schritt 2: Migrieren der Datensatzprozessor-Factory
- Schritt 3: Migrieren des Workers
- Schritt 4: Überblick über die Konfiguration und Empfehlungen für KCL 3.x
- Schritt 5: Migrieren von KCL 2.x zu KCL 3.x

Schritt 1: Migrieren des Datensatzprozessors

Das folgende Beispiel zeigt einen Datensatzprozessor, der für die Version KCL 1.x des DynamoDB-Streams-Kinesis-Adapters implementiert wurde:

```
package com.amazonaws.kcl;

import com.amazonaws.services.kinesis.clientlibrary.exceptions.InvalidStateException;
```

```
import com.amazonaws.services.kinesis.clientlibrary.exceptions.ShutdownException;
import
 com.amazonaws.services.kinesis.clientlibrary.interfaces.IRecordProcessorCheckpointer;
import com.amazonaws.services.kinesis.clientlibrary.interfaces.v2.IRecordProcessor;
import
 com.amazonaws.services.kinesis.clientlibrary.interfaces.v2.IShutdownNotificationAware;
import com.amazonaws.services.kinesis.clientlibrary.lib.worker.ShutdownReason;
import com.amazonaws.services.kinesis.clientlibrary.types.InitializationInput;
import com.amazonaws.services.kinesis.clientlibrary.types.ProcessRecordsInput;
import com.amazonaws.services.kinesis.clientlibrary.types.ShutdownInput;

public class StreamsRecordProcessor implements IRecordProcessor,
 IShutdownNotificationAware {
    @Override
    public void initialize(InitializationInput initializationInput) {
        //
        // Setup record processor
        //
    }

    @Override
    public void processRecords(ProcessRecordsInput processRecordsInput) {
        for (Record record : processRecordsInput.getRecords()) {
            String data = new String(record.getData().array(),
 Charset.forName("UTF-8"));
            System.out.println(data);
            if (record instanceof RecordAdapter) {
                // record processing and checkpointing logic
            }
        }
    }

    @Override
    public void shutdown(ShutdownInput shutdownInput) {
        if (shutdownInput.getShutdownReason() == ShutdownReason.TERMINATE) {
            try {
                shutdownInput.getCheckpointer().checkpoint();
            } catch (ShutdownException | InvalidStateException e) {
                throw new RuntimeException(e);
            }
        }
    }

    @Override
```

```
    public void shutdownRequested(IRecordProcessorCheckpointer checkpointer) {
        try {
            checkpointer.checkpoint();
        } catch (ShutdownException | InvalidStateException e) {
            //
            // Swallow exception
            //
            e.printStackTrace();
        }
    }
}
```

Um die RecordProcessor Klasse zu migrieren

1. Ändern Sie die Schnittstellen von `com.amazonaws.services.kinesis.clientlibrary.interfaces.v2.IRecordProcessor` und `com.amazonaws.services.kinesis.clientlibrary.interfaces.v2.IShutdownNotificat` folgendermaßen zu `com.amazonaws.services.dynamodbv2.streamsadapter.processor.DynamoDBStreamsSha`

   ```
   // import
    com.amazonaws.services.kinesis.clientlibrary.interfaces.v2.IRecordProcessor;
   // import
    com.amazonaws.services.kinesis.clientlibrary.interfaces.v2.IShutdownNotificationAware;

   import
    com.amazonaws.services.dynamodbv2.streamsadapter.processor.DynamoDBStreamsShardRecordProce
   ```

2. Aktualisieren Sie die Importanweisungen für die Methoden `initialize` und `processRecords`:

   ```
   // import com.amazonaws.services.kinesis.clientlibrary.types.InitializationInput;
   import software.amazon.kinesis.lifecycle.events.InitializationInput;

   // import com.amazonaws.services.kinesis.clientlibrary.types.ProcessRecordsInput;
   import
    com.amazonaws.services.dynamodbv2.streamsadapter.model.DynamoDBStreamsProcessRecordsInput;
   ```

3. Ersetzen Sie die Methode `shutdownRequested` durch die folgenden neuen Methoden: `leaseLost`, `shardEnded` und `shutdownRequested`.

```
//    @Override
//    public void shutdownRequested(IRecordProcessorCheckpointer checkpointer) {
//        //
//        // This is moved to shardEnded(...) and
 shutdownRequested(ShutdownReauestedInput)
//        //
//        try {
//            checkpointer.checkpoint();
//        } catch (ShutdownException | InvalidStateException e) {
//            //
//            // Swallow exception
//            //
//            e.printStackTrace();
//        }
//    }

    @Override
    public void leaseLost(LeaseLostInput leaseLostInput) {

    }

    @Override
    public void shardEnded(ShardEndedInput shardEndedInput) {
        try {
            shardEndedInput.checkpointer().checkpoint();
        } catch (ShutdownException | InvalidStateException e) {
            //
            // Swallow the exception
            //
            e.printStackTrace();
        }
    }

    @Override
    public void shutdownRequested(ShutdownRequestedInput shutdownRequestedInput) {
        try {
            shutdownRequestedInput.checkpointer().checkpoint();
        } catch (ShutdownException | InvalidStateException e) {
            //
            // Swallow the exception
            //
            e.printStackTrace();
        }
```

```
    }
```

Nachstehend finden Sie die aktualisierte Version der Datensatzprozessorklasse:

```
package com.amazonaws.codesamples;

import software.amazon.kinesis.exceptions.InvalidStateException;
import software.amazon.kinesis.exceptions.ShutdownException;
import software.amazon.kinesis.lifecycle.events.InitializationInput;
import software.amazon.kinesis.lifecycle.events.LeaseLostInput;
import
 com.amazonaws.services.dynamodbv2.streamsadapter.model.DynamoDBStreamsProcessRecordsInput;
import software.amazon.kinesis.lifecycle.events.ShardEndedInput;
import software.amazon.kinesis.lifecycle.events.ShutdownRequestedInput;
import
 software.amazon.dynamodb.streamsadapter.processor.DynamoDBStreamsShardRecordProcessor;
import
 software.amazon.dynamodb.streamsadapter.adapter.DynamoDBStreamsKinesisClientRecord;
import
 com.amazonaws.services.dynamodbv2.streamsadapter.processor.DynamoDBStreamsShardRecordProcessor
import
 com.amazonaws.services.dynamodbv2.streamsadapter.adapter.DynamoDBStreamsClientRecord;
import software.amazon.awssdk.services.dynamodb.model.Record;

public class StreamsRecordProcessor implements DynamoDBStreamsShardRecordProcessor {

    @Override
    public void initialize(InitializationInput initializationInput) {

    }

    @Override
    public void processRecords(DynamoDBStreamsProcessRecordsInput processRecordsInput)
 {
        for (DynamoDBStreamsKinesisClientRecord record: processRecordsInput.records())
            Record ddbRecord = record.getRecord();
            // processing and checkpointing logic for the ddbRecord
        }
    }

    @Override
    public void leaseLost(LeaseLostInput leaseLostInput) {
```

```
    }

    @Override
    public void shardEnded(ShardEndedInput shardEndedInput) {
        try {
            shardEndedInput.checkpointer().checkpoint();
        } catch (ShutdownException | InvalidStateException e) {
            //
            // Swallow the exception
            //
            e.printStackTrace();
        }
    }

    @Override
    public void shutdownRequested(ShutdownRequestedInput shutdownRequestedInput) {
        try {
            shutdownRequestedInput.checkpointer().checkpoint();
        } catch (ShutdownException | InvalidStateException e) {
            //
            // Swallow the exception
            //
            e.printStackTrace();
        }
    }
}
```

> **Note**
>
> DynamoDB Streams Kinesis Adapter verwendet SDKv2 jetzt das Record-Modell. In SDKv2 geben komplexe `AttributeValue` Objekte (`BS`,,, `NS` `ML`,`SS`) niemals Null zurück. Verwenden Sie die Methoden `hasBs()`, `hasNs()`, `hasM()`, `hasL()`, `hasSs()`, um zu überprüfen, ob diese Werte existieren.

Schritt 2: Migrieren der Datensatzprozessor-Factory

Die Datensatzprozessor-Factory ist für das Erstellen von Prozessoren verantwortlich, wenn eine Lease erworben wird. Nachfolgend sehen Sie ein Beispiel für eine KCL-1.x-Factory:

```
package com.amazonaws.codesamples;
```

```
import software.amazon.dynamodb.AmazonDynamoDB;
import com.amazonaws.services.kinesis.clientlibrary.interfaces.v2.IRecordProcessor;
import
 com.amazonaws.services.kinesis.clientlibrary.interfaces.v2.IRecordProcessorFactory;

public class StreamsRecordProcessorFactory implements IRecordProcessorFactory {

    @Override
    public IRecordProcessor createProcessor() {
        return new StreamsRecordProcessor(dynamoDBClient, tableName);
    }
}
```

So migrieren Sie die **`RecordProcessorFactory`**

- Ändern Sie die implementierte Schnittstelle von `com.amazonaws.services.kinesis.clientlibrary.interfaces.v2.IRecordProcessorFa` folgendermaßen zu `software.amazon.kinesis.processor.ShardRecordProcessorFactory`:

  ```
  // import
   com.amazonaws.services.kinesis.clientlibrary.interfaces.v2.IRecordProcessor;
  import software.amazon.kinesis.processor.ShardRecordProcessor;

  // import
   com.amazonaws.services.kinesis.clientlibrary.interfaces.v2.IRecordProcessorFactory;
  import software.amazon.kinesis.processor.ShardRecordProcessorFactory;

  // public class TestRecordProcessorFactory implements IRecordProcessorFactory {
  public class StreamsRecordProcessorFactory implements ShardRecordProcessorFactory {

  Change the return signature for createProcessor.

  // public IRecordProcessor createProcessor() {
  public ShardRecordProcessor shardRecordProcessor() {
  ```

Nachfolgend sehen Sie ein Beispiel für die Datensatzprozessor-Factory in 3.0:

```
package com.amazonaws.codesamples;
```

```
import software.amazon.kinesis.processor.ShardRecordProcessor;
import software.amazon.kinesis.processor.ShardRecordProcessorFactory;

public class StreamsRecordProcessorFactory implements ShardRecordProcessorFactory {

    @Override
    public ShardRecordProcessor shardRecordProcessor() {
        return new StreamsRecordProcessor();
    }
}
```

Schritt 3: Migrieren des Workers

In Version 3.0 der KCL wird die Worker-Klasse durch eine neue Klasse namens Scheduler ersetzt. Nachfolgend sehen Sie ein Beispiel für einen KCL-1.x-Worker.

```
final KinesisClientLibConfiguration config = new KinesisClientLibConfiguration(...)
final IRecordProcessorFactory recordProcessorFactory = new RecordProcessorFactory();
final Worker worker = StreamsWorkerFactory.createDynamoDbStreamsWorker(
        recordProcessorFactory,
        workerConfig,
        adapterClient,
        amazonDynamoDB,
        amazonCloudWatchClient);
```

So migrieren Sie den Worker

1. Ändern Sie die `import`-Anweisung für die `Worker`-Klasse in die Import-Anweisungen für die Klassen `Scheduler` und `ConfigsBuilder`.

   ```
   // import com.amazonaws.services.kinesis.clientlibrary.lib.worker.Worker;
   import software.amazon.kinesis.coordinator.Scheduler;
   import software.amazon.kinesis.common.ConfigsBuilder;
   ```

2. Importieren Sie `StreamTracker` und ändern Sie den Import von `StreamsWorkerFactory` zu `StreamsSchedulerFactory`.

   ```
   import software.amazon.kinesis.processor.StreamTracker;
   // import software.amazon.dynamodb.streamsadapter.StreamsWorkerFactory;
   import software.amazon.dynamodb.streamsadapter.StreamsSchedulerFactory;
   ```

3. Wählen Sie die Position, von der aus die Anwendung gestartet werden soll. Möglich sind `TRIM_HORIZON` oder `LATEST`.

```
import software.amazon.kinesis.common.InitialPositionInStream;
import software.amazon.kinesis.common.InitialPositionInStreamExtended;
```

4. Erstellen Sie eine `StreamTracker`-Instance.

```
StreamTracker streamTracker = StreamsSchedulerFactory.createSingleStreamTracker(
        streamArn,

 InitialPositionInStreamExtended.newInitialPosition(InitialPositionInStream.TRIM_HORIZON)
);
```

5. Erstellen Sie das Objekt `AmazonDynamoDBStreamsAdapterClient`.

```
import software.amazon.dynamodb.streamsadapter.AmazonDynamoDBStreamsAdapterClient;
import software.amazon.awssdk.auth.credentials.AwsCredentialsProvider;
import software.amazon.awssdk.auth.credentials.DefaultCredentialsProvider;

...

AwsCredentialsProvider credentialsProvider = DefaultCredentialsProvider.create();

AmazonDynamoDBStreamsAdapterClient adapterClient = new
 AmazonDynamoDBStreamsAdapterClient(
        credentialsProvider, awsRegion);
```

6. Erstellen Sie das Objekt `ConfigsBuilder`.

```
import software.amazon.kinesis.common.ConfigsBuilder;

...
ConfigsBuilder configsBuilder = new ConfigsBuilder(
                streamTracker,
                applicationName,
                adapterClient,
                dynamoDbAsyncClient,
                cloudWatchAsyncClient,
                UUID.randomUUID().toString(),
                new StreamsRecordProcessorFactory());
```

7. Erstellen Sie den `Scheduler` mithilfe von `ConfigsBuilder`, wie im folgenden Beispiel gezeigt:

```
import java.util.UUID;

import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.dynamodb.DynamoDbAsyncClient;
import software.amazon.awssdk.services.cloudwatch.CloudWatchAsyncClient;
import software.amazon.awssdk.services.kinesis.KinesisAsyncClient;

import software.amazon.kinesis.common.KinesisClientUtil;
import software.amazon.kinesis.coordinator.Scheduler;

...


DynamoDbAsyncClient dynamoClient =
 DynamoDbAsyncClient.builder().region(region).build();
CloudWatchAsyncClient cloudWatchClient =
 CloudWatchAsyncClient.builder().region(region).build();


DynamoDBStreamsPollingConfig pollingConfig = new
 DynamoDBStreamsPollingConfig(adapterClient);
pollingConfig.idleTimeBetweenReadsInMillis(idleTimeBetweenReadsInMillis);

// Use ConfigsBuilder to configure settings
RetrievalConfig retrievalConfig = configsBuilder.retrievalConfig();
retrievalConfig.retrievalSpecificConfig(pollingConfig);

CoordinatorConfig coordinatorConfig = configsBuilder.coordinatorConfig();
coordinatorConfig.clientVersionConfig(CoordinatorConfig.ClientVersionConfig.CLIENT_VERSION_

Scheduler scheduler = StreamsSchedulerFactory.createScheduler(
                configsBuilder.checkpointConfig(),
                coordinatorConfig,
                configsBuilder.leaseManagementConfig(),
                configsBuilder.lifecycleConfig(),
                configsBuilder.metricsConfig(),
                configsBuilder.processorConfig(),
                retrievalConfig,
                adapterClient
        );
```

> **Important**
>
> Die Einstellung `CLIENT_VERSION_CONFIG_COMPATIBLE_WITH_2X` gewährleistet die Kompatibilität zwischen dem DynamoDB-Streams-Kinesis-Adapter für KCL v3 und KCL v1, nicht zwischen KCL v2 und v3.

Schritt 4: Überblick über die Konfiguration und Empfehlungen für KCL 3.x

Eine ausführliche Beschreibung der nach KCL 1.x eingeführten Konfigurationen, die für KCL 3.x relevant sind, finden Sie unter KCL Configurations und KCL Migration Client Configuration.

> **Important**
>
> Anstatt Objekte von `checkpointConfig`, `coordinatorConfig`, `leaseManagementConfig`, `metricsConfig`, `processorConfig` und `retrievalConfig` direkt zu erstellen, empfehlen wir, `ConfigsBuilder` zu verwenden, um Konfigurationen in KCL 3.x und aktuelleren Versionen einzustellen. Das verhindert Probleme mit der Scheduler-Initialisierung. `ConfigsBuilder` bietet eine flexiblere und wartungsfreundlichere Methode zur Konfiguration Ihrer KCL-Anwendung.

Konfigurationen mit dem Standardwert für Aktualisierungen in KCL 3.x

`billingMode`

In der KCL-Version 1.x ist der Standardwert für `billingMode` auf `PROVISIONED` eingestellt. Bei der KCL-Version 3.x ist die Standardeinstellung für `billingMode` jedoch `PAY_PER_REQUEST` (On-Demand-Modus). Wir empfehlen Ihnen, den On-Demand-Kapazitätsmodus für Ihre Leasetabelle zu verwenden, um die Kapazität automatisch an die Nutzung anzupassen. Anleitungen zur Verwendung der bereitgestellten Kapazität für Ihre Leasetabellen finden Sie unter Best practices for the lease table with provisioned capacity mode.

`idleTimeBetweenReadsInMillis`

In der KCL-Version 1.x ist der Standardwert für `idleTimeBetweenReadsInMillis` auf 1 000 (oder 1 Sekunde) eingestellt. Die KCL-Version 3.x legt den Standardwert für `dleTimeBetweenReadsInMillis` auf 1 500 (oder 1,5 Sekunden) fest. Der Amazon-DynamoDB-Streams-Kinesis-Adapter überschreibt den Standardwert jedoch mit 1 000 (oder 1 Sekunde).

Neue Konfigurationen in KCL 3.x

`leaseAssignmentIntervalMillis`

Diese Konfiguration definiert das Zeitintervall, bis die Verarbeitung neu erkannter Shards beginnt. Es wird nach der Formel 1,5 × `leaseAssignmentIntervalMillis` berechnet. Wenn diese Einstellung nicht explizit konfiguriert ist, beträgt das Zeitintervall standardmäßig 1,5 × `failoverTimeMillis`. Die Verarbeitung neuer Shards beinhaltet das Scannen der Leasetabelle und das Abfragen eines globalen sekundären Index (GSI) in der Leasetabelle. Eine Absenkung des `leaseAssignmentIntervalMillis`-Werts erhöht die Häufigkeit dieser Scan- und Abfragevorgänge, was zu höheren DynamoDB-Kosten führt. Wir empfehlen, diesen Wert auf 2 000 (oder 2 Sekunden) einzustellen, um die Verzögerung bei der Verarbeitung neuer Shards zu minimieren.

`shardConsumerDispatchPollIntervalMillis`

Diese Konfiguration definiert das Intervall zwischen aufeinanderfolgenden Abfragen durch den Shard-Verbraucher, um Zustandsübergänge auszulösen. In KCL-Version 1.x wurde dieses Verhalten durch den Parameter `idleTimeInMillis` gesteuert, der nicht als konfigurierbare Einstellung verfügbar war. Bei KCL-Version 3.x empfehlen wir, diese Konfiguration auf den Wert einzustellen, der in Ihrer KCL-Version 1.x für `idleTimeInMillis` verwendet wurde.

Schritt 5: Migrieren von KCL 2.x zu KCL 3.x

Um für einen reibungslosen Übergang und Kompatibilität mit der neuesten Version der Kinesis Client Library (KCL) zu sorgen, folgen Sie den Schritten 5–8 der Anleitung für das Upgrade von KCL 2.x auf KCL 3.x im Migrationsleitfaden.

Informationen zur Fehlerbehebung in KCL 3.x finden Sie unter Troubleshooting KCL consumer applications.

Rollback zu einer früheren KCL-Version

In diesem Abschnitt wird erläutert, wie Sie Ihre Verbraucheranwendung auf die vorherige KCL-Version zurücksetzen können. Der Rollback-Prozess besteht aus zwei Schritten:

1. Führen Sie das KCL Migration Tool aus.
2. Stellen Sie den Code der vorherigen KCL-Version erneut bereit.

Schritt 1: Ausführen des KCL Migration Tools

Wenn Sie zur vorherigen KCL-Version zurückkehren möchten, müssen Sie das KCL Migration Tool ausführen. Das Tool erfüllt zwei wichtige Aufgaben:

- Es entfernt die Metadatentabelle für Worker-Metriken und den globalen sekundären Index aus der Leasetabelle in DynamoDB. Diese Artefakte werden von KCL 3.x erstellt, werden aber nicht benötigt, wenn Sie zur vorherigen Version zurückkehren.
- Dadurch werden alle Worker in einem mit KCL 1.x kompatiblen Modus ausgeführt und verwenden wieder den Load-Balancing-Algorithmus aus früheren KCL-Versionen. Eventuelle Probleme mit dem neuen Load-Balancing-Algorithmus in KCL 3.x werden dadurch sofort behoben.

> **Important**
>
> Die Koordinatorstatustabelle in DynamoDB muss vorhanden sein und darf während des Migrations-, Rollback- und Rollforward-Prozesses nicht gelöscht werden.

> **Note**
>
> Es ist wichtig, dass alle Worker in Ihrer Verbraucheranwendung zu einem bestimmten Zeitpunkt denselben Load-Balancing-Algorithmus verwenden. Das KCL Migration Tool sorgt dafür, dass alle Worker in Ihrer KCL-3.x-Verbraucheranwendung in den KCL-1.x-kompatiblen Modus wechseln. So führen alle Worker während des Rollbacks zur vorherigen KCL-Version denselben Load-Balancing-Algorithmus aus.

Sie können das KCL-Migrationstool im Skriptverzeichnis des GitHubKCL-Repositorys herunterladen. Führen Sie das Skript über einen Worker oder Host aus, der berechtigt ist, in die Koordinatorstatustabelle, die Tabelle mit den Worker-Metriken und die Leasetabelle zu schreiben. Stellen Sie sicher, dass die entsprechenden IAM-Berechtigungen für KCL-Verbraucheranwendungen konfiguriert sind. Führen Sie das Skript mit dem folgenden Befehl nur einmal pro KCL-Anwendung aus:

```
python3 ./KclMigrationTool.py --region region --mode rollback [--
application_name applicationName] [--lease_table_name leaseTableName]
```

```
 [--coordinator_state_table_name coordinatorStateTableName] [--
worker_metrics_table_name workerMetricsTableName]
```

Parameters

`--region`

Ersetzen Sie durch *region* Ihr. AWS-Region

`--application_name`

Dieser Parameter ist erforderlich, wenn Sie Standardnamen für Ihre DynamoDB-Metadatentabellen (Leasetabelle, Koordinatorstatustabelle und Worker-Metriktabelle) verwenden. Wenn Sie benutzerdefinierte Namen für diese Tabellen festgelegt haben, können Sie diesen Parameter weglassen. *applicationName*Ersetzen Sie durch Ihren tatsächlichen KCL-Anwendungsnamen. Das Tool verwendet diesen Namen, um die Standardtabellennamen abzuleiten, wenn keine benutzerdefinierten Namen angegeben werden.

`--lease_table_name`

Dieser Parameter wird benötigt, wenn Sie in Ihrer KCL-Konfiguration einen benutzerdefinierten Namen für die Leasetabelle festgelegt haben. Wenn Sie den Standardtabellennamen verwenden, können Sie diesen Parameter weglassen. *leaseTableName*Ersetzen Sie durch den benutzerdefinierten Tabellennamen, den Sie für Ihre Leasingtabelle angegeben haben.

`--coordinator_state_table_name`

Dieser Parameter wird benötigt, wenn Sie in Ihrer KCL-Konfiguration einen benutzerdefinierten Namen für die Koordinatorstatustabelle festgelegt haben. Wenn Sie den Standardtabellennamen verwenden, können Sie diesen Parameter weglassen. *coordinatorStateTableName*Ersetzen Sie es durch den benutzerdefinierten Tabellennamen, den Sie für Ihre Koordinatorstatentabelle angegeben haben.

`--worker_metrics_table_name`

Dieser Parameter wird benötigt, wenn Sie in Ihrer KCL-Konfiguration einen benutzerdefinierten Namen für die Worker-Metriktabelle festgelegt haben. Wenn Sie den Standardtabellennamen verwenden, können Sie diesen Parameter weglassen. *workerMetricsTableName*Ersetzen Sie es durch den Namen der benutzerdefinierten Tabelle, den Sie für Ihre Tabelle mit Arbeitskennzahlen angegeben haben.

Schritt 2: Erneutes Bereitstellen des Codes mit der vorherigen KCL-Version

> **Important**
>
> Jede Erwähnung von Version 2.x in der vom KCL Migration Tool generierten Ausgabe sollte als Bezugnahme auf KCL-Version 1.x interpretiert werden. Durch die Ausführung des Skripts erfolgt kein vollständiges Rollback. Es wird lediglich der Load-Balancing-Algorithmus auf den in KCL Version 1.x verwendeten umgestellt.

Nachdem das KCL Migration Tool den Rollback ausgeführt hat, wird eine der folgenden Meldungen angezeigt:

Nachricht 1

„Rollback completed. Your application was running 2x compatible functionality. Please rollback to your previous application binaries by deploying the code with your previous KCL version.“

Erforderliche Maßnahme: Dies bedeutet, dass Ihre Worker im KCL-1.x-kompatiblen Modus ausgeführt wurden. Stellen Sie den Code für Ihre Worker erneut mit der vorherigen KCL-Version bereit.

Nachricht 2

„Rollback completed. Your KCL Application was running 3x functionality and will rollback to 2x compatible functionality. If you don't see mitigation after a short period of time, please rollback to your previous application binaries by deploying the code with your previous KCL version.“

Erforderliche Maßnahme: Dies bedeutet, dass Ihre Worker im KCL-3.x-kompatiblen Modus ausgeführt wurden und das KCL Migration Tool alle Worker auf den KCL-1.x-kompatiblen Modus zurückgesetzt hat. Stellen Sie den Code für Ihre Worker erneut mit der vorherigen KCL-Version bereit.

Nachricht 3

„Application was already rolled back. Alle KCLv3 Ressourcen, die gelöscht werden könnten, wurden gelöscht, um Gebühren zu vermeiden, bis die Anwendung im Rahmen der Migration weiterentwickelt werden kann.“

Erforderliche Maßnahme: Dies bedeutet, dass Ihre Worker bereits in den KCL-1.x-kompatiblen Modus zurückversetzt wurden. Stellen Sie den Code für Ihre Worker erneut mit der vorherigen KCL-Version bereit.

Rollforward zu KCL 3.x nach einem Rollback

In diesem Abschnitt wird erläutert, wie Sie Ihre Verbraucheranwendung nach einem Rollback wieder auf KCL 3.x umstellen können. Wenn Sie einen Rollforward durchführen müssen, sind zwei Schritte erforderlich:

1. Führen Sie das KCL Migration Tool aus.
2. Stellen Sie den Code mit KCL 3.x bereit.

Schritt 1: Ausführen des KCL Migration Tools

Führen Sie das KCL Migration Tool mit dem folgenden Befehl aus, um KCL 3.x wiederherzustellen:

```
python3 ./KclMigrationTool.py --region region --mode rollforward [--
application_name applicationName] [--
coordinator_state_table_name coordinatorStateTableName]
```

Parameters

`--region`

Ersetzen Sie *region* durch Ihre AWS-Region.

`--application_name`

Dieser Parameter ist erforderlich, wenn Sie Standardnamen für die Koordinatorstatustabelle verwenden. Wenn Sie benutzerdefinierte Namen für diese Tabelle festgelegt haben, können Sie diesen Parameter weglassen. *applicationName*Ersetzen Sie durch Ihren tatsächlichen KCL-Anwendungsnamen. Das Tool verwendet diesen Namen, um die Standardtabellennamen abzuleiten, wenn keine benutzerdefinierten Namen angegeben werden.

`--coordinator_state_table_name`

Dieser Parameter wird benötigt, wenn Sie in Ihrer KCL-Konfiguration einen benutzerdefinierten Namen für die Koordinatorstatustabelle festgelegt haben. Wenn Sie den Standardtabellennamen verwenden, können Sie diesen Parameter weglassen. *coordinatorStateTableName*Ersetzen Sie durch den benutzerdefinierten Tabellennamen, den Sie für Ihre Koordinatorstatustabelle angegeben haben.

Nachdem Sie das Migrationstool im Roll-Forward-Modus ausgeführt haben, erstellt KCL die folgenden DynamoDB-Ressourcen, die für KCL 3.x erforderlich sind:

- einen globalen sekundären Index für die Leasetabelle
- eine Tabelle mit Worker-Metriken

Schritt 2: Bereitstellen des Codes mit KCL 3.x

Nachdem Sie den Rollforward mit dem KCL Migration Tool ausgeführt haben, stellen Sie Ihren Code mit KCL 3.x für Ihre Worker bereit. Informationen zum Abschließen Ihrer Migration finden Sie unter Schritt 8: Abschließen der Migration.

Walkthrough: DynamoDB-Streams-Kinesis-Adapter

In diesem Abschnitt wird eine Anleitung für eine Java-Anwendung gegeben, die die Amazon-Kinesis-Client-Library und den Amazon-DynamoDB-Streams-Kinesis-Adapter verwendet. Die Anwendung zeigt ein Beispiel für die Datenreplikation, wobei Schreibaktivitäten einer Tabelle auf eine zweite Tabelle angewendet werden und die Inhalte beider Tabellen synchron bleiben. Sie finden den Quellcode unter Vollständiges Programm: DynamoDB-Streams-Kinesis-Adapter.

Das Programm führt Folgendes aus:

1. Erstellt zwei DynamoDB-Tabellen namens `KCL-Demo-src` und `KCL-Demo-dst`. Für jede dieser Tabellen ist ein Stream aktiviert.
2. Generiert Aktualisierungsaktivitäten in der Quelltabelle durch Hinzufügen, Aktualisieren und Löschen von Elementen. Dies bewirkt, dass Daten in den Tabellenstream geschrieben werden.
3. Liest die Datensätze aus dem Stream, rekonstruiert diese als DynamoDB-Anforderungen und wendet die Anforderungen auf die Zieltabelle an.
4. Scannt die Quell- und Zieltabellen, um sicherzustellen, dass ihre Inhalte identisch sind.
5. Bereinigt die Daten durch Löschen der Tabellen.

Diese Schritte werden in den folgenden Abschnitten beschrieben und die vollständige Anwendung wird am Ende der Anleitung angezeigt.

Themen

- Schritt 1: Erstellen einer DynamoDB-Tabelle
- Schritt 2: Generieren von Aktualisierungsaktivitäten in der Quelltabelle
- Schritt 3: Verarbeiten des Streams
- Schritt 4: Sicherstellen, dass beide Tabellen über identische Inhalte verfügen

- Schritt 5: Bereinigen
- Vollständiges Programm: DynamoDB-Streams-Kinesis-Adapter

Schritt 1: Erstellen einer DynamoDB-Tabelle

Im ersten Schritt erstellen Sie zwei DynamoDB-Tabellen—eine Quelltabelle und eine Zieltabelle. Der `StreamViewType` des Streams der Quelltabelle lautet `NEW_IMAGE`. Das bedeutet, dass sobald ein Element in dieser Tabelle geändert wird, das Image des Elements nach der Änderung in den Stream geschrieben wird. So verfolgt der Stream alle Schreibaktivitäten der Tabelle.

Das folgende Beispiel zeigt den Code für das Erstellen von beiden Tabellen.

```
java.util.List<AttributeDefinition> attributeDefinitions = new
 ArrayList<AttributeDefinition>();
attributeDefinitions.add(new
 AttributeDefinition().withAttributeName("Id").withAttributeType("N"));

java.util.List<KeySchemaElement> keySchema = new ArrayList<KeySchemaElement>();
keySchema.add(new
 KeySchemaElement().withAttributeName("Id").withKeyType(KeyType.HASH)); // Partition

  // key

ProvisionedThroughput provisionedThroughput = new
 ProvisionedThroughput().withReadCapacityUnits(2L)
    .withWriteCapacityUnits(2L);

StreamSpecification streamSpecification = new StreamSpecification();
streamSpecification.setStreamEnabled(true);
streamSpecification.setStreamViewType(StreamViewType.NEW_IMAGE);
CreateTableRequest createTableRequest = new
 CreateTableRequest().withTableName(tableName)
    .withAttributeDefinitions(attributeDefinitions).withKeySchema(keySchema)

 .withProvisionedThroughput(provisionedThroughput).withStreamSpecification(streamSpecification)
```

Schritt 2: Generieren von Aktualisierungsaktivitäten in der Quelltabelle

Im nächsten Schritt erstellen Sie einige Schreibaktivitäten in der Quelltabelle. Während diese Aktivitäten ausgeführt werden, wird der Stream der Quelltabelle nahezu in Echtzeit ebenfalls aktualisiert.

Die Anwendung definiert die Hilfsprogrammklasse mit Methoden zum Aufrufen der API-Operationen `PutItem`, `UpdateItem` und `DeleteItem` zum Schreiben der Daten. Das folgende Codebeispiel zeigt, wie diese Methoden verwendet werden.

```
StreamsAdapterDemoHelper.putItem(dynamoDBClient, tableName, "101", "test1");
StreamsAdapterDemoHelper.updateItem(dynamoDBClient, tableName, "101", "test2");
StreamsAdapterDemoHelper.deleteItem(dynamoDBClient, tableName, "101");
StreamsAdapterDemoHelper.putItem(dynamoDBClient, tableName, "102", "demo3");
StreamsAdapterDemoHelper.updateItem(dynamoDBClient, tableName, "102", "demo4");
StreamsAdapterDemoHelper.deleteItem(dynamoDBClient, tableName, "102");
```

Schritt 3: Verarbeiten des Streams

Das Programm beginnt mit der Verarbeitung des Streams. Der DynamoDB Streams Kinesis Adapter fungiert als transparente Ebene zwischen der KCL und dem DynamoDB Streams-Endpunkt, sodass der Code die KCL vollständig nutzen kann, statt DynamoDB-Streams-Low-Level-Aufrufe tätigen zu müssen. Das Programm führt die folgenden Aufgaben durch:

- Es definiert eine Datensatzprozessor-Klasse, `StreamsRecordProcessor`, mit Methoden, die mit der KCL-Schnittstellendefinition übereinstimmen: `initialize`, `processRecords` und `shutdown`. Die `processRecords`-Methode enthält die Logik, die zum Lesen von der Quelltabelle des Streams und zum Schreiben in die Zieltabelle erforderlich ist.
- Sie definiert eine ClassFactory für die Datensatzprozessor-Klasse (`StreamsRecordProcessorFactory`). Dies ist für Java-Programme, die die KCL verwenden, erforderlich.
- Die Methode instanziiert eine neue KCL `Worker`, die der Class Factory zugeordnet ist.
- Sie fährt `Worker` herunter, wenn die Datensatzverarbeitung abgeschlossen ist.

Aktivieren Sie optional den Aufholmodus in Ihrer Streams-KCL-Adapterkonfiguration, um die GetRecords API-Aufrufrate automatisch um das Dreifache zu skalieren (Standard), wenn die

Verzögerung der Stream-Verarbeitung eine Minute überschreitet (Standard), sodass Ihr Stream-Consumer hohe Durchsatzspitzen in Ihrer Tabelle bewältigen kann.

Weitere Informationen zur KCL-Schnittstellendefinition finden Sie unter Entwickeln von Konsumenten mithilfe der Kinesis-Clientbibliothek im Amazon-Kinesis-Data-Streams-Entwicklerhandbuch.

Das folgende Codebeispiel zeigt die Hauptschleife in `StreamsRecordProcessor`. Die `case`-Anweisung bestimmt, welche Aktion basierend auf dem `OperationType`, der im Stream-Datensatz erscheint, durchgeführt werden soll.

```
for (Record record : records) {
    String data = new String(record.getData().array(), Charset.forName("UTF-8"));
    System.out.println(data);
    if (record instanceof RecordAdapter) {
                software.amazon.dynamodb.model.Record streamRecord = ((RecordAdapter)
 record)
                    .getInternalObject();

                switch (streamRecord.getEventName()) {
                    case "INSERT":
                    case "MODIFY":
                        StreamsAdapterDemoHelper.putItem(dynamoDBClient, tableName,
                            streamRecord.getDynamodb().getNewImage());
                        break;
                    case "REMOVE":
                        StreamsAdapterDemoHelper.deleteItem(dynamoDBClient, tableName,
                            streamRecord.getDynamodb().getKeys().get("Id").getN());
                }
    }
    checkpointCounter += 1;
    if (checkpointCounter % 10 == 0) {
        try {
            checkpointer.checkpoint();
        }
        catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

Schritt 4: Sicherstellen, dass beide Tabellen über identische Inhalte verfügen

An diesem Punkt sind die Inhalte der Quell- und Zieltabellen synchronisiert. Die Anwendung gibt `Scan`-Anforderungen für beide Tabellen aus, um sicherzustellen, dass ihre Inhalte identisch sind.

Die `DemoHelper`-Klasse enthält eine `ScanTable`-Methode, die die `Scan`-Low-Level-API aufruft. Das Verfahren wird im folgenden Beispiel beschrieben.

```
if (StreamsAdapterDemoHelper.scanTable(dynamoDBClient, srcTable).getItems()
    .equals(StreamsAdapterDemoHelper.scanTable(dynamoDBClient, destTable).getItems()))
 {
    System.out.println("Scan result is equal.");
}
else {
    System.out.println("Tables are different!");
}
```

Schritt 5: Bereinigen

Die Demonstration ist abgeschlossen, so dass die Anwendung Quell- und Zieltabellen löscht. Beachten Sie hierzu das folgende Codebeispiel. Nachdem die Tabellen gelöscht wurden, bleiben die Streams für bis zu 24 Stunden verfügbar. Anschließend werden sie automatisch gelöscht.

```
dynamoDBClient.deleteTable(new DeleteTableRequest().withTableName(srcTable));
dynamoDBClient.deleteTable(new DeleteTableRequest().withTableName(destTable));
```

Vollständiges Programm: DynamoDB-Streams-Kinesis-Adapter

Hier finden Sie das vollständige Java-Programm, das die in Walkthrough: DynamoDB-Streams-Kinesis-Adapter beschriebenen Aufgaben durchführt. Wenn Sie das Programm ausführen, wird eine Ausgabe ähnlich der folgenden angezeigt:

```
Creating table KCL-Demo-src
Creating table KCL-Demo-dest
Table is active.
Creating worker for stream: arn:aws:dynamodb:us-west-2:111122223333:table/KCL-Demo-src/
stream/2015-05-19T22:48:56.601
Starting worker...
Scan result is equal.
Done.
```

> **Important**
>
> Um dieses Programm auszuführen, stellen Sie sicher, dass die Client-Anwendung CloudWatch mithilfe von Richtlinien Zugriff auf DynamoDB und Amazon hat. Weitere Informationen finden Sie unter Identitätsbasierte Richtlinien für DynamoDB.

Der Quellcode besteht aus vier `.java` Dateien. Um dieses Programm zu erstellen, fügen Sie die folgende Abhängigkeit hinzu, zu der die Amazon Kinesis Client Library (KCL) 3.x und das AWS SDK for Java v2 als transitive Abhängigkeiten gehören:

Maven

```
<dependency>
    <groupId>com.amazonaws</groupId>
    <artifactId>dynamodb-streams-kinesis-adapter</artifactId>
    <version>2.1.0</version>
</dependency>
```

Gradle

```
implementation 'com.amazonaws:dynamodb-streams-kinesis-adapter:2.1.0'
```

Die Quelldateien sind:

- `StreamsAdapterDemo.java`
- `StreamsRecordProcessor.java`
- `StreamsRecordProcessorFactory.java`
- `StreamsAdapterDemoHelper.java`

StreamsAdapterDemo.java

```
package com.amazonaws.codesamples;

import
 com.amazonaws.services.dynamodbv2.streamsadapter.AmazonDynamoDBStreamsAdapterClient;
import com.amazonaws.services.dynamodbv2.streamsadapter.StreamsSchedulerFactory;
```

```
import
 com.amazonaws.services.dynamodbv2.streamsadapter.polling.DynamoDBStreamsPollingConfig;
import software.amazon.awssdk.auth.credentials.AwsCredentialsProvider;
import software.amazon.awssdk.auth.credentials.DefaultCredentialsProvider;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.cloudwatch.CloudWatchAsyncClient;
import software.amazon.awssdk.services.dynamodb.DynamoDbAsyncClient;
import software.amazon.awssdk.services.dynamodb.model.DeleteTableRequest;
import software.amazon.awssdk.services.dynamodb.model.DescribeTableResponse;
import software.amazon.kinesis.common.ConfigsBuilder;
import software.amazon.kinesis.common.InitialPositionInStream;
import software.amazon.kinesis.common.InitialPositionInStreamExtended;
import software.amazon.kinesis.coordinator.Scheduler;
import software.amazon.kinesis.processor.ShardRecordProcessorFactory;
import software.amazon.kinesis.processor.StreamTracker;
import software.amazon.kinesis.retrieval.RetrievalConfig;

public class StreamsAdapterDemo {

    private static DynamoDbAsyncClient dynamoDbAsyncClient;
    private static CloudWatchAsyncClient cloudWatchAsyncClient;
    private static AmazonDynamoDBStreamsAdapterClient
 amazonDynamoDbStreamsAdapterClient;

    private static String tablePrefix = "KCL-Demo";
    private static String streamArn;

    private static Region region = Region.US_EAST_1;
    private static AwsCredentialsProvider credentialsProvider =
 DefaultCredentialsProvider.create();

    public static void main( String[] args ) throws Exception {
        System.out.println("Starting demo...");
        dynamoDbAsyncClient = DynamoDbAsyncClient.builder()
                .credentialsProvider(credentialsProvider)
                .region(region)
                .build();
        cloudWatchAsyncClient = CloudWatchAsyncClient.builder()
                .credentialsProvider(credentialsProvider)
                .region(region)
                .build();
        amazonDynamoDbStreamsAdapterClient = new
 AmazonDynamoDBStreamsAdapterClient(credentialsProvider, region);
```

```
        String srcTable = tablePrefix + "-src";
        String destTable = tablePrefix + "-dest";

        setUpTables();

        StreamTracker streamTracker =
StreamsSchedulerFactory.createSingleStreamTracker(streamArn,

InitialPositionInStreamExtended.newInitialPosition(InitialPositionInStream.TRIM_HORIZON));

        ShardRecordProcessorFactory shardRecordProcessorFactory =
                new StreamsAdapterDemoProcessorFactory(dynamoDbAsyncClient, destTable);

        ConfigsBuilder configsBuilder = new ConfigsBuilder(
                streamTracker,
                "streams-adapter-demo",
                amazonDynamoDbStreamsAdapterClient,
                dynamoDbAsyncClient,
                cloudWatchAsyncClient,
                "streams-demo-worker",
                shardRecordProcessorFactory
        );

        DynamoDBStreamsPollingConfig pollingConfig = new
DynamoDBStreamsPollingConfig(amazonDynamoDbStreamsAdapterClient);
        RetrievalConfig retrievalConfig = configsBuilder.retrievalConfig();
        retrievalConfig.retrievalSpecificConfig(pollingConfig);

        System.out.println("Creating scheduler for stream " + streamArn);
        Scheduler scheduler = StreamsSchedulerFactory.createScheduler(
                configsBuilder.checkpointConfig(),
                configsBuilder.coordinatorConfig(),
                configsBuilder.leaseManagementConfig(),
                configsBuilder.lifecycleConfig(),
                configsBuilder.metricsConfig(),
                configsBuilder.processorConfig(),
                retrievalConfig,
                amazonDynamoDbStreamsAdapterClient
        );

        System.out.println("Starting scheduler...");
        Thread t = new Thread(scheduler);
        t.start();
```

```
        Thread.sleep(250000);

        System.out.println("Stopping scheduler...");
        scheduler.shutdown();
        t.join();

        if (StreamsAdapterDemoHelper.scanTable(dynamoDbAsyncClient, srcTable).items()
                .equals(StreamsAdapterDemoHelper.scanTable(dynamoDbAsyncClient,
destTable).items())) {
            System.out.println("Scan result is equal.");
        } else {
            System.out.println("Tables are different!");
        }

        System.out.println("Done.");
        cleanupAndExit(0);
    }

    private static void setUpTables() {
        String srcTable = tablePrefix + "-src";
        String destTable = tablePrefix + "-dest";
        streamArn = StreamsAdapterDemoHelper.createTable(dynamoDbAsyncClient,
srcTable);
        StreamsAdapterDemoHelper.createTable(dynamoDbAsyncClient, destTable);

        awaitTableCreation(srcTable);

        performOps(srcTable);
    }

    private static void awaitTableCreation(String tableName) {
        Integer retries = 0;
        Boolean created = false;
        while (!created && retries < 100) {
            DescribeTableResponse result =
StreamsAdapterDemoHelper.describeTable(dynamoDbAsyncClient, tableName);
            created = result.table().tableStatusAsString().equals("ACTIVE");
            if (created) {
                System.out.println("Table is active.");
                return;
            } else {
                retries++;
                try {
                    Thread.sleep(1000);
```

```
                } catch (InterruptedException e) {
                    // do nothing
                }
            }
        }
        System.out.println("Timeout after table creation. Exiting...");
        cleanupAndExit(1);
    }

    private static void performOps(String tableName) {
        StreamsAdapterDemoHelper.putItem(dynamoDbAsyncClient, tableName, "101",
 "test1");
        StreamsAdapterDemoHelper.updateItem(dynamoDbAsyncClient, tableName, "101",
 "test2");
        StreamsAdapterDemoHelper.deleteItem(dynamoDbAsyncClient, tableName, "101");
        StreamsAdapterDemoHelper.putItem(dynamoDbAsyncClient, tableName, "102",
 "demo3");
        StreamsAdapterDemoHelper.updateItem(dynamoDbAsyncClient, tableName, "102",
 "demo4");
        StreamsAdapterDemoHelper.deleteItem(dynamoDbAsyncClient, tableName, "102");
    }

    private static void cleanupAndExit(Integer returnValue) {
        String srcTable = tablePrefix + "-src";
        String destTable = tablePrefix + "-dest";

 dynamoDbAsyncClient.deleteTable(DeleteTableRequest.builder().tableName(srcTable).build());

 dynamoDbAsyncClient.deleteTable(DeleteTableRequest.builder().tableName(destTable).build());
        System.exit(returnValue);
    }
}
```

StreamsRecordProcessor.java

```
package com.amazonaws.codesamples;

import
 com.amazonaws.services.dynamodbv2.streamsadapter.adapter.DynamoDBStreamsClientRecord;
import
 com.amazonaws.services.dynamodbv2.streamsadapter.model.DynamoDBStreamsProcessRecordsInput;
import
 com.amazonaws.services.dynamodbv2.streamsadapter.processor.DynamoDBStreamsShardRecordProcessor
```

```
import software.amazon.awssdk.services.dynamodb.DynamoDbAsyncClient;
import software.amazon.awssdk.services.dynamodb.model.Record;
import software.amazon.kinesis.exceptions.InvalidStateException;
import software.amazon.kinesis.exceptions.ShutdownException;
import software.amazon.kinesis.lifecycle.events.InitializationInput;
import software.amazon.kinesis.lifecycle.events.LeaseLostInput;
import software.amazon.kinesis.lifecycle.events.ShardEndedInput;
import software.amazon.kinesis.lifecycle.events.ShutdownRequestedInput;

import java.nio.charset.Charset;
import java.nio.charset.StandardCharsets;

public class StreamsRecordProcessor implements DynamoDBStreamsShardRecordProcessor {

    private Integer checkpointCounter;

    private final DynamoDbAsyncClient dynamoDbAsyncClient;
    private final String tableName;

    public StreamsRecordProcessor(DynamoDbAsyncClient dynamoDbAsyncClient, String
 tableName) {
        this.dynamoDbAsyncClient = dynamoDbAsyncClient;
        this.tableName = tableName;
    }

    @Override
    public void initialize(InitializationInput initializationInput) {
        this.checkpointCounter = 0;
    }

    @Override
    public void processRecords(DynamoDBStreamsProcessRecordsInput
 dynamoDBStreamsProcessRecordsInput) {
        for (DynamoDBStreamsClientRecord record:
 dynamoDBStreamsProcessRecordsInput.records()) {
            String data = new String(record.data().array(), StandardCharsets.UTF_8);
            System.out.println(data);
            Record streamRecord = record.getRecord();

            switch (streamRecord.eventName()) {
                case INSERT:
                case MODIFY:
                    StreamsAdapterDemoHelper.putItem(dynamoDbAsyncClient, tableName,
                            streamRecord.dynamodb().newImage());
```

```
                case REMOVE:
                    StreamsAdapterDemoHelper.deleteItem(dynamoDbAsyncClient, tableName,
                            streamRecord.dynamodb().keys().get("Id").n());
            }
            checkpointCounter += 1;
            if (checkpointCounter % 10 == 0) {
                try {
                    dynamoDBStreamsProcessRecordsInput.checkpointer().checkpoint();
                } catch (Exception e) {
                    e.printStackTrace();
                }
            }
        }
    }

    @Override
    public void leaseLost(LeaseLostInput leaseLostInput) {
        System.out.println("Lease Lost");
    }

    @Override
    public void shardEnded(ShardEndedInput shardEndedInput) {
        try {
            shardEndedInput.checkpointer().checkpoint();
        } catch (ShutdownException | InvalidStateException e) {
            e.printStackTrace();
        }
    }

    @Override
    public void shutdownRequested(ShutdownRequestedInput shutdownRequestedInput) {
        try {
            shutdownRequestedInput.checkpointer().checkpoint();
        } catch (ShutdownException | InvalidStateException e) {
            e.printStackTrace();
        }
    }
}
```

StreamsRecordProcessorFactory.java

```
package com.amazonaws.codesamples;
```

```
import software.amazon.awssdk.services.dynamodb.DynamoDbAsyncClient;
import software.amazon.kinesis.processor.ShardRecordProcessor;
import software.amazon.kinesis.processor.ShardRecordProcessorFactory;

public class StreamsAdapterDemoProcessorFactory implements ShardRecordProcessorFactory
 {
    private final String tableName;
    private final DynamoDbAsyncClient dynamoDbAsyncClient;

    public StreamsAdapterDemoProcessorFactory(DynamoDbAsyncClient asyncClient, String
 tableName) {
        this.tableName = tableName;
        this.dynamoDbAsyncClient = asyncClient;
    }

    @Override
    public ShardRecordProcessor shardRecordProcessor() {
        return new StreamsRecordProcessor(dynamoDbAsyncClient, tableName);
    }
}
```

StreamsAdapterDemoHelper.java

```
package com.amazonaws.codesamples;

import software.amazon.awssdk.services.dynamodb.DynamoDbAsyncClient;
import software.amazon.awssdk.services.dynamodb.model.AttributeDefinition;
import software.amazon.awssdk.services.dynamodb.model.AttributeValue;
import software.amazon.awssdk.services.dynamodb.model.BillingMode;
import software.amazon.awssdk.services.dynamodb.model.CreateTableRequest;
import software.amazon.awssdk.services.dynamodb.model.CreateTableResponse;
import software.amazon.awssdk.services.dynamodb.model.DeleteItemRequest;
import software.amazon.awssdk.services.dynamodb.model.DescribeTableRequest;
import software.amazon.awssdk.services.dynamodb.model.DescribeTableResponse;
import software.amazon.awssdk.services.dynamodb.model.KeySchemaElement;
import software.amazon.awssdk.services.dynamodb.model.KeyType;
import software.amazon.awssdk.services.dynamodb.model.OnDemandThroughput;
import software.amazon.awssdk.services.dynamodb.model.PutItemRequest;
import software.amazon.awssdk.services.dynamodb.model.ResourceInUseException;
import software.amazon.awssdk.services.dynamodb.model.ScanRequest;
import software.amazon.awssdk.services.dynamodb.model.ScanResponse;
import software.amazon.awssdk.services.dynamodb.model.StreamSpecification;
import software.amazon.awssdk.services.dynamodb.model.StreamViewType;
```

```
import software.amazon.awssdk.services.dynamodb.model.UpdateItemRequest;

import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;

public class StreamsAdapterDemoHelper {

    /**
     * @return StreamArn
     */
    public static String createTable(DynamoDbAsyncClient client, String tableName) {
        List<AttributeDefinition> attributeDefinitions = new ArrayList<>();
        attributeDefinitions.add(AttributeDefinition.builder()
                .attributeName("Id")
                .attributeType("N")
                .build());

        List<KeySchemaElement> keySchema = new ArrayList<>();
        keySchema.add(KeySchemaElement.builder()
                .attributeName("Id")
                .keyType(KeyType.HASH) // Partition key
                .build());

        StreamSpecification streamSpecification = StreamSpecification.builder()
                .streamEnabled(true)
                .streamViewType(StreamViewType.NEW_IMAGE)
                .build();

        CreateTableRequest createTableRequest = CreateTableRequest.builder()
                .tableName(tableName)
                .attributeDefinitions(attributeDefinitions)
                .keySchema(keySchema)
                .billingMode(BillingMode.PAY_PER_REQUEST)
                .streamSpecification(streamSpecification)
                .build();

        try {
            System.out.println("Creating table " + tableName);
            CreateTableResponse result = client.createTable(createTableRequest).join();
            return result.tableDescription().latestStreamArn();
        } catch (Exception e) {
            if (e.getCause() instanceof ResourceInUseException) {
```

```
                System.out.println("Table already exists.");
                return describeTable(client, tableName).table().latestStreamArn();
            }
            throw e;
        }
    }

    public static DescribeTableResponse describeTable(DynamoDbAsyncClient client,
String tableName) {
        return client.describeTable(DescribeTableRequest.builder()
                        .tableName(tableName)
                        .build())
                .join();
    }

    public static ScanResponse scanTable(DynamoDbAsyncClient dynamoDbClient, String
tableName) {
        return dynamoDbClient.scan(ScanRequest.builder()
                        .tableName(tableName)
                        .build())
                .join();
    }

    public static void putItem(DynamoDbAsyncClient dynamoDbClient, String tableName,
String id, String val) {
        Map<String, AttributeValue> item = new HashMap<>();
        item.put("Id", AttributeValue.builder().n(id).build());
        item.put("attribute-1", AttributeValue.builder().s(val).build());

        putItem(dynamoDbClient, tableName, item);
    }

    public static void putItem(DynamoDbAsyncClient dynamoDbClient, String tableName,
                               Map<String, AttributeValue> items) {
        PutItemRequest putItemRequest = PutItemRequest.builder()
                .tableName(tableName)
                .item(items)
                .build();
        dynamoDbClient.putItem(putItemRequest).join();
    }

    public static void updateItem(DynamoDbAsyncClient dynamoDbClient, String tableName,
String id, String val) {
        Map<String, AttributeValue> key = new HashMap<>();
```

```
        key.put("Id", AttributeValue.builder().n(id).build());

        Map<String, String> expressionAttributeNames = new HashMap<>();
        expressionAttributeNames.put("#attr2", "attribute-2");

        Map<String, AttributeValue> expressionAttributeValues = new HashMap<>();
        expressionAttributeValues.put(":val", AttributeValue.builder().s(val).build());

        UpdateItemRequest updateItemRequest = UpdateItemRequest.builder()
                .tableName(tableName)
                .key(key)
                .updateExpression("SET #attr2 = :val")
                .expressionAttributeNames(expressionAttributeNames)
                .expressionAttributeValues(expressionAttributeValues)
                .build();

        dynamoDbClient.updateItem(updateItemRequest).join();
    }

    public static void deleteItem(DynamoDbAsyncClient dynamoDbClient, String tableName,
 String id) {
        Map<String, AttributeValue> key = new HashMap<>();
        key.put("Id", AttributeValue.builder().n(id).build());

        DeleteItemRequest deleteItemRequest = DeleteItemRequest.builder()
                .tableName(tableName)
                .key(key)
                .build();
        dynamoDbClient.deleteItem(deleteItemRequest).join();
    }
}
```

DynamoDB-Streams-Low-Level-API: Java-Beispiel

> **Note**
>
> Der Code auf dieser Seite ist nicht umfassend und behandelt nicht alle Szenarien für die Nutzung von Amazon DynamoDB Streams. Die empfohlene Methode für die Nutzung von Stream-Datensätzen aus DynamoDB erfolgt anhand des Amazon-Kinesis-Adapters unter Verwendung der Kinesis-Client-Library (KCL) gemäß der Beschreibung in Verwenden des DynamoDB-Streams-Kinesis-Adapters zum Verarbeiten von Stream-Datensätzen.

Dieser Abschnitt enthält ein Java-Programm, das DynamoDB Streams in Aktion zeigt. Das Programm führt Folgendes aus:

1. Erstellt eine DynamoDB-Tabelle mit einem aktivierten Stream.
2. Beschreibt die Stream-Einstellungen für diese Tabelle.
3. Ändert die Daten in der Tabelle.
4. Beschreibt die Shards im Stream.
5. Liest die Stream-Datensätze aus den Shards.
6. Ruft untergeordnete Shards ab und setzt das Lesen von Datensätzen fort.
7. Bereinigt die Daten.

Wenn Sie das Programm ausführen, wird eine Ausgabe ähnlich der folgenden angezeigt.

```
Testing Streams Demo
Creating an Amazon DynamoDB table TestTableForStreams with a simple primary key: Id
Waiting for TestTableForStreams to be created...
Current stream ARN for TestTableForStreams: arn:aws:dynamodb:us-
east-2:123456789012:table/TestTableForStreams/stream/2018-03-20T16:49:55.208
Stream enabled: true
Update view type: NEW_AND_OLD_IMAGES

Performing write activities on TestTableForStreams
Processing item 1 of 100
Processing item 2 of 100
Processing item 3 of 100
...
Processing item 100 of 100
Shard: {ShardId: shardId-1234567890-...,SequenceNumberRange: {StartingSequenceNumber:
 100002572486797508907,},}
    Shard iterator: EjYFEkX2a26eVTWe...
        StreamRecord(ApproximateCreationDateTime=2025-04-09T13:11:58Z,
 Keys={Id=AttributeValue(S=4)}, NewImage={Message=AttributeValue(S=New Item!),
 Id=AttributeValue(S=4)}, SequenceNumber=200000158404754583390 9, SizeBytes=22,
 StreamViewType=NEW_AND_OLD_IMAGES)
        StreamRecord(ApproximateCreationDateTime=2025-04-09T13:11:58Z,
 Keys={Id=AttributeValue(S=4)}, NewImage={Message=AttributeValue(S=This is an updated
 item), Id=AttributeValue(S=4)}, OldImage={Message=AttributeValue(S=New Item!),
 Id=AttributeValue(S=4)}, SequenceNumber=210000360486976789270 1, SizeBytes=55,
 StreamViewType=NEW_AND_OLD_IMAGES)
```

```
        StreamRecord(ApproximateCreationDateTime=2025-04-09T13:11:58Z,
 Keys={Id=AttributeValue(S=4)}, OldImage={Message=AttributeValue(S=This is an updated
 item), Id=AttributeValue(S=4)}, SequenceNumber=220000109977111289843 4, SizeBytes=36,
 StreamViewType=NEW_AND_OLD_IMAGES)
...
Deleting the table...
Table StreamsDemoTable deleted.
Demo complete
```

Example Beispiel

```
import java.util.HashMap;
import java.util.List;
import java.util.Map;

import software.amazon.awssdk.core.waiters.WaiterResponse;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.AttributeAction;
import software.amazon.awssdk.services.dynamodb.model.AttributeDefinition;
import software.amazon.awssdk.services.dynamodb.model.AttributeValue;
import software.amazon.awssdk.services.dynamodb.model.AttributeValueUpdate;
import software.amazon.awssdk.services.dynamodb.model.BillingMode;
import software.amazon.awssdk.services.dynamodb.model.CreateTableRequest;
import software.amazon.awssdk.services.dynamodb.model.CreateTableResponse;
import software.amazon.awssdk.services.dynamodb.model.DeleteItemRequest;
import software.amazon.awssdk.services.dynamodb.model.DeleteTableRequest;
import software.amazon.awssdk.services.dynamodb.model.DescribeStreamRequest;
import software.amazon.awssdk.services.dynamodb.model.DescribeStreamResponse;
import software.amazon.awssdk.services.dynamodb.model.DescribeTableRequest;
import software.amazon.awssdk.services.dynamodb.model.DescribeTableResponse;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.model.GetRecordsRequest;
import software.amazon.awssdk.services.dynamodb.model.GetRecordsResponse;
import software.amazon.awssdk.services.dynamodb.model.GetShardIteratorRequest;
import software.amazon.awssdk.services.dynamodb.model.GetShardIteratorResponse;
import software.amazon.awssdk.services.dynamodb.model.KeySchemaElement;
import software.amazon.awssdk.services.dynamodb.model.KeyType;
import software.amazon.awssdk.services.dynamodb.model.PutItemRequest;
import software.amazon.awssdk.services.dynamodb.model.Record;
import software.amazon.awssdk.services.dynamodb.model.ScalarAttributeType;
import software.amazon.awssdk.services.dynamodb.model.Shard;
import software.amazon.awssdk.services.dynamodb.model.ShardFilter;
```

```
import software.amazon.awssdk.services.dynamodb.model.ShardFilterType;
import software.amazon.awssdk.services.dynamodb.model.ShardIteratorType;
import software.amazon.awssdk.services.dynamodb.model.StreamSpecification;
import software.amazon.awssdk.services.dynamodb.model.TableDescription;
import software.amazon.awssdk.services.dynamodb.model.UpdateItemRequest;
import software.amazon.awssdk.services.dynamodb.streams.DynamoDbStreamsClient;
import software.amazon.awssdk.services.dynamodb.waiters.DynamoDbWaiter;

public class StreamsLowLevelDemo {


    public static void main(String[] args) {
        final String usage = "Testing Streams Demo";
        try {
            System.out.println(usage);

            String tableName = "StreamsDemoTable";
            String key = "Id";
            System.out.println("Creating an Amazon DynamoDB table " + tableName + "
 with a simple primary key: " + key);
            Region region = Region.US_WEST_2;
            DynamoDbClient ddb = DynamoDbClient.builder()
                    .region(region)
                    .build();

            DynamoDbStreamsClient ddbStreams = DynamoDbStreamsClient.builder()
                    .region(region)
                    .build();
            DescribeTableRequest describeTableRequest = DescribeTableRequest.builder()
                    .tableName(tableName)
                    .build();
            TableDescription tableDescription = null;
            try{
                tableDescription = ddb.describeTable(describeTableRequest).table();
            }catch (Exception e){
                System.out.println("Table " + tableName + " does not exist.");
                tableDescription = createTable(ddb, tableName, key);
            }

            // Print the stream settings for the table
            String streamArn = tableDescription.latestStreamArn();

            StreamSpecification streamSpec = tableDescription.streamSpecification();
```

```
            System.out.println("Current stream ARN for " + tableDescription.tableName()
+ ": " +
                   streamArn);
            System.out.println("Stream enabled: " + streamSpec.streamEnabled());
            System.out.println("Update view type: " + streamSpec.streamViewType());
            System.out.println();
            // Generate write activity in the table
            System.out.println("Performing write activities on " + tableName);
            int maxItemCount = 100;
            for (Integer i = 1; i <= maxItemCount; i++) {
                System.out.println("Processing item " + i + " of " + maxItemCount);
                // Write a new item
                putItemInTable(key, i, tableName, ddb);
                // Update the item
                updateItemInTable(key, i, tableName, ddb);
                // Delete the item
                deleteDynamoDBItem(key, i, tableName, ddb);
            }

            // Process Stream
            processStream(streamArn, maxItemCount, ddb, ddbStreams, tableName);

            // Delete the table
            System.out.println("Deleting the table...");
            DeleteTableRequest deleteTableRequest = DeleteTableRequest.builder()
                    .tableName(tableName)
                    .build();
            ddb.deleteTable(deleteTableRequest);
            System.out.println("Table " + tableName + " deleted.");
            System.out.println("Demo complete");
            ddb.close();
        } catch (Exception e) {
            System.out.println("Error: " + e.getMessage());
        }
    }

    private static void processStream(String streamArn, int maxItemCount,
DynamoDbClient ddb, DynamoDbStreamsClient ddbStreams, String tableName) {
        // Get all the shard IDs from the stream. Note that DescribeStream returns
        // the shard IDs one page at a time.
        String lastEvaluatedShardId = null;
        do {
            DescribeStreamRequest describeStreamRequest =
DescribeStreamRequest.builder()
```

```
                    .streamArn(streamArn)
                    .exclusiveStartShardId(lastEvaluatedShardId).build();
            DescribeStreamResponse describeStreamResponse =
ddbStreams.describeStream(describeStreamRequest);

            List<Shard> shards = describeStreamResponse.streamDescription().shards();

            // Process each shard on this page

            fetchShardsAndReadRecords(streamArn, maxItemCount, ddbStreams, shards);

            // If LastEvaluatedShardId is set, then there is
            // at least one more page of shard IDs to retrieve
            lastEvaluatedShardId =
describeStreamResponse.streamDescription().lastEvaluatedShardId();

        } while (lastEvaluatedShardId != null);

    }

    private static void fetchShardsAndReadRecords(String streamArn, int maxItemCount,
DynamoDbStreamsClient ddbStreams, List<Shard> shards) {
        for (Shard shard : shards) {
            String shardId = shard.shardId();
            System.out.println("Shard: " + shard);

            // Get an iterator for the current shard
            GetShardIteratorRequest shardIteratorRequest =
GetShardIteratorRequest.builder()
                    .streamArn(streamArn).shardId(shardId)
                    .shardIteratorType(ShardIteratorType.TRIM_HORIZON).build();

            GetShardIteratorResponse getShardIteratorResult =
ddbStreams.getShardIterator(shardIteratorRequest);

            String currentShardIter = getShardIteratorResult.shardIterator();

            // Shard iterator is not null until the Shard is sealed (marked as
READ_ONLY).
            // To prevent running the loop until the Shard is sealed, we process only
the
            // items that were written into DynamoDB and then exit.
            int processedRecordCount = 0;
            while (currentShardIter != null && processedRecordCount < maxItemCount) {
```

```
                // Use the shard iterator to read the stream records
                GetRecordsRequest getRecordsRequest = GetRecordsRequest.builder()
                        .shardIterator(currentShardIter).build();
                GetRecordsResponse getRecordsResult =
ddbStreams.getRecords(getRecordsRequest);
                List<Record> records = getRecordsResult.records();
                for (Record record : records) {
                    System.out.println("        " + record.dynamodb());
                }
                processedRecordCount += records.size();
                currentShardIter = getRecordsResult.nextShardIterator();
            }
            if (currentShardIter == null){
                System.out.println("Shard has been fully processed. Shard iterator is
null.");
                System.out.println("Fetch the child shard to continue processing
instead of bulk fetching all shards");
                DescribeStreamRequest describeStreamRequestForChildShards =
DescribeStreamRequest.builder()
                        .streamArn(streamArn)
                        .shardFilter(ShardFilter.builder()
                                .type(ShardFilterType.CHILD_SHARDS)
                                .shardId(shardId).build())
                        .build();
                DescribeStreamResponse describeStreamResponseChildShards =
ddbStreams.describeStream(describeStreamRequestForChildShards);
                fetchShardsAndReadRecords(streamArn, maxItemCount, ddbStreams,
describeStreamResponseChildShards.streamDescription().shards());
            }
        }
    }

    private static void putItemInTable(String key, Integer i, String tableName,
DynamoDbClient ddb) {
        Map<String, AttributeValue> item = new HashMap<>();
        item.put(key, AttributeValue.builder()
                .s(i.toString())
                .build());
        item.put("Message", AttributeValue.builder()
                .s("New Item!")
                .build());
        PutItemRequest request = PutItemRequest.builder()
                .tableName(tableName)
                .item(item)
```

```
                .build();
        ddb.putItem(request);
    }

    private static void updateItemInTable(String key, Integer i, String tableName,
DynamoDbClient ddb) {

        HashMap<String, AttributeValue> itemKey = new HashMap<>();
        itemKey.put(key, AttributeValue.builder()
                .s(i.toString())
                .build());


        HashMap<String, AttributeValueUpdate> updatedValues = new HashMap<>();
        updatedValues.put("Message", AttributeValueUpdate.builder()
                .value(AttributeValue.builder().s("This is an updated item").build())
                .action(AttributeAction.PUT)
                .build());

        UpdateItemRequest request = UpdateItemRequest.builder()
                .tableName(tableName)
                .key(itemKey)
                .attributeUpdates(updatedValues)
                .build();
        ddb.updateItem(request);
    }

    public static void deleteDynamoDBItem(String key, Integer i, String tableName,
DynamoDbClient ddb) {
        HashMap<String, AttributeValue> keyToGet = new HashMap<>();
        keyToGet.put(key, AttributeValue.builder()
                .s(i.toString())
                .build());

        DeleteItemRequest deleteReq = DeleteItemRequest.builder()
                .tableName(tableName)
                .key(keyToGet)
                .build();
        ddb.deleteItem(deleteReq);
    }

    public static TableDescription createTable(DynamoDbClient ddb, String tableName,
String key) {
        DynamoDbWaiter dbWaiter = ddb.waiter();
```

```
        StreamSpecification streamSpecification = StreamSpecification.builder()
                .streamEnabled(true)
                .streamViewType("NEW_AND_OLD_IMAGES")
                .build();
        CreateTableRequest request = CreateTableRequest.builder()
                .attributeDefinitions(AttributeDefinition.builder()
                        .attributeName(key)
                        .attributeType(ScalarAttributeType.S)
                        .build())
                .keySchema(KeySchemaElement.builder()
                        .attributeName(key)
                        .keyType(KeyType.HASH)
                        .build())
                .billingMode(BillingMode.PAY_PER_REQUEST) //  DynamoDB automatically
scales based on traffic.
                .tableName(tableName)
                .streamSpecification(streamSpecification)
                .build();

        TableDescription newTable;
        try {
            CreateTableResponse response = ddb.createTable(request);
            DescribeTableRequest tableRequest = DescribeTableRequest.builder()
                    .tableName(tableName)
                    .build();

            System.out.println("Waiting for " + tableName + " to be created...");

            // Wait until the Amazon DynamoDB table is created.
            WaiterResponse<DescribeTableResponse> waiterResponse =
dbWaiter.waitUntilTableExists(tableRequest);
            waiterResponse.matched().response().ifPresent(System.out::println);
            newTable = response.tableDescription();
            return newTable;

        } catch (DynamoDbException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
        return null;
    }
```

```
}
```

DynamoDB Streams und -Trigger AWS Lambda

Amazon DynamoDB ist integriert, AWS Lambda sodass Sie Trigger erstellen können — Codeteile, die automatisch auf Ereignisse in DynamoDB Streams reagieren. Mit Auslösern können Sie Anwendungen erstellen, die auf Datenänderungen in DynamoDB-Tabellen reagieren.

Themen

- Tutorial #1: Verwenden von Filtern zur Verarbeitung aller Ereignisse mit Amazon DynamoDB und AWS Lambda Verwendung der AWS CLI
- Tutorial 2: Verwenden von Filtern, um einige Ereignisse mit DynamoDB und Lambda zu verarbeiten
- Bewährte Methoden zur Verwendung von DynamoDB Streams mit Lambda

Wenn Sie DynamoDB Streams für eine Tabelle aktivieren, können Sie den Stream Amazon Resource Name (ARN) mit einer von Ihnen geschriebenen AWS Lambda Funktion verknüpfen. Alle Mutationsaktionen für diese DynamoDB-Tabelle können dann als Element im Stream erfasst werden. Sie können beispielsweise einen Auslöser festlegen, sodass beim Ändern eines Elements in einer Tabelle sofort ein neuer Datensatz im Stream der betreffenden Tabelle angezeigt wird.

> **Note**
>
> Wenn Sie mehr als zwei Lambda-Funktionen für einen DynamoDB-Stream abonnieren, kann es zu einer Drosselung der Lesegeschwindigkeit kommen.

Der AWS Lambda-Service fragt den Stream viermal pro Sekunde nach neuen Datensätzen ab. Wenn neue Stream-Datensätze verfügbar sind, wird Ihre Lambda-Funktion synchron aufgerufen. Sie können bis zu zwei Lambda-Funktionen für denselben DynamoDB-Stream abonnieren. Wenn Sie mehr als zwei Lambda-Funktionen für denselben DynamoDB-Stream abonnieren, kann es zu einer Drosselung der Lesegeschwindigkeit kommen.

Die Lambda-Funktion kann eine Benachrichtigung senden, einen Workflow einleiten oder viele andere von Ihnen festgelegte Aktionen durchführen. Sie können eine Lambda-Funktion schreiben, um jeden Stream-Datensatz einfach in persistenten Speicher wie z. B. Amazon S3 File Gateway (Amazon S3) zu kopieren und einen permanenten Prüfungs-Trail der Schreibaktivitäten in Ihrer Tabelle zu erstellen. Angenommen, Sie verfügen über eine mobile Gaming-App, die Schreibvorgänge

in einer `GameScores`-Tabelle vornimmt. Bei jedem Aktualisieren des `TopScore`-Attributs der `GameScores`-Tabelle wird ein entsprechender Stream-Datensatz in den Stream der Tabelle geschrieben. Dieses Ereignis kann dann automatisch eine Lambda-Funktion auslösen, die einen Glückwunsch in einem Social-Media-Netzwerk postet. Diese Funktion könnte auch so geschrieben werden, dass sie alle Stream-Datensätze ignoriert, die kein Update von `GameScores` sind oder das `TopScore`-Attribut nicht ändern.

Wenn Ihre Funktion einen Fehler zurückgibt, wiederholt Lambda den Vorgang mit dem Batch, bis die Verarbeitung erfolgreich ist oder die Daten ablaufen. Sie können Lambda auch so konfigurieren, dass eine Wiederholung mit einem kleineren Batch erfolgt, die Anzahl der Wiederholungen eingeschränkt wird, Datensätze verworfen werden, sobald sie zu alt sind, und vieles mehr.

Gemäß bewährten Methoden für die Leistung muss die Lambda-Funktion kurzlebig sein. Um unnötige Verarbeitungsverzögerungen zu vermeiden, sollte sie auch keine komplexe Logik ausführen. Insbesondere bei einem Hochgeschwindigkeits-Stream ist es besser, asynchrone Nachbearbeitungs-Schrittfunktions-Workflows auszulösen als synchrone Lambdas mit langer Laufzeit.

Sie können Lambda-Trigger für verschiedene AWS Konten verwenden, indem Sie eine ressourcenbasierte Richtlinie für den DynamoDB-Stream konfigurieren, um der kontoübergreifenden Lesezugriff auf die Lambda-Funktion zu gewähren. Weitere Informationen darüber, wie Sie Ihren Stream so konfigurieren, dass er kontoübergreifenden Zugriff ermöglicht, finden Sie unter Zugriff mit kontoübergreifenden AWS Lambda-Funktionen teilen im DynamoDB-Entwicklerhandbuch.

Weitere Informationen AWS Lambda zu finden Sie im Entwicklerhandbuch.AWS Lambda

Tutorial #1: Verwenden von Filtern zur Verarbeitung aller Ereignisse mit Amazon DynamoDB und AWS Lambda Verwendung der AWS CLI

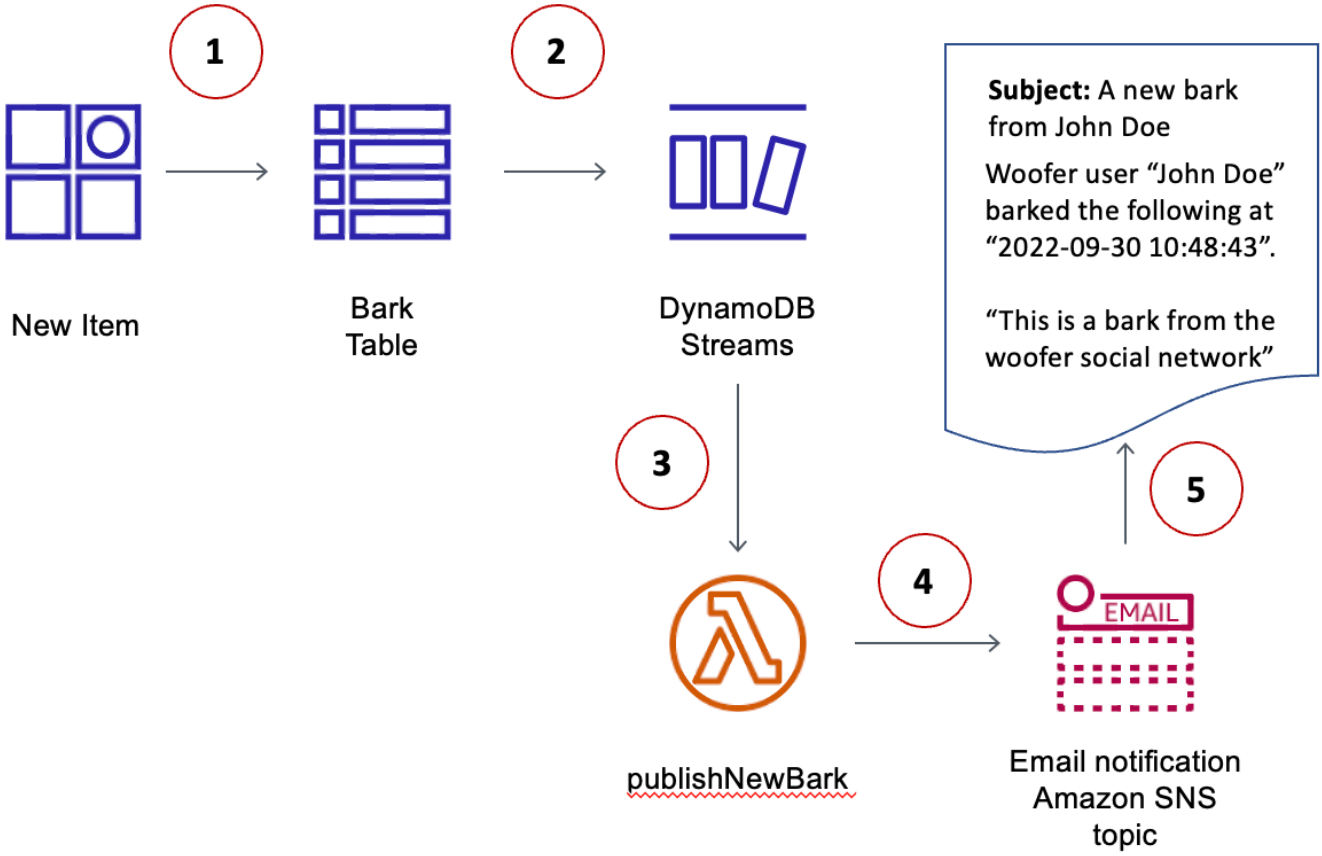
In diesem Tutorial erstellen Sie einen AWS Lambda Trigger zur Verarbeitung eines Streams aus einer DynamoDB-Tabelle.

Themen

- Schritt 1: Erstellen einer DynamoDB-Tabelle mit einem aktivierten Stream
- Schritt 2: Erstellen einer Lambda-Ausführungsrolle
- Schritt 3: Erstellen eines Amazon-SNS-Themas
- Schritt 4: Erstellen und Testen einer Lambda-Funktion

- Schritt 5: Erstellen und Testen eines Auslösers

Das Szenario für dieses Tutorial ist Woofer, ein einfaches soziales Netzwerk. Woofer-Benutzer kommunizieren mit einer kurzen Textnachricht Barks („Bellen”), die an andere Woofer-Benutzer gesendet werden. Das folgende Diagramm zeigt die Komponenten und den Workflow für diese Anwendung.



1. Ein Benutzer schreibt ein Element in eine DynamoDB-Tabelle (`BarkTable`). Jedes Element in der Tabelle steht für eine Textnachricht („Bark”).
2. Ein neuer Stream-Datensatz wird erstellt, um zu berücksichtigen, dass ein neues Element der `BarkTable` hinzugefügt wurde.
3. Der neue Stream-Record löst eine AWS Lambda Funktion () `publishNewBark` aus.
4. Wenn der Stream-Datensatz angibt, dass ein neues Element `BarkTable` hinzugefügt wurde, liest die Lambda-Funktion die Daten aus dem Stream-Datensatz und veröffentlicht eine Nachricht für ein Thema im Amazon Simple Notification Service (Amazon SNS).
5. Die Nachricht wird von den Abonnenten des Amazon-SNS-Themas empfangen. (In diesem Tutorial ist der einzige Abonnent eine E-Mail-Adresse.)

Bevor Sie beginnen

Dieses Tutorial verwendet die AWS Command Line Interface AWS CLI. Folgen Sie den Anweisungen im AWS Command Line Interface -Benutzerhandbuch zum Installieren und Konfigurieren der AWS CLI.

Schritt 1: Erstellen einer DynamoDB-Tabelle mit einem aktivierten Stream

In diesem Schritt erstellen Sie eine DynamoDB-Tabelle (`BarkTable`), um alle Textnachrichten von Woofer-Benutzern zu speichern. Der Primärschlüssel besteht aus `Username` (Partitionsschlüssel) und `Timestamp` (Sortierschlüssel). Bei beiden Attributen handelt es sich um Zeichenfolgen.

Für `BarkTable` ist ein Stream aktiviert. Später in diesem Tutorial erstellen Sie einen Trigger, indem Sie dem Stream eine AWS Lambda Funktion zuordnen.

1. Verwenden Sie den folgenden Befehl, um die Tabelle zu erstellen.

```
aws dynamodb create-table \
    --table-name BarkTable \
    --attribute-definitions AttributeName=Username,AttributeType=S
 AttributeName=Timestamp,AttributeType=S \
    --key-schema AttributeName=Username,KeyType=HASH
 AttributeName=Timestamp,KeyType=RANGE \
    --provisioned-throughput ReadCapacityUnits=5,WriteCapacityUnits=5 \
    --stream-specification StreamEnabled=true,StreamViewType=NEW_AND_OLD_IMAGES
```

2. Suchen Sie in der Ausgabe nach dem `LatestStreamArn`.

```
...
"LatestStreamArn": "arn:aws:dynamodb:region:accountID:table/BarkTable/
stream/timestamp
...
```

   Notieren Sie sich die *`region`* und die *`accountID`*, da Sie sie für die anderen Schritte in diesem Tutorial benötigen.

Schritt 2: Erstellen einer Lambda-Ausführungsrolle

In diesem Schritt erstellen Sie eine AWS Identity and Access Management (IAM-) Rolle (`WooferLambdaRole`) und weisen ihr Berechtigungen zu. Diese Rolle wird von der Lambda-Funktion verwendet, die Sie in Schritt 4: Erstellen und Testen einer Lambda-Funktion erstellen.

Außerdem legen Sie eine Richtlinie für die Rolle fest. Die Richtlinie enthält alle Berechtigungen, die die Lambda-Funktion zur Laufzeit benötigt.

1. Erstellen Sie eine Datei mit dem Namen `trust-relationship.json` und dem folgenden Inhalt.

   JSON

   ```
   {
      "Version":"2012-10-17",
      "Statement": [
        {
          "Effect": "Allow",
          "Principal": {
            "Service": "lambda.amazonaws.com"
          },
          "Action": "sts:AssumeRole"
        }
      ]
    }
   ```

2. Geben Sie den folgenden Befehl ein, um `WooferLambdaRole` zu erstellen.

   ```
   aws iam create-role --role-name WooferLambdaRole \
       --path "/service-role/" \
       --assume-role-policy-document file://trust-relationship.json
   ```

3. Erstellen Sie eine Datei mit dem Namen `role-policy.json` und dem folgenden Inhalt. (Ersetzen Sie *`region`* und *`accountID`* durch Ihre AWS Region und Konto-ID.)

   JSON

   ```
   {
       "Version":"2012-10-17",
       "Statement": [
           {
               "Effect": "Allow",
               "Action": [
                   "logs:CreateLogGroup",
                   "logs:CreateLogStream",
                   "logs:PutLogEvents"
   ```

```
            ],
            "Resource": "arn:aws:logs:us-east-1:111122223333:*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "dynamodb:DescribeStream",
                "dynamodb:GetRecords",
                "dynamodb:GetShardIterator",
                "dynamodb:ListStreams"
            ],
            "Resource": "arn:aws:dynamodb:us-east-1:111122223333:table/
BarkTable/stream/*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "sns:Publish"
            ],
            "Resource": [
                "*"
            ]
        }
    ]
}
```

Die Richtlinie enthält vier Anweisungen, damit `WooferLambdaRole` folgende Aufgaben ausführen kann:

- Führen Sie eine Lambda-Funktion (`publishNewBark`). Die Funktion erstellen Sie zu einem späteren Zeitpunkt in diesem Tutorial.
- Greifen Sie auf Amazon CloudWatch Logs zu. Die Lambda-Funktion schreibt zur Laufzeit Diagnosen in CloudWatch Logs.
- Lesen von Daten aus dem DynamoDB Stream für `BarkTable`.
- Veröffentlichen Sie Nachrichten in Amazon SNS.

4. Führen Sie den folgenden Befehl aus, um die Richtlinie `WooferLambdaRole` anzufügen:

```
aws iam put-role-policy --role-name WooferLambdaRole \
    --policy-name WooferLambdaRolePolicy \
```

```
    --policy-document file://role-policy.json
```

Schritt 3: Erstellen eines Amazon-SNS-Themas

In diesem Schritt erstellen Sie ein Amazon-SNS-Thema (`wooferTopic`) und abonnieren es für eine E-Mail-Adresse. Ihre Lambda-Funktion verwendet dieses Thema zum Veröffentlichen neuer Barks von Woofer-Benutzern.

1. Geben Sie den folgenden Befehl ein, um ein neues Amazon-SNS-Thema zu erstellen.

   ```
   aws sns create-topic --name wooferTopic
   ```

2. Geben Sie den folgenden Befehl ein, um `wooferTopic` für eine E-Mail-Adresse zu abonnieren. (Ersetzen Sie *`region`* bzw. *`accountID`* durch Ihre AWS -Region bzw. die Konto-ID und *`example@example.com`* durch eine gültige E-Mail-Adresse.)

   ```
   aws sns subscribe \
       --topic-arn arn:aws:sns:region:accountID:wooferTopic \
       --protocol email \
       --notification-endpoint example@example.com
   ```

3. Amazon SNS sendet eine Bestätigungsnachricht an Ihre E-Mail-Adresse. Klicken Sie auf den Link Confirm subscription (Abonnement bestätigen) in dieser E-Mail, um Ihr Abonnement abzuschließen.

Schritt 4: Erstellen und Testen einer Lambda-Funktion

In diesem Schritt erstellen Sie eine AWS Lambda Funktion (`publishNewBark`) zur Verarbeitung von `BarkTable` Stream-Datensätzen.

Die `publishNewBark`-Funktion verarbeitet nur die Stream-Ereignisse, die neuen Elementen in `BarkTable` entsprechen. Die Funktion liest Daten eines entsprechenden Ereignisses und ruft Amazon SNS auf, um es zu veröffentlichen.

1. Erstellen Sie eine Datei mit dem Namen `publishNewBark.js` und dem folgenden Inhalt. Ersetzen Sie *`region`* und *`accountID`* durch Ihre AWS Region und Konto-ID.

   ```
   'use strict';
   var AWS = require("aws-sdk");
   ```

```
var sns = new AWS.SNS();

exports.handler = (event, context, callback) => {

    event.Records.forEach((record) => {
        console.log('Stream record: ', JSON.stringify(record, null, 2));

        if (record.eventName == 'INSERT') {
            var who = JSON.stringify(record.dynamodb.NewImage.Username.S);
            var when = JSON.stringify(record.dynamodb.NewImage.Timestamp.S);
            var what = JSON.stringify(record.dynamodb.NewImage.Message.S);
            var params = {
                Subject: 'A new bark from ' + who,
                Message: 'Woofer user ' + who + ' barked the following at ' + when
 + ':\n\n ' + what,
                TopicArn: 'arn:aws:sns:region:accountID:wooferTopic'
            };
            sns.publish(params, function(err, data) {
                if (err) {
                    console.error("Unable to send message. Error JSON:",
 JSON.stringify(err, null, 2));
                } else {
                    console.log("Results from sending message: ",
 JSON.stringify(data, null, 2));
                }
            });
        }
    });
    callback(null, `Successfully processed ${event.Records.length} records.`);
};
```

2. Erstellen Sie eine ZIP-Datei mit `publishNewBark.js`. Wenn Sie das ZIP-Befehlszeilen-Dienstprogramm verwenden, geben Sie dazu den folgenden Befehl ein.

```
zip publishNewBark.zip publishNewBark.js
```

3. Beim Erstellen der Lambda-Funktion geben Sie den Amazon-Ressourcennamen (ARN) für `WooferLambdaRole` an, den Sie in Schritt 2: Erstellen einer Lambda-Ausführungsrolle erstellt haben. Geben Sie den folgenden Befehl ein, um diesen ARN abzurufen.

```
aws iam get-role --role-name WooferLambdaRole
```

Suchen Sie in der Ausgabe nach dem ARN für `WooferLambdaRole`.

```
...
"Arn": "arn:aws:iam::region:role/service-role/WooferLambdaRole"
...
```

Verwenden Sie den folgenden Befehl, um die Lambda-Funktion zu erstellen. *roleARN*Ersetzen Sie es durch den ARN für`WooferLambdaRole`.

```
aws lambda create-function \
    --region region \
    --function-name publishNewBark \
    --zip-file fileb://publishNewBark.zip \
    --role roleARN \
    --handler publishNewBark.handler \
    --timeout 5 \
    --runtime nodejs16.x
```

4. Testen Sie die `publishNewBark`-Funktion. Stellen Sie dazu eine Eingabe bereit, die einem echten Datensatz aus DynamoDB Streams ähnelt.

   Erstellen Sie eine Datei mit dem Namen `payload.json` und dem folgenden Inhalt. Ersetzen Sie *region* und *accountID* durch Ihre AWS-Region und die Konto-ID:

```
{
    "Records": [
        {
            "eventID": "7de3041dd709b024af6f29e4fa13d34c",
            "eventName": "INSERT",
            "eventVersion": "1.1",
            "eventSource": "aws:dynamodb",
            "awsRegion": "region",
            "dynamodb": {
                "ApproximateCreationDateTime": 1479499740,
                "Keys": {
                    "Timestamp": {
                        "S": "2016-11-18:12:09:36"
                    },
                    "Username": {
                        "S": "John Doe"
```

```
                    }
                },
                "NewImage": {
                    "Timestamp": {
                        "S": "2016-11-18:12:09:36"
                    },
                    "Message": {
                        "S": "This is a bark from the Woofer social network"
                    },
                    "Username": {
                        "S": "John Doe"
                    }
                },
                "SequenceNumber": "13021600000000001596893679",
                "SizeBytes": 112,
                "StreamViewType": "NEW_IMAGE"
            },
            "eventSourceARN": "arn:aws:dynamodb:region:account ID:table/BarkTable/
stream/2016-11-16T20:42:48.104"
        }
    ]
}
```

Verwenden Sie den folgenden Befehl, um die `publishNewBark`-Funktion zu erstellen.

```
aws lambda invoke --function-name publishNewBark --payload file://payload.json --
cli-binary-format raw-in-base64-out output.txt
```

Wenn der Test erfolgreich war, sehen Sie die folgende Ausgabe.

```
{
    "StatusCode": 200,
    "ExecutedVersion": "$LATEST"
}
```

Darüber hinaus enthält die `output.txt`-Datei folgenden Text.

```
"Successfully processed 1 records."
```

Außerdem erhalten Sie in wenigen Minuten eine neue E-Mail-Nachricht.

> **Note**
>
> AWS Lambda schreibt Diagnoseinformationen in Amazon CloudWatch Logs. Wenn Fehler mit Ihrer Lambda-Funktion auftreten, können Sie folgende Diagnoseverfahren zur Fehlerbehebung verwenden:
>
> 1. Öffnen Sie die CloudWatch Konsole unter https://console.aws.amazon.com/cloudwatch/.
> 2. Wählen Sie im Navigationsbereich Logs (Logs) aus.
> 3. Wählen Sie die folgende Protokollgruppe aus: `/aws/lambda/publishNewBark`
> 4. Wählen Sie den aktuellen Protokoll-Stream aus, um die Ausgabe (und Fehler) der Funktion zu sehen.

Schritt 5: Erstellen und Testen eines Auslösers

In Schritt 4: Erstellen und Testen einer Lambda-Funktion haben Sie die Lambda-Funktion getestet, um sicherzustellen, dass sie korrekt ausgeführt wird. In diesem Schritt erstellen Sie einen Auslöser, indem Sie die Lambda-Funktion (`publishNewBark`) mit einer Ereignisquelle (dem `BarkTable`-Stream) verknüpfen.

1. Wenn Sie den Auslöser erstellen, müssen Sie den ARN für den `BarkTable`-Stream angeben. Geben Sie den folgenden Befehl ein, um diesen ARN abzurufen.

   ```
   aws dynamodb describe-table --table-name BarkTable
   ```

   Suchen Sie in der Ausgabe nach dem `LatestStreamArn`.

   ```
   ...
    "LatestStreamArn": "arn:aws:dynamodb:region:accountID:table/BarkTable/
   stream/timestamp
   ...
   ```

2. Geben Sie den folgenden Befehl ein, um den Auslöser zu erstellen. Ersetzen Sie *`streamARN`* durch den tatsächlichen Stream-ARN.

   ```
   aws lambda create-event-source-mapping \
       --region region \
   ```

```
    --function-name publishNewBark \
    --event-source streamARN  \
    --batch-size 1 \
    --starting-position TRIM_HORIZON
```

3. Testen Sie den Auslöser. Geben Sie dazu den folgenden Befehl ein, um `BarkTable` ein Element hinzuzufügen.

```
aws dynamodb put-item \
    --table-name BarkTable \
    --item Username={S="Jane
 Doe"},Timestamp={S="2016-11-18:14:32:17"},Message={S="Testing...1...2...3"}
```

   Sie erhalten in wenigen Minuten eine neue E-Mail-Nachricht.

4. Öffnen Sie die DynamoDB-Konsole und fügen Sie `BarkTable` weitere Elemente hinzu. Sie müssen Werte für die Attribute `Username` und `Timestamp` angeben. (Sie sollten auch einen Wert für `Message` angeben, auch wenn er nicht erforderlich ist.) Sie erhalten eine neue E-Mail-Nachricht für jedes Element, das Sie `BarkTable` hinzufügen.

   Die Lambda-Funktion verarbeitet nur neue Elemente, die Sie `BarkTable` hinzufügen. Wenn Sie ein Element in der Tabelle aktualisieren oder löschen, hat die Funktion keine Auswirkung.

> **Note**
>
> AWS Lambda schreibt Diagnoseinformationen in Amazon CloudWatch Logs. Wenn Fehler mit Ihrer Lambda-Funktion auftreten, können Sie folgende Diagnoseverfahren zur Fehlerbehebung verwenden.
>
> 1. Öffnen Sie die CloudWatch Konsole unter https://console.aws.amazon.com/cloudwatch/.
> 2. Wählen Sie im Navigationsbereich Logs (Logs) aus.
> 3. Wählen Sie die folgende Protokollgruppe aus: `/aws/lambda/publishNewBark`
> 4. Wählen Sie den aktuellen Protokoll-Stream aus, um die Ausgabe (und Fehler) der Funktion zu sehen.

Tutorial 2: Verwenden von Filtern, um einige Ereignisse mit DynamoDB und Lambda zu verarbeiten

In diesem Tutorial erstellen Sie einen AWS Lambda Trigger, um nur einige Ereignisse in einem Stream aus einer DynamoDB-Tabelle zu verarbeiten.

Themen

- Alles zusammenfügen - CloudFormation
- Zusammenführung – CDK

Über die Lambda-Ereignisfilterung können Sie mithilfe von Filterausdrücken steuern, welche Ereignisse Lambda zur Verarbeitung an Ihre Funktion sendet. Sie können bis zu 5 verschiedene Filter pro DynamoDB-Streams konfigurieren. Wenn Sie Batching-Fenster verwenden, wendet Lambda die Filterkriterien auf jedes neue Ereignis an, um festzustellen, ob es dem aktuellen Batch hinzugefügt werden soll.

Filter werden über Strukturen, sogenannte `FilterCriteria`, angewendet. Die 3 Hauptattribute von `FilterCriteria` sind `metadata properties`, `data properties` und `filter patterns`.

Hier ist eine Beispielstruktur eines DynamoDB-Streams-Ereignisses:

```
{
  "eventID": "c9fbe7d0261a5163fcb6940593e41797",
  "eventName": "INSERT",
  "eventVersion": "1.1",
  "eventSource": "aws:dynamodb",
  "awsRegion": "us-east-2",
  "dynamodb": {
    "ApproximateCreationDateTime": 1664559083.0,
    "Keys": {
      "SK": { "S": "PRODUCT#CHOCOLATE#DARK#1000" },
      "PK": { "S": "COMPANY#1000" }
    },
    "NewImage": {
      "quantity": { "N": "50" },
      "company_id": { "S": "1000" },
      "fabric": { "S": "Florida Chocolates" },
      "price": { "N": "15" },
      "stores": { "N": "5" },
      "product_id": { "S": "1000" },
      "SK": { "S": "PRODUCT#CHOCOLATE#DARK#1000" },
      "PK": { "S": "COMPANY#1000" },
```

```
      "state": { "S": "FL" },
      "type": { "S": "" }
    },
    "SequenceNumber": "700000000000888747038",
    "SizeBytes": 174,
    "StreamViewType": "NEW_AND_OLD_IMAGES"
  },
  "eventSourceARN": "arn:aws:dynamodb:us-east-2:111122223333:table/chocolate-table-
StreamsSampleDDBTable-LUOI6UXQY7J1/stream/2022-09-30T17:05:53.209"
}
```

`metadata properties` sind die Felder des Ereignisobjekts. Im Falle von DynamoDB-Streams sind `metadata properties` Felder wie dynamodb oder `eventName`.

`data properties` sind die Felder des Ereignistexts. Um nach `data properties` zu filtern, müssen sie in `FilterCriteria` im richtigen Schlüssel eingeschlossen sein. Für DynamoDB-Ereignisquellen lautet der Datenschlüssel `NewImage` oder `OldImage`.

Schließlich definieren Filterregeln den Filterausdruck, den Sie auf eine bestimmte Eigenschaft anwenden möchten. Hier sind einige Beispiele:

Vergleichsoperator	Beispiel	Regelsyntax (partiell)
Null	Der Produkttyp ist null.	`{ "product_type": { "S": null } }`
Leer	Der Produktname ist leer.	`{ "product_name": { "S": [ ""] } }`
Gleich	Der Bundesstaat ist Florida.	`{ "state": { "S": ["FL"] } }`
Bedingung 2	Der Produktstatus ist Florida und die Produktkategorie „Chocolate“ (Schokolade).	`{ "state": { "S": ["FL"] } , "category": { "S": [ "CHOCOLATE"] } }`
Oder	Der Produktstatus ist Florida oder Kalifornien.	`{ "state": { "S": ["FL","CA"] } }`

Vergleichsoperator	Beispiel	Regelsyntax (partiell)
Nicht	Der Produktstatus ist nicht Florida.	`{"state": {"S": [{"anything-but": ["FL"]}]}}`
Vorhanden	Produkt „Homemade “ (Hausgemacht) ist vorhanden.	`{"homemade": {"S": [{"exists": true}]}}`
Nicht vorhanden	Produkt „Homemade “ (Hausgemacht) ist nicht vorhanden.	`{"homemade": {"S": [{"exists": false}]}}`
Beginnt mit	PK beginnt mit COMPANY (Unternehmen).	`{"PK": {"S": [{"prefix": "COMPANY"}]}}`

Sie können bis zu 5 Ereignisfilterungsmuster für eine Lambda-Funktion angeben. Beachten Sie, dass jedes dieser 5 Ereignisse als logisches ODER ausgewertet wird. Wenn Sie also zwei Filter namens `Filter_One` und `Filter_Two` konfigurieren, führt die Lambda-Funktion `Filter_One` ODER `Filter_Two` aus.

> **Note**
>
> Auf der Seite Lambda-Ereignisfilterung sind einige Optionen zum Filtern und Vergleichen numerischer Werte vorhanden. Im Falle von DynamoDB-Filterereignissen gelten diese jedoch nicht, da Zahlen in DynamoDB als Zeichenfolgen gespeichert werden. Bei `"quantity": { "N": "50" }` beispielsweise wissen wir aufgrund der Eigenschaft `"N"`, dass es sich um eine Zahl handelt.

Alles zusammenfügen - CloudFormation

Um die Funktionalität der Ereignisfilterung in der Praxis zu demonstrieren, finden Sie hier eine CloudFormation Beispielvorlage. Diese Vorlage generiert eine einfache DynamoDB-Tabelle mit einem Partitionsschlüssel PK und einem Sortierschlüssel SK mit aktiviertem Amazon DynamoDB Streams. Sie erstellt eine Lambda-Funktion und eine einfache Lambda-Ausführungsrolle, die das Schreiben von Protokollen in Amazon Cloudwatch und das Lesen der Ereignisse aus dem Amazon-

DynamoDB-Stream ermöglichen. Zudem fügt sie die Ereignisquellenzuordnung zwischen den DynamoDB-Streams und der Lambda-Funktion hinzu, sodass die Funktion bei jedem Ereignis im Amazon-DymamoDB-Stream ausgeführt werden kann.

```
AWSTemplateFormatVersion: "2010-09-09"

Description: Sample application that presents AWS Lambda event source filtering
with Amazon DynamoDB Streams.

Resources:
  StreamsSampleDDBTable:
    Type: AWS::DynamoDB::Table
    Properties:
      AttributeDefinitions:
        - AttributeName: "PK"
          AttributeType: "S"
        - AttributeName: "SK"
          AttributeType: "S"
      KeySchema:
        - AttributeName: "PK"
          KeyType: "HASH"
        - AttributeName: "SK"
          KeyType: "RANGE"
      StreamSpecification:
        StreamViewType: "NEW_AND_OLD_IMAGES"
      ProvisionedThroughput:
        ReadCapacityUnits: 5
        WriteCapacityUnits: 5

  LambdaExecutionRole:
    Type: AWS::IAM::Role
    Properties:
      AssumeRolePolicyDocument:
        Version: "2012-10-17",
        Statement:
          - Effect: Allow
            Principal:
              Service:
                - lambda.amazonaws.com
            Action:
              - sts:AssumeRole
      Path: "/"
      Policies:
```

```
        - PolicyName: root
          PolicyDocument:
            Version: "2012-10-17",
            Statement:
              - Effect: Allow
                Action:
                  - logs:CreateLogGroup
                  - logs:CreateLogStream
                  - logs:PutLogEvents
                Resource: arn:aws:logs:*:*:*
              - Effect: Allow
                Action:
                  - dynamodb:DescribeStream
                  - dynamodb:GetRecords
                  - dynamodb:GetShardIterator
                  - dynamodb:ListStreams
                Resource: !GetAtt StreamsSampleDDBTable.StreamArn

  EventSourceDDBTableStream:
    Type: AWS::Lambda::EventSourceMapping
    Properties:
      BatchSize: 1
      Enabled: True
      EventSourceArn: !GetAtt StreamsSampleDDBTable.StreamArn
      FunctionName: !GetAtt ProcessEventLambda.Arn
      StartingPosition: LATEST

  ProcessEventLambda:
    Type: AWS::Lambda::Function
    Properties:
      Runtime: python3.7
      Timeout: 300
      Handler: index.handler
      Role: !GetAtt LambdaExecutionRole.Arn
      Code:
        ZipFile: |
          import logging

          LOGGER = logging.getLogger()
          LOGGER.setLevel(logging.INFO)

          def handler(event, context):
            LOGGER.info('Received Event: %s', event)
            for rec in event['Records']:
```

```
            LOGGER.info('Record: %s', rec)

Outputs:
  StreamsSampleDDBTable:
    Description: DynamoDB Table ARN created for this example
    Value: !GetAtt StreamsSampleDDBTable.Arn
  StreamARN:
    Description: DynamoDB Table ARN created for this example
    Value: !GetAtt StreamsSampleDDBTable.StreamArn
```

Nachdem Sie diese Cloud-Formation-Vorlage bereitgestellt haben, können Sie das folgende Amazon-DynamoDB-Element einfügen:

```
{
 "PK": "COMPANY#1000",
 "SK": "PRODUCT#CHOCOLATE#DARK",
 "company_id": "1000",
 "type": "",
 "state": "FL",
 "stores": 5,
 "price": 15,
 "quantity": 50,
 "fabric": "Florida Chocolates"
}
```

Dank der einfachen Lambda-Funktion, die direkt in dieser Cloud-Formation-Vorlage enthalten ist, sehen Sie die Ereignisse in den CloudWatch Amazon-Protokollgruppen für die Lambda-Funktion wie folgt:

```
{
  "eventID": "c9fbe7d0261a5163fcb6940593e41797",
  "eventName": "INSERT",
  "eventVersion": "1.1",
  "eventSource": "aws:dynamodb",
  "awsRegion": "us-east-2",
  "dynamodb": {
    "ApproximateCreationDateTime": 1664559083.0,
    "Keys": {
      "SK": { "S": "PRODUCT#CHOCOLATE#DARK#1000" },
      "PK": { "S": "COMPANY#1000" }
    },
    "NewImage": {
```

```
      "quantity": { "N": "50" },
      "company_id": { "S": "1000" },
      "fabric": { "S": "Florida Chocolates" },
      "price": { "N": "15" },
      "stores": { "N": "5" },
      "product_id": { "S": "1000" },
      "SK": { "S": "PRODUCT#CHOCOLATE#DARK#1000" },
      "PK": { "S": "COMPANY#1000" },
      "state": { "S": "FL" },
      "type": { "S": "" }
    },
    "SequenceNumber": "70000000000888747038",
    "SizeBytes": 174,
    "StreamViewType": "NEW_AND_OLD_IMAGES"
  },
  "eventSourceARN": "arn:aws:dynamodb:us-east-2:111122223333:table/chocolate-table-
StreamsSampleDDBTable-LUOI6UXQY7J1/stream/2022-09-30T17:05:53.209"
}
```

Filterbeispiele

- Nur Produkte, die einem bestimmten Bundesstaat entsprechen

In diesem Beispiel wird die CloudFormation Vorlage dahingehend geändert, dass sie einen Filter enthält, der allen Produkten aus Florida mit der Abkürzung „FL“ entspricht.

```
EventSourceDDBTableStream:
    Type: AWS::Lambda::EventSourceMapping
    Properties:
      BatchSize: 1
      Enabled: True
      FilterCriteria:
        Filters:
          - Pattern: '{ "dynamodb": { "NewImage": { "state": { "S": ["FL"] } } } }'
      EventSourceArn: !GetAtt StreamsSampleDDBTable.StreamArn
      FunctionName: !GetAtt ProcessEventLambda.Arn
      StartingPosition: LATEST
```

Wenn Sie den Stack erneut bereitstellen, können Sie das folgende DynamoDB-Element zur Tabelle hinzufügen. Beachten Sie, dass es nicht in den Lambda-Funktionsprotokollen angezeigt wird, da das Produkt in diesem Beispiel aus Kalifornien stammt.

```
{
 "PK": "COMPANY#1000",
 "SK": "PRODUCT#CHOCOLATE#DARK#1000",
 "company_id": "1000",
 "fabric": "Florida Chocolates",
 "price": 15,
 "product_id": "1000",
 "quantity": 50,
 "state": "CA",
 "stores": 5,
 "type": ""
}
```

- Nur die Elemente, die mit einigen Werten im PK und SK beginnen

In diesem Beispiel wird die CloudFormation Vorlage so geändert, dass sie die folgende Bedingung enthält:

```
EventSourceDDBTableStream:
    Type: AWS::Lambda::EventSourceMapping
    Properties:
      BatchSize: 1
      Enabled: True
      FilterCriteria:
        Filters:
          - Pattern: '{"dynamodb": {"Keys": {"PK": { "S": [{ "prefix":
 "COMPANY" }] },"SK": { "S": [{ "prefix": "PRODUCT" }] }}}}'
      EventSourceArn: !GetAtt StreamsSampleDDBTable.StreamArn
      FunctionName: !GetAtt ProcessEventLambda.Arn
      StartingPosition: LATEST
```

Beachten Sie Folgendes: Die AND-Bedingung erfordert, dass sich die Bedingung innerhalb des Musters befindet, wobei sich die Schlüssel PK und SK in demselben Ausdruck, durch ein Komma getrennt, befinden.

Entweder mit einigen Werten für PK und SK beginnen oder stammt aus bestimmten Bundesstaat.

In diesem Beispiel wird die CloudFormation Vorlage so geändert, dass sie die folgenden Bedingungen enthält:

```
  EventSourceDDBTableStream:
```

```
    Type: AWS::Lambda::EventSourceMapping
    Properties:
      BatchSize: 1
      Enabled: True
      FilterCriteria:
        Filters:
          - Pattern: '{"dynamodb": {"Keys": {"PK": { "S": [{ "prefix":
"COMPANY" }] },"SK": { "S": [{ "prefix": "PRODUCT" }] }}}}'
          - Pattern: '{ "dynamodb": { "NewImage": { "state": { "S": ["FL"] } } } }'
      EventSourceArn: !GetAtt StreamsSampleDDBTable.StreamArn
      FunctionName: !GetAtt ProcessEventLambda.Arn
      StartingPosition: LATEST
```

Beachten Sie, dass die OR-Bedingung durch Einführung neuer Muster in den Filterabschnitt hinzugefügt wird.

Zusammenführung – CDK

In der folgenden Beispielvorlage für die CDK-Projektbildung werden die Funktionen zum Filtern von Ereignissen veranschaulicht. Bevor Sie mit diesem CDK-Projekt arbeiten können, müssen Sie die Voraussetzungen installieren. Dies beinhaltet auch die Ausführung von Vorbereitungsskripten.

Erstellen eines CDK-Projekts

Erstellen Sie zunächst ein neues AWS CDK Projekt, indem Sie es `cdk init` in einem leeren Verzeichnis aufrufen.

```
mkdir ddb_filters
cd ddb_filters
cdk init app --language python
```

Im Befehl `cdk init` wird der Name des Projektordners zur Benennung verschiedener Elemente des Projekts verwendet, einschließlich Klassen, Unterordnern und Dateien. Bindestriche im Ordnernamen werden in Unterstriche umgewandelt. Ansonsten sollte der Name dem Format eines Python-Bezeichners folgen. Er sollte beispielsweise nicht mit einer Zahl beginnen und keine Leerzeichen enthalten.

Um mit dem neuen Projekt zu arbeiten, aktivieren Sie seine virtuelle Umgebung. Dadurch können die Abhängigkeiten des Projekts lokal im Projektordner installiert werden und müssen nicht global installiert werden.

```
source .venv/bin/activate
```

```
python -m pip install -r requirements.txt
```

> **Note**
>
> Möglicherweise erkennen Sie dies als den Mac/Linux Befehl zur Aktivierung einer virtuellen Umgebung. Die Python-Vorlagen enthalten eine Batch-Datei, `source.bat`, die die Verwendung desselben Befehls unter Windows ermöglicht. Der traditionelle Windows-Befehl `.venv\Scripts\activate.bat` funktioniert ebenfalls. Wenn Sie Ihr AWS CDK Projekt mit AWS CDK Toolkit v1.70.0 oder früher initialisiert haben, befindet sich Ihre virtuelle Umgebung im `.env` Verzeichnis statt. `.venv`

Grundlegende Infrastruktur

Öffnen Sie die Datei `./ddb_filters/ddb_filters_stack.py` in einem Texteditor Ihrer Wahl. Diese Datei wurde auto generiert, als Sie das AWS CDK Projekt erstellt haben.

Fügen Sie als Nächstes die Funktionen `_create_ddb_table` und `_set_ddb_trigger_function` hinzu. Diese Funktionen erstellen eine DynamoDB-Tabelle mit dem Partitionsschlüssel PK und dem Sortierschlüssel SK im Bereitstellungsmodus/On-Demand-Modus, wobei Amazon DynamoDB Streams standardmäßig aktiviert ist, um neue und alte Bilder anzuzeigen.

Die Lambda-Funktion wird im Ordner `lambda` unter der Datei `app.py` gespeichert. Diese Datei wird später erstellt. Sie wird eine Umgebungsvariable, `APP_TABLE_NAME`, enthalten. Hierbei wird es sich um den Namen der Amazon-DynamoDB-Tabelle handeln, die von diesem Stack erstellt wurde. In derselben Funktion werden wir der Lambda-Funktion Stream-Leseberechtigungen erteilen. Schließlich wird sie die DynamoDB Streams als Ereignisquelle für die Lambda-Funktion abonnieren.

Am Ende der Datei, in der `__init__`-Methode, werden Sie die entsprechenden Konstrukte aufrufen, um sie im Stack zu initialisieren. Bei größeren Projekten, die zusätzliche Komponenten und Services erfordern, ist es möglicherweise am besten, diese Konstrukte außerhalb des Basis-Stacks zu definieren.

```
import os
import json

import aws_cdk as cdk
from aws_cdk import (
    Stack,
```

```
    aws_lambda as _lambda,
    aws_dynamodb as dynamodb,
)
from constructs import Construct


class DdbFiltersStack(Stack):

    def _create_ddb_table(self):
        dynamodb_table = dynamodb.Table(
            self,
            "AppTable",
            partition_key=dynamodb.Attribute(
                name="PK", type=dynamodb.AttributeType.STRING
            ),
            sort_key=dynamodb.Attribute(
                name="SK", type=dynamodb.AttributeType.STRING),
            billing_mode=dynamodb.BillingMode.PAY_PER_REQUEST,
            stream=dynamodb.StreamViewType.NEW_AND_OLD_IMAGES,
            removal_policy=cdk.RemovalPolicy.DESTROY,
        )

        cdk.CfnOutput(self, "AppTableName", value=dynamodb_table.table_name)
        return dynamodb_table

    def _set_ddb_trigger_function(self, ddb_table):
        events_lambda = _lambda.Function(
            self,
            "LambdaHandler",
            runtime=_lambda.Runtime.PYTHON_3_9,
            code=_lambda.Code.from_asset("lambda"),
            handler="app.handler",
            environment={
                "APP_TABLE_NAME": ddb_table.table_name,
            },
        )

        ddb_table.grant_stream_read(events_lambda)

        event_subscription = _lambda.CfnEventSourceMapping(
            scope=self,
            id="companyInsertsOnlyEventSourceMapping",
            function_name=events_lambda.function_name,
            event_source_arn=ddb_table.table_stream_arn,
```

```
            maximum_batching_window_in_seconds=1,
            starting_position="LATEST",
            batch_size=1,
        )

    def __init__(self, scope: Construct, construct_id: str, **kwargs) -> None:
        super().__init__(scope, construct_id, **kwargs)

        ddb_table = self._create_ddb_table()
        self._set_ddb_trigger_function(ddb_table)
```

Jetzt werden wir eine sehr einfache Lambda-Funktion erstellen, die die Protokolle in Amazon CloudWatch druckt. Erstellen Sie zu diesem Zweck einen neuen Ordner namens `lambda`.

```
mkdir lambda
touch app.py
```

Fügen Sie der Datei `app.py` über Ihren bevorzugten Texteditor den folgenden Inhalt hinzu:

```
import logging

LOGGER = logging.getLogger()
LOGGER.setLevel(logging.INFO)


def handler(event, context):
    LOGGER.info('Received Event: %s', event)
    for rec in event['Records']:
        LOGGER.info('Record: %s', rec)
```

Stellen Sie sicher, dass Sie sich im Ordner `/ddb_filters/` befinden, und geben Sie zum Erstellen der Beispielanwendung den folgenden Befehl ein:

```
cdk deploy
```

Irgendwann werden Sie aufgefordert, zu bestätigen, dass Sie die Lösung bereitstellen möchten. Akzeptieren Sie die Änderungen durch Eingabe von `Y`.

```
##################################################################################################
# + # ${LambdaHandler/ServiceRole} # arn:${AWS::Partition}:iam::aws:policy/service-
role/AWSLambdaBasicExecutionRole #
```

```
##############################################################################################

Do you wish to deploy these changes (y/n)? y

...

#  Deployment time: 67.73s

Outputs:
DdbFiltersStack.AppTableName = DdbFiltersStack-AppTable815C50BC-1M1W7209V5YPP
Stack ARN:
arn:aws:cloudformation:us-east-2:111122223333:stack/
DdbFiltersStack/66873140-40f3-11ed-8e93-0a74f296a8f6
```

Sobald die Änderungen bereitgestellt sind, öffnen Sie Ihre AWS Konsole und fügen Sie Ihrer Tabelle ein Element hinzu.

```
{
 "PK": "COMPANY#1000",
 "SK": "PRODUCT#CHOCOLATE#DARK",
 "company_id": "1000",
 "type": "",
 "state": "FL",
 "stores": 5,
 "price": 15,
 "quantity": 50,
 "fabric": "Florida Chocolates"
}
```

Die CloudWatch Protokolle sollten jetzt alle Informationen aus diesem Eintrag enthalten.

Filterbeispiele

- Nur Produkte, die einem bestimmten Bundesstaat entsprechen

Öffnen Sie die Datei `ddb_filters/ddb_filters/ddb_filters_stack.py` und ändern Sie sie so, dass sie den Filter enthält, der alle Produkte abgleicht, die gleich „FL“ sind. Dies kann direkt unter der `event_subscription` in Zeile 45 geändert werden.

```
event_subscription.add_property_override(
    property_path="FilterCriteria",
```

```
    value={
        "Filters": [
            {
                "Pattern": json.dumps(
                    {"dynamodb": {"NewImage": {"state": {"S": ["FL"]}}}}
                )
            },
        ]
    },
)
```

- Nur die Elemente, die mit einigen Werten im PK und SK beginnen

Ändern Sie das Python-Skript, um die folgende Bedingung aufzunehmen:

```
event_subscription.add_property_override(
    property_path="FilterCriteria",
    value={
        "Filters": [
            {
                "Pattern": json.dumps(
                    {
                        {
                            "dynamodb": {
                                "Keys": {
                                    "PK": {"S": [{"prefix": "COMPANY"}]},
                                    "SK": {"S": [{"prefix": "PRODUCT"}]},
                                }
                            }
                        }
                    }
                )
            },
        ]
    },
```

- Entweder mit einigen Werten für PK und SK beginnen oder stammt aus bestimmten Bundesstaat.

Ändern Sie das Python-Skript, um die folgenden Bedingungen aufzunehmen:

```
event_subscription.add_property_override(
```

```
    property_path="FilterCriteria",
    value={
        "Filters": [
            {
                "Pattern": json.dumps(
                    {
                        {
                            "dynamodb": {
                                "Keys": {
                                    "PK": {"S": [{"prefix": "COMPANY"}]},
                                    "SK": {"S": [{"prefix": "PRODUCT"}]},
                                }
                            }
                        }
                    }
                )
            },
            {
                "Pattern": json.dumps(
                    {"dynamodb": {"NewImage": {"state": {"S": ["FL"]}}}}
                )
            },
        ]
    },
)
```

Beachten Sie, dass die OR-Bedingung durch Hinzufügen weiterer Elemente zum Filter-Array hinzugefügt wird.

Bereinigen

Suchen Sie den Filter-Stack in der Basis Ihres Arbeitsverzeichnisses und führen Sie `cdk destroy` aus. Sie werden aufgefordert, das Löschen der Ressource zu bestätigen:

```
cdk destroy
Are you sure you want to delete: DdbFiltersStack (y/n)? y
```

Bewährte Methoden zur Verwendung von DynamoDB Streams mit Lambda

Eine AWS Lambda Funktion wird in einem Container ausgeführt — einer Ausführungsumgebung, die von anderen Funktionen isoliert ist. Wenn Sie eine Funktion zum ersten Mal ausführen, AWS Lambda wird ein neuer Container erstellt und mit der Ausführung des Codes der Funktion begonnen.

Eine Lambda-Funktion hat einen Handler, der einmal pro Aufruf ausgeführt wird. Der Handler enthält die Hauptgeschäftslogik für die Funktion. Die Lambda-Funktion in Schritt 4: Erstellen und Testen einer Lambda-Funktion verfügt z. B. über einen Handler, der Datensätze in einem DynamoDB-Stream verarbeiten kann.

Sie können auch Initialisierungscode bereitstellen, der nur einmal ausgeführt wird — nachdem der Container erstellt wurde, aber bevor der Handler zum ersten Mal AWS Lambda ausgeführt wird. Die unter gezeigte Lambda-Funktion Schritt 4: Erstellen und Testen einer Lambda-Funktion hat einen Initialisierungscode, der das SDK für JavaScript in Node.js importiert und einen Client für Amazon SNS erstellt. Diese Objekte sollten nur einmal außerhalb des Handlers definiert werden.

Nachdem die Funktion ausgeführt wurde, AWS Lambda können Sie sich dafür entscheiden, den Container für nachfolgende Aufrufe der Funktion wiederzuverwenden. In diesem Fall kann Ihr Funktions-Handler die Ressourcen, die Sie in Ihrem Initialisierungscode definiert haben, erneut nutzen. (Sie können nicht steuern, wie lange AWS Lambda den Container beibehält oder ob der Container überhaupt wiederverwendet wird.)

Für die Verwendung AWS Lambda von DynamoDB-Triggern empfehlen wir Folgendes:

- AWS Service-Clients sollten im Initialisierungscode instanziiert werden, nicht im Handler. Dies ermöglicht AWS Lambda die Wiederverwendung vorhandener Verbindungen für die Dauer der Lebensdauer des Containers.
- Im Allgemeinen müssen Sie Verbindungen nicht explizit verwalten oder Verbindungspooling implementieren, da AWS Lambda dies für Sie erledigt wird.

Ein Lambda-Verbraucher für einen DynamoDB-Stream garantiert nicht genau eine Lieferung und kann gelegentlich zu Duplikaten führen. Stellen Sie sicher, dass Ihr Lambda-Funktionscode idempotent ist, um zu verhindern, dass unerwartete Probleme aufgrund doppelter Verarbeitung auftreten.

Weitere Informationen finden Sie im AWS Lambda Entwicklerhandbuch unter Bewährte Methoden für die Arbeit mit AWS Lambda Funktionen.

DynamoDB Streams und Apache Flink

Sie können Amazon-DynamoDB-Streams-Datensätze mit Apache Flink aufnehmen. Mit Amazon Managed Service für Apache Flink lassen sich Streaming-Daten mithilfe von Apache Flink in Echtzeit umwandeln und analysieren. Apache Flink ist ein Open-Source-Verarbeitungsframework für Streams, mit dem Echtzeitdaten verarbeitet werden können. Der Amazon-DynamoDB-Streams-Connector

für Apache Flink vereinfacht die Erstellung und Verwaltung von Apache Flink-Workloads. Zudem ermöglicht er die Integration von Anwendungen in andere AWS-Services.

Amazon Managed Service für Apache Flink hilft Ihnen dabei, schnell end-to-end Stream-Verarbeitungsanwendungen für Protokollanalysen, Clickstream-Analysen, Internet der Dinge (IoT), Werbetechnologie, Spiele und mehr zu erstellen. Die vier häufigsten Anwendungsfälle sind Streaming extract-transform-load (ETL), ereignisgesteuerte Anwendungen, reaktionsschnelle Echtzeitanalysen und interaktive Abfragen von Datenströmen. Weitere Informationen dazu, wie Sie aus Amazon DynamoDB Streams in Apache Flink schreiben, finden Sie unter Amazon DynamoDB Streams Connector.

In-Memory-Beschleunigung mit DynamoDB Accelerator (DAX)

Amazon DynamoDB ist für die Skalierbarkeit und Leistung entwickelt. In den meisten Fällen sind die DynamoDB-Reaktionszeiten im einstelligen Millisekundenbereich. Allerdings gibt es gewisse Anwendungsfälle, die Reaktionszeiten in Mikrosekunden erfordern. Für diese Anwendungsfälle bietet DynamoDB Accelerator (DAX) schnelle Reaktionszeiten für den Zugriff auf Eventually-Consistent-Daten.

DAX ist ein DynamoDB-kompatibler Caching-Service, der es Ihnen ermöglicht, von schneller In-Memory-Leistung für anspruchsvolle Anwendungen zu profitieren. DAX befasst sich mit drei Kernszenarien:

1. Als In-Memory-Cache reduziert DAX die Reaktionszeiten der Workloads von Eventually-Consistent-Lesevorgängen um eine Zehnerpotenz von einstelligen Millisekunden zu Mikrosekunden.
2. DAX verringert die operative und Anwendungskomplexität, indem ein verwalteter Service, der API-kompatibel mit DynamoDB ist, bereitgestellt wird. Daher sind nur minimale funktionale Änderungen für die Nutzung mit einer vorhandenen Anwendung erforderlich.
3. Für leseintensive oder stoßweise auftretende Workloads bietet DAX einen erhöhten Durchsatz und potenzielle Kosteneinsparungen, indem weniger überdimensionierte Lesekapazitätseinheiten bereitgestellt werden. Dies ist besonders vorteilhaft für Anwendungen, die wiederholte Lesevorgänge für einzelne Schlüssel benötigen.

DAX unterstützt serverseitige Verschlüsselung. Wenn die Verschlüsselung im Ruhezustand ist, werden die Daten, die von DAX auf der Festplatte gespeichert sind, verschlüsselt. DAX schreibt Daten auf den Datenträger als Teil der Weitergabe von Änderungen vom primären Knoten an Read Replicas. Weitere Informationen finden Sie unter DAX-Verschlüsselung im Ruhezustand.

DAX unterstützt auch die Verschlüsselung bei der Übertragung. Dadurch wird sichergestellt, dass alle Anforderungen und Antworten zwischen Ihrer Anwendung und dem Cluster durch TLS (Transport Level Security) verschlüsselt werden und Verbindungen zum Cluster durch Überprüfung eines x509-Cluster-Zertifikats authentifiziert werden können. Weitere Informationen finden Sie unter DAX-Verschlüsselung während der Übertragung.

Themen

- Anwendungsfälle für DAX
- Nutzungshinweise für DAX
- DAX: So funktioniert es
- DAX-Cluster-Komponenten
- Erstellen eines DAX-Clusters
- DAX- und DynamoDB-Konsistenzmodelle
- Entwickeln mit dem DynamoDB-Accelerator-(DAX)-Client
- Verwalten von DAX-Clustern
- Überwachen von DynamoDB Accelerator
- DAX-T3/T2-Instances mit Spitzenlastleistung
- DAX-Zugriffskontrolle
- DAX-Verschlüsselung im Ruhezustand
- DAX-Verschlüsselung während der Übertragung
- Verwendung von serviceverknüpften IAM-Rollen für DAX
- AWS Kontenübergreifender Zugriff auf DAX
- DAX-Clustergrößenleitfaden

Anwendungsfälle für DAX

DAX bietet Zugriff auf Eventually-Consistent-Daten aus DynamoDB-Tabellen mit einer Latenz von Mikrosekunden. Ein Multi-AZ-DAX-Cluster kann Millionen von Abfragen pro Sekunde verarbeiten.

DAX eignet sich für folgende Anwendungsarten:

- Anwendungen, die eine schnellstmögliche Reaktionszeit für Lesevorgänge benötigen. Einige Beispiele sind Real Time Bidding, Social Gaming und Trading-Anwendungen. DAX liefert schnelle In-Memory-Leseleistung für diese Anwendungsfälle.
- Anwendungen, die eine geringe Anzahl von Elementen häufiger lesen als andere. Stellen Sie sich zum Beispiel ein E-Commerce-System vor, das einen eintägigen Verkauf eines beliebten Produkts anbietet. Während des Verkaufs würde die Nachfrage für dieses Produkt (und seiner Daten in DynamoDB) im Vergleich zu allen anderen Produkten drastisch steigen Um die Auswirkungen eines aktiven Schlüssels und einer nicht-einheitlichen Datenverteilung zu vermeiden, könnten Sie die Leseaktivität in einen DAX-Cache verlagern bis der eintägige Verkauf vorbei ist.

- Anwendungen, die leseintensiv, aber auch kostensensibel sind. Mit DynamoDB stellen Sie die Anzahl der Lesevorgänge pro Sekunde bereit, die Ihre Anwendung benötigt. Wenn die Leseaktivität steigt, können Sie den bereitgestellten Lesedurchsatz Ihrer Tabelle erhöhen (gegen Aufpreis). Sie können auch die Aktivitäten Ihrer Anwendung in einen DAX-Cluster verlagern und die Menge an Lesekapazitätseinheiten, die Sie sonst erwerben müssten, verringern.
- Anwendungen, die wiederholte Lesevorgänge für große Datensätze benötigen. Solch eine Anwendung könnte möglicherweise Datenbankressourcen von anderen Anwendungen umleiten. Beispielsweise könnte eine lang andauernde Analyse von regionalen Wetterdaten zeitweise die gesamte Lesekapazität in einer DynamoDB-Tabelle verbrauchen. Dies hätte negative Auswirkungen auf andere Anwendungen, die auf dieselben Daten zugreifen müssen. Mit DAX könnte die Wetteranalyse stattdessen gegen zwischengespeicherte Daten durchgeführt werden.

DAX ist für folgende Anwendungsarten nicht geeignet:

- Anwendungen, die Strongly Consistent-Lesevorgänge erfordern (oder keine Eventually Consistent-Lesevorgänge akzeptieren können).
- Anwendungen, die keine Reaktionszeiten im Mikrosekundenbereich für Lesevorgänge benötigen oder die keine wiederholte Leseaktivitäten von zugrunde liegenden Tabellen auslagern müssen.
- Anwendungen, die schreibintensiv sind. Ein hohes Schreibvolumen führt zu einer erhöhten Replikation zwischen DAX-Knoten in einem Cluster. Dies führt wiederum zu einem erhöhten Ressourcenverbrauch und dem Risiko von Verfügbarkeitsproblemen.
- Anwendungen ohne viele wiederholte Lesevorgänge. DAX bietet die beste Leistung, wenn die Cache-Trefferraten 90 % überschreiten. Niedrigere Cache-Trefferraten erhöhen die Anzahl der Cache-Fehler, wodurch mehr Ressourcen im DAX-Cluster verbraucht werden.

Nutzungshinweise für DAX

- Eine Liste der AWS Regionen, in denen DAX verfügbar ist, finden Sie unter Amazon DynamoDB DynamoDB-Preise.
- DAX unterstützt Anwendungen, die in Go, Java, Node.js, Python und .NET geschrieben wurden, und verwendet AWS dafür bereitgestellte Clients für diese Programmiersprachen.
- DAX ist ausschließlich für die EC2-VPC-Plattform verfügbar.
- Die DAX-Cluster-Service-Rolle-Richtlinie muss die Aktion `dynamodb:DescribeTable` zulassen, damit Metadaten über die DynamoDB-Tabelle beibehalten werden.

- DAX-Cluster halten Metadaten zu den Attributnamen der von ihnen gespeicherten Elemente bei. Diese Metadaten werden unbegrenzt gespeichert (und zwar sogar, nachdem das Element abgelaufen ist oder aus dem Cache entfernt wurde). Anwendungen, die eine unbegrenzte Anzahl von Attributnamen verwenden, können im Laufe der Zeit zu einer Erschöpfung des Speichers im DAX-Cluster führen. Diese Einschränkung gilt nur für Attributnamen auf oberster Ebene und nicht für verschachtelte Attributnamen. Beispiele für problematische Attributnamen der obersten Ebene sind Zeitstempel und Sitzung UUIDs. IDs

  Diese Einschränkung gilt nur für Attributnamen und nicht für deren Werte. Elemente wie die Folgenden stellen kein Problem dar.

```
{
    "Id": 123,
    "Title": "Bicycle 123",
    "CreationDate": "2017-10-24T01:02:03+00:00"
}
```

  Bei Elementen wie den folgenden ist dies jedoch der Fall, wenn sie in entsprechender Anzahl vorhanden sind und jedes von ihnen über einen anderen Zeitstempel verfügt:

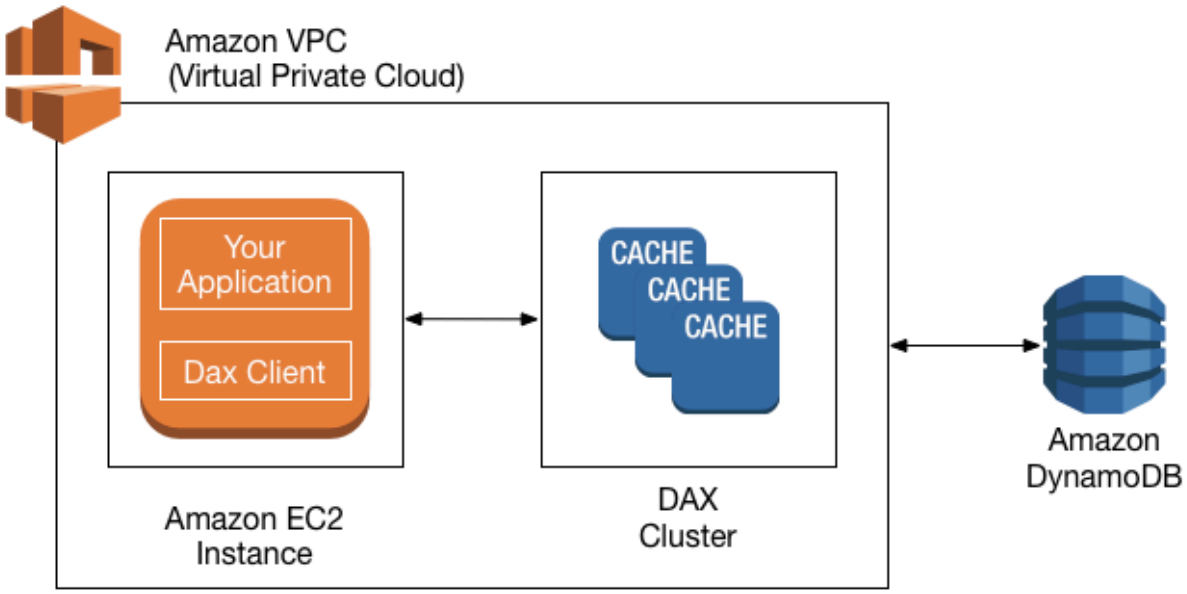
```
{
    "Id": 123,
    "Title": "Bicycle 123",
    "2017-10-24T01:02:03+00:00": "created"
}
```

DAX: So funktioniert es

Amazon DynamoDB Accelerator (DAX) wurde für die Ausführung in einer Amazon-Virtual-Private-Cloud-(Amazon-VPC)-Umgebung entwickelt. Der Amazon-VPC-Service definiert ein virtuelles Netzwerk, das einem herkömmlichen Rechenzentrum sehr ähnlich ist. Mit einer VPC können Sie den zugehörigen IP-Adressbereich, die Subnetze, Routing-Tabellen, Netzwerk-Gateways und Sicherheitseinstellungen steuern. Sie können einen DAX-Cluster in Ihrem virtuellen Netzwerk starten und den Zugriff auf den Cluster über Amazon-VPC-Sicherheitsgruppen steuern.

> **Note**
>
> Wenn Sie Ihr AWS Konto nach dem 4. Dezember 2013 erstellt haben, haben Sie in jeder AWS Region bereits eine Standard-VPC. Die VPC kann sofort verwendet werden – ohne zusätzliche Konfigurationsschritte ausführen zu müssen.
> Weitere Informationen finden Sie unter Standard-VPC und Standard-Subnetze im Amazon-VPC-Benutzerhandbuch.

Das folgende Diagramm zeigt einen allgemeinen Überblick über DAX.



Für die Erstellung eines DAX-Clusters verwenden Sie AWS-Managementkonsole. Sofern nicht anders von Ihnen festgelegt, wird Ihr DAX-Cluster in Ihrer Standard-VPC ausgeführt. Um Ihre Anwendung auszuführen, starten Sie eine Amazon-EC2-Instance in Ihrer Amazon VPC. Anschließend stellen Sie Ihre Anwendung (mit dem bereitstellen-Client) auf der EC2-Instance bereit.

Der DAX-Client leitet zur Laufzeit alle DynamoDB-API-Anforderungen der Anwendung an den DAX-Cluster weiter. Wenn DAX eine dieser API-Anforderungen direkt verarbeiten kann, erfolgt die Verarbeitung auch direkt. Andernfalls wird die Anforderung an DynamoDB übergeben.

Schließlich gibt der DAX-Cluster die Ergebnisse an Ihre Anwendung zurück.

Themen

- Wie DAX-Anforderungen verarbeitet
- Element-Cache
- Abfrage-Cache

Wie DAX-Anforderungen verarbeitet

Ein DAX-Cluster besteht aus einem oder mehreren Knoten. Auf jedem Knoten wird eine eigene Instance der DAX-Caching-Software ausgeführt. Einer der Knoten dient als primärer Knoten für den Cluster. Weitere Knoten (sofern vorhanden) dienen als Read Replicas. Weitere Informationen finden Sie unter Knoten.

Ihre Anwendung kann auf DAX zugreifen, wenn der Endpunkt für den DAX-Cluster angegeben wird. Die DAX-Client-Software arbeitet mit dem Cluster-Endpunkt zusammen, um einen intelligenten Lastausgleich und intelligentes Routing zu erreichen.

Lesevorgänge

DAX kann auf die folgenden API-Aufrufe antworten:

- `GetItem`
- `BatchGetItem`
- `Query`
- `Scan`

Wenn die Anforderung Eventually-Consistent-Lesevorgänge (Standardverhalten) angibt, versucht sie, das Element in DAX zu lesen:

- Wenn DAX das Element verfügbar hat (Cache-Treffer), gibt DAX das Element an die Anwendung ohne Zugriff auf DynamoDB zurück.
- Wenn DAX nicht über das Element verfügt (Cache-Fehlschlag), leitet DAX die Anforderung an DynamoDB weiter. Wenn die Antwort von DynamoDB eingeht, gibt DAX die Ergebnisse an die Anwendung zurück. Die Ergebnisse werden jedoch auch in den Cache auf dem primären Knoten geschrieben.

> **Note**
>
> Wenn im Cluster Lesereplikate vorhanden sind, sorgt DAX automatisch dafür, dass die Replikate mit dem primären Knoten synchron sind. Weitere Informationen finden Sie unter Cluster.

Wenn die Anforderung Strongly-Consistent-Lesevorgänge angibt, leitet DAX die Anforderung an DynamoDB weiter. Die Ergebnisse von DynamoDB werden nicht im Cache von DAX gespeichert. Sie werden lediglich an die Anwendung zurückgegeben.

Schreibvorgänge

Die folgenden DAX-API-Operationen werden als „Write-Through“ bezeichnet:

- `BatchWriteItem`
- `UpdateItem`
- `DeleteItem`
- `PutItem`

Mit diesen Operationen werden die Daten zuerst in die DynamoDB-Tabelle geschrieben und anschließend in den DAX-Cluster. Die Operation ist nur erfolgreich, wenn die Daten erfolgreich in die Tabelle und in DAX geschrieben werden.

Andere Operationen

DAX erkennt keine DynamoDB-Operationen für die Verwaltung von Tabellen (z. B. `CreateTable`, `UpdateTable` und usw.). Wenn die Anwendung diese Operationen ausführen muss, muss sie direkt auf DynamoDB zugreifen, anstatt DAX zu verwenden.

Ausführliche Informationen zu DAX und DynamoDB-Konsistenz finden Sie unter DAX- und DynamoDB-Konsistenzmodelle.

Hinweise zur Funktionsweise von Transaktionen in DAX finden Sie unter Verwenden von Transactional APIs in DynamoDB Accelerator (DAX).

Anforderungsratenbegrenzung

Wenn die Anzahl der an DAX gesendeten Anfragen die Kapazität eines Knotens übersteigt, begrenzt DAX die Rate, mit der zusätzliche Anfragen akzeptiert werden, indem es a ThrottlingExceptionzurückgibt. Zur Bestimmung der Menge der Anforderungen, die verarbeitet werden können, wertet DAX Ihre CPU-Auslastung kontinuierlich aus, während ein fehlerfreier Clusterstatus beibehalten wird.

Sie können die ThrottledRequestCount Metrik überwachen, die DAX auf Amazon veröffentlicht CloudWatch. Wenn diese Ausnahmen regelmäßig angezeigt werden, sollten Sie die Skalierung des Clusters in Erwägung ziehen.

Element-Cache

DAX pflegt einen Element-Cache, um die Ergebnisse von `GetItem`- und `BatchGetItem`-Operationen zu speichern. Die Elemente im Cache stellen letztlich konsistente Daten von DynamoDB dar. Sie werden anhand ihrer Primärschlüsselwerte gespeichert.

Wenn eine Anwendung eine `GetItem`- oder `BatchGetItem`-Anforderung sendet, versucht DAX, die Elemente unter Verwendung der angegebenen Schlüsselwerte direkt aus dem Element-Cache zu lesen. Werden die Elemente gefunden (Cache-Treffer), gibt DAX sie umgehend an die Anwendung zurück. Wenn die Elemente nicht gefunden werden (Cache-Fehler), sendet DAX die Anforderung an DynamoDB. DynamoDB verarbeitet die Anforderungen mithilfe von Eventually-Consistent-Lesevorgängen und gibt die Elemente an DAX zurück. DAX speichert sie im Element-Cache und gibt sie dann an die Anwendung zurück.

Der Element-Cache verfügt über eine Time-to-Live-Einstellung (TTL), die standardmäßig auf 5 Minuten festgelegt ist. DAX weist jedem Element einen Zeitstempel zu, das in den Element-Cache geschrieben wird. Ein Element läuft ab, wenn es im Cache länger als wie in der TTL-Einstellung angegeben vorhanden ist. Wenn Sie eine `GetItem`-Anforderung für ein abgelaufenes Element ausgeben, wird dies als Cache-Fehler betrachtet und DAX sendet die `GetItem`-Anforderung an DynamoDB.

> **Note**
>
> Sie können die TTL-Einstellung für den Element-Cache beim Erstellen des neuen DAX-Clusters festlegen. Weitere Informationen finden Sie unter Verwalten von DAX-Clustern.

DAX verwaltet auch eine LRU-Liste (Least Recently Used) für den Element-Cache. Mit der LRU-Liste wird verfolgt, wann ein Element zuerst in den Cache geschrieben und wann es zuletzt aus dem Cache gelesen wurde. Wenn der Element-Cache voll wird, bereinigt DAX ältere Elemente (auch wenn sie noch nicht abgelaufen sind), um Platz für neue Elemente zu schaffen. Der LRU-Algorithmus ist für den Element-Cache immer aktiviert. Er ist vom Benutzer nicht konfigurierbar.

Wenn Sie Null als Element-Cache-TTL-Einstellung angeben, werden Elemente im Element-Cache nur aufgrund einer LRU-Bereinigung oder einer „Write-Through“-Operation aktualisiert.

Ausführliche Hinweise zur Konsistenz des Element-Caches in DAX finden Sie unter Verhalten des DAX-Element-Caches.

Abfrage-Cache

DAX pflegt auch einen Abfrage-Cache, um die Ergebnisse von `Query`- und `Scan`-Operationen zu speichern. Die Elemente in diesem Cache stellen die Ergebnissätze von Abfragen und Scans auf DynamoDB-Tabellen dar. Diese Ergebnissätze werden anhand ihrer Parameterwerte gespeichert.

Wenn eine Anwendung eine `Query`- oder `Scan`-Anforderung sendet, versucht DAX unter Verwendung der angegebenen Parameterwerte, einen übereinstimmenden Ergebnissatz aus dem Abfrage-Cache zu lesen. Wird der Ergebnissatz gefunden (Cache Hit), gibt DAX ihn umgehend an die Anwendung zurück. Wenn der Ergebnissatz nicht gefunden wird (Cache-Fehler), sendet DAX die Anforderung an DynamoDB. DynamoDB verarbeitet die Anforderungen mit Eventually-Consistent-Lesevorgängen und gibt den Ergebnissatz an DAX zurück. DAX speichert ihn im Abfrage-Cache und gibt ihn dann an die Anwendung zurück.

> **Note**
>
> Sie können die TTL-Einstellung für den Abfrage-Cache beim Erstellen eines neuen DAX-Clusters festlegen. Weitere Informationen finden Sie unter Verwalten von DAX-Clustern.

DAX verwaltet auch eine LRU-Liste (Least Recently Used) für den Abfrage-Cache. Mit der LRU-Liste wird verfolgt, wann ein Ergebnissatz zuerst in den Cache geschrieben und wann das Ergebnis zuletzt aus dem Cache gelesen wurde. Wenn der Abfrage-Cache voll wird, bereinigt DAX ältere Ergebnissätze (auch wenn sie noch nicht abgelaufen sind), um Platz für neue Ergebnissätze zu schaffen. Der LRU-Algorithmus ist für den Abfrage-Cache immer aktiviert. Er ist vom Benutzer nicht konfigurierbar.

Wenn Sie Null als Abfrage-Cache TTL-Einstellung, wird die Abfrageantwort nicht zwischengespeichert.

Ausführliche Hinweise zur Konsistenz des Abfrage-Caches in DAX finden Sie unter Verhalten des DAX-Abfrage-Caches.

DAX-Cluster-Komponenten

Ein Amazon-DynamoDB-Accelerator-(DAX)-Cluster besteht aus AWS-Infrastrukturkomponenten. In diesem Abschnitt werden diese Komponenten und ihre Zusammenarbeit beschrieben.

Themen

- Knoten
- Cluster
- Regionen und Availability Zones
- Parametergruppen
- Sicherheitsgruppen
- Cluster-ARN
- Cluster-Endpunkt
- Knotenendpunkte
- Subnetzgruppen
- --Ereignisse
- Wartungsfenster

Knoten

Ein Knoten ist der kleinste Baustein eines DAX-Clusters. Jeder Knoten führt eine Instance der DAX-Software aus und unterhält ein einzelnes Replikat der zwischengespeicherten Daten.

Sie können Ihren DAX-Cluster auf zwei Arten skalieren:

- Durch Hinzufügen weiterer Knoten im Cluster. Dies erhöht den Gesamt-Lesedurchsatz des Clusters.
- Durch das Verwenden eines größeren Knotentyps. Größere Knotentypen bieten mehr Kapazität und können den Durchsatz erhöhen. (Sie müssen einen neuen Cluster mit dem neuen Knotentyp erstellen.)

Jeder Knoten in einem Cluster hat denselben Knotentyp und führt dieselbe DAX-Caching-Software aus. Eine Liste der verfügbaren Knotentypen finden Sie unter Amazon-DynamoDB-Preise.

Cluster

Ein Cluster ist eine logische Gruppierung von einem oder mehreren Knoten, die DAX als eine Einheit verwaltet. Einer der Knoten im Cluster wird als primärer Knoten ausgewiesen und die anderen Knoten (falls vorhanden) sind Read Replicas.

Der primäre Knoten ist für Folgendes verantwortlich:

- Erfüllen der Anwendungsanforderungen für zwischengespeicherte Daten.
- Handhaben der Schreibvorgänge in DynamoDB.
- Bereinigen von Daten aus dem Cache, entsprechend der Bereinigungsrichtlinie des Clusters.

Wenn Änderungen an zwischengespeicherten Daten auf dem Primärknoten vorgenommen werden, verteilt DAX die Änderungen unter Verwendung von Replikationsprotokollen auf alle Lesereplikat-Knoten. Nachdem die Bestätigung von allen Lesereplikaten eingegangen ist, löscht DynamoDB die Replikationsprotokolle vom Primärknoten.

Ein DAX-Cluster unterstützt bis zu elf Knoten pro Cluster (den primären Knoten, plus maximal zehn Lesereplikate).

Read Replicas sind verantwortlich für Folgendes:

- Erfüllen der Anwendungsanforderungen für zwischengespeicherte Daten.
- Bereinigen von Daten aus dem Cache, entsprechend der Bereinigungsrichtlinie des Clusters.

Im Gegensatz zu dem primären Knoten schreiben Lesereplikate jedoch nicht in DynamoDB.

Read Replicas dienen zwei zusätzlichen Zwecken:

- Skalierbarkeit. Wenn Sie über eine große Anzahl von Anwendungs-Clients verfügen, die gleichzeitig auf DAX zugreifen müssen, können Sie weitere Replikate für die Leseskalierung hinzufügen. DAX verteilt die Last gleichmäßig auf alle Knoten im Cluster. (Eine weitere Möglichkeit zur Erhöhung des Durchsatzes ist die Verwendung größerer Cache-Knoten-Typen.)
- Hohe Verfügbarkeit. Beim Ausfall eines primären Knotens, schaltet DAX automatisch auf ein Lesereplikat um und bestimmt es zum neuen primären Knoten. Wenn ein Replikat-Knoten

ausfällt, können andere Instances im DAX-Cluster nach wie vor für die Bearbeitung von Anfragen verwendet werden, bis der ausgefallene Knoten wiederhergestellt werden kann. Für maximale Fehlertoleranz sollten Sie Read Replicas in separaten Availability Zones (Verfügbarkeitszonen) bereitstellen. Diese Konfiguration stellt sicher, dass Ihr DAX-Cluster weiterhin ausgeführt werden kann, auch wenn die gesamte Availability Zone nicht mehr verfügbar ist.

Important

Für die Produktionsnutzung empfehlen wir dringend, DAX mit mindestens drei Knoten zu verwenden, in dem die Knoten in verschiedene Availability Zones platziert werden. Drei Knoten sind aus Gründen der Fehlertoleranz für einen DAX-Cluster erforderlich.
Ein DAX-Cluster kann für Bereitstellungs- oder Test-Workloads mit nur einem oder zwei Knoten bereitgestellt werden. Cluster mit einem oder zwei Knoten sind jedoch nicht fehlertolerant. Für die Produktionsnutzung empfehlen wir daher die Nutzung von mindestens drei Knoten. Wenn bei Clustern mit einem oder zwei Knoten Software- oder Hardwarefehler auftreten, ist der Cluster möglicherweise nicht mehr verfügbar oder es gehen zwischengespeicherte Daten verloren.

Important

Ein DAX-Cluster unterstützt maximal 500 DynamoDB-Tabellen. Wenn Sie mehr als 500 Tabellen verwenden, kann es bei Ihrem Cluster zu einer Verschlechterung der Verfügbarkeit und Leistung kommen.

Regionen und Availability Zones

Ein DAX-Cluster in einer AWS-Region kann nur mit DynamoDB-Tabellen interagieren, die sich in derselben Region befinden. Stellen Sie daher sicher, dass Sie Ihren DAX-Cluster in der richtigen Region starten. Wenn Sie über DynamoDB-Tabellen in anderen Regionen verfügen, müssen Sie die DAX-Cluster in diesen Regionen ebenfalls starten.

Jede -Region ist darauf ausgelegt, vollständig von den anderen -Regionen getrennt zu sein. Innerhalb jeder Region gibt es mehrere Availability Zones. Durch das Starten Ihrer Knoten in verschiedenen Availability Zones können Sie eine größtmögliche Fehlertoleranz zu erreichen.

> **Important**
>
> Platzieren Sie nicht alle Ihre Cluster-Knoten in eine einzige Availability Zone. Bei dieser Konfiguration ist Ihr DAX-Cluster bei einem Ausfall einer Availability Zone nicht mehr verfügbar.
> Für die Produktionsnutzung empfehlen wir dringend, DAX mit mindestens drei Knoten zu verwenden, in dem die Knoten in verschiedene Availability Zones platziert werden. Drei Knoten sind aus Gründen der Fehlertoleranz für einen DAX-Cluster erforderlich.
> Ein DAX-Cluster kann für Bereitstellungs- oder Test-Workloads mit nur einem oder zwei Knoten bereitgestellt werden. Cluster mit einem oder zwei Knoten sind jedoch nicht fehlertolerant. Für die Produktionsnutzung empfehlen wir daher die Nutzung von mindestens drei Knoten. Wenn bei Clustern mit einem oder zwei Knoten Software- oder Hardwarefehler auftreten, ist der Cluster möglicherweise nicht mehr verfügbar oder es gehen zwischengespeicherte Daten verloren.

Parametergruppen

Parametergruppen werden verwendet, um Laufzeiteinstellungen für DAX-Clusters zu verwalten. DAX verfügt über mehrere Parameter, die Sie verwenden können, um die Leistung zu optimieren (z. B. Definieren einer TTL-Richtlinie für zwischengespeicherte Daten). Eine Parametergruppe ist eine benannte Sammlung von Parametern, die Sie einem Cluster zuweisen können. Dadurch stellen Sie sicher, dass alle Knoten in diesem Cluster identisch konfiguriert werden.

Sicherheitsgruppen

Ein DAX-Cluster wird in einer Amazon-Virtual-Private-Cloud-Umgebung (Amazon VPC) ausgeführt. Bei dieser Umgebung handelt es sich um ein virtuelles Netzwerk, das für das betreffende AWS-Konto spezifisch und zugleich von anderen VPCs isoliert ist. Eine Sicherheitsgruppe dient als virtuelle Firewall für die VPC und ermöglicht das Steuern des ein- und ausgehenden Netzwerkdatenverkehrs.

Wenn Sie einen Cluster in Ihrer VPC starten, fügen Sie Ihrer Sicherheitsgruppe eine Zugangsregel hinzu, um eingehenden Netzwerkverkehr zu erlauben. Die Zugangsregel legt das Protokoll (TCP) und die Portnummer (8111) für den Cluster fest. Nachdem Sie diese Zugangsregel Ihrer Sicherheitsgruppe hinzugefügt haben, können Ihre Anwendungen, die innerhalb Ihrer VPC ausgeführt werden, auf den DAX-Cluster zugreifen.

Cluster-ARN

Jedem DAX-Cluster wird ein Amazon-Ressourcenname (ARN) zugewiesen. Das ARN-Format lautet folgendermaßen.

```
arn:aws:dax:region:accountID:cache/clusterName
```

Sie verwenden den Cluster-ARN in einer IAM-Richtlinie, um Berechtigungen für DAX-API-Operationen zu definieren. Weitere Informationen finden Sie unter DAX-Zugriffskontrolle.

Cluster-Endpunkt

Jeder DAX-Cluster bietet einen Cluster-Endpunkt für Ihre Anwendung. Durch den Zugriff auf den Cluster mithilfe des Endpunkts muss Ihre Anwendung die Hostnamen und Portnummern einzelner Knoten im Cluster nicht kennen. Ihre Anwendung "kennt" automatisch alle Knoten im Cluster, auch wenn Sie Read Replicas hinzufügen oder entfernen.

Das folgende Beispiel zeigt einen Cluster-Endpunkt in der Region us-east-1, der nicht für die Verwendung der Verschlüsselung beim Transit konfiguriert ist.

```
dax://my-cluster.l6fzcv.dax-clusters.us-east-1.amazonaws.com
```

Das folgende Beispiel zeigt einen Cluster-Endpunkt in derselben Region, der für die Verwendung der Verschlüsselung beim Transit konfiguriert ist.

```
daxs://my-encrypted-cluster.l6fzcv.dax-clusters.us-east-1.amazonaws.com
```

Knotenendpunkte

Jeder der einzelnen Knoten in einem DAX-Cluster hat seinen eigenen Hostnamen und seine eigene Portnummer. Das folgende Beispiel zeigt einen Knotenendpunkt.

```
myDAXcluster-a.2cmrwl.clustercfg.dax.use1.cache.amazonaws.com:8111
```

Ihre Anwendung kann direkt auf einen Knoten zugreifen, indem sie seinen Endpunkt nutzt. Allerdings empfehlen wir, dass Sie den DAX-Cluster als Einzeleinheit behandeln und stattdessen mithilfe des Cluster-Endpunkts darauf zugreifen. Der Cluster-Endpunkt schützt Ihre Anwendung davor, eine Liste von Knoten pflegen zu müssen und diese Liste auf dem neuesten Stand zu halten, wenn Sie dem Cluster Knoten hinzufügen oder entfernen.

Subnetzgruppen

Der Zugriff auf DAX-Cluster-Knoten ist auf Anwendungen beschränkt, die auf Amazon-EC2-Instances in einer Amazon-VPC-Umgebung ausgeführt werden. Sie können Subnetzgruppen verwenden, um Clusters Zugriff von Amazon-EC2-Instances zu gewähren, die auf bestimmten Subnetzen ausgeführt werden. Eine Subnetzgruppe ist eine Sammlung von Subnetzen (in der Regel private Subnetze), die Sie für Ihre in einer Amazon-VPC-Umgebung ausgeführten Cluster festlegen können.

Wenn Sie einen DAX-Cluster erstellen, müssen Sie eine Subnetzgruppe angeben. DAX verwendet diese Subnetzgruppe, um ein Subnetz und IP-Adressen innerhalb dieses Subnetzes auszuwählen und diese mit Ihrem Knoten zu verknüpfen.

--Ereignisse

DAX zeichnet wichtige Ereignisse innerhalb Ihres Clusters auf, wie beispielsweise einen Fehlschlag beim Hinzufügen eines Knotens, einem erfolgreich hinzugefügten Knoten oder Änderungen an Sicherheitsgruppen. Durch die Überwachung wichtiger Schlüsselereignisse, können Sie den aktuellen Status Ihrer Clusters erfahren und, je nach Ereignis, in der Lage sein, Abhilfemaßnahmen zu ergreifen. Sie können auf diese Ereignisse mit AWS-Managementkonsole oder der `DescribeEvents`-Aktion in der DAX-Verwaltungs-API zugreifen.

Sie können auch anfordern, dass Benachrichtigungen an ein bestimmtes Amazon-SNS-Thema (Amazon Simple Notification Service) gesendet werden. So erfahren Sie es sofort, wenn ein Ereignis im DAX-Cluster auftritt.

Wartungsfenster

Jeder Cluster verfügt über ein wöchentliches Wartungsfenster, in dem Systemänderungen angewendet werden. Die Änderungen werden nacheinander angewendet. Dabei wird ein vorhandener Knoten ersetzt und ein neuer Knoten mit den angewendeten Änderungen wird dem Cluster hinzugefügt. Während dieses Zeitraums kann es in Ihrer Anwendung zu vorübergehenden Fehlern oder Drosselungen kommen. Daher empfehlen wir, dass Sie das Wartungsfenster auf die Zeiten mit der geringsten Nutzung legen und diesen Zeitplan bei Bedarf regelmäßig anpassen. Sie können einen Zeitraum mit einer Dauer von bis zu 24 Stunden festlegen, in dem alle angeforderten Wartungsaktivitäten durchgeführt werden sollten.

Falls Sie bei der Erstellung oder Änderung eines Cache-Clusters kein bevorzugtes Wartungsfenster angegeben, weist DAX ein 60-minütiges Wartungsfenster an einem zufällig ausgewählten Wochentag zu. Das 60-Minuten-Wartungsfenster wird zufällig aus einem 8-Stunden-Zeitraum für jede AWS-

Region ausgewählt. Die folgende Tabelle listet die Blöcke für jede Region auf, von denen die Standard-Wartungsfenster zugewiesen werden.

Regionscode	Name der Region	Wartungsfenster
ap-northeast-1	Region Asien-Pazifik (Tokio)	13:00 - 21:00 UHR UTC
ap-southeast-1	Region Asien-Pazifik (Singapur)	14:00 - 22:00 UHR UTC
ap-southeast-2	Region Asien-Pazifik (Sydney)	12:00 - 20:00 UHR UTC
ap-south-1	Region Asien-Pazifik (Mumbai)	17:30 - 1:30 UHR UTC
cn-northwest-1	Region China (Ningxia)	23:00 - 07:00 UHR UTC
cn-north-1	Region China (Peking)	14:00 - 22:00 UHR UTC
eu-central-1	Region Europa (Frankfurt)	23:00 - 07:00 UTC
eu-north-1	Region Europa (Stockholm)	01:00 - 09:00 UHR UTC
eu-south-2	Region Europa (Spanien)	21:00 - 05:00 UTC
eu-west-1	Region Europa (Irland)	22:00 - 06:00 UHR UTC
eu-west-2	Region Europa (London)	23:00 - 07:00 UHR UTC
eu-west-3	Region Europa (Paris)	23:00 - 07:00 UHR UTC
sa-east-1	Region Südamerika (São Paulo)	01:00 - 09:00 UHR UTC
us-east-1	Region USA Ost (Nord-Virginia)	03:00 - 11:00 UHR UTC
us-east-2	Region USA Ost (Ohio)	23:00 - 07:00 UHR UTC
us-west-1	Region USA West (Nordkalifornien)	06:00 - 14:00 UHR UTC

Regionscode	Name der Region	Wartungsfenster
us-west-2	Region USA West (Oregon)	06:00 bis 14:00 Uhr UTC

Erstellen eines DAX-Clusters

In diesem Abschnitt werden die Ersteinrichtung und Verwendung von Amazon DynamoDB Accelerator (DAX) in Ihrer Amazon-Virtual-Private-Cloud-(Amazon-VPC)-Standardumgebung beschrieben. Sie können Ihren ersten DAX-Cluster entweder mit AWS Command Line Interface (AWS CLI) oder dem erstellen. AWS-Managementkonsole

Nachdem Sie den DAX-Cluster erstellt haben, können Sie darauf von einer Amazon-EC2-Instance zugreifen, die in der gleichen VPC ausgeführt wird. Sie können dann Ihren DAX-Cluster mit einem Anwendungsprogramm verwenden. Weitere Informationen finden Sie unter Entwickeln mit dem DynamoDB-Accelerator-(DAX)-Client.

Themen

- Erstellen einer IAM-Servicerolle für DAX für den Zugriff auf DynamoDB
- DAX und IPv6
- Erstellen eines DAX-Clusters mit dem AWS CLI
- Erstellen eines DAX-Clusters mit dem AWS-Managementkonsole

Erstellen einer IAM-Servicerolle für DAX für den Zugriff auf DynamoDB

Damit der DAX-Cluster in Ihrem Namen auf DynamoDB-Tabellen zugreifen kann, müssen Sie eine Servicerolle erstellen. Eine Servicerolle ist eine AWS Identity and Access Management (IAM-) Rolle, die einen AWS Dienst autorisiert, in Ihrem Namen zu handeln. Die Servicerolle ermöglicht DAX den Zugriff auf Ihre DynamoDB-Tabellen, als ob Sie selbst auf diese Tabellen zugreifen würden. Sie müssen die Servicerolle erstellen, bevor Sie den DAX-Cluster erstellen können.

Wenn Sie die Konsole verwenden, überprüft der Workflow für die Cluster-Erstellung, ob bereits eine DAX-Servicerolle vorhanden ist. Wenn keine gefunden wird, erstellt die Konsole eine neue Servicerolle für Sie. Weitere Informationen finden Sie unter the section called “Schritt 2: Erstellen eines DAX-Clusters”.

Wenn Sie die verwenden AWS CLI, müssen Sie eine DAX-Servicerolle angeben, die Sie zuvor erstellt haben. Gegebenenfalls müssen Sie vorab eine neue Servicerolle erstellen. Weitere Informationen finden Sie unter Schritt 1: Erstellen Sie eine IAM-Servicerolle für DAX für den Zugriff auf DynamoDB mithilfe der AWS CLI.

Erforderliche Berechtigungen zum Erstellen einer Service-Rolle

Die von AWS verwaltete `AdministratorAccess`-Richtlinie enthält alle Berechtigungen, die erforderlich sind, um einen DAX-Cluster und eine Servicerolle zu erstellen. Wenn dem Benutzer die `AdministratorAccess`-Richtlinie angefügt wurde, ist keine weitere Aktion erforderlich.

Andernfalls müssen Sie der IAM-Richtlinie die folgenden Berechtigungen hinzufügen, damit der Benutzer die Servicerolle erstellen kann:

- `iam:CreateRole`
- `iam:CreatePolicy`
- `iam:AttachRolePolicy`
- `iam:PassRole`

Fügen Sie diese Berechtigungen dem Benutzer an, der die Aktion ausführen möchte.

> **Note**
>
> Die `iam:PassRole` Berechtigungen `iam:CreateRoleiam:CreatePolicy`,`iam:AttachRolePolicy`, und sind nicht in den AWS verwalteten Richtlinien für DynamoDB enthalten. Dies ist beabsichtigt, da diese Berechtigungen die Möglichkeit der Berechtigungseskalation bieten. Ein Benutzer könnte diese Berechtigungen nutzen, um eine neue Administratorrichtlinie zu erstellen und diese dann einer vorhandenen Rolle zuzuweisen. Deshalb müssen Sie (als Administrator Ihres DAX-Clusters) diese Berechtigungen explizit zu Ihrer Richtlinie hinzufügen.

Fehlerbehebung

Wenn in Ihrer Benutzerrichtlinie die Berechtigungen `iam:CreateRole`, `iam:CreatePolicy` und `iam:AttachPolicy` fehlen, erhalten Sie Fehlermeldungen. In der folgenden Tabelle sind diese Fehlermeldungen mit den zugehörigen Lösungen aufgeführt.

Fehlermeldung	Gehen Sie wie folgt vor:
`User: arn:aws:iam:: accountID :user/userName is not authorized to perform: iam:CreateRole on resource: arn:aws:iam:: accountID :role/service-role/ roleName`	Fügen Sie `iam:CreateRole` zu Ihrer Benutzerrichtlinie hinzu.
`User: arn:aws:iam:: accountID :user/userName is not authorized to perform: iam:CreatePolicy on resource: policy policyName`	Fügen Sie `iam:CreatePolicy` zu Ihrer Benutzerrichtlinie hinzu.
`User: arn:aws:iam:: accountID :user/userName is not authorized to perform: iam:AttachRolePolicy on resource: role daxServiceRole`	Fügen Sie `iam:AttachRolePolicy` zu Ihrer Benutzerrichtlinie hinzu.

Weitere Informationen zu den IAM-Richtlinien, die für die Verwaltung von DAX-Clustern erforderlich sind, finden Sie unter DAX-Zugriffskontrolle.

DAX und IPv6

DynamoDB DAX unterstützt jetzt IPv6 Adressierung, sodass Sie Cluster erstellen können, die im Nur-Only-, IPv4 -only- oder IPv6 Dual-Stack-Netzwerkmodus betrieben werden. Dies hilft bei der Verbesserung der Netzwerkkapazitäten, um den sich ändernden Infrastrukturanforderungen gerecht zu werden.

Netzwerktypen:

Sie können Cluster mit den folgenden Netzwerktypen erstellen:

- IPv4-nur
- IPv6-nur
- Dual-Stack (unterstützt sowohl als auch IPv4) IPv6

Die wichtigsten Funktionen:

Mit IPv6 Support können Sie Folgendes tun:

- Optionen für die Netzwerkkonfiguration:
  - IPv4-only und Dual-Stack-Cluster aktiviert. `dual_stack subnets`
  - IPv6-Nur-Cluster in Nur-Subnetzen. IPv6
- Verwaltung von Subnetzgruppen:
  - Erstellen Sie Subnetzgruppen mit Nur-Unterstützung, IPv4 -Nur-Unterstützung oder IPv6 Dual-Stack-Unterstützung
  - Ändern Sie bestehende Subnetzgruppen mit zusätzlichen VPC-Subnetzen
  - Fügen Sie IPv6 -only Subnetze zu -konfigurierten Subnetzgruppen hinzu IPv6
  - Fügen Sie zu konfigurierten Gruppen IPv4 Subnetze hinzu oder stapeln Sie sie in zwei Gruppen IPv4
- Client-Konfiguration:
  - Bei Aufrufen auf der Datenebene können Sie das bevorzugte IP-Protokoll für Dual_Stack-Cluster wie folgt festlegen:
    - `ip_discovery`Parameter im Python-SDK
    - `ipDiscovery`Parameter in anderen SDKs
  - Standard: IPv4 wenn keine Protokollpräferenz angegeben ist

Vor der Implementierung IPv6 in Ihren DAX-Clustern müssen Sie Folgendes berücksichtigen:

- Der Netzwerktyp kann nach der Clustererstellung nicht geändert werden
- Bei Dual-Stack-Clustern bestimmt der `ip_discovery/ipDiscovery` Parameter in der Client-Konfiguration, welches IP-Protokoll verwendet werden soll (IPv4 oder IPv6)
- Verschiedene Anwendungen können je nach Konfiguration über unterschiedliche IP-Protokolle eine Verbindung zu demselben Dual-Stack-Cluster herstellen

Example Beispiel für die Clientkonfiguration

```
DynamoDbAsyncClient client = ClusterDaxAsyncClient.builder()
        .overrideConfiguration(Configuration.builder()
            .url(endpoint)             // DAX cluster endpoint
            .ipDiscovery(ipDiscovery)       // IP discovery type (IPv4 or IPv6)
            .build())
```

```
        .build();
```

Important

Wenn Sie ressourcenbasierte IAM-Richtlinien verwenden, um IP-Adressen für DynamoDB-Tabellen in IPv6 reinen DAX-Umgebungen einzuschränken, müssen Sie eine Ausnahme für die IAM-Rolle Ihres DAX-Clusters erstellen, wenn Sie den Adressraum blockieren (). IPv4 `0.0.0.0/0` Fügen Sie Ihrer Richtlinie eine `ArnNotEquals` Bedingung hinzu, die speziell den Zugriff für die IAM-Rolle des DAX-Clusters ermöglicht und gleichzeitig IP-basierte Einschränkungen für andere Zugriffspfade beibehält. Ohne diese Ausnahme kann DAX nicht auf Ihre DynamoDB-Tabelle zugreifen.
Beispiel:

Example

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Deny",
      "Principal": "*",
      "Action": "dynamodb:PutItem",
      "Resource": "arn:aws:dynamodb:us-
east-1:123456789012:table/MusicCollection",
      "Condition": {
        "ArnNotEquals": {
          "aws:PrincipalArn": "arn:aws:iam::123456789012:role/
DAXServiceRoleForDynamoDBAccess"
        },
        "IpAddress": {
          "aws:SourceIp": "0.0.0.0/0"
        }
      }
    }
  ]
}
```

Erstellen eines DAX-Clusters mit dem AWS CLI

In diesem Abschnitt wird beschrieben, wie Sie einen Amazon-DynamoDB-Accelerator-(DAX)-Cluster mit der AWS Command Line Interface (AWS CLI) erstellen. Wenn Sie es noch nicht getan haben, müssen Sie AWS CLI installieren und konfigurieren. Eine Anleitung finden Sie unter den folgenden Themen im AWS Command Line Interface -Benutzerhandbuch:

- Installation des AWS CLI
- Konfigurieren von AWS CLI

> **Important**
>
> Um DAX-Cluster mit dem zu verwalten AWS CLI, installieren Sie Version 1.11.110 oder höher oder führen Sie ein Upgrade auf Version 1.11.110 durch.

In allen AWS CLI Beispielen werden die `us-west-2` Region und das fiktive Konto verwendet. IDs

Themen

- Schritt 1: Erstellen Sie eine IAM-Servicerolle für DAX für den Zugriff auf DynamoDB mithilfe der AWS CLI
- Schritt 2: Erstellung einer Subnetzgruppe
- Schritt 3: Erstellen Sie einen DAX-Cluster mit dem AWS CLI
- Schritt 4: Konfigurieren Sie Regeln für eingehende Sicherheitsgruppen mithilfe der AWS CLI

Schritt 1: Erstellen Sie eine IAM-Servicerolle für DAX für den Zugriff auf DynamoDB mithilfe der AWS CLI

Bevor Sie einen Amazon-DynamoDB-Accelerator-(DAX)-Cluster erstellen können, müssen Sie eine Servicerolle dafür erstellen. Eine Servicerolle ist eine AWS Identity and Access Management (IAM-) Rolle, die einen AWS Dienst autorisiert, in Ihrem Namen zu handeln. Die Servicerolle ermöglicht DAX den Zugriff auf Ihre DynamoDB-Tabellen, als würden Sie selbst darauf zugreifen.

In diesem Schritt erstellen Sie eine IAM-Richtlinie und fügen diese dann einer IAM-Rolle hinzu. So können Sie die Rolle einem DAX-Cluster zuweisen, sodass er DynamoDB-Operationen in Ihrem Namen durchführen kann.

So erstellen Sie eine IAM-Service-Rolle für DAX

1. Erstellen Sie eine Datei mit dem Namen `service-trust-relationship.json` und dem folgenden Inhalt.

   JSON

   ```
   {
       "Version":"2012-10-17",
       "Statement": [
          {
               "Effect": "Allow",
               "Principal": {
                   "Service": "dax.amazonaws.com"
               },
               "Action": "sts:AssumeRole"
           }
       ]
   }
   ```

2. Erstellen Sie die Servicerolle.

   ```
   aws iam create-role \
       --role-name DAXServiceRoleForDynamoDBAccess \
       --assume-role-policy-document file://service-trust-relationship.json
   ```

3. Erstellen Sie eine Datei mit dem Namen `service-role-policy.json` und dem folgenden Inhalt.

   JSON

   ```
   {
       "Version":"2012-10-17",
       "Statement": [
           {
               "Action": [
                   "dynamodb:DescribeTable",
                   "dynamodb:PutItem",
                   "dynamodb:GetItem",
                   "dynamodb:UpdateItem",
                   "dynamodb:DeleteItem",
   ```

```
                "dynamodb:Query",
                "dynamodb:Scan",
                "dynamodb:BatchGetItem",
                "dynamodb:BatchWriteItem",
                "dynamodb:ConditionCheckItem"
            ],
            "Effect": "Allow",
            "Resource": [
                "arn:aws:dynamodb:us-west-2:111122223333:*"
            ]
        }
    ]
}
```

Ersetzen Sie es *accountID* durch Ihre AWS Konto-ID. Um Ihre AWS Konto-ID zu finden, wählen Sie in der oberen rechten Ecke der Konsole Ihre Login-ID aus. Ihre AWS Konto-ID wird im Drop-down-Menü angezeigt.

Im Beispiel *accountID* muss der Amazon-Ressourcenname (ARN) eine 12-stellige Zahl sein. Verwenden Sie keine Bindestriche oder anderen Satzzeichen.

4. Erstellen Sie eine IAM-Richtlinie für die Servicerolle:

```
aws iam create-policy \
    --policy-name DAXServicePolicyForDynamoDBAccess \
    --policy-document file://service-role-policy.json
```

Beachten Sie in der Ausgabe den ARN für die Richtlinie, die Sie erstellt haben.

```
arn:aws:iam::123456789012:policy/DAXServicePolicyForDynamoDBAccess
```

5. Fügen Sie die Richtlinie der Servicerolle an. Ersetzen Sie *arn* im folgenden Code durch den tatsächlichen Rollen-ARN aus dem vorherigen Schritt.

```
aws iam attach-role-policy \
    --role-name DAXServiceRoleForDynamoDBAccess \
    --policy-arn arn
```

Als Nächstes geben Sie eine Subnetzgruppe für Ihre Standard-VPC an. Eine Subnetzgruppe ist eine Sammlung eines oder mehrerer Subnetze innerhalb der VPC. Siehe Schritt 2: Erstellung einer Subnetzgruppe.

Schritt 2: Erstellung einer Subnetzgruppe

Gehen Sie wie folgt vor, um mithilfe von () eine Subnetzgruppe für Ihren Amazon DynamoDB Accelerator (DAX) -Cluster zu erstellen. AWS Command Line Interface AWS CLI

> **Note**
>
> Wenn Sie bereits eine Subnetzgruppe für die Standard-VPC erstellt haben, können Sie diesen Schritt überspringen.

DAX wurde für die Ausführung in einer Amazon-Virtual-Private-Cloud-Umgebung (Amazon VPC) entwickelt. Wenn Sie Ihr AWS -Konto nach dem 4. Dezember 2013 erstellt haben, verfügen Sie bereits in jeder AWS -Region über eine Standard-VPC. Weitere Informationen finden Sie unter Standard-VPC und Standard-Subnetze im Amazon-VPC-Benutzerhandbuch.

So erstellen Sie eine Subnetzgruppe

1. Um die ID des Standard-VPCs zu bestimmen, geben Sie den folgenden Befehl ein.

   ```
   aws ec2 describe-vpcs
   ```

   Beachten Sie in der Ausgabe die ID der Standard-VPC, wie im folgenden Beispiel dargestellt.

   `vpc-12345678`

2. Ermitteln Sie das Subnetz, das Ihrer Standard-VPC IDs zugeordnet ist. *vpcID*Ersetzen Sie es durch Ihre tatsächliche VPC-ID, z. B. `vpc-12345678`

   ```
   aws ec2 describe-subnets \
       --filters "Name=vpc-id,Values=vpcID" \
       --query "Subnets[*].SubnetId"
   ```

   Notieren Sie sich in der Ausgabe die Subnetz-IDs – z. B. `subnet-11111111`.

3. Erstellen Sie die Subnetzgruppe. Stellen Sie sicher, dass Sie mindestens eine Subnetz-ID im Parameter `--subnet-ids` angeben.

```
aws dax create-subnet-group \
    --subnet-group-name my-subnet-group \
    --subnet-ids subnet-11111111 subnet-22222222 subnet-33333333 subnet-44444444
```

Informationen zum Erstellen des Clusters finden Sie unter Schritt 3: Erstellen Sie einen DAX-Cluster mit dem AWS CLI.

Schritt 3: Erstellen Sie einen DAX-Cluster mit dem AWS CLI

Gehen Sie wie folgt vor, um mit AWS Command Line Interface (AWS CLI) einen Amazon DynamoDB Accelerator (DAX) -Cluster in Ihrer Standard-Amazon-VPC zu erstellen.

So erstellen Sie einen DAX-Cluster

1. Rufen Sie den Amazon-Ressourcennamen (ARN) für die Servicerolle ab.

   ```
   aws iam get-role \
       --role-name DAXServiceRoleForDynamoDBAccess \
       --query "Role.Arn" --output text
   ```

   Beachten Sie in der Ausgabe den Servicerollen-ARN, wie im folgenden Beispiel dargestellt.

   ```
   arn:aws:iam::123456789012:role/DAXServiceRoleForDynamoDBAccess
   ```

2. DAX-Cluster erstellen. Ersetzen Sie *roleARN* mit der ARN aus dem vorherigen Schritt.

   ```
   aws dax create-cluster \
       --cluster-name mydaxcluster \
       --node-type dax.r4.large \
       --replication-factor 3 \
       --iam-role-arn roleARN \
       --subnet-group my-subnet-group \
       --sse-specification Enabled=true \
       --region us-west-2
   ```

   Alle Knoten in dem Cluster sind vom Typ `dax.r4.large` (`--node-type`). Es gibt drei Knoten (`--replication-factor`) – einen primären Knoten und zwei Replikate.

> **Note**
>
> Da `sudo` und `grep`-reservierte Schlüsselwörter sind, können Sie keinen DAX-Cluster mit diesen Wörtern im Cluster-Namen erstellen. Beispiel: `sudo` und `sudocluster` sind ungültige Cluster-Namen.

Zum Anzeigen des Cluster-Status geben Sie den folgenden Befehl ein:

```
aws dax describe-clusters
```

Der Status wird in der Ausgabe angezeigt - z. B. `"Status": "creating"`.

> **Note**
>
> Das Erstellen des Clusters kann einige Minuten dauern. Sobald der Cluster bereit ist, ändert sich sein Status zu `available`. Fahren Sie in der Zwischenzeit mit Schritt 4: Konfigurieren Sie Regeln für eingehende Sicherheitsgruppen mithilfe der AWS CLI fort und folgen Sie den Anweisungen.

Schritt 4: Konfigurieren Sie Regeln für eingehende Sicherheitsgruppen mithilfe der AWS CLI

Die Knoten in dem Amazon-DynamoDB-Accelerator-(DAX)-Cluster verwenden die Standardsicherheitsgruppe für Ihre Amazon VPC. Für die Standardsicherheitsgruppe müssen Sie eingehenden Datenverkehr auf TCP-Port 8111 für unverschlüsselte Cluster oder Port 9111 für verschlüsselte Cluster autorisieren. Auf diese Weise können Amazon-EC2-Instances in der Amazon VPC auf den DAX-Cluster zugreifen.

> **Note**
>
> Wenn Sie den DAX-Cluster mit einer anderen Sicherheitsgruppe (einer anderen als `default`) gestartet haben, müssen Sie stattdessen den hier beschriebenen Prozess für diese Gruppe durchführen.

So konfigurieren Sie Regeln für eingehenden Datenverkehr für Sicherheitsgruppen

1. Um die Standard-Sicherheitsgruppen-ID zu bestimmen, geben Sie den folgenden Befehl ein. Ersetzen Sie *vpcID* mit der tatsächlichen VPC-ID (aus Schritt 2: Erstellung einer Subnetzgruppe).

   ```
   aws ec2 describe-security-groups \
       --filters Name=vpc-id,Values=vpcID Name=group-name,Values=default \
       --query "SecurityGroups[*].{GroupName:GroupName,GroupId:GroupId}"
   ```

   Beachten Sie in der Ausgabe die Sicherheitsgruppen-ID - z. B. `sg-01234567`.

2. Geben Sie dann Folgendes ein. Ersetzen Sie *sgID* mit der tatsächlichen Sicherheitsgruppen-ID. Verwenden Sie Port 8111 für unverschlüsselte Cluster und 9111 für verschlüsselte Cluster.

   ```
   aws ec2 authorize-security-group-ingress \
       --group-id sgID --protocol tcp --port 8111
   ```

Erstellen eines DAX-Clusters mit dem AWS-Managementkonsole

In diesem Abschnitt wird beschrieben, wie Sie einen Amazon-DynamoDB-Accelerator-(DAX)-Cluster mit AWS-Managementkonsole.

Themen

- Schritt 1: Erstellen Sie eine Subnetzgruppe mit dem AWS-Managementkonsole
- Schritt 2: Erstellen Sie einen DAX-Cluster mit dem AWS-Managementkonsole
- Schritt 3: Konfigurieren Sie Regeln für eingehende Sicherheitsgruppen mithilfe der AWS-Managementkonsole

Schritt 1: Erstellen Sie eine Subnetzgruppe mit dem AWS-Managementkonsole

Gehen Sie folgendermaßen vor, um mithilfe AWS-Managementkonsole eine Subnetzgruppe für Ihren Amazon-DynamoDB-Accelerator-(DAX)-Cluster zu erstellen.

> **Note**
>
> Wenn Sie bereits eine Subnetzgruppe für die Standard-VPC erstellt haben, können Sie diesen Schritt überspringen.

DAX wurde für die Ausführung in einer Amazon-Virtual-Private-Cloud-Umgebung (Amazon VPC) entwickelt. Wenn Sie Ihr AWS -Konto nach dem 4. Dezember 2013 erstellt haben, verfügen Sie bereits in jeder AWS -Region über eine Standard-VPC. Weitere Informationen finden Sie unter Standard-VPC und Standard-Subnetze im Amazon-VPC-Benutzerhandbuch.

Im Rahmen des Erstellungsprozesses eines DAX-Clusters müssen Sie eine Subnetzgruppe angeben. Eine Subnetzgruppe ist eine Sammlung eines oder mehrerer Subnetze innerhalb der VPC. Beim Erstellen Ihres DAX-Clusters werden die Knoten für die Subnetze innerhalb der Subnetzgruppen bereitgestellt.

> **Note**
>
> Die VPC mit diesem DAX-Cluster kann andere Ressourcen und sogar VPC-Endpunkte für die anderen Dienste mit Ausnahme des VPC-Endpunkts für enthalten ElastiCache und kann zu Fehlern bei den DAX-Clustervorgängen führen.

So erstellen Sie eine Subnetzgruppe

1. Öffnen Sie die DynamoDB-Konsole unter. https://console.aws.amazon.com/dynamodb/
2. Klicken Sie im Navigationsbereich unter DAX auf Subnet groups (Subnetz-Gruppen).
3. Wählen Sie Create subnet group (Subnetz-Grupp erstellen) aus.
4. Führen Sie im Fenster Create subnet group Folgendes aus:
    a. Name – Geben Sie eine Kurzbezeichnung für die Subnetzgruppe ein.
    b. Beschreibung – Geben Sie eine Beschreibung für die Subnetzgruppe ein.
    c. VPC-ID – Wählen Sie den Bezeichner für Ihre Amazon-VPC-Umgebung aus.
    d. Subnets (Subnetze)—Wählen Sie ein oder mehrere Subnetze aus der Liste aus.

> **Note**
>
> Die Subnetze sind über mehrere Availability Zones verteilt. Wenn Sie planen, einen DAX-Cluster mit mehreren Knoten (einen Primärknoten und eine oder mehrere Read Replicas) zu erstellen, empfehlen wir Ihnen, mehrere Subnetze zu wählen. IDs DAX kann dann die Cluster-Knoten in mehreren Availability Zones bereitstellen. Wenn eine Availability Zone ausfällt, nimmt DAX automatisch einen Failover zu einer aktiven Availability Zone vor. Der DAX-Cluster funktioniert ohne Unterbrechung weiter.

Wenn Sie die gewünschten Einstellungen vorgenommen haben, klicken Sie auf Create subnet group (Subnetzgruppe erstellen).

Informationen zum Erstellen des Clusters finden Sie unter Schritt 2: Erstellen Sie einen DAX-Cluster mit dem AWS-Managementkonsole.

Schritt 2: Erstellen Sie einen DAX-Cluster mit dem AWS-Managementkonsole

Befolgen Sie dieses Verfahren zum Erstellen eines Amazon-DynamoDB-Accelerator-(DAX)-Clusters in der Standard-Amazon VPC.

So erstellen Sie einen DAX-Cluster

1. Öffnen Sie die DynamoDB-Konsole unter. https://console.aws.amazon.com/dynamodb/
2. Klicken Sie im Navigationsbereich unter DAX auf Cluster.
3. Wählen Sie Create cluster (Cluster erstellen).
4. Führen Sie im Fenster Create cluster Folgendes aus:

   a. Cluster-Name – Geben Sie unter einen Kurznamen für den DAX-Cluster ein.

> **Note**
>
> Da `sudo` und `grep` reservierte Schlüsselwörter sind, können Sie keinen DAX-Cluster mit diesen Wörtern im Cluster-Namen erstellen. Zum Beispiel sind `sudo` und `sudocluster` sind ungültige Cluster-Namen.

b. Cluster-Bescheibung – Geben Sie eine Beschreibung für den Cluster ein.

c. Node types (Knotentypen)—Wählen Sie den Knotentyp für alle Knoten im Cluster.

d. Cluster size (Cluster-Größe)—Wählen Sie die Anzahl der Knoten im Cluster. Ein Cluster besteht aus einem Primärknoten und bis zu neun Read Replicas.

> **Note**
>
> Wenn Sie einen Einzelknoten-Cluster erstellen möchten, wählen Sie 1 aus. Ihr Cluster besteht aus einem primären Knoten.
> Wenn Sie einen Cluster mit mehreren Knoten erstellen möchten, wählen Sie eine Zahl zwischen 3 (ein Primärknoten und zwei Read Replicas) und 10 (ein Primärknoten und neun Read Replicas) aus.
>
> > **Important**
> >
> > Für die Produktionsnutzung empfehlen wir dringend, DAX mit mindestens drei Knoten zu verwenden, wobei die Knoten in verschiedene Availability Zonen platziert werden. Drei Knoten sind aus Gründen der Fehlertoleranz für einen DAX-Cluster erforderlich.
> > Ein DAX-Cluster kann für Bereitstellungs- oder Test-Workloads mit nur einem oder zwei Knoten bereitgestellt werden. Cluster mit einem oder zwei Knoten sind jedoch nicht fehlertolerant. Für die Produktionsnutzung empfehlen wir daher die Nutzung von mindestens drei Knoten. Wenn bei Clustern mit einem oder zwei Knoten Software- oder Hardwarefehler auftreten, ist der Cluster möglicherweise nicht mehr verfügbar oder es gehen zwischengespeicherte Daten verloren.

e. Wählen Sie Next (Weiter).

f. Subnet group (Subnetz-Gruppe)—Auswählen Wählen Sie existierende und wählen Sie die Subnetzgruppe, die Sie in Schritt 1: Erstellen Sie eine Subnetzgruppe mit dem AWS-Managementkonsole erstellt haben aus.

g. Access control (Zugriffskontrolle)—Wählen Sie die Sicherheitsgruppe default (Standard).

h. Availability Zones (AZ)—Wählen Sie Automatic (Automatisch) aus.

i. Wählen Sie Next (Weiter) aus.

j. IAM service role for DynamoDB access (IAM-Servicerolle für DynamoDB-Zugriff)—Wählen Sie Create new (Neu erstellen) aus und geben Sie die folgenden Informationen ein:

- IAM-Rollenname – Geben Sie unter Rollenname einen Namen für Ihre IAM-Rolle ein, z. B `DAXServiceRole`. In der Konsole wird eine neue IAM-Rolle erstellt und der DAX-Cluster übernimmt diese Rolle zur Laufzeit.
- Wählen Sie das Feld neben Create policy (Richtlinie erstellen) aus.
- IAM role policy (IAM-Rollenrichtlinie)—Wählen Sie Read/Write (Lesen/Schreiben) aus. Der DAX-Cluster kann so Lese- und Schreibvorgänge in DynamoDB ausführen.
- New IAM policy name (Neuer Name der IAM-Richtlinie) – Dieses Feld wird aufgefüllt, wenn Sie den Namen der IAM-Rolle eingeben. Sie können auch einen Namen für eine IAM-Richtlinie eingeben, z. B. `DAXServicePolicy`. In der Konsole wird eine neue IAM-Richtlinie erstellt und die Richtlinie der IAM-Rolle zugeordnet.
- Access to DynamoDB tables (Zugriff auf DynamoDB-Tabellen) – Wählen Sie All tables (Alle Tabellen).

k. Encryption (Verschlüsselung) – Wählen Sie Turn on encryption at rest (Verschlüsselung im Ruhezustand aktivieren) und Turn on encryption in transit (Verschlüsselung während der Übertragung aktivieren) aus. Weitere Informationen finden Sie unter DAX-Verschlüsselung im Ruhezustand und DAX-Verschlüsselung während der Übertragung.

Eine separate Service-Rolle für DAX für den Zugriff auf Amazon EC2 ist ebenfalls erforderlich. DAX erstellt diese Servicerolle automatisch für Sie. Weitere Informationen finden Sie unter Verwenden von serviceverknüpften Rollen für DAX.

5. Wenn Sie die gewünschten Einstellungen vorgenommen haben, wählen Sie Next (Weiter).
6. Parameter group (Gruppe „Parameter“) – Wählen Sie Choose existing (Vorhandene wählen).
7. Maintenance window (Wartungsfenster) – Wählen Sie No Preference (Keine Präferenz), falls Sie für Software-Upgrades keine Präferenz haben, oder wählen Sie Specify time window (Zeitfenster angeben) und legen Sie die Optionen Weekday (Wochentag),Time (UTC) (Uhrzeit (UTC)) und Start within (hours) (In (Stunden) anfangen) fest, um Ihr Wartungsfenster zu planen.
8. Tags — Wählen Sie Neues Tag hinzufügen, um ein key/value Paar für Tagging-Zwecke einzugeben.
9. Wählen Sie Weiter aus.

Auf dem Bildschirm Überprüfen und erstellen können Sie alle Einstellungen überprüfen. Wenn Sie bereit sind, den Cluster zu erstellen, wählen Sie Cluster erstellen.

Auf dem Bildschirm Clusters wird Ihr DAX-Cluster mit dem Status Wird erstellt aufgeführt.

> **Note**
>
> Das Erstellen des Clusters kann einige Minuten dauern. Sobald der Cluster bereit ist, ändert sich sein Status in Available (Verfügbar).
> Fahren Sie in der Zwischenzeit mit Schritt 3: Konfigurieren Sie Regeln für eingehende Sicherheitsgruppen mithilfe der AWS-Managementkonsole fort und folgen Sie den Anweisungen.

Schritt 3: Konfigurieren Sie Regeln für eingehende Sicherheitsgruppen mithilfe der AWS-Managementkonsole

Ihr Amazon-DynamoDB-Accelerator-(DAX)-Cluster kommuniziert über TCP-Port 8111 (für unverschlüsselte Cluster) oder 9111 (für verschlüsselte Cluster), sodass Sie eingehenden Datenverkehr auf diesem Port autorisieren müssen. Auf diese Weise können Amazon-EC2-Instances in der Amazon VPC auf den DAX-Cluster zugreifen.

> **Note**
>
> Wenn Sie den DAX-Cluster mit einer anderen Sicherheitsgruppe (einer anderen als `default`) gestartet haben, müssen Sie stattdessen den hier beschriebenen Prozess für diese Gruppe durchführen.

So konfigurieren Sie Regeln für eingehenden Datenverkehr für Sicherheitsgruppen

1. Öffnen Sie die Amazon-EC2-Konsole unter https://console.aws.amazon.com/ec2/.
2. Wählen Sie im Navigationsbereich Security Groups (Sicherheitsgruppen) aus.
3. Wählen Sie die Sicherheitsgruppe default (Standard) aus. Wählen Sie im Menü Actions (Aktionen) die Option Edit inbound rules (Regeln für eingehenden Datenverkehr bearbeiten) aus.
4. Wählen Sie Add Rule (Regel hinzufügen) aus und geben Sie die folgenden Informationen ein:

- Portbereich – Geben Sie 8111 (wenn Ihr Cluster unverschlüsselt ist) oder 9111 (wenn Ihr Cluster verschlüsselt ist) ein.
- Quelle – Lassen Sie den Wert unverändert auf Custom (Benutzerdefiniert), und wählen Sie das Suchfeld auf der rechten Seite aus. Ein Dropdown-Menü wird angezeigt. Wählen Sie die ID für Ihre Standardsicherheitsgruppe aus.

5. Wählen Sie Save rules (Regeln speichern), um Ihre Änderungen zu speichern.
6. Um den Namen in der Konsole zu aktualisieren, rufen Sie die Eigenschaft Name auf und wählen Sie die Option Edit (Bearbeiten), die angezeigt wird.

DAX- und DynamoDB-Konsistenzmodelle

Amazon DynamoDB Accelerator (DAX) ist ein Write-Through-Cache-Service, der das Hinzufügen eines Caches zu DynamoDB-Tabellen vereinfachen soll. Da DAX getrennt von DynamoDB arbeitet, ist es wichtig, dass Sie die Konsistenzmodelle von DAX und DynamoDB verstehen, um sicherzustellen, dass sich Ihre Anwendungen wie erwartet verhalten.

In vielen Anwendungsfällen wirkt sich die Art und Weise, wie Ihre Anwendung verwendet, auf die Konsistenz der Daten im DAX-Cluster sowie auf die Konsistenz der Daten zwischen DAX und DynamoDB aus.

Themen

- Konsistenz zwischen DAX-Cluster-Knoten
- Verhalten des DAX-Element-Caches
- Verhalten des DAX-Abfrage-Caches
- Strongly Consistent- und Transactional-Lesevorgänge
- Negative Cache-Speicherung
- Strategien für Schreibvorgänge

Konsistenz zwischen DAX-Cluster-Knoten

Um eine hohe Verfügbarkeit für Ihre Anwendung zu erreichen, empfehlen wir, dass Sie mindestens drei Knoten für Ihren DAX-Cluster bereitstellen. Platzieren Sie diese Knoten dann in mehrere Availability Zones einer Region.

Wenn Ihr DAX-Cluster ausgeführt wird, repliziert er die Daten zwischen allen Knoten im Cluster (unter der Annahme, dass Sie mehr als einen Knoten bereitgestellt haben). Erwägen Sie eine Anwendung, die eine erfolgreiche `UpdateItem`-Operation mit DAX ausführt. Mit dieser Aktion wird der Element-Cache im primären Knoten mit dem neuen Wert geändert. Dieser Wert wird dann auf alle anderen Knoten im Cluster repliziert. Diese Replikation ist ein Eventually Consistent-Vorgang und dauert in der Regel weniger als eine Sekunde.

In diesem Szenario ist es möglich, dass zwei Clients denselben Schlüssel aus demselben DAX-Cluster lesen, jedoch abhängig von dem Knoten, auf den jeder Client zugegriffen hat, verschiedene Werte erhalten. Die Knoten sind alle konsistent, wenn die Aktualisierung vollständig auf alle Knoten im Cluster repliziert wurde. (Dieses Verhalten ähnelt dem Eventually-Consistent-Verhalten von DynamoDB.)

Wenn Sie eine Anwendung erstellen, die DAX verwendet, sollten Sie diese so konzipieren, dass sie Eventually-Consistent-Daten tolerieren kann.

Verhalten des DAX-Element-Caches

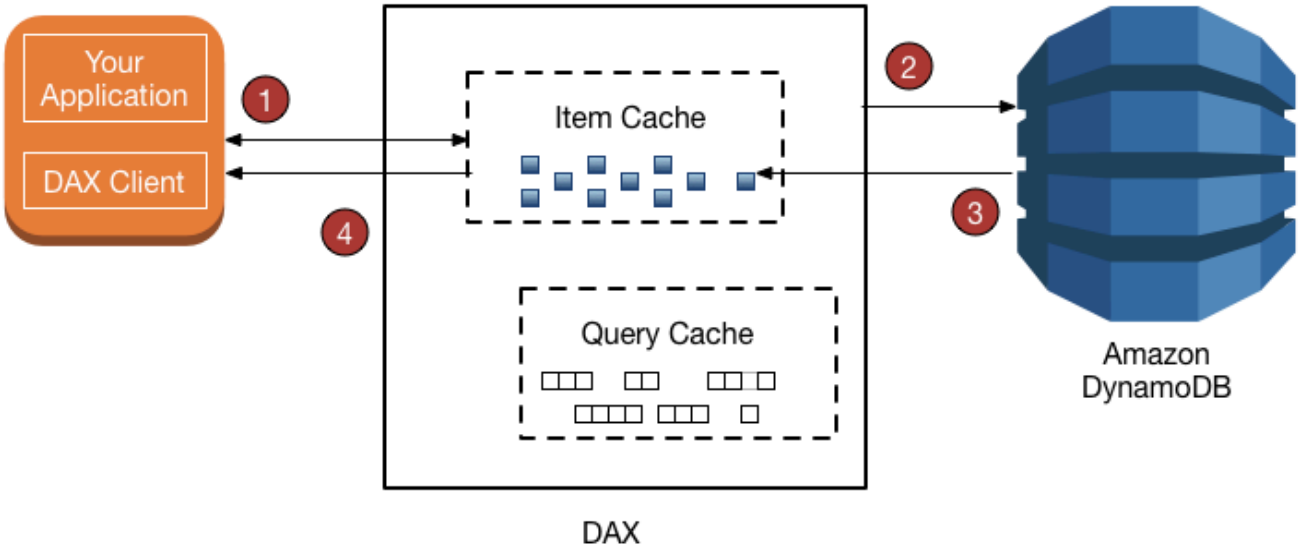
Jeder DAX-Cluster hat zwei getrennte Caches – einen Element- und einen Abfrage-Cache. Weitere Informationen finden Sie unter DAX: So funktioniert es.

In diesem Abschnitt werden die Konsistenzimplikationen für das Lesen aus dem und Schreiben in den DAX-Element-Cache beschrieben.

Konsistente Lesezugriffe

Mit DynamoDB führt die `GetItem`-Operation standardmäßig einen Eventually-Consistent-Lesevorgang aus. Angenommen, Sie verwenden `UpdateItem` mit dem DynamoDB-Client. Wenn Sie versuchen, dasselbe Element anschließend sofort zu lesen, kann es sein, dass die Daten so wie vor der Aktualisierung erscheinen. Der Grund hierfür ist die Verzögerung bei der Verbreitung auf alle DynamoDB-Speicherorte. Die Konsistenz wird in der Regel innerhalb von Sekunden erzielt. Wenn Sie den Lesevorgang erneut versuchen, sehen Sie wahrscheinlich das aktualisierte Element.

Wenn Sie `GetItem` mit dem DAX-Client verwenden, erfolgt die Operation (in diesem Fall ein Eventually-Consistent-Lesevorgang) wie folgt:



1. Der DAX-Client erstellt eine `GetItem`-Anforderung. DAX versucht, das angeforderte Element aus dem Element-Cache zu lesen. Wenn sich das Element im Cache befindet (Cache-Treffer) gibt DAX es an die Anwendung zurück.
2. Wenn das Element nicht verfügbar ist (Cache-Fehler), führt DAX eine `GetItem`-Operation für DynamoDB aus.
3. DynamoDB gibt das angeforderte Element zurück und DAX speichert es im Element-Cache.
4. DAX gibt das Element an die Anwendung zurück.
5. (nicht dargestellt) Wenn der DAX-Cluster mehr als einen Knoten enthält, wird das Element auf alle anderen Knoten im Cluster repliziert.

Das Element bleibt im Element-Cache von DAX, abhängig von der (TTL)-Einstellung und vom LRU-Algorithmus des Caches. Weitere Informationen finden Sie unter DAX: So funktioniert es.

Während dieses Zeitraums liest DAX das Element nicht erneut aus DynamoDB. Wenn das Element von einem anderen Benutzer mit einem DynamoDB-Client unter Umgehung von DAX aktualisiert wird, liefert eine `GetItem`-Anforderung, die den DAX-Client verwendet, andere Ergebnisse als eine `GetItem`-Anforderung, die den DynamoDB-Client verwendet. In diesem Szenario enthalten DAX und DynamoDB inkonsistente Werte für denselben Schlüssel, bis die TTL für das DAX-Element abgelaufen ist.

Wenn eine Anwendung Daten in einer zugrunde liegenden DynamoDB-Tabelle unter Umgehung von DAX ändert, muss die Anwendung etwaige Dateninkonsistenzen vorhersehen und tolerieren.

> **Note**
>
> Neben `GetItem` unterstützt der DAX-Client auch `BatchGetItem`-Anforderungen. `BatchGetItem` ist in erster Linie ein Wrapper für einzelne oder mehrere `GetItem`-Anforderungen, sodass DAX diese als individuelle `GetItem`-Operationen behandelt.

Konsistente Schreibzugriffe

DAX ist ein Write-Through-Cache-Service, der die Konsistenz des DAX-Element-Caches mit den zugrunde liegenden DynamoDB-Tabellen vereinfachen soll.

Der DAX-Client unterstützt dieselben API-Operationen wie DynamoDB (`PutItem`, `UpdateItem`, `DeleteItem`, `BatchWriteItem` und `TransactWriteItems`). Wenn Sie diese Operationen mit dem DAX-Client verwenden, werden die Elemente sowohl in DAX als auch DynamoDB geändert. DAX aktualisiert die Elemente in seinem Element-Cache, und zwar unabhängig vom TTL-Wert für diese Elemente.

Beispiel: Sie erstellen eine `GetItem`-Anforderung vom DAX-Client zum Lesen eines Elements in der `ProductCatalog` Tabelle. (Der Partitionsschlüssel ist `Id`. Ein Sortierschlüssel ist nicht vorhanden.) Sie rufen das Element mit der `Id` `101` ab. Der `QuantityOnHand`-Wert für dieses Element ist `42`. DAX speichert das Element im Element-Cache mit einem bestimmten TTL-Format. In diesem Beispiel ist der TTL-Wert zehn Minuten. Drei Minuten später verwendet eine andere Anwendung den DAX-Client, um dasselbe Element zu aktualisieren, sodass der `QuantityOnHand`-Wert jetzt `41` lautet. Vorausgesetzt, das Element wird nicht erneut aktualisiert, geben alle nachfolgenden Lesevorgänge für dasselbe Element während der nächsten zehn Minuten den im Cache gespeicherten Wert für `QuantityOnHand` (`41`) zurück.

Wie DAX Schreibzugriffe verarbeitet

DAX ist für Anwendungen konzipiert, die Lesevorgänge mit hoher Leistung erfordern. Als Write-Through-Cache übergibt DAX Ihre Schreibvorgänge synchron an DynamoDB und repliziert dann automatisch und asynchron resultierende Aktualisierungen in den Elementcache über alle Knoten im Cluster. Sie brauchen keine Cache-Invalidierungslogik verwalten, da DAX diesen Vorgang für Sie übernimmt.

DAX unterstützt die folgenden Schreibvorgänge: `PutItem`, `UpdateItem`, `DeleteItem`, `BatchWriteItem` und `TransactWriteItems`.

Wenn Sie eine `PutItem`-, `UpdateItem`-, `DeleteItem`- oder `BatchWriteItem`-Anforderung an DAX senden, werden die folgenden Vorgänge ausgeführt:

- DAX sendet die Anforderung an DynamoDB.
- DynamoDB antwortet DAX und bestätigt den erfolgreichen Schreibvorgang.
- DAX schreibt das Element in den Element-Cache.
- DAX gibt eine Erfolgsmeldung an den Anforderer zurück.

Wenn Sie eine `TransactWriteItems`-Anforderung an DAX senden, werden die folgenden Vorgänge ausgeführt:

- DAX sendet die Anforderung an DynamoDB.
- DynamoDB antwortet DAX und bestätigt, dass die Transaktion abgeschlossen ist.
- DAX gibt eine Erfolgsmeldung an den Anforderer zurück.
- DAX stellt im Hintergrund eine `TransactGetItems`-Anforderung für jedes Element in der `TransactWriteItems`-Anforderung, um das Element im Element-Cache zu speichern. `TransactGetItems` wird verwendet, um die serialisierbare Isolierung sicherzustellen.

Wenn bei einem Schreibvorgang für DynamoDB ein Fehler auftritt, einschließlich Drosselung, wird das Element in DAX nicht im Cache gespeichert. Die Ausnahme für den Fehler wird an den Anforderer zurückgegeben. So wird sichergestellt, dass keine Daten in den DAX-Cache geschrieben werden, es sei denn, sie wurden vorher erfolgreich in DynamoDB geschrieben.

> **Note**
>
> Mit jedem Schreibvorgang in DAX wird der Status des Element-Caches geändert. Schreibvorgänge in den Element-Cache wirken sich nicht auf den Abfrage-Cache aus. (Der DAX-Element-Cache und Abfrage-Cache werden für unterschiedliche Zwecke verwendet und arbeiten unabhängig voneinander.)

Verhalten des DAX-Abfrage-Caches

DAX speichert die Ergebnisse von `Query` und `Scan`-Anforderungen im Abfrage-Cache. Diese Ergebnisse wirken sich jedoch nicht auf den Element-Cache aus. Wenn Ihre Anwendung eine `Query`- oder `Scan`- Anforderung mit DAX erstellt, wird der Ergebnissatz im Abfrage-Cache und nicht im Element-Cache gespeichert. Sie können den Element-Cache nicht "warmlaufen" lassen, indem Sie eine `Scan`-Operation ausführen, da der Element-Cache und der Abfrage-Cache zwei verschiedene Entitäten sind.

Kohärenz von query-update-query

Durch Aktualisierungen des Element-Caches oder der zugrunde liegenden DynamoDB-Tabelle werden die im Abfrage-Cache gespeicherten Ergebnisse nicht ungültig oder geändert.

Berücksichtigen Sie zur Veranschaulichung das folgende Szenario. Eine Anwendung arbeitet mit der `DocumentRevisions`-Tabelle, deren Partitionsschlüssel `DocId` und Sortierschlüssel `RevisionNumber` lautet.

1. Ein Client erstellt eine `Query` für `DocId101`, für alle Elemente mit `RevisionNumber` größer als oder gleich `5`. DAX speichert den Ergebnissatz im Abfrage-Cache und gibt ihn an den Benutzer zurück.
2. Der Client gibt eine `PutItem`-Anforderung für die `DocId` `101` mit dem `RevisionNumber`-Wert `20` aus.
3. Der Client erstellt dieselbe `Query` wie in Schritt 1 beschrieben (`DocId` `101` und `RevisionNumber` **>=** `5`).

In diesem Szenario ist der zwischengespeicherte Ergebnissatz für die in Schritt 3 erstellte `Query` mit dem Ergebnissatz identisch, der in Schritt 1 zwischengespeichert wurde. Der Grund ist, dass DAX `Query` - oder `Scan`-Ergebnissätze basierend auf Aktualisierungen einzelner Elemente nicht ungültig macht. Die `PutItem`-Operation aus Schritt 2 wirkt sich nur auf den DAX-Abfrage-Cache aus, wenn die TTL für die `Query` abläuft.

Ihre Anwendung sollte den TTL-Wert für den Abfrage-Cache und die Dauer berücksichtigen, wie lange Ihre Anwendung inkonsistente Ergebnisse zwischen dem Abfrage- und dem Element-Cache tolerieren kann.

Strongly Consistent- und Transactional-Lesevorgänge

Zum Ausführen einer Strongly-Consistent- `GetItem`, `BatchGetItem`, `Query`, oder `Scan`Anforderung, legen Sie den `ConsistentRead` Parameter auf „true“ fest. DAX übergibt Strongly-Consistent-Leseanforderungen an DynamoDB. Wenn eine Antwort von DynamoDB eingeht, gibt DAX die Ergebnisse an den Client zurück, speichert sie jedoch nicht im Cache. DAX kann selbst keine Strongly-Consistent-Lesevorgänge ausführen, da keine enge Kopplung an DynamoDB besteht. Aus diesem Grund müssen alle nachfolgenden Lesevorgänge aus DAX vom Typ Eventually Consistent sein. Nachfolgende Strongly-Consistent-Lesevorgänge müssen an DynamoDB übergeben werden

DAX verarbeitet `TransactGetItems`-Anforderungen genauso wie Strongly Consistent-Lesevorgänge. DAX übergibt alle `TransactGetItems`-Anforderungen an DynamoDB. Wenn eine Antwort von DynamoDB eingeht, gibt DAX die Ergebnisse an den Client zurück, speichert sie jedoch nicht im Cache.

Negative Cache-Speicherung

DAX unterstützt negative Cache-Einträge, und zwar sowohl im Element- als auch im Abfrage-Cache. Ein negativer Cache-Eintrag tritt auf, wenn DAX in einer zugrunde liegenden DynamoDB-Tabelle keine angeforderten Elemente finden kann. Statt einen Fehler zu melden, legt DAX ein leeres Ergebnis im Cache ab und gibt dieses Ergebnis an den Benutzer zurück.

Angenommen, eine Anwendung sendet eine `GetItem`-Anforderung an einen DAX-Cluster und es befindet sich kein übereinstimmendes Element im DAX-Element-Cache. Dies führt dazu, dass DAX das entsprechende Element aus der zugrunde liegenden DynamoDB-Tabelle liest. Wenn das Element in DynamoDB nicht existiert, speichert DAX ein leeres Element im Element-Cache und gibt dieses leere Element an die Anwendung zurück. Angenommen, die Anwendung sendet eine andere `GetItem`-Anforderung für das gleiche Element. DAX findet das leere Element im Element-Cache und gibt es sofort an die Anwendung zurück. Auf DynamoDB wird dabei nicht zurückgegriffen.

Ein negativer Cache-Eintrag bleibt im DAX-Element-Cache, bis die Element-TTL abgelaufen ist, LRU aufgerufen oder das Element mit `PutItem`, `UpdateItem`, oder `DeleteItem` geändert wird.

Der DAX-Abfrage-Cache verarbeitet negative Cache-Ergebnisse ähnlich. Wenn eine Anwendung eine `Query`- oder `Scan`-Operation ausführt und der DAX-Abfrage-Cache kein Ergebnis enthält, sendet DAX die Anforderung an DynamoDB. Wenn keine übereinstimmenden Elemente im Ergebnissatz vorhanden sind, speichert DAX einen leeren Ergebnissatz im Abfrage-Cache und gibt den leeren Ergebnissatz an die Anwendung zurück. Bei nachfolgenden `Query`- oder `Scan`-

Anforderungen ergibt sich derselbe (leere) Ergebnissatz, bis die TTL für diesen Ergebnissatz abgelaufen ist.

Strategien für Schreibvorgänge

Das Write-Through-Verhalten von DAX eignet sich für viele Anwendungsmuster. Für einige Anwendungsmuster ist ein Write-Through-Modell allerdings nicht geeignet.

Bei Anwendungen, die latenzempfindlich sind, führt die Write-Through-Methode für DAX zu einem zusätzlichen Netzwerk-Hop. Ein Schreibvorgang in DAX ist daher etwas langsamer als ein Schreibvorgang, der direkt in DynamoDB ausgeführt wird. Wenn Ihre Anwendung schreiblatenzempfindlich ist, können Sie die Latenz reduzieren, indem Sie stattdessen direkt in DynamoDB schreiben. Weitere Informationen finden Sie unter Write-Around.

Für schreibintensive Anwendungen (z. B. Anwendungen, die ein Massenladen von Daten ausführen) empfiehlt es sich nicht, alle Daten über DAX zu schreiben, da nur ein kleiner Anteil der Daten von der Anwendung gelesen wird. Wenn Sie große Datenmengen über DAX schreiben, muss der entsprechende LRU-Algorithmus aufgerufen werden, um Platz im Cache für die neuen, zu lesenden Elemente zu schaffen. Hierdurch wird die Effektivität von DAX als Lese-Cache beeinträchtigt.

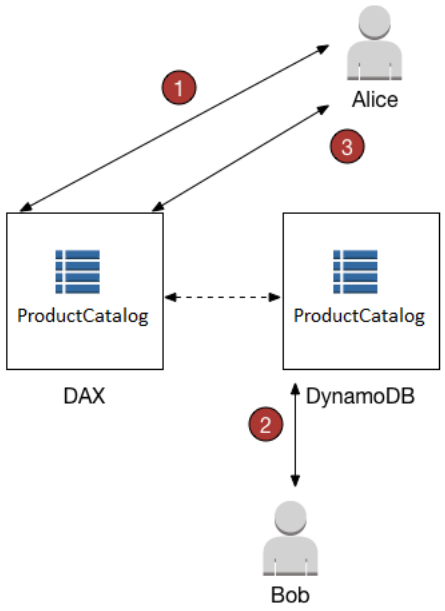
Wenn Sie ein Element in DAX schreiben, wird der Status des Element-Caches für das neue Element geändert. (DAX muss z. B. ältere Daten aus dem Element-Cache entfernen, um Platz für das neue Element bereitzustellen.) Das neue Element bleibt im Element-Cache, abhängig vom LRU-Algorithmus des Caches und von der TTL-Einstellung für den Cache. Solange das Element im Element-Cache gespeichert bleibt, liest DAX das Element nicht erneut aus DynamoDB.

Write-Through

Der DAX-Element-Cache implementiert eine Write-Through-Richtlinie. Weitere Informationen finden Sie unter Wie DAX Schreibzugriffe verarbeitet.

Wenn Sie ein Element schreiben, stellt DAX sicher, dass das zwischengespeicherte Element mit dem Element so synchronisiert wird, wie es in DynamoDB vorhanden ist. Dies ist hilfreich für Anwendungen, die ein Element erneut lesen müssen, unmittelbar nachdem sie es geschrieben haben. Wenn andere Anwendungen jedoch direkt in eine DynamoDB-Tabelle schreiben, ist das Element im DAX-Element-Cache nicht mehr mit DynamoDB synchronisiert.

Nehmen wir folgendes Beispiel zur Veranschaulichung. Angenommen, zwei Benutzer (Alice und Bob) arbeiten mit der Tabelle `ProductCatalog`. Alice greift mit DAX auf die Tabelle zu, während Bob DAX umgeht und direkt in DynamoDB auf die Tabelle zugreift.



1. Alice aktualisiert ein Element in der `ProductCatalog`-Tabelle. DAX leitet die Anforderung an DynamoDB weiter und die Aktualisierung wird erfolgreich ausgeführt. Anschließend schreibt DAX das Element in den Element-Cache und gibt die Antwort, dass der Vorgang erfolgreich ausgeführt wurde, an Alice zurück. Von diesem Zeitpunkt an sieht jeder Benutzer, der das Element aus DAX liest, das Element mit der Aktualisierung von Alice, bis das Element letztendlich aus dem Cache entfernt wird.
2. Kurze Zeit später aktualisiert Bob dasselbe `ProductCatalog`-Element, das Alice geschrieben hat. Bob aktualisiert das Element jedoch direkt in DynamoDB. DAX aktualisiert den Element-Cache nicht automatisch als Reaktion auf Aktualisierungen über DynamoDB. Daher sehen DAX-Benutzer Bobs Aktualisierung nicht.
3. Alice liest das Element erneut aus DAX. Das Element befindet sich im Element-Cache, sodass DAX es an Alice zurückgibt, ohne auf die DynamoDB-Tabelle zuzugreifen.

In diesem Szenario sehen Alice und Bob unterschiedliche Darstellungen desselben `ProductCatalog`-Elements. Dies ist der Fall, bis DAX das Element letztendlich aus dem Element-Cache entfernt oder bis ein anderer Benutzer dasselbe Element mit DAX erneut aktualisiert.

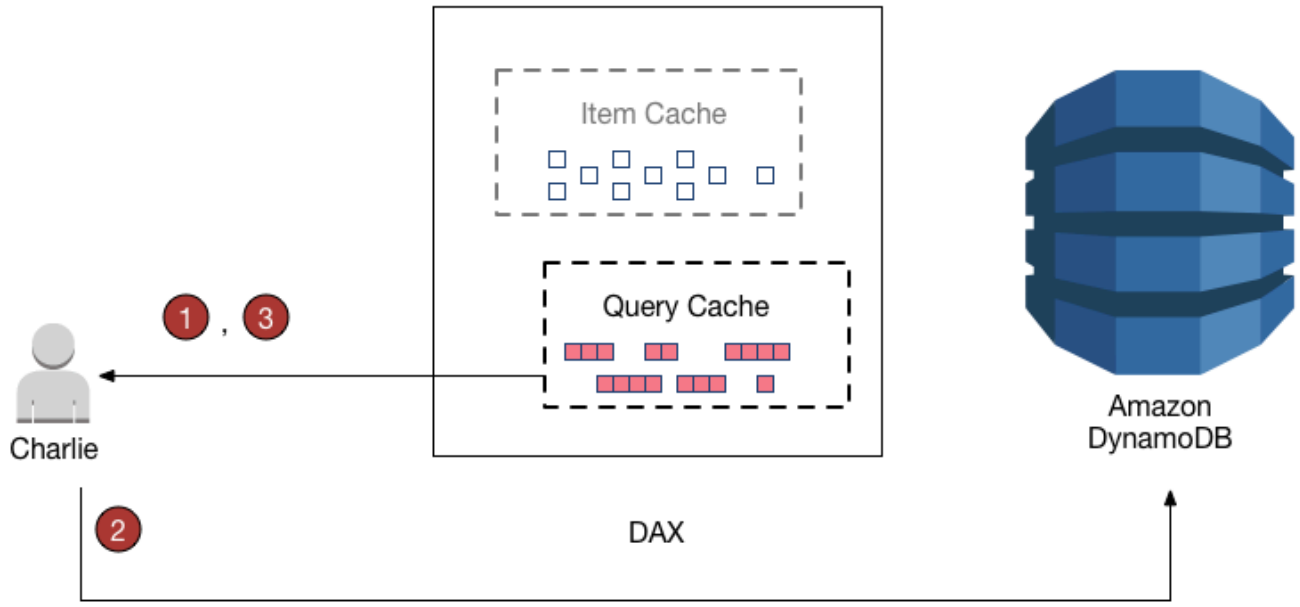
Write-Around

Wenn die Anwendung große Datenmengen (z. B. ein Massenladen von Daten) schreiben muss, kann es sinnvoll sein, DAX zu umgehen und die Daten direkt in DynamoDB zu schreiben. Diese Write-

Around-Strategie reduziert die Schreiblatenz. Der Element-Cache bleibt allerdings nicht mit den Daten in DynamoDB synchronisiert.

Wenn Sie eine Write-Around-Strategie verwenden möchten, denken Sie daran, dass DAX den Element-Cache füllt, sobald Anwendungen den DAX-Client zum Lesen von Daten verwenden. Dies kann in einigen Fällen von Vorteil sind, da sichergestellt wird, dass nur die am häufigsten gelesenen Daten zwischengespeichert werden (im Gegensatz zu den am häufigsten geschriebenen Daten).

Beispiel: Ein Benutzer (Charlie) möchte unter Verwendung von mit einer anderen Tabelle arbeiten, und zwar mit der `GameScores`Tabelle, die DAX nutzt. Der Partitionsschlüssel für `GameScores` ist `UserId`, sodass alle Punktzahlen von Charlie dieselbe `UserId` hätten.



1. Charlie möchte seine gesamten Ergebnisse abrufen und sendet eine `Query` an DAX. Unter der Annahme, dass diese Abfrage noch nicht erstellt wurde, leitet DAX die Abfrage zur Verarbeitung an DynamoDB weiter. Die Ergebnisse werden im DAX-Abfrage-Cache gespeichert und an Charlie zurückgegeben. Der Ergebnissatz bleibt im Abfrage-Cache verfügbar, bis er entfernt wird.
2. Angenommen, Charlie spielt das Spiel Meteor Blaster und erzielt eine hohe Punktzahl. Charlie sendet eine `UpdateItem`-Anforderung an DynamoDB und ändert ein Element in der `GameScores` Tabelle.
3. Schließlich beschließt Charlie, seine frühere `Query` erneut auszuführen, um alle seine Daten aus `GameScores` abzurufen. In den Ergebnissen ist die hohe Punktzahl für Meteor Blaster nicht

enthalten. Der Grund hierfür ist, dass die Abfrageergebnisse aus dem Abfrage-Cache und nicht aus dem Element-Cache stammen. Die beiden Caches sind unabhängig voneinander, sodass sich eine Änderung des einen Caches nicht auf den anderen auswirkt.

DAX aktualisiert keine Ergebnissätze im Abfrage-Cache mit den neuesten Daten aus DynamoDB. Jeder Ergebnissatz im Abfrage-Cache entspricht dem Zeitpunkt, an dem die `Query`- oder `Scan`-Operation durchgeführt wurde. Daher werden Charlies `Query`-Ergebnisse nicht in seiner `PutItem`-Operation berücksichtigt. Dies ist so lange der Fall, bis DAX den Ergebnissatz aus dem Abfrage-Cache entfernt.

Entwickeln mit dem DynamoDB-Accelerator-(DAX)-Client

Um DAX von einer Anwendung aus zu nutzen, verwenden Sie den DAX-Client für Ihre Programmiersprache. Der DAX-Client ist so konzipiert, dass Ihre vorhandenen Amazon-DynamoDB-Anwendungen nur minimal unterbrochen werden. Sie müssen nur einige einfache Änderungen am Code vornehmen.

> **Note**
>
> DAX-Clients für verschiedene Programmiersprachen stehen auf der folgenden Website zur Verfügung:
>
> - http://dax-sdk.s3 - website-us-west -2.amazonaws.com

In diesem Abschnitt wird gezeigt, wie Sie eine Amazon-EC2-Instance in der Standard-Amazon-VPC starten, eine Verbindung mit der Instance herstellen und eine Beispielanwendung ausführen. Außerdem enthalten sind Informationen dazu, wie Sie Ihre bestehende Anwendung so ändern, dass sie Ihren DAX-Cluster verwenden kann.

Themen

- Tutorial: Ausführen einer Beispielanwendung mit DynamoDB Accelerator (DAX)
- Ändern einer vorhandenen Anwendung für die Verwendung von DAX

Tutorial: Ausführen einer Beispielanwendung mit DynamoDB Accelerator (DAX)

In dieser Anleitung wird dargestellt, wie Sie eine Amazon-EC2-Instance in der standardmäßigen Virtual Private Cloud (VPC) starten, eine Verbindung mit der Instance herstellen und eine Beispielanwendung ausführen, die Amazon DynamoDB Accelerator (DAX) nutzt.

> **Note**
>
> Um diese Anleitung abzuschließen, müssen Sie über einen DAX-Cluster verfügen, der in Ihrer Standard-VPC ausgeführt wird. Wenn Sie noch keinen DAX-Cluster eingerichtet haben, finden Sie unter Erstellen eines DAX-Clusters eine Anleitung.

Themen

- Schritt 1: Starten einer Amazon-EC2-Instance
- Schritt 2: Erstellen eines Benutzers und einer Richtlinie
- Schritt 3: Konfigurieren einer Amazon-EC2-Instance
- Schritt 4: Ausführen einer Beispielanwendung

Schritt 1: Starten einer Amazon-EC2-Instance

Wenn der Amazon-DynamoDB-Accelerator-(DAX)-Cluster verfügbar ist, können Sie eine Amazon-EC2-Instance in Ihrer standardmäßigen Amazon Virtual Private Cloud (Amazon VPC) starten. Damit können Sie DAX-Client-Software auf dieser Instance installieren und ausführen.

Starten Sie EC2-Instances wie folgt:

1. Melden Sie sich bei der an AWS-Managementkonsole und öffnen Sie die Amazon EC2 EC2-Konsole unter https://console.aws.amazon.com/ec2/.
2. Wählen Sie Launch Instance (Instance starten) aus und gehen Sie folgendermaßen vor:

   Schritt 1: Auswählen eines Amazon Machine Images (AMI)

   1. Suchen Sie in der Liste von AMIs nach dem Amazon Linux AMI und wählen Sie Select aus.

   Schritt 2: Auswählen eines Instance-Typs

1. Wählen Sie in der Liste der Instance-Typen t2.micro aus.
2. Wählen Sie Next: Configure Instance Details aus.

Schritt 3: Konfigurieren der Instance-Details

1. Wählen Sie bei Network (Netzwerk) Ihre Standard-VPC aus.
2. Wählen Sie Next: Add Storage aus.

Schritt 4: Hinzufügen von Speicher

1. Überspringen Sie diesen Schritt, indem Sie Next: Add tags (Weiter: Tags hinzufügen) auswählen.

Schritt 5: Hinzufügen von Tags

1. Überspringen Sie diesen Schritt, indem Sie Next: Configure Security Group auswählen.

Schritt 6: Konfigurieren einer Sicherheitsgruppe

1. Wählen Sie Bestehende Sicherheitsgruppe auswählen aus.
2. Wählen Sie in der Liste der Sicherheitsgruppen default aus. Dies ist die Standard-Sicherheitsgruppe für Ihre VPC.
3. Wählen Sie Next: Review and Launch aus.

Schritt 7: Prüfen eines Starts einer Instance

1. Wählen Sie Launch (Starten) aus.

3. Führen Sie im Fenster Select an existing key pair or create a new key pair einen der folgenden Schritte aus:

   - Wenn Sie nicht über ein Amazon-EC2-Schlüsselpaar verfügen, wählen Sie Neues Schlüsselpaar erstellen aus und befolgen Sie die Anweisungen. Sie werden dazu aufgefordert, eine private Schlüsseldatei (`.pem`-Datei) herunterzuladen. Sie benötigen diese Datei später, wenn Sie sich bei Ihrer Amazon-EC2-Instance anmelden.

- Wenn Sie bereits über ein Amazon-EC2-Schlüsselpaar verfügen, wechseln Sie zu Schlüsselpaar auswählen und wählen Sie Ihr Schlüsselpaar aus der Liste aus. Beachten Sie, dass Sie bereits über die private Schlüsseldatei (`.pem`-Datei) verfügen müssen, um sich bei Ihrer Amazon-EC2-Instance anmelden zu können.

4. Nachdem Sie Ihr Schlüsselpaar konfiguriert haben, wählen Sie Launch Instances (Instances starten) aus.
5. Wählen Sie im Navigationsbereich der Konsole zuerst EC2 Dashboard und anschließend die gestartete Instance aus. Suchen Sie im unteren Bereich der Registerkarte Description (Beschreibung) das Public DNS (Öffentliches DNS) für Ihre Instance, beispielsweise: `ec2-11-22-33-44.us-west-2.compute.amazonaws.com`. Notieren Sie sich diesen öffentlichen DNS-Namen, da Sie ihn für Schritt 3: Konfigurieren einer Amazon-EC2-Instance benötigen.

> **Note**
>
> Es dauert einige Minuten, bis Ihre Amazon-EC2-Instance verfügbar ist. Fahren Sie in der Zwischenzeit mit Schritt 2: Erstellen eines Benutzers und einer Richtlinie fort und folgen Sie den Anweisungen.

Schritt 2: Erstellen eines Benutzers und einer Richtlinie

In diesem Schritt erstellen Sie einen Benutzer mit einer Richtlinie, die Zugriff auf Ihren Amazon DynamoDB Accelerator (DAX) -Cluster und auf DynamoDB unter Verwendung gewährt. AWS Identity and Access Management Sie können dann Anwendungen verwenden, die mit dem DAX-Cluster interagieren.

Melden Sie sich an für ein AWS-Konto

Wenn Sie noch keine haben AWS-Konto, führen Sie die folgenden Schritte aus, um eine zu erstellen.

Um sich für eine anzumelden AWS-Konto

1. Öffnen Sie https://portal.aws.amazon.com/billing/die Anmeldung.
2. Folgen Sie den Online-Anweisungen.

   Während der Anmeldung erhalten Sie einen Telefonanruf oder eine Textnachricht und müssen einen Verifizierungscode über die Telefontasten eingeben.

Wenn Sie sich für eine anmelden AWS-Konto, Root-Benutzer des AWS-Kontoswird eine erstellt. Der Root-Benutzer hat Zugriff auf alle AWS-Services und Ressourcen des Kontos. Als bewährte Sicherheitsmethode weisen Sie einem Benutzer Administratorzugriff zu und verwenden Sie nur den Root-Benutzer, um Aufgaben auszuführen, die Root-Benutzerzugriff erfordern.

AWS sendet Ihnen nach Abschluss des Anmeldevorgangs eine Bestätigungs-E-Mail. Du kannst jederzeit deine aktuellen Kontoaktivitäten einsehen und dein Konto verwalten, indem du zu https://aws.amazon.com/gehst und Mein Konto auswählst.

Erstellen eines Benutzers mit Administratorzugriff

Nachdem Sie sich für einen angemeldet haben AWS-Konto, sichern Sie Ihren Root-Benutzer des AWS-Kontos AWS IAM Identity Center, aktivieren und erstellen Sie einen Administratorbenutzer, sodass Sie den Root-Benutzer nicht für alltägliche Aufgaben verwenden.

Sichern Sie Ihre Root-Benutzer des AWS-Kontos

1. Melden Sie sich AWS-Managementkonsoleals Kontoinhaber an, indem Sie Root-Benutzer auswählen und Ihre AWS-Konto E-Mail-Adresse eingeben. Geben Sie auf der nächsten Seite Ihr Passwort ein.

   Hilfe bei der Anmeldung mit dem Root-Benutzer finden Sie unter Anmelden als Root-Benutzer im AWS-Anmeldung -Benutzerhandbuch zu.
2. Aktivieren Sie die Multi-Faktor-Authentifizierung (MFA) für den Root-Benutzer.

   Anweisungen finden Sie unter Aktivieren eines virtuellen MFA-Geräts für Ihren AWS-Konto Root-Benutzer (Konsole) im IAM-Benutzerhandbuch.

Erstellen eines Benutzers mit Administratorzugriff

1. Aktivieren Sie das IAM Identity Center.

   Anweisungen finden Sie unter Aktivieren AWS IAM Identity Center im AWS IAM Identity Center -Benutzerhandbuch.
2. Gewähren Sie einem Administratorbenutzer im IAM Identity Center Benutzerzugriff.

Ein Tutorial zur Verwendung von IAM-Identity-Center-Verzeichnis als Identitätsquelle finden Sie IAM-Identity-Center-Verzeichnis im Benutzerhandbuch unter Benutzerzugriff mit der Standardeinstellung konfigurieren.AWS IAM Identity Center

Anmelden als Administratorbenutzer

- Um sich mit Ihrem IAM-Identity-Center-Benutzer anzumelden, verwenden Sie die Anmelde-URL, die an Ihre E-Mail-Adresse gesendet wurde, als Sie den IAM-Identity-Center-Benutzer erstellt haben.

  Hilfe bei der Anmeldung mit einem IAM Identity Center-Benutzer finden Sie im AWS-Anmeldung Benutzerhandbuch unter Anmeldung beim AWS Access-Portal.

Weiteren Benutzern Zugriff zuweisen

1. Erstellen Sie im IAM-Identity-Center einen Berechtigungssatz, der den bewährten Vorgehensweisen für die Anwendung von geringsten Berechtigungen folgt.

   Anweisungen hierzu finden Sie unter Berechtigungssatz erstellen im AWS IAM Identity Center - Benutzerhandbuch.
2. Weisen Sie Benutzer einer Gruppe zu und weisen Sie der Gruppe dann Single Sign-On-Zugriff zu.

   Eine genaue Anleitung finden Sie unter Gruppen hinzufügen im AWS IAM Identity Center - Benutzerhandbuch.

Um Zugriff zu gewähren, fügen Sie Ihren Benutzern, Gruppen oder Rollen Berechtigungen hinzu:

- Benutzer und Gruppen in AWS IAM Identity Center:

  Erstellen Sie einen Berechtigungssatz. Befolgen Sie die Anweisungen unter Erstellen eines Berechtigungssatzes im AWS IAM Identity Center -Benutzerhandbuch.
- Benutzer, die in IAM über einen Identitätsanbieter verwaltet werden:

  Erstellen Sie eine Rolle für den Identitätsverbund. Befolgen Sie die Anleitung unter Eine Rolle für einen externen Identitätsanbieter (Verbund) erstellen im IAM-Benutzerhandbuch.
- IAM-Benutzer:

- Erstellen Sie eine Rolle, die Ihr Benutzer annehmen kann. Befolgen Sie die Anleitung unter Eine Rolle für einen IAM-Benutzer erstellen im IAM-Benutzerhandbuch.
- (Nicht empfohlen) Weisen Sie einem Benutzer eine Richtlinie direkt zu oder fügen Sie einen Benutzer zu einer Benutzergruppe hinzu. Befolgen Sie die Anweisungen unter Hinzufügen von Berechtigungen zu einem Benutzer (Konsole) im IAM-Benutzerhandbuch.

So verwenden Sie den JSON-Richtlinieneditor zum Erstellen einer Richtlinie

1. Melden Sie sich bei der an AWS-Managementkonsole und öffnen Sie die IAM-Konsole unter https://console.aws.amazon.com/iam/.
2. Wählen Sie im Navigationsbereich auf der linken Seite Policies (Richtlinien).

   Wenn Sie zum ersten Mal Policies (Richtlinien) auswählen, erscheint die Seite Welcome to Managed Policies (Willkommen bei verwalteten Richtlinien). Wählen Sie Get Started.
3. Wählen Sie oben auf der Seite Create policy (Richtlinie erstellen) aus.
4. Wählen Sie im Bereich Policy editor (Richtlinien-Editor) die Option JSON aus.
5. Geben oder fügen Sie ein JSON-Richtliniendokument ein. Weitere Informationen zur IAM-Richtliniensprache finden Sie in der IAM-JSON-Richtlinienreferenz.
6. Beheben Sie alle Sicherheitswarnungen, Fehler oder allgemeinen Warnungen, die während der Richtlinien-Validierung erzeugt wurden, und wählen Sie dann Weiter.

   > **Note**
   >
   > Sie können jederzeit zwischen den Editoroptionen Visual und JSON wechseln. Wenn Sie jedoch Änderungen vornehmen oder im Visual-Editor Weiter wählen, strukturiert IAM Ihre Richtlinie möglicherweise um, um sie für den visuellen Editor zu optimieren. Weitere Informationen finden Sie unter Richtlinienrestrukturierung im IAM-Benutzerhandbuch.
7. (Optional) Wenn Sie eine Richtlinie in der erstellen oder bearbeiten AWS-Managementkonsole, können Sie eine JSON- oder YAML-Richtlinienvorlage generieren, die Sie in CloudFormation Vorlagen verwenden können.

   Wählen Sie dazu im Richtlinien-Editor Aktionen und anschließend CloudFormationVorlage generieren aus. Weitere Informationen CloudFormation finden Sie in der Referenz zum AWS Identity and Access Management Ressourcentyp im AWS CloudFormation Benutzerhandbuch.

8. Wenn Sie mit dem Hinzufügen von Berechtigungen zur Richtlinie fertig sind, wählen Sie Next (Weiter) aus.
9. Geben Sie auf der Seite Prüfen und erstellen unter Richtlinienname einen Namen und unter Beschreibung (optional) eine Beschreibung für die Richtlinie ein, die Sie erstellen. Überprüfen Sie Permissions defined in this policy (In dieser Richtlinie definierte Berechtigungen), um die Berechtigungen einzusehen, die von Ihrer Richtlinie gewährt werden.
10. (Optional) Fügen Sie der Richtlinie Metadaten hinzu, indem Sie Tags als Schlüssel-Wert-Paare anfügen. Weitere Informationen zur Verwendung von Tags in IAM finden Sie unter Tags für AWS Identity and Access Management Ressourcen im IAM-Benutzerhandbuch.
11. Wählen Sie Create policy (Richtlinie erstellen) aus, um Ihre neue Richtlinie zu speichern.

Richtliniendokument – Kopieren Sie das folgende Dokument und fügen Sie es ein, um die JSON-Richtlinie zu erstellen.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Action": [
                "dax:*"
            ],
            "Effect": "Allow",
            "Resource": [
                "*"
            ]
        },
        {
            "Action": [
                "dynamodb:*"
            ],
            "Effect": "Allow",
            "Resource": [
                "*"
            ]
        }
    ]
}
```

Schritt 3: Konfigurieren einer Amazon-EC2-Instance

Wenn Ihre Amazon-EC2-Instance verfügbar ist, können Sie sich bei der Instance anmelden und sie für die Verwendung vorbereiten.

> **Note**
>
> In den folgenden Schritten wird davon ausgegangen, dass Sie eine Verbindung mit Ihrer Amazon-EC2-Instance von einem Linux-Computer aus herstellen. Informationen über andere Möglichkeiten zum Herstellen einer Verbindung finden Sie unter Herstellen einer Verbindung mit Ihrer Linux-Instance im Amazon-EC2-Benutzerhandbuch.

So konfigurieren Sie die EC2-Instance

1. Öffnen Sie die Amazon-EC2-Konsole unter https://console.aws.amazon.com/ec2/.
2. Verwenden Sie den `ssh`-Befehl, wie im folgenden Beispiel gezeigt, um sich bei der Amazon-EC2-Instance anzumelden.

   ```
   ssh -i my-keypair.pem ec2-user@public-dns-name
   ```

   Sie müssen Ihre private Schlüsseldatei (`.pem`-Datei) und den öffentlichen DNS-Namen Ihrer Instance angeben. (Siehe Schritt 1: Starten einer Amazon-EC2-Instance).

   Die Anmelde-ID lautet `ec2-user`. Es ist kein Passwort erforderlich.
3. Nachdem Sie sich bei Ihrer EC2-Instance angemeldet haben, konfigurieren Sie Ihre AWS Anmeldeinformationen wie folgt. Geben Sie Ihre AWS Zugriffsschlüssel-ID und Ihren geheimen Schlüssel (vonSchritt 2: Erstellen eines Benutzers und einer Richtlinie) ein und legen Sie als Standardregionsnamen Ihre aktuelle Region fest. (Im folgenden Beispiel lautet der Standard-Regionsname `us-west-2`.)

   ```
   aws configure
   AWS Access Key ID [None]: AKIAIOSFODNN7EXAMPLE
   AWS Secret Access Key [None]: wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY
   Default region name [None]: us-west-2
   Default output format [None]:
   ```

Nach dem Start und der Konfiguration der Amazon-EC2-Instance können Sie die Funktionalität von DAX mit einer der verfügbaren Beispielanwendungen testen. Weitere Informationen finden Sie unter Schritt 4: Ausführen einer Beispielanwendung.

Schritt 4: Ausführen einer Beispielanwendung

Um Sie beim Testen der Amazon-DynamoDB-Accelerator-(DAX)-Funktionalität zu unterstützen, können Sie eine der verfügbaren Beispielanwendungen auf der Amazon-EC2-Instance ausführen

Themen

- Node.js und DAX
- DAX SDK für Go
- Java und DAX
- .NET und DAX
- Python und DAX

Node.js und DAX

Standard-Client-Konfiguration für Node.js

Bei der Konfiguration des DAX JavaScript SDK-Clients können Sie verschiedene Parameter anpassen, um Leistung, Verbindungsbehandlung und Fehlerresistenz zu optimieren. In der folgenden Tabelle werden die Standardkonfigurationseinstellungen beschrieben, mit denen gesteuert wird, wie Ihr Client mit dem DAX-Cluster interagiert, u. a. Timeout-Werte, Wiederholungsmechanismen, Verwaltung von Anmeldeinformationen und Optionen zur Systemüberwachung. Weitere Informationen finden Sie unter Dynamo DBClient Operations.

Standardeinstellungen für den DAX-JS-SDK-Client

Parameter	Typ	Description
`region` optional	`string`	Der für den DAX-Client AWS-Region zu verwendende (Beispiel - 'us-east-1'). Dies ist ein erforderlicher Parameter , sofern er nicht über die Umgebungsvariable bereitgestellt wird.

Parameter	Typ	Description
`endpoint` Erforderlich	`string`	Der Endpunkt des Clusters, mit dem das SDK eine Verbindung herstellt. Beispiele: Unverschlüsselt — .region.amazonaws.com dax-cluster-name Verschlüsselt – daxs://Mein-Cluster.l6fzcv.dax-clusters.us-east-1.amazonaws.com
`requestTimeout` Standard: 6 000 ms	`number`	Dies definiert die maximale Zeit, die der Client auf eine Antwort von DAX wartet.
`writeRetries` Standard: 1	`number`	Die Anzahl der Wiederholungsversuche für fehlgeschlagene Schreibanforderungen.
`readRetries` Standard: 1	`number`	Die Anzahl der Wiederholungsversuche für fehlgeschlagene Leseanforderungen.
`maxRetries` Standard: 1	`number`	Die maximale Anzahl von Versuchen bei fehlgeschlagenen Anforderungen. Wenn festgelegt readRetries/writeRetries sind, hat die in ReadRetries und WriteRetries festgelegte Konfiguration Vorrang vor MaxRetries.

Parameter	Typ	Description
`connectTimeout` Standard: 10 000 ms	`number`	Das Timeout (in Millisekunden) für das Herstellen einer Verbindung zu einem der Cluster-Knoten.
`maxConcurrentConnections` Standard: 100	`number`	Beschränkt die Gesamtzahl der gleichzeitigen Verbindungen, die eine Client-Instance pro Knoten in einem DAX-Cluster herstellen kann.

Parameter	Typ	Description
`maxRetryDelay` Standard: 7 000 ms	`number`	Wenn der DAX-Server angibt, dass eine Wiederherstellung erforderlich ist, indem das `waitForRecoveryBeforeRetrying` -Flag auf „true“ gesetzt wird, hält der Client an, bevor er es erneut versucht. Während dieser Wiederherstellungsperioden bestimmt der `maxRetryDelay` -Parameter die maximale Wartezeit zwischen Wiederholungsversuchen. Diese für die Wiederherstellung spezifische Konfiguration gilt nur, wenn sich der DAX-Server im Wiederherstellungsmodus befindet. In allen anderen Szenarien folgt das Wiederholungsverhalten einem von zwei Mustern: entweder einer exponentiellen Verzögerung auf der Grundlage der Anzahl der Wiederholungen (bestimmt durch den Parameter `writeRetries` , `readRetries` oder `maxRetries`) oder einer sofortigen Wiederholung, abhängig vom Ausnahmetyp.

Parameter	Typ	Description
`credentials` optional	`AwsCredentialIdentity` | `AwsCredentialIdentityProvider`	Die AWS Anmeldeinformationen, die für die Authentifizierung von Anfragen verwendet werden sollen. Dies kann als AwsCredentialIdentity oder als angegeben werden. AwsCredentialIdentityProvider Wenn nicht angegeben, verwendet das AWS -SDK automatisch die standardmäßige Anbieterkette für Anmeldeinformationen. Beispiel: `{ accessKeyId: 'AKIA... ' , secretAccessKey: ' ... ', sessionToken:'... '} `* @default Verwendet die Standard-Anbieterkette AWS für Anmeldeinformationen.
`healthCheckInterval` Standard: 5 000 ms	`number`	Das Intervall (in Millisekunden) zwischen den Cluster-Zustandsprüfungen. Bei einem kürzeren Intervall werden die Prüfungen häufiger durchgeführt.
`healthCheckTimeout` Standard: 1 000 ms	`number`	Das Timeout (in Millisekunden) für den Abschluss der Zustandsprüfung.

Parameter	Typ	Description
`skipHostnameVerification` Standard: falsch	`boolean`	Die Hostnamen-Überprüfung von TLS-Verbindungen wird übersprungen. Dies hat keine Auswirkungen auf unverschlüsselte Cluster. Standardmäßig wird die Überprüfung des Hostnamens durchgeführt. Wenn Sie diesen Wert auf „True“ setzen, wird die Überprüfung übersprungen. Sie müssen sich bewusst sein, welche Folgen eine Deaktivierung hat, d. h. dass der Cluster, zu dem Sie eine Verbindung herstellen, nicht authentifiziert werden kann.
`unhealthyConsecutiveErrorCount` Standard: 5	`number`	Legt die Anzahl aufeinander erfolgender Fehler fest, die erforderlich sind, um innerhalb des Intervalls für die Zustandsprüfung zu signalisieren, dass der Knoten fehlerhaft ist.
`clusterUpdateInterval` Standard: 4 000 ms	`number`	Gibt das Intervall zwischen der Abfrage von Mitgliedschaftsänderungen bei Cluster-Mitgliedern zurück.
`clusterUpdateThreshold` Standard: 125	`number`	Gibt den Schwellenwert zurück, unter dem der Cluster nicht nach Mitgliedschaftsänderungen abgefragt wird.

Parameter	Typ	Description
`credentailProvider` optional | Standard null	`AwsCredentialIdentityProvider`	Benutzerdefinierter Anbieter für AWS Anmeldeinformationen, die zur Authentifizierung von Anfragen an DAX verwendet werden.

Paginierungskonfiguration für DaxDocument

Name	Typ	Detail
`client`	DaxDocument	Instanz des DaxDocument Typs.
`pageSize`	number	Bestimmt die Anzahl der Elemente pro Seite.
`startingToken` Optional	any	LastEvaluatedKey aus der vorherigen Antwort kann für nachfolgende Anfragen verwendet werden.

Informationen zur Verwendung der Paginierung finden Sie unter the section called “TryDax.js”.

Migration zum DAX-Node.js-SDK V3

Dieser Migrationsleitfaden hilft Ihnen bei der Umstellung Ihrer vorhandenen DAX-Node.js-Anwendungen. Das neue SDK erfordert Node.js 18 oder höher und führt mehrere wichtige Änderungen in der Strukturierung Ihres DynamoDB-Accelerator-Codes ein. Dieser Leitfaden führt Sie durch die wichtigsten Unterschiede, darunter Syntaxänderungen, neue Importmethoden und aktualisierte asynchrone Programmiermuster.

Verwendung von V2 Node.js DAX

```
const AmazonDaxClient = require('amazon-dax-client');
const AWS = require('aws-sdk');
```

```
var region = "us-west-2";

AWS.config.update({
  region: region,
});

var client = new AWS.DynamoDB.DocumentClient();

if (process.argv.length > 2) {
  var dax = new AmazonDaxClient({
    endpoints: [process.argv[2]],
    region: region,
  });
  client = new AWS.DynamoDB.DocumentClient({ service: dax });
}

// Make Get Call using Dax
var params = {
    TableName: 'TryDaxTable',
    pk: 1,
    sk: 1
}
client.get(params, function (err, data) {
    if (err) {
        console.error(
            "Unable to read item. Error JSON:",
            JSON.stringify(err, null, 2)
          );
    } else {
        console.log(data);
    }
});
```

Verwendung von V3 Node.js DAX

Für die Verwendung von DAX Node.js ist V3-Node-Version 18 oder höher die bevorzugte Version. Um zu Node 18 zu wechseln, verwenden Sie Folgendes:

```
// Import AWS DAX V3
import { DaxDocument } from '@amazon-dax-sdk/lib-dax';

// Import AWS SDK V3 DynamoDBDocument ~ DocumentClient in V2
import { DynamoDBDocument } from '@aws-sdk/lib-dynamodb';
```

```
import { DynamoDBClient } from '@aws-sdk/client-dyanmodb';

// Create DynamoDBDocument
var client = DynamoDBDocument.from(new DynamoDB({region: 'us-west-2'});

// Override DynamoDBDocument Client with DaxDocument
if (process.argv.length > 2) {
  client = new DaxDocument({
    endpoints: [process.argv[2]],
    region: 'us-west-2',
  });
}

var params = {
    TableName: 'TryDaxTable',
    pk: 1,
    sk: 1
}
// Dax Shifted it's API Calls to await/promise
try {
  const results = await client.get(params);
  console.log(results);
} catch (err) {
  console.error(err)
}
```

Das DAX-SDK für Node.js v3.x ist mit dem AWS -SDK für Node.js v3.x kompatibel. Das DAX-SDK für Node.js v3.x unterstützt die Verwendung aggregierter Clients. Beachten Sie, dass DAX die Erstellung von Bare-Bone-Clients nicht unterstützt. Weitere Informationen zu nicht unterstützten Funktionen finden Sie unter the section called “Funktionen, die nicht mit AWS SDK V3 identisch sind”.

Um die Node.js-Beispielanwendung auf Ihrer Amazon-EC2-Instance auszuführen, gehen Sie wie folgt vor.

So führen Sie das Node.js-Beispiel für DAX aus

1. Richten Sie Node.js auf Ihrer Amazon-EC2-Instance wie folgt ein:

   a. Installieren Sie den Node Version Manager (nvm).

   ```
   curl -o- https://raw.githubusercontent.com/nvm-sh/nvm/v0.40.1/install.sh | bash
   ```

   b. Installieren Sie Node.js mit dem nvm.

```
nvm install 18
```

c. Verwenden Sie nvm, um Node 18 zu verwenden.

```
nvm use 18
```

d. Testen Sie, ob Node.js installiert ist und ordnungsgemäß ausgeführt wird.

```
node -e "console.log('Running Node.js ' + process.version)"
```

Die folgende Meldung sollte angezeigt werden.

```
Running Node.js v18.x.x
```

2. Installieren Sie den Client DaxDocument Node.js mit dem Node Package Manager (npm).

```
npm install @amazon-dax-sdk/lib-dax
```

TryDax Beispielcode

Wenn Sie die Leistungsvorteile von DynamoDB Accelerator (DAX) bewerten möchten, gehen Sie wie folgt vor, um einen Beispieltest durchzuführen, bei dem die Lesevorgangszeiten zwischen Standard-DynamoDB und einem DAX-Cluster verglichen werden.

1. Nachdem Sie Ihren Workspace eingerichtet und `lib-dax` als Abhängigkeit hinzugefügt haben, kopieren Sie the section called “TryDax.js” in Ihr Projekt.
2. Führen Sie das Programm für Ihren DAX-Cluster aus. Um den Endpunkt für Ihren DAX-Cluster zu bestimmen, wählen Sie einen der folgenden Schritte aus:

   - Using the DynamoDB console (Verwenden der DynamoDB-Konsole) — Wählen Sie Ihren DAX-Cluster aus. Der Cluster-Endpunkt wird auf der Konsole angezeigt, wie im folgenden Beispiel gezeigt.

     ```
     dax://my-cluster.l6fzcv.dax-clusters.us-east-1.amazonaws.com
     ```

   - Verwenden Sie AWS CLI— Geben Sie den folgenden Befehl ein.

     ```
     aws dax describe-clusters --query "Clusters[*].ClusterDiscoveryEndpoint"
     ```

Der Cluster-Endpunkt wird in der Ausgabe angezeigt, wie im folgenden Beispiel gezeigt.

```
{
    "Address": "my-cluster.l6fzcv.dax-clusters.us-east-1.amazonaws.com",
    "Port": 8111,
    "URL": "dax://my-cluster.l6fzcv.dax-clusters.us-east-1.amazonaws.com"
}
```

3. Führen Sie jetzt das Programm aus, indem Sie den Cluster-Endpunkt als Befehlszeilenparameter angeben.

```
node TryDax.js dax://my-cluster.l6fzcv.dax-clusters.us-east-1.amazonaws.com
```

Die Ausgabe sollte folgendermaßen oder ähnlich aussehen:

```
Attempting to create table; please wait...
Successfully created table. Table status: ACTIVE
Writing data to the table...
Writing 20 items for partition key:  1
Writing 20 items for partition key:  2
Writing 20 items for partition key:  3
...
Running GetItem Test
        Total time: 153555.10 µs - Avg time: 383.89 µs
        Total time: 44679.96 µs - Avg time: 111.70 µs
        Total time: 36885.86 µs - Avg time: 92.21 µs
        Total time: 32467.25 µs - Avg time: 81.17 µs
        Total time: 32202.60 µs - Avg time: 80.51 µs
Running Query Test
        Total time: 14869.25 µs - Avg time: 2973.85 µs
        Total time: 3036.31 µs - Avg time: 607.26 µs
        Total time: 2468.92 µs - Avg time: 493.78 µs
        Total time: 2062.53 µs - Avg time: 412.51 µs
        Total time: 2178.22 µs - Avg time: 435.64 µs
Running Scan Test
        Total time: 2395.88 µs - Avg time: 479.18 µs
        Total time: 2207.16 µs - Avg time: 441.43 µs
        Total time: 2443.14 µs - Avg time: 488.63 µs
        Total time: 2038.24 µs - Avg time: 407.65 µs
        Total time: 1972.17 µs - Avg time: 394.43 µs
```

```
Running Pagination Test
Scan Pagination
[
  { pk: 1, sk: 1, someData: 'XXXXXXXXXX' },
  { pk: 1, sk: 2, someData: 'XXXXXXXXXX' },
  { pk: 1, sk: 3, someData: 'XXXXXXXXXX' }
]
[
  { pk: 1, sk: 4, someData: 'XXXXXXXXXX' },
  { pk: 1, sk: 5, someData: 'XXXXXXXXXX' },
  { pk: 1, sk: 6, someData: 'XXXXXXXXXX' }
]
...
Query Pagination
[
  { pk: 1, sk: 1, someData: 'XXXXXXXXXX' },
  { pk: 1, sk: 2, someData: 'XXXXXXXXXX' },
  { pk: 1, sk: 3, someData: 'XXXXXXXXXX' }
]
[
  { pk: 1, sk: 4, someData: 'XXXXXXXXXX' },
  { pk: 1, sk: 5, someData: 'XXXXXXXXXX' },
  { pk: 1, sk: 6, someData: 'XXXXXXXXXX' }
]
...
Attempting to delete table; please wait...
Successfully deleted table.
```

Beachten Sie die Zeitinformationen. Die Anzahl der Mikrosekunden, die für die `GetItem`-, `Query`-, `Scan`-Tests benötigt werden.

4. In diesem Fall haben Sie die Programme für den DAX-Cluster ausgeführt. Jetzt führen Sie das Programm erneut aus, diesmal für DynamoDB.

5. Führen Sie nun das Programm erneut aus, diesmal aber ohne die Cluster-Endpunkt-URL als Befehlszeilenparameter.

```
node TryDax.js
```

Sehen Sie sich die Ausgabe an und notieren Sie die Zeitinformationen. Die verstrichene Zeit sollte für `GetItem`, `Query` und `Scan` mit DAX deutlich kürzer sein als mit DynamoDB.

Funktionen, die nicht mit AWS SDK V3 identisch sind

- Bare-Bone-Clients: Dax Node.js V3 unterstützt keine Bare-Bone-Clients.

```
const dynamoDBClient = new DynamoDBClient({ region: 'us-west-2' });
const regularParams = {
    TableName: 'TryDaxTable',
    Key: {
        pk: 1,
        sk: 1
    }
};
// The DynamoDB client supports the send operation.
const dynamoResult = await dynamoDBClient.send(new GetCommand(regularParams));

// However, the DaxDocument client does not support the send operation.
const daxClient = new DaxDocument({
    endpoints: ['your-dax-endpoint'],
    region: 'us-west-2',
});

const params = {
    TableName: 'TryDaxTable',
    Key: {
        pk: 1,
        sk: 1
    }
};

// This will throw an error - send operation is not supported for DAX. Please refer
 to documentation.
const result = await daxClient.send(new GetCommand(params));
console.log(result);
```

- Middleware-Stack: Dax Node.js V3 unterstützt die Verwendung von Middleware-Funktionen nicht.

```


const dynamoDBClient = new DynamoDBClient({ region: 'us-west-2' });
// The DynamoDB client supports the middlewareStack.
dynamoDBClient.middlewareStack.add(
  (next, context) =>> async (args) => {
    console.log("Before operation:", args);
```

```
    const result = await next(args);
    console.log("After operation:", result);
    return result;
  },
  {
    step: "initialize", // or "build", "finalizeRequest", "deserialize"
    name: "loggingMiddleware",
  }
);

// However, the DaxDocument client does not support the middlewareStack.
const daxClient = new DaxDocument({
    endpoints: ['your-dax-endpoint'],
    region: 'us-west-2',
});

// This will throw an error - custom middleware and middlewareStacks are not
 supported for DAX. Please refer to documentation.
daxClient.middlewareStack.add(
  (next, context) => async (args) => {
    console.log("Before operation:", args);
    const result = await next(args);
    console.log("After operation:", result);
    return result;
  },
  {
    step: "initialize", // or "build", "finalizeRequest", "deserialize"
    name: "loggingMiddleware",
  }
);
```

TryDax.js

```
import { DynamoDB, waitUntilTableExists, waitUntilTableNotExists } from "@aws-sdk/
client-dynamodb";
import { DaxDocument, daxPaginateScan, daxPaginateQuery } from "@amazon-dax-sdk/lib-
dax";
import { DynamoDBDocument, paginateQuery, paginateScan } from "@aws-sdk/lib-dynamodb";

const region = "us-east-1";
const tableName = "TryDaxTable";
```

```
// Determine the client (DynamoDB or DAX)
let client = DynamoDBDocument.from(new DynamoDB({ region }));
if (process.argv.length > 2) {
  client = new DaxDocument({ region, endpoint: process.argv[2] });
}

// Function to create table
async function createTable() {
  const dynamodb = new DynamoDB({ region });
  const params = {
    TableName: tableName,
    KeySchema: [
      { AttributeName: "pk", KeyType: "HASH" },
      { AttributeName: "sk", KeyType: "RANGE" },
    ],
    AttributeDefinitions: [
      { AttributeName: "pk", AttributeType: "N" },
      { AttributeName: "sk", AttributeType: "N" },
    ],
    ProvisionedThroughput: { ReadCapacityUnits: 25, WriteCapacityUnits: 25 },
  };

  try {
    console.log("Attempting to create table; please wait...");
    await dynamodb.createTable(params);
    await waitUntilTableExists({ client: dynamodb, maxWaitTime: 30 }, { TableName:
 tableName });
    console.log("Successfully created table. Table status: ACTIVE");
  } catch (err) {
    console.error("Error in table creation:", err);
  }
}

// Function to insert data
async function writeData() {
  console.log("Writing data to the table...");
  const someData = "X".repeat(10);
  for (let ipk = 1; ipk <= 20; ipk++) {
    console.log("Writing 20 items for partition key: ", ipk)
    for (let isk = 1; isk <= 20; isk++) {
      try {
        await client.put({ TableName: tableName, Item: { pk: ipk, sk: isk,
 someData } });
```

```
      } catch (err) {
        console.error("Error inserting data:", err);
      }
    }
  }
}

// Function to test GetItem
async function getItemTest() {
  console.log("Running GetItem Test");
  for (let i = 0; i < 5; i++) {
    const startTime = performance.now();
    const promises = [];
    for (let ipk = 1; ipk <= 20; ipk++) {
      for (let isk = 1; isk <= 20; isk++) {
        promises.push(client.get({ TableName: tableName, Key: { pk: ipk, sk: isk } }));
      }
    }
    await Promise.all(promises);
    const endTime = performance.now();
    const iterTime = (endTime - startTime) * 1000;
    const totalTime = iterTime.toFixed(2).padStart(3, ' ');
    const avgTime = (iterTime / 400).toFixed(2).padStart(3, ' ');
    console.log(`\tTotal time: ${totalTime} \u00B5s - Avg time: ${avgTime} \u00B5s`);
  }
}

// Function to test Query
async function queryTest() {
  console.log("Running Query Test");
  for (let i = 0; i < 5; i++) {
    const startTime = performance.now();
    const promises = [];
    for (let pk = 1; pk <= 5; pk++) {
      const params = {
        TableName: tableName,
        KeyConditionExpression: "pk = :pkval and sk between :skval1 and :skval2",
        ExpressionAttributeValues: { ":pkval": pk, ":skval1": 1, ":skval2": 2 },
      };
      promises.push(client.query(params));
    }
    await Promise.all(promises);
    const endTime = performance.now();
    const iterTime = (endTime - startTime) * 1000;
```

```
    const totalTime = iterTime.toFixed(2).padStart(3, ' ');
    const avgTime = (iterTime / 5).toFixed(2).padStart(3, ' ');
    console.log(`\tTotal time: ${totalTime} \u00B5s - Avg time: ${avgTime} \u00B5s`);
  }
}

// Function to test Scan
async function scanTest() {
  console.log("Running Scan Test");
  for (let i = 0; i < 5; i++) {
    const startTime = performance.now();
    const promises = [];
    for (let pk = 1; pk <= 5; pk++) {
      const params = {
        TableName: tableName,
        FilterExpression: "pk = :pkval and sk between :skval1 and :skval2",
        ExpressionAttributeValues: { ":pkval": pk, ":skval1": 1, ":skval2": 2 },
      };
      promises.push(client.scan(params));
    }
    await Promise.all(promises);
    const endTime = performance.now();
    const iterTime = (endTime - startTime) * 1000;
    const totalTime = iterTime.toFixed(2).padStart(3, ' ');
    const avgTime = (iterTime / 5).toFixed(2).padStart(3, ' ');
    console.log(`\tTotal time: ${totalTime} \u00B5s - Avg time: ${avgTime} \u00B5s`);
  }
}

// Function to test Pagination
async function paginationTest() {
  console.log("Running Pagination Test");
  console.log("Scan Pagination");
  const scanParams = { TableName: tableName };
  const paginator = process.argv.length > 2 ? daxPaginateScan : paginateScan;
  for await (const page of paginator({ client, pageSize: 3 }, scanParams)) {
    console.log(page.Items);
  }

  console.log("Query Pagination");
  const queryParams = {
    TableName: tableName,
    KeyConditionExpression: "pk = :pkval and sk between :skval1 and :skval2",
    ExpressionAttributeValues: { ":pkval": 1, ":skval1": 1, ":skval2": 10 },
```

```
  };
  const queryPaginator = process.argv.length > 2 ? daxPaginateQuery : paginateQuery;
  for await (const page of queryPaginator({ client, pageSize: 3 }, queryParams)) {
    console.log(page.Items);
  }
}

// Function to delete the table
async function deleteTable() {
  const dynamodb = new DynamoDB({ region });
  console.log("Attempting to delete table; please wait...")
  try {
    await dynamodb.deleteTable({ TableName: tableName });
    await waitUntilTableNotExists({ client: dynamodb, maxWaitTime: 30 }, { TableName:
 tableName });
    console.log("Successfully deleted table.");
  } catch (err) {
    console.error("Error deleting table:", err);
  }
}

// Execute functions sequentially
(async function () {
  await createTable();
  await writeData();
  await getItemTest();
  await queryTest();
  await scanTest();
  await paginationTest();
  await deleteTable();
})();
```

DAX SDK für Go

Gehen Sie wie folgt vor, um die Amazon-DynamoDB-Accelerator-(DAX)-SDK für Go auf Ihrer Amazon-EC2-Instance auszuführen.

So führen Sie das SDK-für-Go-Beispiel für DAX aus

1. Richten Sie das SDK für Go auf Ihrer Amazon-EC2-Instance ein:

    a. Installieren Sie die Go-Programmiersprache (`Golang`).

```
sudo yum install -y golang
```

b. Testen Sie, ob Golang installiert ist und ordnungsgemäß ausgeführt wird.

```
go version
```

Eine Nachricht wie diese sollte erscheinen.

```
go version go1.23.4 linux/amd64
```

2. Installieren Sie die Golang-Beispielanwendung.

```
go get github.com/aws-samples/sample-aws-dax-go-v2
```

3. Führen Sie die folgenden Golang-Programme aus. Das erste Programm erstellt eine DynamoDB-Tabelle mit dem Namen `TryDaxGoTable`. Das zweite Programm schreibt Daten in die Tabelle.

```
go run ~/go/pkg/mod/github.com/aws-samples/sample-aws-dax-go-v2@v1.0.0/try_dax.go -
service dynamodb -command create-table
```

```
go run ~/go/pkg/mod/github.com/aws-samples/sample-aws-dax-go-v2@v1.0.0/try_dax.go -
service dynamodb -command put-item
```

4. Führen Sie die folgenden Golang-Programme aus.

```
go run ~/go/pkg/mod/github.com/aws-samples/sample-aws-dax-go-v2@v1.0.0/try_dax.go -
service dynamodb -command get-item
```

```
go run ~/go/pkg/mod/github.com/aws-samples/sample-aws-dax-go-v2@v1.0.0/try_dax.go -
service dynamodb -command query
```

```
go run ~/go/pkg/mod/github.com/aws-samples/sample-aws-dax-go-v2@v1.0.0/try_dax.go -
service dynamodb -command scan
```

Beachten Sie die Zeitinformationen – die Anzahl der benötigten Millisekunden für den `GetItem`-, `Query`- und `Scan`-Test.

5. Im vorherigen Schritt haben Sie die Programme für den DynamoDB-Endpunkt ausgeführt. Führen Sie die Programme jetzt erneut aus. Dieses Mal werden die `GetItem`-, `Query`- und `Scan`-Operationen aber vom DAX-Cluster verarbeitet.

   Um den Endpunkt für Ihren DAX-Cluster zu bestimmen, wählen Sie einen der folgenden Schritte aus:

   - Using the DynamoDB console (Verwenden der DynamoDB-Konsole) — Wählen Sie Ihren DAX-Cluster aus. Der Cluster-Endpunkt wird auf der Konsole angezeigt, wie im folgenden Beispiel gezeigt.

     ```
     dax://my-cluster.l6fzcv.dax-clusters.us-east-1.amazonaws.com
     ```

   - Verwenden von AWS CLI — Geben Sie den folgenden Befehl ein.

     ```
     aws dax describe-clusters --query "Clusters[*].ClusterDiscoveryEndpoint"
     ```

     Der Cluster-Endpunkt wird wie im folgenden Beispiel in der Ausgabe angezeigt.

     ```
     {
         "Address": "my-cluster.l6fzcv.dax-clusters.us-east-1.amazonaws.com",
         "Port": 8111,
         "URL": "dax://my-cluster.l6fzcv.dax-clusters.us-east-1.amazonaws.com"
     }
     ```

   Führen Sie jetzt die Programme erneut aus. Geben Sie dieses Mal jedoch den Cluster-Endpunkt als Befehlszeilenparameter an.

   ```
   go run ~/go/pkg/mod/github.com/aws-samples/sample-aws-dax-go-v2@v1.0.0/try_dax.go
    -service dax -command get-item -endpoint my-cluster.l6fzcv.dax-clusters.us-
   east-1.amazonaws.com:8111
   ```

   ```
   go run ~/go/pkg/mod/github.com/aws-samples/sample-aws-dax-go-v2@v1.0.0/try_dax.go
    -service dax -command query -endpoint my-cluster.l6fzcv.dax-clusters.us-
   east-1.amazonaws.com:8111
   ```

```
go run ~/go/pkg/mod/github.com/aws-samples/sample-aws-dax-go-v2@v1.0.0/try_dax.go
 -service dax -command scan -endpoint my-cluster.l6fzcv.dax-clusters.us-
east-1.amazonaws.com:8111
```

```
go run ~/go/pkg/mod/github.com/aws-samples/sample-aws-dax-go-v2@v1.0.0/try_dax.go
 -service dax -command paginated-scan -endpoint my-cluster.l6fzcv.dax-clusters.us-
east-1.amazonaws.com:8111
```

```
go run ~/go/pkg/mod/github.com/aws-samples/sample-aws-dax-go-v2@v1.0.0/try_dax.go
 -service dax -command paginated-query -endpoint my-cluster.l6fzcv.dax-clusters.us-
east-1.amazonaws.com:8111
```

```
go run ~/go/pkg/mod/github.com/aws-samples/sample-aws-dax-go-v2@v1.0.0/try_dax.go
 -service dax -command paginated-batch-get -endpoint my-cluster.l6fzcv.dax-
clusters.us-east-1.amazonaws.com:8111
```

Sehen Sie sich den Rest der Ausgabe an und notieren Sie die Zeitinformationen. Die verstrichene Zeit sollte für `GetItem`, `Query` und `Scan` mit DAX deutlich kürzer sein als mit DynamoDB.

6. Führen Sie das folgende Golang-Programm aus, um `TryDaxGoTable` zu löschen:

```
go run ~/go/pkg/mod/github.com/aws-samples/sample-aws-dax-go-v2@v1.0.0/try_dax.go -
service dynamodb -command delete-table
```

Funktionen, die nicht mit AWS SDK für Go V2 gleichwertig sind

Middleware Stack — DAX Go V2 unterstützt die Verwendung von Middleware Stacks bis jetzt nicht. APIoptions Weitere Informationen finden Sie unter Anpassen der v2-Client-Anfragen mit Middleware. AWS SDK für Go

Beispiel:

```
// Custom middleware implementation
type customSerializeMiddleware struct{}
// ID returns the identifier for the middleware
func (m *customSerializeMiddleware) ID() string {
    return "CustomMiddleware"
```

```
}
// HandleSerialize implements the serialize middleware handler
func (m *customSerializeMiddleware) HandleSerialize(
    ctx context.Context,
    in middleware.SerializeInput,
    next middleware.SerializeHandler,
) (
    out middleware.SerializeOutput,
    metadata middleware.Metadata,
    err error,
) {
    // Add your custom logic here before the request is serialized
    fmt.Printf("Executing custom middleware for request: %v\n", in)
    // Call the next handler in the middleware chain
    return next.HandleSerialize(ctx, in)
}

func executeGetItem(ctx context.Context) error {
    client, err := initItemClient(ctx)
    if err != nil {
        os.Stderr.WriteString(fmt.Sprintf("failed to initialize client: %v\n", err))
        return err
    }

    st := time.Now()
    for c := 0; c < iterations; c++ {
        for i := 0; i < pkMax; i++ {
            for j := 0; j < skMax; j++ {
                // Create key using attributevalue.Marshal for type safety
                pk, err := attributevalue.Marshal(fmt.Sprintf("%s_%d", keyPrefix, i))
                if err != nil {
                    return fmt.Errorf("error marshaling pk: %v", err)
                }
                sk, err := attributevalue.Marshal(fmt.Sprintf("%d", j))
                if err != nil {
                    return fmt.Errorf("error marshaling sk: %v", err)
                }
                key := map[string]types.AttributeValue{
                    "pk": pk,
                    "sk": sk,
                }
                in := &dynamodb.GetItemInput{
                    TableName: aws.String(table),
                    Key:       key,
```

```
                }

                // Custom middleware option
                customMiddleware := func(o *dynamodb.Options) {
                    o.APIOptions = append(o.APIOptions, func(stack *middleware.Stack)
 error {
                        // Add custom middleware to the stack
                        return stack.Serialize.Add(&customSerializeMiddleware{},
 middleware.After)
                    })
                }

                // Apply options to the GetItem call
                out, err := client.GetItem(ctx, in, customMiddleware)
                if err != nil {
                    return err
                }
                writeVerbose(out)
            }
        }
    }
    d := time.Since(st)
    os.Stdout.WriteString(fmt.Sprintf("Total Time: %v, Avg Time: %v\n", d, d/
iterations))
    return nil
}
```

Ausgabe:

```
failed to execute command: custom middleware through APIOptions is not supported in DAX
 client
exit status 1
```

Standard-Client-Konfiguration für Go

In diesem Handbuch werden Sie durch die Konfigurationsoptionen geführt, mit denen Sie die Leistung, das Verbindungsmanagement und das Protokollierungsverhalten Ihres DAX-Clients optimieren können. Wenn Sie sich mit den Standardeinstellungen und deren Anpassung vertraut machen, können Sie die Interaktion Ihrer Go-Anwendung mit DAX optimieren.

In diesem Abschnitt

- Standardeinstellungen für den DAX-Go-SDK-Client

- Client-Erstellung

Standardeinstellungen für den DAX-Go-SDK-Client

Parameter	Typ	Description
`Region` Erforderlich	`string`	Der für den DAX-Client AWS-Region zu verwendende (Beispiel- 'us-east-1'). Dies ist ein erforderlicher Parameter , sofern er nicht über die Umgebung bereitgestellt wird.
`HostPorts` Erforderlich	`[] string`	Liste der DAX-Cluster-Endpunkte, mit denen das SDK eine Verbindung herstellt. Beispiel: Nicht verschlüsselt – dax://Mein-Cluster.l6fzcv.dax-clusters.us-east-1.amazonaws.com Verschlüsselt – daxs://Mein-Cluster.l6fzcv.dax-clusters.us-east-1.amazonaws.com
`MaxPendingConnectionsPerHost` Standard: 10	`number`	Die Anzahl der gleichzeitigen Verbindungsversuche. (Verbindungen können gleichzeitig hergestellt werden.)
`ClusterUpdateThreshold` Standard: 125 * time.Millisecond	`time.Duration`	Die Mindestzeit, die zwischen Clusteraktualisierungen vergehen muss.

Parameter	Typ	Description
`ClusterUpdateInterval` Standard: 4 * time.Second	`time.Duration`	Das Intervall, in dem der Client die DAX-Clusterinformationen automatisch aktualisiert.
`IdleConnectionsReapDelay` Standard: 30 * time.Second	`time.Duration`	Das Intervall, in dem der Client inaktive Verbindungen im DAX-Client schließt.
`ClientHealthCheckInterval` Standard: 5 * time.Second	`time.Duration`	Das Intervall, in dem der Client Zustandsprüfungen der DAX-Cluster-Endpunkte durchführt.
`Credentials` default	`aws.CredentialsProvider`	Die AWS Anmeldeinformationen, die vom DAX-Client zur Authentifizierung von Anfragen an den DAX-Dienst verwendet werden. Siehe Credentials and Credential Providers.
`DialContext` default	`func`	Eine benutzerdefinierte Funktion, die vom DAX-Client verwendet wird, um Verbindungen zum DAX-Cluster herzustellen.

Parameter	Typ	Description
`SkipHostnameVerification` Standard: falsch	bool	Die Hostnamen-Überprüfung von TLS-Verbindungen wird übersprungen. Diese Einstellung wirkt sich nur auf verschlüsselte Cluster aus. Wenn sie auf „True“ gesetzt ist, wird die Überprüfung des Hostnamens deaktiviert. Wenn Sie die Überprüfung deaktivieren, können Sie die Identität des Clusters, zu dem Sie eine Verbindung herstellen, nicht authentifizieren, was ein Sicherheitsrisiko darstellt. Standardmäßig ist die Überprüfung des Hostnamens aktiviert.
`RouteManagerEnabled` Standard: falsch	`bool`	Dieses Flag wird verwendet, um Routen zu entfernen, bei denen Netzwerkfehler auftreten.
`RequestTimeout` Standard: 60 * time.Second	`time.Duration`	Dies definiert die maximale Zeit, die der Client auf eine Antwort von DAX wartet. Priorität: Kontext-Timeout (falls festgelegt) > `RequestTimmeout` (falls festgelegt) > Standard von 60 s für `RequestTimeout` .

Parameter	Typ	Description
`WriteRetries` Standard: 2	`number`	Die Anzahl der Wiederholungsversuche für fehlgeschlagene Schreibanforderungen.
`ReadRetries` Standard: 2	`number`	Die Anzahl der Wiederholungsversuche für fehlgeschlagene Leseanforderungen.
`RetryDelay` Standard: 0	`time.Duration`	Die Verzögerung bei nicht gedrosselten Fehlern (in Sekunden) bei Wiederholungsversuchen, wenn eine Anforderung fehlschlägt.
`Logger` optional	`logging.Logger`	Logger ist eine Schnittstelle zum Protokollieren von Einträgen bei bestimmten Klassifizierungen.
`LogLevel` Standard-Utils. LogOff	`number`	Dieses loglevel ist nur für DAX definiert. Es kann mit Github importiert werden. com/aws/aws-dax-go-v2/tree/main/dax/utils. `const ( LogOff LogLevelType = 0 LogDebug LogLevelType = 1 LogDebugWithRequestRetries LogLevelType = 2 )`

> **Note**
>
> Für `time.Duration` ist die Standardeinheit Nanosekunde. Wenn für einen Parameter keine Einheit angegeben wird, wird dies als Nanosekunden betrachtet: `daxCfg.ClusterUpdateInterval = 10` bedeutet 10 Nanosekunden. (`daxCfg.ClusterUpdateInterval = 10 * time.Millisecond` bedeutet 10 Millisekunden).

Client-Erstellung

Erstellen Sie einen DAX-Client wie folgt:

- Erstellen Sie die DAX-Konfiguration und erstellen Sie dann den DAX-Client mithilfe der DAX-Konfiguration. Auf diese Weise können Sie eine DAX-Konfiguration bei Bedarf überschreiben.

```
import (
"github.com/aws/aws-dax-go-v2/dax/utils"
"github.com/aws/aws-dax-go-v2/dax"
)

// Non - Encrypted : 'dax://my-cluster.l6fzcv.dax-clusters.us-east-1.amazonaws.com'.
// Encrypted : daxs://my-cluster.l6fzcv.dax-clusters.us-east-1.amazonaws.com'.

config := dax.DefaultConfig()
config.HostPorts = []string{endpoint}
config.Region = region
config.LogLevel = utils.LogDebug
daxClient, err := dax.New(config)
```

Migration zu DAX Go SDK V2

Dieser Migrationsleitfaden hilft Ihnen bei der Umstellung Ihrer vorhandenen DAX-Go-Anwendungen.

Nutzung von V1 DAX Go SDK

```
package main

import (
    "fmt"
    "os"
```

```
    "github.com/aws/aws-dax-go/dax"
    "github.com/aws/aws-sdk-go/aws"
    "github.com/aws/aws-sdk-go/aws/session"
    "github.com/aws/aws-sdk-go/service/dynamodb"
)

func main() {
    region := "us-west-2"
    endpoint := "dax://my-cluster.l6fzcv.dax-clusters.us-east-1.amazonaws.com"

    // Create session
    sess, err := session.NewSession(&aws.Config{
        Region: aws.String(region),
    })
    if err != nil {
        fmt.Printf("Failed to create session: %v\n", err)
        os.Exit(1)
    }

    // Configure DAX client
    cfg := dax.DefaultConfig()
    cfg.HostPorts = []string{endpoint}
    cfg.Region = region

    // Create DAX client
    daxClient, err := dax.New(cfg)
    if err != nil {
        fmt.Printf("Failed to create DAX client: %v\n", err)
        os.Exit(1)
    }
    defer daxClient.Close() // Don't forget to close the client

    // Create GetItem input
    input := &dynamodb.GetItemInput{
        TableName: aws.String("TryDaxTable"),
        Key: map[string]*dynamodb.AttributeValue{
            "pk": {
                N: aws.String("1"),
            },
            "sk": {
                N: aws.String("1"),
            },
        },
```

```
    }

    // Make the GetItem call
    result, err := daxClient.GetItem(input)
    if err != nil {
        fmt.Printf("Failed to get item: %v\n", err)
        os.Exit(1)
    }

    // Print the result
    fmt.Printf("GetItem succeeded: %+v\n", result)
}
```

Nutzung von V2 DAX Go SDK

```
package main

import (
    "context"
    "fmt"
    "os"

    "github.com/aws/aws-dax-go-v2/dax"
    "github.com/aws/aws-sdk-go-v2/config"
    "github.com/aws/aws-sdk-go-v2/service/dynamodb"
    "github.com/aws/aws-sdk-go-v2/service/dynamodb/types"
    "github.com/aws/aws-sdk-go-v2/feature/dynamodb/attributevalue"
    "github.com/aws/aws-sdk-go-v2/aws"
)

func main() {
    ctx := context.Background()
    region := "us-west-2"
    endpoint := "dax://my-cluster.l6fzcv.dax-clusters.us-east-1.amazonaws.com"

    // Create DAX config
    config := dax.DefaultConfig()
    // Specify Endpoint and Region
    config.HostPorts = []string{endpoint}
    config.Region = region
    // Enabling logging
    config.LogLevel = utils.LogDebug
    // Create DAX client
```

```
    daxClient, err := dax.New(config)
    if err != nil {
        fmt.Printf("Failed to create DAX client: %v\n", err)
        os.Exit(1)
    }
    defer daxClient.Close() // Don't forget to close the client

    // Create key using attributevalue.Marshal for type safety
    pk, err := attributevalue.Marshal(fmt.Sprintf("%s_%d", keyPrefix, i))
    if err != nil {
        return fmt.Errorf("error marshaling pk: %v", err)
    }
    sk, err := attributevalue.Marshal(fmt.Sprintf("%d", j))
    if err != nil {
        return fmt.Errorf("error marshaling sk: %v", err)
    }

    // Create GetItem input
    input := &dynamodb.GetItemInput{
        TableName: aws.String("TryDaxTable"),
        Key: map[string]types.AttributeValue{
             "pk": pk,
             "sk": sk,
        },
    }

    // Make the GetItem call
    result, err := daxClient.GetItem(ctx, input)
    if err != nil {
        fmt.Printf("Failed to get item: %v\n", err)
        os.Exit(1)
    }

    // Print the result
    fmt.Printf("GetItem succeeded: %+v\n", result)
}
```

Weitere Informationen zur API-Nutzung finden Sie unter AWS -Beispiele.

Java und DAX

DAX SDK für Java 2.x ist kompatibel mit AWS SDK für Java 2.x. Es basiert auf Java 8+ und bietet Unterstützung für blockierungsfreie I/O. Informationen zur Verwendung von DAX mit AWS SDK for Java 1.x finden Sie unter. Verwenden von DAX mit AWS SDK for Java 1.x

Verwenden des Clients als Maven-Abhängigkeit

Führen Sie die folgenden Schritte aus, um den Client für das DAX SDK for Java als Abhängigkeit in Ihrer Anwendung zu verwenden.

1. Laden Sie Apache Maven herunter und installieren Sie es. Weitere Informationen finden Sie unter Downloading Apache Maven und Installing Apache Maven.
2. Fügen Sie die Client-Maven-Abhängigkeit der POM-Datei (Project Object Model) Ihrer Anwendung hinzu. In diesem Beispiel ersetzen Sie es *x.x.x* durch die tatsächliche Versionsnummer des Clients.

```
<!--Dependency:-->
<dependencies>
    <dependency>
        <groupId>software.amazon.dax</groupId>
        <artifactId>amazon-dax-client</artifactId>
        <version>x.x.x</version>
    </dependency>
</dependencies>
```

TryDax Beispielcode

Nachdem Sie Ihren Workspace eingerichtet und das DAX-SDK als Abhängigkeit hinzugefügt haben, kopieren Sie TryDax.java in Ihr Projekt.

Führen Sie den Code mit diesem Befehl aus.

```
java -cp classpath TryDax
```

Die Ausgabe sollte in etwa wie folgt aussehen:

```
Creating a DynamoDB client
```

```
Attempting to create table; please wait...
Successfully created table.  Table status: ACTIVE
Writing data to the table...
Writing 10 items for partition key: 1
Writing 10 items for partition key: 2
Writing 10 items for partition key: 3
...

Running GetItem and Query tests...
First iteration of each test will result in cache misses
Next iterations are cache hits

GetItem test - partition key 1-100 and sort keys 1-10
  Total time: 4390.240 ms - Avg time: 4.390 ms
  Total time: 3097.089 ms - Avg time: 3.097 ms
  Total time: 3273.463 ms - Avg time: 3.273 ms
  Total time: 3353.739 ms - Avg time: 3.354 ms
  Total time: 3533.314 ms - Avg time: 3.533 ms
Query test - partition key 1-100 and sort keys between 2 and 9
  Total time: 475.868 ms - Avg time: 4.759 ms
  Total time: 423.333 ms - Avg time: 4.233 ms
  Total time: 460.271 ms - Avg time: 4.603 ms
  Total time: 397.859 ms - Avg time: 3.979 ms
  Total time: 466.644 ms - Avg time: 4.666 ms

Attempting to delete table; please wait...
Successfully deleted table.
```

Beachten Sie die Zeitinformationen – die Anzahl der benötigten Millisekunden für die `GetItem`- und `Query`-Prüfungen. In diesem Fall führten Sie das Programm über den DynamoDB-Endpunkt aus. Jetzt führen Sie das Programm erneut aus, diesmal gegen Ihren DAX-Cluster.

Um den Endpunkt für Ihren DAX-Cluster zu bestimmen, wählen Sie einen der folgenden Schritte aus:

- Wählen Sie in der DynamoDB-Konsole den DAX-Cluster aus. Der Cluster-Endpunkt wird auf der Konsole angezeigt, wie im folgenden Beispiel gezeigt.

  ```
  dax://my-cluster.l6fzcv.dax-clusters.us-east-1.amazonaws.com
  ```

- Geben Sie mit dem AWS CLI den folgenden Befehl ein:

  ```
  aws dax describe-clusters --query "Clusters[*].ClusterDiscoveryEndpoint"
  ```

Die Adresse, der Port und die URL des Clusterendpunkts werden wie im folgenden Beispiel in der Ausgabe angezeigt.

```
{
    "Address": "my-cluster.l6fzcv.dax-clusters.us-east-1.amazonaws.com",
    "Port": 8111,
    "URL": "dax://my-cluster.l6fzcv.dax-clusters.us-east-1.amazonaws.com"
}
```

Führen Sie nun das Programm erneut aus, geben Sie jedoch diesmal die Clusterendpunkt-URL als Befehlszeilenparameter an.

```
java -cp classpath TryDax dax://my-cluster.l6fzcv.dax-clusters.us-east-1.amazonaws.com
```

Sehen Sie sich die Ausgabe an und notieren Sie die Zeitinformationen. Die verstrichene Zeit für `GetItem` und `Query` sollten mit DAX deutlich geringer sein als mit DynamoDB.

SDK-Metriken

Mit dem DAX SDK for Java 2.x können Sie Metriken über die Service-Clients in Ihrer Anwendung sammeln und die Ergebnisse in Amazon CloudWatch analysieren. Weitere Informationen finden Sie unter Aktivieren von SDK-Metriken.

> **Note**
>
> Das DAX-SDK für Java erfasst nur `ApiCallSuccessful`- und `ApiCallDuration`-Metriken.

TryDax.java

```
import java.util.Map;

import software.amazon.awssdk.services.dynamodb.DynamoDbAsyncClient;
import software.amazon.awssdk.services.dynamodb.model.AttributeDefinition;
import software.amazon.awssdk.services.dynamodb.model.AttributeValue;
import software.amazon.awssdk.services.dynamodb.model.BillingMode;
import software.amazon.awssdk.services.dynamodb.model.CreateTableRequest;
import software.amazon.awssdk.services.dynamodb.model.DeleteTableRequest;
```

```
import software.amazon.awssdk.services.dynamodb.model.DescribeTableRequest;
import software.amazon.awssdk.services.dynamodb.model.GetItemRequest;
import software.amazon.awssdk.services.dynamodb.model.KeySchemaElement;
import software.amazon.awssdk.services.dynamodb.model.KeyType;
import software.amazon.awssdk.services.dynamodb.model.PutItemRequest;
import software.amazon.awssdk.services.dynamodb.model.QueryRequest;
import software.amazon.awssdk.services.dynamodb.model.ScalarAttributeType;
import software.amazon.dax.ClusterDaxAsyncClient;
import software.amazon.dax.Configuration;

public class TryDax {
    public static void main(String[] args) throws Exception {
        DynamoDbAsyncClient ddbClient = DynamoDbAsyncClient.builder()
                .build();

        DynamoDbAsyncClient daxClient = null;
        if (args.length >= 1) {
            daxClient = ClusterDaxAsyncClient.builder()
                    .overrideConfiguration(Configuration.builder()
                            .url(args[0]) // e.g. dax://my-cluster.l6fzcv.dax-
clusters.us-east-1.amazonaws.com
                            .build())
                    .build();
        }

        String tableName = "TryDaxTable";

        System.out.println("Creating table...");
        createTable(tableName, ddbClient);

        System.out.println("Populating table...");
        writeData(tableName, ddbClient, 100, 10);

        DynamoDbAsyncClient testClient = null;
        if (daxClient != null) {
            testClient = daxClient;
        } else {
            testClient = ddbClient;
        }

        System.out.println("Running GetItem and Query tests...");
        System.out.println("First iteration of each test will result in cache misses");
        System.out.println("Next iterations are cache hits\n");
```

```
        // GetItem
        getItemTest(tableName, testClient, 100, 10, 5);

        // Query
        queryTest(tableName, testClient, 100, 2, 9, 5);

        System.out.println("Deleting table...");
        deleteTable(tableName, ddbClient);
    }

    private static void createTable(String tableName, DynamoDbAsyncClient client) {
        try {
            System.out.println("Attempting to create table; please wait...");

            client.createTable(CreateTableRequest.builder()
                    .tableName(tableName)
                    .keySchema(KeySchemaElement.builder()
                            .keyType(KeyType.HASH)
                            .attributeName("pk")
                            .build(), KeySchemaElement.builder()
                            .keyType(KeyType.RANGE)
                            .attributeName("sk")
                            .build())
                    .attributeDefinitions(AttributeDefinition.builder()
                            .attributeName("pk")
                            .attributeType(ScalarAttributeType.N)
                            .build(), AttributeDefinition.builder()
                            .attributeName("sk")
                            .attributeType(ScalarAttributeType.N)
                            .build())
                    .billingMode(BillingMode.PAY_PER_REQUEST)
                    .build()).get();
            client.waiter().waitUntilTableExists(DescribeTableRequest.builder()
                    .tableName(tableName)
                    .build()).get();
            System.out.println("Successfully created table.");

        } catch (Exception e) {
            System.err.println("Unable to create table: ");
            e.printStackTrace();
        }
    }

    private static void deleteTable(String tableName, DynamoDbAsyncClient client) {
```

```
        try {
            System.out.println("\nAttempting to delete table; please wait...");
            client.deleteTable(DeleteTableRequest.builder()
                    .tableName(tableName)
                    .build()).get();
            client.waiter().waitUntilTableNotExists(DescribeTableRequest.builder()
                    .tableName(tableName)
                    .build()).get();
            System.out.println("Successfully deleted table.");

        } catch (Exception e) {
            System.err.println("Unable to delete table: ");
            e.printStackTrace();
        }
    }

    private static void writeData(String tableName, DynamoDbAsyncClient client, int
pkmax, int skmax) {
        System.out.println("Writing data to the table...");

        int stringSize = 1000;
        StringBuilder sb = new StringBuilder(stringSize);
        for (int i = 0; i < stringSize; i++) {
            sb.append('X');
        }
        String someData = sb.toString();

        try {
            for (int ipk = 1; ipk <= pkmax; ipk++) {
                System.out.println(("Writing " + skmax + " items for partition key: " +
ipk));
                for (int isk = 1; isk <= skmax; isk++) {
                    client.putItem(PutItemRequest.builder()
                            .tableName(tableName)
                            .item(Map.of("pk", attr(ipk), "sk", attr(isk), "someData",
attr(someData)))
                            .build()).get();
                }
            }
        } catch (Exception e) {
            System.err.println("Unable to write item:");
            e.printStackTrace();
        }
    }
```

```
   private static AttributeValue attr(int n) {
       return AttributeValue.builder().n(String.valueOf(n)).build();
   }

   private static AttributeValue attr(String s) {
       return AttributeValue.builder().s(s).build();
   }

   private static void getItemTest(String tableName, DynamoDbAsyncClient client, int
pk, int sk, int iterations) {
       long startTime, endTime;
       System.out.println("GetItem test - partition key 1-" + pk + " and sort keys 1-"
+ sk);

       for (int i = 0; i < iterations; i++) {
           startTime = System.nanoTime();
           try {
               for (int ipk = 1; ipk <= pk; ipk++) {
                   for (int isk = 1; isk <= sk; isk++) {
                       client.getItem(GetItemRequest.builder()
                               .tableName(tableName)
                               .key(Map.of("pk", attr(ipk), "sk", attr(isk)))
                               .build()).get();
                   }
               }
           } catch (Exception e) {
               System.err.println("Unable to get item:");
               e.printStackTrace();
           }
           endTime = System.nanoTime();
           printTime(startTime, endTime, pk * sk);
       }
   }

   private static void queryTest(String tableName, DynamoDbAsyncClient client, int pk,
int sk1, int sk2, int iterations) {
       long startTime, endTime;
       System.out.println("Query test - partition key 1-" + pk + " and sort keys
between " + sk1 + " and " + sk2);

       for (int i = 0; i < iterations; i++) {
           startTime = System.nanoTime();
           for (int ipk = 1; ipk <= pk; ipk++) {
```

```
                try {
                    // Pagination API for Query.
                    client.queryPaginator(QueryRequest.builder()
                            .tableName(tableName)
                            .keyConditionExpression("pk = :pkval and sk between :skval1
 and :skval2")
                            .expressionAttributeValues(Map.of(":pkval", attr(ipk),
 ":skval1", attr(sk1), ":skval2", attr(sk2)))
                            .build()).items().subscribe((item) -> {
                    }).get();
                } catch (Exception e) {
                    System.err.println("Unable to query table:");
                    e.printStackTrace();
                }
            }
            endTime = System.nanoTime();
            printTime(startTime, endTime, pk);
        }
    }

    private static void printTime(long startTime, long endTime, int iterations) {
        System.out.format("\tTotal time: %.3f ms - ", (endTime - startTime) /
 (1000000.0));
        System.out.format("Avg time: %.3f ms\n", (endTime - startTime) / (iterations *
 1000000.0));
    }
}
```

.NET und DAX

Gehen Sie wie folgt vor, um das .NET-Beispiel auf Ihrer Amazon-EC2-Instance auszuführen.

> **Note**
>
> Dieses Tutorial verwendet das.NET 9 SDK. Sie zeigt, wie Sie ein Programm in der Standard-Amazon VPC ausführen können, um auf den Amazon-DynamoDB-Accelerator-(DAX)-Cluster zuzugreifen. Es funktioniert mit dem AWS SDK v4 für .NET. Einzelheiten zu Änderungen in V4 und Informationen zur Migration finden Sie unter Migration auf Version 4 des AWS SDK for .NET. Wenn Sie möchten, können Sie die verwenden, AWS Toolkit for Visual Studio um eine.NET-Anwendung zu schreiben und sie in Ihrer VPC bereitzustellen.

Weitere Informationen finden Sie unter Erstellen und Bereitstellen von Elastic-Beanstalk-Anwendungen in .NET mit AWS -Toolkit for Visual Studio im AWS Elastic Beanstalk - Entwicklerhandbuch.

So führen Sie das .NET-Beispiel für DAX aus

1. Gehen Sie zur Microsoft-Downloadseite und laden Sie das neueste.NET 9-SDK für Linux herunter. Der Name der heruntergeladenen Datei lautet `dotnet-sdk-N.N.N-linux-x64.tar.gz`.
2. Extrahieren Sie die SDK-Dateien.

   ```
   mkdir dotnet
   tar zxvf dotnet-sdk-N.N.N-linux-x64.tar.gz -C dotnet
   ```

   Ersetzen Sie *N.N.N* durch die tatsächliche Versionsnummer des .NET SDK (z. B.: `9.0.305`).
3. Überprüfen Sie die Installation.

   ```
   alias dotnet=$HOME/dotnet/dotnet
   dotnet --version
   ```

   Damit sollte die Versionsnummer des .NET SDK ausgegeben werden.

   > **Note**
   >
   > Statt der Versionsnummer erhalten Sie möglicherweise die folgende Fehlermeldung: error: libunwind.so.8: cannot open shared object file: No such file or directory (Fehler: libunwind.so.8: Die freigegebene Datei kann nicht geöffnet werden: Datei oder Verzeichnis nicht vorhanden)
   > Um den Fehler zu beheben, installieren Sie das `libunwind`-Paket.
   >
   > ```
   > sudo yum install -y libunwind
   > ```
   >
   > Anschließend sollte es möglich sein, den `dotnet --version`-Befehl fehlerfrei auszuführen.
4. Erstellen Sie ein neues .NET-Projekt.

```
dotnet new console -o myApp
```

Die Durchführung eines one-time-only Setups dauert einige Minuten. Nachdem sie fertig ist, führen Sie das Beispielprojekt aus.

```
dotnet run --project myApp
```

Sie sollten die folgende Meldung erhalten: `Hello World!`

5. Die Datei `myApp/myApp.csproj` enthält Metadaten über Ihr Projekt. Um den DAX-Client in Ihrer Anwendung nutzen zu können, ändern Sie die Datei, sodass sie wie folgt aussieht.

```
<Project Sdk="Microsoft.NET.Sdk">
    <PropertyGroup>
        <OutputType>Exe</OutputType>
        <TargetFramework>net9.0</TargetFramework>
    </PropertyGroup>
    <ItemGroup>
        <PackageReference Include="AWSSDK.DAX.Client" Version="*" />
    </ItemGroup>
</Project>
```

6. Downloaden Sie den Quellcode des Beispielprogramms (`.zip`-Datei):

```
wget http://docs.aws.amazon.com/amazondynamodb/latest/developerguide/samples/
TryDax.zip
```

Wenn der Download abgeschlossen ist, extrahieren Sie die Quelldateien.

```
unzip TryDax.zip
```

7. Führen Sie nun nacheinander die Beispielprogramme von dotNet aus. Kopieren Sie für jedes Programm seinen Inhalt in die Datei `myApp/Program.cs` und führen Sie dann das `MyApp`-Projekt aus.

Führen Sie die folgenden .NET-Programme aus. Das erste Programm erstellt eine DynamoDB-Tabelle mit dem Namen `TryDaxTable`. Das zweite Programm schreibt Daten in die Tabelle.

```
cp TryDax/dotNet/01-CreateTable.cs myApp/Program.cs
```

```
dotnet run --project myApp

cp TryDax/dotNet/02-Write-Data.cs myApp/Program.cs
dotnet run --project myApp
```

8. Führen Sie jetzt einige Programme zur Durchführung von `GetItem`-, `Query`- und `Scan`-Operationen auf dem DAX-Cluster aus. Um den Endpunkt für Ihren DAX-Cluster zu bestimmen, wählen Sie einen der folgenden Schritte aus:

   - Using the DynamoDB console (Verwenden der DynamoDB-Konsole) — Wählen Sie Ihren DAX-Cluster aus. Der Cluster-Endpunkt wird auf der Konsole angezeigt, wie im folgenden Beispiel gezeigt.

     ```
     dax://my-cluster.l6fzcv.dax-clusters.us-east-1.amazonaws.com
     ```

   - Verwenden Sie AWS CLI— Geben Sie den folgenden Befehl ein.

     ```
     aws dax describe-clusters --query "Clusters[*].ClusterDiscoveryEndpoint"
     ```

     Der Clusterendpunkt wird wie im folgenden Beispiel in der Ausgabe angezeigt.

     ```
     {
         "Address": "my-cluster.l6fzcv.dax-clusters.us-east-1.amazonaws.com",
         "Port": 8111,
         "URL": "dax://my-cluster.l6fzcv.dax-clusters.us-east-1.amazonaws.com"
     }
     ```

   Führen Sie jetzt die folgenden Programme aus und geben Sie den Cluster-Endpunkt als Befehlszeilenparameter an. (Ersetzen Sie den Beispiel-Endpunkt durch den tatsächlichen DAX-Cluster-Endpunkt.)

   ```
   cp TryDax/dotNet/03-GetItem-Test.cs myApp/Program.cs
   dotnet run --project myApp dax://my-cluster.l6fzcv.dax-clusters.us-
   east-1.amazonaws.com

   cp TryDax/dotNet/04-Query-Test.cs myApp/Program.cs
   dotnet run --project myApp dax://my-cluster.l6fzcv.dax-clusters.us-
   east-1.amazonaws.com

   cp TryDax/dotNet/05-Scan-Test.cs myApp/Program.cs
   ```

```
dotnet run --project myApp dax://my-cluster.l6fzcv.dax-clusters.us-
east-1.amazonaws.com
```

Beachten Sie die Zeitinformationen – die Anzahl der benötigten Millisekunden für den `GetItem`-, `Query`- und `Scan`-Test.

9. Führen Sie das folgende .NET-Programm aus, um `TryDaxTable` zu löschen.

```
cp TryDax/dotNet/06-DeleteTable.cs myApp/Program.cs
dotnet run --project myApp
```

Weitere Informationen zu diesen Programmen finden Sie in folgenden Abschnitten:

- 0-1 CreateTable .cs
- 02-Write-Data.cs
- 03- -Test.cs GetItem
- 04-Query-Test.cs
- 05-Scan-Test.cs
- 0-6. DeleteTable cs

0-1 CreateTable .cs

Das Programm `01-CreateTable.cs` erstellt eine Tabelle (`TryDaxTable`). Die restlichen .NET-Programme in diesem Abschnitt hängen von dieser Tabelle ab.

```
using System;
using System.Collections.Generic;
using System.Threading.Tasks;
using Amazon.DynamoDBv2;
using Amazon.DynamoDBv2.Model;

namespace ClientTest
{
    class Program
    {
        public static async Task Main(string[] args)
        {
            AmazonDynamoDBClient client = new AmazonDynamoDBClient();
```

```
            var tableName = "TryDaxTable";

            var request = new CreateTableRequest()
            {
                TableName = tableName,
                KeySchema = new List<KeySchemaElement>()
                {
                    new KeySchemaElement{ AttributeName = "pk",KeyType = "HASH"},
                    new KeySchemaElement{ AttributeName = "sk",KeyType = "RANGE"}
                },
                AttributeDefinitions = new List<AttributeDefinition>() {
                    new AttributeDefinition{ AttributeName = "pk",AttributeType = "N"},
                    new AttributeDefinition{ AttributeName = "sk",AttributeType  = "N"}
                },
                ProvisionedThroughput = new ProvisionedThroughput()
                {
                    ReadCapacityUnits = 10,
                    WriteCapacityUnits = 10
                }
            };

            var response = await client.CreateTableAsync(request);

            Console.WriteLine("Hit <enter> to continue...");
            Console.ReadLine();
        }
    }
}
```

02-Write-Data.cs

Das Programm `02-Write-Data.cs` schreibt Testdaten in `TryDaxTable`.

```
using System;
using System.Collections.Generic;
using System.Threading.Tasks;
using Amazon.DynamoDBv2;
using Amazon.DynamoDBv2.Model;

namespace ClientTest
{
```

```
    class Program
    {
        public static async Task Main(string[] args)
        {
            AmazonDynamoDBClient client = new AmazonDynamoDBClient();

            var tableName = "TryDaxTable";

            string someData = new string('X', 1000);
            var pkmax = 10;
            var skmax = 10;

            for (var ipk = 1; ipk <= pkmax; ipk++)
            {
                Console.WriteLine($"Writing {skmax} items for partition key: {ipk}");
                for (var isk = 1; isk <= skmax; isk++)
                {
                    var request = new PutItemRequest()
                    {
                        TableName = tableName,
                        Item = new Dictionary<string, AttributeValue>()
                       {
                            { "pk", new AttributeValue{N = ipk.ToString() } },
                            { "sk", new AttributeValue{N = isk.ToString() } },
                            { "someData", new AttributeValue{S = someData } }
                       }
                    };

                    var response = await client.PutItemAsync(request);
                }
            }

            Console.WriteLine("Hit <enter> to continue...");
            Console.ReadLine();
        }
    }
}
```

03- -Test.cs GetItem

Das Programm `03-GetItem-Test.cs` führt `GetItem`-Operationen für `TryDaxTable` aus.

```
using System;
using System.Collections.Generic;
using System.Threading.Tasks;
using Amazon.DAX;
using Amazon.DynamoDBv2.Model;
using Amazon.Runtime;

namespace ClientTest
{
    class Program
    {
        public static async Task Main(string[] args)
        {
            string endpointUri = args[0];
            Console.WriteLine($"Using DAX client - endpointUri={endpointUri}");

            var clientConfig = new DaxClientConfig(endpointUri)
            {
                AwsCredentials = FallbackCredentialsFactory.GetCredentials()
            };
            var client = new ClusterDaxClient(clientConfig);

            var tableName = "TryDaxTable";

            var pk = 1;
            var sk = 10;
            var iterations = 5;

            var startTime = System.DateTime.Now;

            for (var i = 0; i < iterations; i++)
            {
                for (var ipk = 1; ipk <= pk; ipk++)
                {
                    for (var isk = 1; isk <= sk; isk++)
                    {
                        var request = new GetItemRequest()
                        {
                            TableName = tableName,
                            Key = new Dictionary<string, AttributeValue>() {
                            {"pk", new AttributeValue {N = ipk.ToString()} },
                            {"sk", new AttributeValue {N = isk.ToString() } }
                        }
                        };
```

```
                        var response = await client.GetItemAsync(request);
                        Console.WriteLine($"GetItem succeeded for pk: {ipk},sk:
 {isk}");
                    }
                }
            }

            var endTime = DateTime.Now;
            TimeSpan timeSpan = endTime - startTime;
            Console.WriteLine($"Total time: {timeSpan.TotalMilliseconds}
 milliseconds");

            Console.WriteLine("Hit <enter> to continue...");
            Console.ReadLine();
        }
    }
}
```

04-Query-Test.cs

Das Programm `04-Query-Test.cs` führt `Query`-Operationen für `TryDaxTable` aus.

```
using System;
using System.Collections.Generic;
using System.Threading.Tasks;
using Amazon.Runtime;
using Amazon.DAX;
using Amazon.DynamoDBv2.Model;

namespace ClientTest
{
    class Program
    {
        public static async Task Main(string[] args)
        {
            string endpointUri = args[0];
            Console.WriteLine($"Using DAX client - endpointUri={endpointUri}");


            var clientConfig = new DaxClientConfig(endpointUri)
            {
                AwsCredentials = FallbackCredentialsFactory.GetCredentials()
```

```
            };
            var client = new ClusterDaxClient(clientConfig);

            var tableName = "TryDaxTable";

            var pk = 5;
            var sk1 = 2;
            var sk2 = 9;
            var iterations = 5;

            var startTime = DateTime.Now;

            for (var i = 0; i < iterations; i++)
            {
                var request = new QueryRequest()
                {
                    TableName = tableName,
                    KeyConditionExpression = "pk = :pkval and sk between :skval1
 and :skval2",
                    ExpressionAttributeValues = new Dictionary<string,
 AttributeValue>() {
                            {":pkval", new AttributeValue {N = pk.ToString()} },
                            {":skval1", new AttributeValue {N = sk1.ToString()} },
                            {":skval2", new AttributeValue {N = sk2.ToString()} }
                    }
                };
                var response = await client.QueryAsync(request);
                Console.WriteLine($"{i}: Query succeeded");

            }

            var endTime = DateTime.Now;
            TimeSpan timeSpan = endTime - startTime;
            Console.WriteLine($"Total time: {timeSpan.TotalMilliseconds}
 milliseconds");

            Console.WriteLine("Hit <enter> to continue...");
            Console.ReadLine();
        }
    }
}
```

05-Scan-Test.cs

Das Programm `05-Scan-Test.cs` führt `Scan`-Operationen für `TryDaxTable` aus.

```
using System;
using System.Threading.Tasks;
using Amazon.Runtime;
using Amazon.DAX;
using Amazon.DynamoDBv2.Model;

namespace ClientTest
{
    class Program
    {
        public static async Task Main(string[] args)
        {
            string endpointUri = args[0];
            Console.WriteLine($"Using DAX client - endpointUri={endpointUri}");

            var clientConfig = new DaxClientConfig(endpointUri)
            {
                AwsCredentials = FallbackCredentialsFactory.GetCredentials()
            };
            var client = new ClusterDaxClient(clientConfig);

            var tableName = "TryDaxTable";

            var iterations = 5;

            var startTime = DateTime.Now;

            for (var i = 0; i < iterations; i++)
            {
                var request = new ScanRequest()
                {
                    TableName = tableName
                };
                var response = await client.ScanAsync(request);
                Console.WriteLine($"{i}: Scan succeeded");
            }

            var endTime = DateTime.Now;
            TimeSpan timeSpan = endTime - startTime;
```

```
            Console.WriteLine($"Total time: {timeSpan.TotalMilliseconds}
 milliseconds");

            Console.WriteLine("Hit <enter> to continue...");
            Console.ReadLine();
        }
    }
}
```

0-6. DeleteTable cs

Das `06-DeleteTable.cs`-Programm löscht `TryDaxTable`. Führen Sie dieses Programm aus, sobald Sie mit dem Testen fertig sind.

```
using System;
using System.Threading.Tasks;
using Amazon.DynamoDBv2.Model;
using Amazon.DynamoDBv2;

namespace ClientTest
{
    class Program
    {
        public static async Task Main(string[] args)
        {
            AmazonDynamoDBClient client = new AmazonDynamoDBClient();

            var tableName = "TryDaxTable";

            var request = new DeleteTableRequest()
            {
                TableName = tableName
            };

            var response = await client.DeleteTableAsync(request);

            Console.WriteLine("Hit <enter> to continue...");
            Console.ReadLine();
        }
    }
}
```

Python und DAX

Um die Python-Beispielanwendung auf Ihrer Amazon-EC2-Instance auszuführen, gehen Sie wie folgt vor.

So führen Sie das Python-Beispiel für DAX aus

1. Installieren Sie den DAX-Python-Client mit dem `pip`-Dienstprogramm.

   ```
   pip install amazon-dax-client
   ```

2. Downloaden Sie den Quellcode des Beispielprogramms (`.zip`-Datei):

   ```
   wget http://docs.aws.amazon.com/amazondynamodb/latest/developerguide/samples/
   TryDax.zip
   ```

   Wenn der Download abgeschlossen ist, extrahieren Sie die Quelldateien.

   ```
   unzip TryDax.zip
   ```

3. Führen Sie die folgenden Python-Programme aus. Das erste Programm erstellt eine Amazon-DynamoDB-Tabelle mit dem Namen `TryDaxTable`. Das zweite Programm schreibt Daten in die Tabelle.

   ```
   python 01-create-table.py
   python 02-write-data.py
   ```

4. Führen Sie die folgenden Python-Programme aus.

   ```
   python 03-getitem-test.py
   python 04-query-test.py
   python 05-scan-test.py
   ```

   Beachten Sie die Zeitinformationen – die Anzahl der benötigten Millisekunden für den `GetItem`-, `Query`- und `Scan`-Test.

5. Im vorherigen Schritt haben Sie die Programme für den DynamoDB-Endpunkt ausgeführt. Führen Sie die Programme jetzt erneut aus. Dieses Mal werden die `GetItem`-, `Query`- und `Scan`-Operationen aber vom DAX-Cluster verarbeitet.

Um den Endpunkt für Ihren DAX-Cluster zu bestimmen, wählen Sie einen der folgenden Schritte aus:

- Using the DynamoDB console (Verwenden der DynamoDB-Konsole) — Wählen Sie Ihren DAX-Cluster aus. Der Cluster-Endpunkt wird auf der Konsole angezeigt, wie im folgenden Beispiel gezeigt.

  ```
  dax://my-cluster.l6fzcv.dax-clusters.us-east-1.amazonaws.com
  ```

- Verwenden Sie AWS CLI— Geben Sie den folgenden Befehl ein.

  ```
  aws dax describe-clusters --query "Clusters[*].ClusterDiscoveryEndpoint"
  ```

  Der Cluster-Endpunkt wird wie in diesem Beispiel in der Ausgabe angezeigt.

  ```
  {
      "Address": "my-cluster.l6fzcv.dax-clusters.us-east-1.amazonaws.com",
      "Port": 8111,
      "URL": "dax://my-cluster.l6fzcv.dax-clusters.us-east-1.amazonaws.com"
  }
  ```

Führen Sie jetzt die Programme erneut aus. Geben Sie dieses Mal jedoch den Cluster-Endpunkt als Befehlszeilenparameter an.

```
python 03-getitem-test.py dax://my-cluster.l6fzcv.dax-clusters.us-
east-1.amazonaws.com
python 04-query-test.py dax://my-cluster.l6fzcv.dax-clusters.us-
east-1.amazonaws.com
python 05-scan-test.py dax://my-cluster.l6fzcv.dax-clusters.us-east-1.amazonaws.com
```

Sehen Sie sich den Rest der Ausgabe an und notieren Sie die Zeitinformationen. Die verstrichene Zeit sollte für `GetItem`, `Query` und `Scan` mit DAX deutlich kürzer sein als mit DynamoDB.

6. Führen Sie das folgende Python-Programm aus, um `TryDaxTable` zu löschen:

   ```
   python 06-delete-table.py
   ```

Weitere Informationen zu diesen Programmen finden Sie in folgenden Abschnitten:

- 01-create-table.py
- 02-write-data.py
- 03-getitem-test.py
- 04-query-test.py
- 05-scan-test.py
- 06-delete-table.py

01-create-table.py

Das Programm `01-create-table.py` erstellt eine Tabelle (`TryDaxTable`). Die restlichen Python-Programme in diesem Abschnitt hängen von dieser Tabelle ab.

```
import boto3


def create_dax_table(dyn_resource=None):
    """
    Creates a DynamoDB table.

    :param dyn_resource: Either a Boto3 or DAX resource.
    :return: The newly created table.
    """
    if dyn_resource is None:
        dyn_resource = boto3.resource("dynamodb")

    table_name = "TryDaxTable"
    params = {
        "TableName": table_name,
        "KeySchema": [
            {"AttributeName": "partition_key", "KeyType": "HASH"},
            {"AttributeName": "sort_key", "KeyType": "RANGE"},
        ],
        "AttributeDefinitions": [
            {"AttributeName": "partition_key", "AttributeType": "N"},
            {"AttributeName": "sort_key", "AttributeType": "N"},
        ],
        "BillingMode": "PAY_PER_REQUEST",
    }
    table = dyn_resource.create_table(**params)
```

```
    print(f"Creating {table_name}...")
    table.wait_until_exists()
    return table


if __name__ == "__main__":
    dax_table = create_dax_table()
    print(f"Created table.")
```

02-write-data.py

Das Programm `02-write-data.py` schreibt Testdaten in `TryDaxTable`.

```
import boto3


def write_data_to_dax_table(key_count, item_size, dyn_resource=None):
    """
    Writes test data to the demonstration table.

    :param key_count: The number of partition and sort keys to use to populate the
                      table. The total number of items is key_count * key_count.
    :param item_size: The size of non-key data for each test item.
    :param dyn_resource: Either a Boto3 or DAX resource.
    """
    if dyn_resource is None:
        dyn_resource = boto3.resource("dynamodb")

    table = dyn_resource.Table("TryDaxTable")
    some_data = "X" * item_size

    for partition_key in range(1, key_count + 1):
        for sort_key in range(1, key_count + 1):
            table.put_item(
                Item={
                    "partition_key": partition_key,
                    "sort_key": sort_key,
                    "some_data": some_data,
                }
            )
            print(f"Put item ({partition_key}, {sort_key}) succeeded.")


if __name__ == "__main__":
```

```
    write_key_count = 10
    write_item_size = 1000
    print(
        f"Writing {write_key_count*write_key_count} items to the table. "
        f"Each item is {write_item_size} characters."
    )
    write_data_to_dax_table(write_key_count, write_item_size)
```

03-getitem-test.py

Das Programm `03-getitem-test.py` führt `GetItem`-Operationen für `TryDaxTable` aus. Dieses Beispiel wird für die Region eu-west-1 gegeben.

```
import argparse
import sys
import time
import amazondax
import boto3


def get_item_test(key_count, iterations, dyn_resource=None):
    """
    Gets items from the table a specified number of times. The time before the
    first iteration and the time after the last iteration are both captured
    and reported.

    :param key_count: The number of items to get from the table in each iteration.
    :param iterations: The number of iterations to run.
    :param dyn_resource: Either a Boto3 or DAX resource.
    :return: The start and end times of the test.
    """
    if dyn_resource is None:
        dyn_resource = boto3.resource('dynamodb')

    table = dyn_resource.Table('TryDaxTable')
    start = time.perf_counter()
    for _ in range(iterations):
        for partition_key in range(1, key_count + 1):
            for sort_key in range(1, key_count + 1):
                table.get_item(Key={
                    'partition_key': partition_key,
                    'sort_key': sort_key
```

```
                })
                print('.', end='')
                sys.stdout.flush()
    print()
    end = time.perf_counter()
    return start, end


if __name__ == '__main__':
    parser = argparse.ArgumentParser()
    parser.add_argument(
        'endpoint_url', nargs='?',
        help="When specified, the DAX cluster endpoint. Otherwise, DAX is not used.")
    args = parser.parse_args()

    test_key_count = 10
    test_iterations = 50
    if args.endpoint_url:
        print(f"Getting each item from the table {test_iterations} times, "
              f"using the DAX client.")
        # Use a with statement so the DAX client closes the cluster after completion.
        with amazondax.AmazonDaxClient.resource(endpoint_url=args.endpoint_url,
 region_name='eu-west-1') as dax:
            test_start, test_end = get_item_test(
                test_key_count, test_iterations, dyn_resource=dax)
    else:
        print(f"Getting each item from the table {test_iterations} times, "
              f"using the Boto3 client.")
        test_start, test_end = get_item_test(
            test_key_count, test_iterations)
    print(f"Total time: {test_end - test_start:.4f} sec. Average time: "
          f"{(test_end - test_start)/ test_iterations}.")
```

04-query-test.py

Das Programm `04-query-test.py` führt `Query`-Operationen für `TryDaxTable` aus.

```
import argparse
import time
import sys
import amazondax
import boto3
from boto3.dynamodb.conditions import Key
```

```
def query_test(partition_key, sort_keys, iterations, dyn_resource=None):
    """
    Queries the table a specified number of times. The time before the
    first iteration and the time after the last iteration are both captured
    and reported.

    :param partition_key: The partition key value to use in the query. The query
                          returns items that have partition keys equal to this value.
    :param sort_keys: The range of sort key values for the query. The query returns
                      items that have sort key values between these two values.
    :param iterations: The number of iterations to run.
    :param dyn_resource: Either a Boto3 or DAX resource.
    :return: The start and end times of the test.
    """
    if dyn_resource is None:
        dyn_resource = boto3.resource("dynamodb")

    table = dyn_resource.Table("TryDaxTable")
    key_condition_expression = Key("partition_key").eq(partition_key) & Key(
        "sort_key"
    ).between(*sort_keys)

    start = time.perf_counter()
    for _ in range(iterations):
        table.query(KeyConditionExpression=key_condition_expression)
        print(".", end="")
        sys.stdout.flush()
    print()
    end = time.perf_counter()
    return start, end


if __name__ == "__main__":
    # pylint: disable=not-context-manager
    parser = argparse.ArgumentParser()
    parser.add_argument(
        "endpoint_url",
        nargs="?",
        help="When specified, the DAX cluster endpoint. Otherwise, DAX is not used.",
    )
    args = parser.parse_args()
```

```
    test_partition_key = 5
    test_sort_keys = (2, 9)
    test_iterations = 100
    if args.endpoint_url:
        print(f"Querying the table {test_iterations} times, using the DAX client.")
        # Use a with statement so the DAX client closes the cluster after completion.
        with amazondax.AmazonDaxClient.resource(endpoint_url=args.endpoint_url) as dax:
            test_start, test_end = query_test(
                test_partition_key, test_sort_keys, test_iterations, dyn_resource=dax
            )
    else:
        print(f"Querying the table {test_iterations} times, using the Boto3 client.")
        test_start, test_end = query_test(
            test_partition_key, test_sort_keys, test_iterations
        )

    print(
        f"Total time: {test_end - test_start:.4f} sec. Average time: "
        f"{(test_end - test_start)/test_iterations}."
    )
```

05-scan-test.py

Das Programm `05-scan-test.py` führt `Scan`-Operationen für `TryDaxTable` aus.

```
import argparse
import time
import sys
import amazondax
import boto3


def scan_test(iterations, dyn_resource=None):
    """
    Scans the table a specified number of times. The time before the
    first iteration and the time after the last iteration are both captured
    and reported.

    :param iterations: The number of iterations to run.
    :param dyn_resource: Either a Boto3 or DAX resource.
    :return: The start and end times of the test.
    """
    if dyn_resource is None:
        dyn_resource = boto3.resource("dynamodb")
```

```
    table = dyn_resource.Table("TryDaxTable")
    start = time.perf_counter()
    for _ in range(iterations):
        table.scan()
        print(".", end="")
        sys.stdout.flush()
    print()
    end = time.perf_counter()
    return start, end


if __name__ == "__main__":
    # pylint: disable=not-context-manager
    parser = argparse.ArgumentParser()
    parser.add_argument(
        "endpoint_url",
        nargs="?",
        help="When specified, the DAX cluster endpoint. Otherwise, DAX is not used.",
    )
    args = parser.parse_args()

    test_iterations = 100
    if args.endpoint_url:
        print(f"Scanning the table {test_iterations} times, using the DAX client.")
        # Use a with statement so the DAX client closes the cluster after completion.
        with amazondax.AmazonDaxClient.resource(endpoint_url=args.endpoint_url) as dax:
            test_start, test_end = scan_test(test_iterations, dyn_resource=dax)
    else:
        print(f"Scanning the table {test_iterations} times, using the Boto3 client.")
        test_start, test_end = scan_test(test_iterations)
    print(
        f"Total time: {test_end - test_start:.4f} sec. Average time: "
        f"{(test_end - test_start)/test_iterations}."
    )
```

06-delete-table.py

Das `06-delete-table.py`-Programm löscht `TryDaxTable`. Führen Sie dieses Programm aus, sobald Sie mit dem Testen der Amazon-DynamoDB-Accelerator-(DAX)-Funktionalität fertig sind.

```
import boto3
```

```
def delete_dax_table(dyn_resource=None):
    """
    Deletes the demonstration table.

    :param dyn_resource: Either a Boto3 or DAX resource.
    """
    if dyn_resource is None:
        dyn_resource = boto3.resource("dynamodb")

    table = dyn_resource.Table("TryDaxTable")
    table.delete()

    print(f"Deleting {table.name}...")
    table.wait_until_not_exists()


if __name__ == "__main__":
    delete_dax_table()
    print("Table deleted!")
```

Ändern einer vorhandenen Anwendung für die Verwendung von DAX

Wenn Sie bereits über eine Java-Anwendung verfügen, die Amazon DynamoDB verwendet, müssen Sie sie so ändern, dass sie auf den DynamoDB-Accelerator-(DAX)-Cluster zugreifen kann. Sie müssen nicht die gesamte Anwendung neu schreiben, da der DAX-Java-Client dem DynamoDB-Low-Level-Client ähnelt, der im AWS SDK for Java 2.x enthalten ist. Weitere Informationen finden Sie unter Arbeiten mit Elementen in DynamoDB.

> **Note**
>
> In diesem Beispiel wird AWS SDK for Java 2.x verwendet. Weitere Informationen zur Legacy-SDK für Java 1.x-Version finden Sie unter Verwenden einer vorhandenen SDK for Java 1.x Anwendung zur Nutzung von DAX.

Zum Ändern des Programms ersetzen Sie den DynamoDB-Client durch einen DAX-Client.

```
Region region = Region.US_EAST_1;

// Create an asynchronous DynamoDB client
```

```
DynamoDbAsyncClient client = DynamoDbAsyncClient.builder()
                .region(region)
                .build();

// Create an asynchronous DAX client
DynamoDbAsyncClient client = ClusterDaxAsyncClient.builder()
                .overrideConfiguration(Configuration.builder()
                    .url(<cluster url>) // for example, "dax://my-cluster.l6fzcv.dax-
clusters.us-east-1.amazonaws.com"
                    .region(region)
                    .addMetricPublisher(cloudWatchMetricsPub) // optionally enable SDK
 metric collection
                    .build())
                .build();
```

Sie können auch die High-Level-Bibliothek verwenden, die Teil des AWS SDK for Java 2.x ist, und den DynamoDB-Client durch einen DAX-Client ersetzen.

```
Region region = Region.US_EAST_1;
DynamoDbAsyncClient dax = ClusterDaxAsyncClient.builder()
        .overrideConfiguration(Configuration.builder()
            .url(<cluster url>) // for example, "dax://my-cluster.l6fzcv.dax-
clusters.us-east-1.amazonaws.com"
            .region(region)
            .build())
        .build();

DynamoDbEnhancedAsyncClient enhancedClient = DynamoDbEnhancedAsyncClient.builder()
        .dynamoDbClient(dax)
        .build();
```

Weitere Informationen finden Sie unter Mapping-Elementen in DynamoDB-Tabellen.

Verwalten von DAX-Clustern

In diesem Abschnitt werden einige der gängigsten Verwaltungsaufgaben für Amazon-DynamoDB-Accelerator-(DAX)-Cluster erörtert.

Themen

- IAM-Berechtigungen zum Verwalten eines DAX-Clusters
- Skalieren eines DAX-Clusters

- Anpassen der DAX-Cluster-Einstellungen
- Konfigurieren der TTL-Einstellungen
- Unterstützung von Markierungen für DAX
- AWS CloudTrail Integration
- Löschen eines DAX-Clusters

IAM-Berechtigungen zum Verwalten eines DAX-Clusters

Wenn Sie einen DAX-Cluster mithilfe von AWS-Managementkonsole oder AWS Command Line Interface (AWS CLI) verwalten, empfehlen wir dringend, den Umfang der Aktionen, die Benutzer ausführen können, einzuschränken. So können Sie die Risiken minimieren und der Regel der geringsten Rechte folgen.

Die folgende Diskussion konzentriert sich auf die Zugriffskontrolle für das DAX-Management APIs. Weitere Informationen finden Sie unter Amazon DynamoDB Accelerator in Amazon-DynamoDB-API-Referenz.

> **Note**
>
> Ausführlichere Informationen zur Verwaltung von AWS Identity and Access Management (IAM-) Berechtigungen finden Sie im Folgenden:
>
> - IAM und Erstellen von DAX-Cluster: Erstellen eines DAX-Clusters.
> - IAM und DAX-Operationen auf Dateneben: DAX-Zugriffskontrolle.

Für das DAX-Management APIs können Sie API-Aktionen nicht auf eine bestimmte Ressource beschränken. Das `Resource` -Element muss auf `"*"` gesetzt sein. Dies ist anders als bei DAX-API-Operationen auf Datenebene, wie z. B. `GetItem`, `Query` und `Scan`. Operationen auf Datenebene werden über den DAX-Client bereitgestellt. Diese Operationen können auf bestimmte Ressourcen begrenzt werden.

Schauen Sie sich zur Veranschaulichung das folgende IAM-Richtliniendokument an.

JSON

```
{
```

```
    "Version":"2012-10-17",
    "Statement": [
        {
            "Action": [
                "dax:*"
            ],
            "Effect": "Allow",
            "Resource": [
                "arn:aws:dax:us-west-2:123456789012:cache/DAXCluster01"
            ]
        }
    ]
}
```

Angenommen, die Absicht dieser Richtlinie besteht darin, DAX-Verwaltungs-API-Aufrufe für den Cluster `DAXCluster01` – und nur für diesen Cluster – zuzulassen.

Nehmen wir nun an, ein Benutzer gibt den folgenden AWS CLI Befehl aus.

```
aws dax describe-clusters
```

Dieser Befehl schlägt mit der Ausnahme Nicht autorisiert fehl, da der zugrunde liegende API-Aufruf `DescribeClusters` nicht auf einen bestimmten Cluster beschränkt werden kann. Die Richtlinie ist syntaktisch zwar gültig, dennoch schlägt der Befehl fehl, da das `Resource`-Element auf `"*"` festgelegt sein muss. Wenn der Benutzer jedoch ein Programm ausführt, das DAX-Datenebenenaufrufe (z. B. `GetItem` oder `Query`) an `DAXCluster01` sendet, sind diese Aufrufe erfolgreich. Dies liegt daran, dass die DAX-Datenebene auf bestimmte Ressourcen beschränkt werden APIs kann (in diesem Fall`DAXCluster01`).

Wenn Sie eine einzige umfassende IAM-Richtlinie verfassen möchten, die sowohl das DAX-Management APIs als auch die DAX-Datenebene umfasst APIs, empfehlen wir Ihnen, zwei unterschiedliche Aussagen in das Richtliniendokument aufzunehmen. Eine dieser Aussagen sollte sich auf die DAX-Datenebene beziehen APIs, während sich die andere Aussage auf das DAX-Management bezieht. APIs

Es folgt eine Beispielrichtlinie, die diesen Ansatz veranschaulicht: Sie sehen, wie die `DAXDataAPIs`-Anweisung auf die Ressource `DAXCluster01` begrenzt ist; die Ressource für `DAXManagementAPIs` muss aber `"*"` lauten. Die in den Anweisungen gezeigten Aktionen dienen lediglich der Veranschaulichung. Sie können sie nach Bedarf für Ihre Anwendung anpassen.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "DAXDataAPIs",
            "Action": [
                "dax:GetItem",
                "dax:BatchGetItem",
                "dax:Query",
                "dax:Scan",
                "dax:PutItem",
                "dax:UpdateItem",
                "dax:DeleteItem",
                "dax:BatchWriteItem"
            ],
            "Effect": "Allow",
            "Resource": [
                "arn:aws:dax:us-west-2:123456789012:cache/DAXCluster01"
            ]},
            {
            "Sid": "DAXManagementAPIs",
            "Action": [
                "dax:CreateParameterGroup",
                "dax:CreateSubnetGroup",
                "dax:DecreaseReplicationFactor",
                "dax:DeleteCluster",
                "dax:DeleteParameterGroup",
                "dax:DeleteSubnetGroup",
                "dax:DescribeClusters",
                "dax:DescribeDefaultParameters",
                "dax:DescribeEvents",
                "dax:DescribeParameterGroups",
                "dax:DescribeParameters",
                "dax:DescribeSubnetGroups",
                "dax:IncreaseReplicationFactor",
                "dax:ListTags",
                "dax:RebootNode",
                "dax:TagResource",
                "dax:UntagResource",
                "dax:UpdateCluster",
                "dax:UpdateParameterGroup",
```

```
                "dax:UpdateSubnetGroup"
                ],
            "Effect": "Allow",
            "Resource": [
                "*"
            ]
        }
    ]
}
```

Skalieren eines DAX-Clusters

Für die Skalierung eines DAX-Clusters gibt es zwei Möglichkeiten. Die erste Option ist die horizontale Skalierung, bei der Sie Read Replicas zum Cluster hinzufügen. Die zweite Option ist die vertikale Skalierung, bei der Sie verschiedene Knotentypen auswählen. Hinweise zur Auswahl einer geeigneten Clustergröße und des entsprechenden Knotentyps für Ihre Anwendung finden Sie unter DAX-Clustergrößenleitfaden.

Horizontale Skalierung

Mit horizontaler Skalierung können Sie den Durchsatz für Lesevorgänge verbessern, indem Sie dem Cluster weitere Read Replicas hinzufügen. Ein einzelner DAX-Cluster unterstützt bis zu 10 Read Replicas, und Sie können Replikate hinzufügen oder entfernen, während der Cluster ausgeführt wird.

Wenn Sie einen neuen Knoten hinzufügen, müssen Sie die Cache-Daten von einem Peer-Knoten synchronisieren. Daher hängt die Dauer für das Hinzufügen von der Cache-Größe und der Workload Ihrer Anwendung ab. Als bewährte Methode empfehlen wir, Ihren Cluster vorab im Hinblick auf die erwarteten Datenverkehrsspitzen zu skalieren. Informationen zum Auswählen der richtigen Größe und Empfehlungen zur Überwachung finden Sie im DAX-Clustergrößenleitfaden.

Die folgenden AWS CLI Beispiele zeigen, wie die Anzahl der Knoten erhöht oder verringert werden kann. Das Argument `--new-replication-factor` gibt die Gesamtzahl der Knoten im Cluster an. Einer der Knoten ist der primäre Knoten und die anderen Knoten sind Read Replicas.

```
aws dax increase-replication-factor \
    --cluster-name MyNewCluster  \
    --new-replication-factor 5
```

```
aws dax decrease-replication-factor \
```

```
    --cluster-name MyNewCluster  \
    --new-replication-factor 3
```

> **Note**
>
> Wenn Sie den Replikationsfaktor ändern, wird der Clusterstatus in `modifying` geändert. Der Status ändert sich in `available`, wenn die Änderung abgeschlossen ist.

Vertikale Skalierung

Wenn Sie mit einer großen Datenmenge arbeiten, profitiert Ihre Anwendung möglicherweise von der Verwendung größerer Knotentypen. Größere Knoten können es dem Cluster ermöglichen, mehr Daten im Arbeitsspeicher zu speichern. Dies reduziert Cache-Fehlgriffe und verbessert die allgemeine Leistung der Anwendung. (Alle Knoten in einem DAX-Cluster müssen vom selben Typ sein.)

Wenn Ihr DAX-Cluster eine hohe Rate von Schreibvorgängen oder Cache-Fehlern aufweist, kann Ihre Anwendung auch von der Verwendung größerer Knotentypen profitieren. Schreibvorgänge und Cachefehler verbrauchen Ressourcen auf dem primären Knoten des Clusters. Daher kann die Verwendung größerer Knotentypen die Leistung des primären Knotens erhöhen und dadurch einen höheren Durchsatz für diese Arten von Vorgängen ermöglichen.

Auf einem DAX-Cluster, der gerade ausgeführt wird, können Sie die Knotentypen nicht ändern. Sie müssen stattdessen einen neuen Cluster mit dem gewünschten Knotentyp erstellen. Eine Liste der unterstützten Knotentypen finden Sie unter Knoten.

Sie können mit dem AWS-Managementkonsole, CloudFormation, oder dem AWS SDK einen neuen DAX-Cluster erstellen. AWS CLI(Verwenden Sie für den den `--node-type` Parameter AWS CLI, um den Knotentyp anzugeben.)

Anpassen der DAX-Cluster-Einstellungen

Beim Erstellen eines DAX-Clusters werden die folgenden Standardeinstellungen verwendet:

- Automatische Cache-Entfernung mit Time to Live (TTL) von 5 Minuten aktiviert
- Keine Präferenz für Availability Zones
- Keine Präferenz für Wartungsfenster
- Benachrichtigungen deaktiviert

Für neue Cluster können Sie die Einstellungen zum Zeitpunkt der Erstellung anpassen. Um dies in der AWS-Managementkonsole auszuführen, deaktivieren Sie Use default settings (Standardeinstellungen verwenden), um die folgenden Einstellungen zu ändern:

- Netzwerk und Sicherheit — Ermöglicht es Ihnen, einzelne DAX-Clusterknoten in verschiedenen Availability Zones innerhalb der aktuellen AWS Region auszuführen. Wenn Sie No Preference (Keine Präferenz) auswählen, werden die Knoten automatisch zwischen den Availability Zones verteilt.
- Parametergruppe – Dies ist ein benannter Satz von Parametern, die auf jeden Knoten im Cluster angewendet werden. Sie können eine Parametergruppe verwenden, um das Cache-TTL-Verhalten anzugeben. Sie können den Wert eines beliebigen Parameters innerhalb einer Parametergruppe (mit Ausnahme der Standardparametergruppe `default.dax.1.0`) jederzeit ändern.
- Wartungsfenster – Dies ist ein wöchentlicher Zeitraum, in dem die Software-Upgrades und -Patches auf die Knoten im Cluster angewendet werden. Sie können den Starttag, den Startzeitpunkt und die Dauer des Wartungsfensters auswählen. Wenn Sie No Preference (Keine Präferenz) auswählen, wird das Wartungsfenster zufällig aus einem 8-Stunden-Block pro Region ausgewählt. Weitere Informationen finden Sie unter Wartungsfenster.

> **Note**
>
> Parameter Group (Parametergruppe) und Maintenance Window (Wartungsfenster) können auch jederzeit auf einem laufenden Cluster geändert werden.

Wenn ein Wartungsereignis auftritt, kann DAX Sie über Amazon Simple Notification Service (Amazon SNS) benachrichtigen. Zum Konfigurieren von Benachrichtigungen wählen Sie eine Option aus der Auswahl Topic for SNS notification aus. Sie können ein neues Amazon-SNS-Thema erstellen oder ein vorhandenes Thema verwenden.

Weitere Informationen zum Erstellen und Abonnieren eines Amazon-SNS-Themas finden Sie unter Erste Schritte mit Amazon SNS im Amazon-Simple-Notification-Service-Entwicklerhandbuch.

Konfigurieren der TTL-Einstellungen

DAX verwaltet zwei Caches für Daten, die von DynamoDB gelesen werden:

- Element-Cache – Für Elemente, die mit `GetItem` oder `BatchGetItem` abgerufen wurden.

- Abfragecache – Für Ergebnismengen, die mit `Query` oder `Scan` abgerufen werden.

Weitere Informationen erhalten Sie unter Element-Cache und Abfrage-Cache.

Die TTL-Standardeinstellung für jeden dieser Caches beträgt 5 Minuten. Wenn Sie andere TTL-Einstellungen verwenden möchten, können Sie einen DAX-Cluster über eine benutzerdefinierte Parametergruppe starten. Um diesen Vorgang in der Konsole auszuführen, wählen Sie DAX | Parameter groups (DAX | Parametergruppen) im Navigationsbereich aus.

Sie können diese Aufgaben auch mit der AWS CLI ausführen. Das folgende Beispiel zeigt, wie Sie einen neuen DAX-Cluster mit einer benutzerdefinierten Parametergruppe starten. In diesem Beispiel wird die TTL-Einstellung für den Element-Cache auf 10 Minuten und für den Abfrage-Cache auf 3 Minuten festgelegt.

1. Neue Parametergruppe erstellen.

```
aws dax create-parameter-group \
    --parameter-group-name custom-ttl
```

2. Legen Sie die TTL-Einstellung für den Element-Cache auf 10 Minuten (600.000 Millisekunden) fest.

```
aws dax update-parameter-group \
    --parameter-group-name custom-ttl \
    --parameter-name-values "ParameterName=record-ttl-millis,ParameterValue=600000"
```

3. Legen Sie die TTL-Einstellung für den Abfrage-Cache auf 3 Minuten (180.000 Millisekunden) fest.

```
aws dax update-parameter-group \
    --parameter-group-name custom-ttl \
    --parameter-name-values "ParameterName=query-ttl-millis,ParameterValue=180000"
```

4. Überprüfen Sie, ob die Parameter korrekt festgelegt wurden.

```
aws dax describe-parameters --parameter-group-name custom-ttl \
    --query "Parameters[*].[ParameterName,Description,ParameterValue]"
```

Sie können nun einen neuen DAX-Cluster mit dieser Parametergruppe starten.

```
aws dax create-cluster \
    --cluster-name MyNewCluster \
    --node-type dax.r3.large \
    --replication-factor 3 \
    --iam-role-arn arn:aws:iam::123456789012:role/DAXServiceRole \
    --parameter-group custom-ttl
```

> **Note**
>
> Sie können eine Parametergruppe ändern, die an einen laufenden DAX-Cluster angehängt ist. Parameteränderungen gelten nur für Elemente, die nach der Änderung geschrieben wurden. Vorhandene zwischengespeicherte Elemente behalten die TTL und die Einstellungen bei, als sie ursprünglich geschrieben wurden.

Unterstützung von Markierungen für DAX

Viele AWS Dienste, einschließlich DynamoDB, unterstützen Tagging — die Fähigkeit, Ressourcen mit benutzerdefinierten Namen zu kennzeichnen. Sie können DAX-Clustern Tags zuweisen, sodass Sie schnell alle Ihre AWS Ressourcen mit demselben Tag identifizieren oder Ihre AWS Rechnungen anhand der von Ihnen zugewiesenen Tags kategorisieren können.

Weitere Informationen finden Sie unter Hinzufügen von Tags und Labels zu Ressourcen in DynamoDB.

Mit dem AWS-Managementkonsole

So verwalten Sie Markierungen für DAX-Cluster

1. Öffnen Sie die DynamoDB-Konsole unter. https://console.aws.amazon.com/dynamodb/
2. Klicken Sie im Navigationsbereich unter DAX auf Cluster.
3. Wählen Sie den Cluster aus, mit dem Sie arbeiten möchten.
4. Wählen Sie die Registerkarte Tags aus. Sie können Ihre Tags hier hinzufügen, auflisten, bearbeiten oder löschen.

   Wenn Sie die gewünschten Einstellungen vorgenommen haben, wählen Sie Apply Changes aus.

Mit dem AWS CLI

Wenn Sie die AWS CLI zur Verwaltung von DAX-Cluster-Tags verwenden, müssen Sie zuerst den Amazon-Ressourcennamen (ARN) für den Cluster ermitteln. Im folgenden Beispiel wird gezeigt, wie Sie einen ARN für einen Cluster namens `MyDAXCluster` bestimmen.

```
aws dax describe-clusters \
    --cluster-name MyDAXCluster \
    --query "Clusters[*].ClusterArn"
```

In der Ausgabe sieht der ARN in etwa wie folgt aus: `arn:aws:dax:us-west-2:123456789012:cache/MyDAXCluster`

Im folgenden Beispiel wird gezeigt, wie Sie den Cluster markieren.

```
aws dax tag-resource  \
    --resource-name arn:aws:dax:us-west-2:123456789012:cache/MyDAXCluster \
    --tags="Key=ClusterUsage,Value=prod"
```

So listen Sie alle Tags für einen Cluster auf.

```
aws dax list-tags \
    --resource-name arn:aws:dax:us-west-2:123456789012:cache/MyDAXCluster
```

Um ein Tag zu entfernen, geben Sie den zugehörigen Schlüssel an.

```
aws dax untag-resource  \
    --resource-name arn:aws:dax:us-west-2:123456789012:cache/MyDAXCluster \
    --tag-keys ClusterUsage
```

AWS CloudTrail Integration

DAX ist integriert AWS CloudTrail, sodass Sie DAX-Cluster-Aktivitäten überprüfen können. Mithilfe von CloudTrail Protokollen können Sie alle Änderungen anzeigen, die auf Clusterebene vorgenommen wurden. Sie können auch Änderungen an Cluster-Komponenten, z. B. Knoten, Subnetzgruppen und Parametergruppen, anzeigen. Weitere Informationen finden Sie unter Protokollieren von DynamoDB-Vorgängen mithilfe von AWS CloudTrail.

Löschen eines DAX-Clusters

Wenn Sie einen DAX-Cluster nicht mehr verwenden, sollten Sie ihn löschen, um zu verhindern, dass Ihnen nicht verwendete Ressourcen in Rechnung gestellt werden.

Sie können einen DAX-Cluster mithilfe der Konsole oder der AWS CLI löschen. Im Folgenden wird ein -Beispiel gezeigt.

```
aws dax delete-cluster --cluster-name mydaxcluster
```

Überwachen von DynamoDB Accelerator

Die Überwachung ist ein wichtiger Bestandteil der Aufrechterhaltung der Zuverlässigkeit, Verfügbarkeit und Leistung von Amazon DynamoDB Accelerator (DAX) und Ihrer AWS Lösungen. Sie sollten Überwachungsdaten aus allen Teilen Ihrer AWS Lösung sammeln, damit Sie einen etwaigen Ausfall an mehreren Stellen leichter debuggen können.

Bevor Sie mit der Überwachung von DAX beginnen, sollten Sie einen Überwachungsplan mit Antworten auf die folgenden Fragen erstellen:

- Was sind Ihre Ziele bei der Überwachung?
- Welche Ressourcen werden überwacht?
- Wie oft werden diese Ressourcen überwacht?
- Welche Überwachungstools werden verwendet?
- Wer soll die Überwachungsaufgaben ausführen?
- Wer soll benachrichtigt werden, wenn Fehler auftreten?

Themen

- Überwachungstools für DynamoDB Accelerator
- Überwachung mit Amazon CloudWatch
- Protokollierung von DAX-Vorgängen mit AWS CloudTrail

Überwachungstools für DynamoDB Accelerator

AWS bietet Tools, mit denen Sie Amazon DynamoDB Accelerator (DAX) überwachen können. Sie können einige dieser Tools konfigurieren, damit diese die Überwachung für Sie ausführen. Einige

Tools erfordern jedoch manuelle Eingriffe. Wir empfehlen, dass Sie die Überwachungsaufgaben möglichst automatisieren.

Themen

- Automatisierte Überwachungstools
- Manuelle Überwachungstools

Automatisierte Überwachungstools

Sie können die folgenden automatisierten Tools zur Überwachung von DAX verwenden und möglicherweise auftretende Probleme melden:

- Amazon CloudWatch Alarms — Überwachen Sie eine einzelne Metrik über einen von Ihnen angegebenen Zeitraum und führen Sie eine oder mehrere Aktionen aus, die auf dem Wert der Metrik im Verhältnis zu einem bestimmten Schwellenwert über mehrere Zeiträume basieren. Die Aktion ist eine Benachrichtigung, die an ein Amazon Simple Notification Service (Amazon SNS) -Thema oder eine Amazon EC2 Auto Scaling Scaling-Richtlinie gesendet wird. CloudWatch Alarme lösen keine Aktionen aus, nur weil sie sich in einem bestimmten Status befinden. Der Status muss sich geändert haben und für eine bestimmte Anzahl von Zeiträumen beibehalten worden sein. Weitere Informationen finden Sie unter Überwachen von Metriken in DynamoDB mit Amazon CloudWatch.
- Amazon CloudWatch Logs — Überwachen, speichern und greifen Sie auf Ihre Protokolldateien aus AWS CloudTrail oder anderen Quellen zu. Weitere Informationen finden Sie unter Überwachung von Protokolldateien im CloudWatch Amazon-Benutzerhandbuch.
- Amazon CloudWatch Events — Ordnen Sie Ereignisse zu und leiten Sie sie an eine oder mehrere Zielfunktionen oder Streams weiter, um Änderungen vorzunehmen, Statusinformationen zu erfassen und Korrekturmaßnahmen zu ergreifen. Weitere Informationen finden Sie unter Was ist Amazon CloudWatch Events im CloudWatch Amazon-Benutzerhandbuch.
- AWS CloudTrail Protokollüberwachung — Teilen Sie Protokolldateien zwischen Konten, überwachen CloudTrail Sie Protokolldateien in Echtzeit, indem Sie sie an CloudWatch Logs senden, schreiben Sie Protokollverarbeitungsanwendungen in Java und überprüfen Sie, ob sich Ihre Protokolldateien nach der Lieferung von nicht geändert haben CloudTrail. Weitere Informationen finden Sie unter Arbeiten mit CloudTrail Protokolldateien im AWS CloudTrail Benutzerhandbuch.

Manuelle Überwachungstools

Ein weiterer wichtiger Teil der DAX-Überwachung ist die manuelle Überwachung der Elemente, die von den CloudWatch Alarmen nicht abgedeckt werden. Die DAX- CloudWatch Trusted Advisor,, und andere AWS-Managementkonsole Dashboards bieten einen at-a-glance Überblick über den Zustand Ihrer AWS Umgebung. Zudem empfehlen wir die Überprüfung der Protokolldateien auf DAX.

- Auf dem DAX-Dashboard wird Folgendes angezeigt:
  - Servicezustand
- Auf der CloudWatch Startseite wird Folgendes angezeigt:
  - Aktuelle Alarme und Status
  - Diagramme mit Alarmen und Ressourcen
  - Servicestatus

  Darüber hinaus können CloudWatch Sie Folgendes verwenden:
  - Erstellen Sie benutzerdefinierte Dashboards zur Überwachung der Services, die Ihnen wichtig sind.
  - Aufzeichnen von Metrikdaten, um Probleme zu beheben und Trends zu erkennen.
  - Suchen und durchsuchen Sie alle Ihre AWS Ressourcenmetriken.
  - Erstellen und Bearbeiten von Alarmen, um über Probleme benachrichtigt zu werden.

Überwachung mit Amazon CloudWatch

Sie können DynamoDB Accelerator (DAX) mithilfe von Amazon überwachen CloudWatch, das Rohdaten aus DAX sammelt und zu lesbaren, nahezu in Echtzeit verfügbaren Metriken verarbeitet. Diese Statistiken werden für einen Zeitraum von zwei Wochen aufgezeichnet. Sie können anschließend auf Verlaufsdaten zugreifen, um einen besseren Eindruck von der Leistung Ihrer Webanwendung oder Ihres Service zu erhalten. Standardmäßig werden DAX-Metrikdaten automatisch an CloudWatch gesendet. Weitere Informationen finden Sie unter Was ist Amazon CloudWatch? im CloudWatch Amazon-Benutzerhandbuch.

Themen

- Wie verwende ich DAX-Metriken?
- Anzeigen von DAX-Metriken und -Dimensionen
- CloudWatch Alarme zur Überwachung von DAX erstellen

- Produktionsüberwachung

Wie verwende ich DAX-Metriken?

Die von DAX gemeldeten Metriken bieten Informationen, die Sie auf unterschiedliche Weise analysieren können. In der folgenden Liste finden Sie einige häufige Verwendungszwecke für die Metriken. Dies sind Vorschläge für den Einstieg. Es handelt sich nicht um eine umfassende Liste.

Wie kann ich ...?	Relevante Metriken
Feststellen, ob Systemfehler aufgetreten sind?	Überwachen Sie `FaultRequestCount` , um festzustellen, ob Anforderungen zu einem HTTP 500-Fehler (Serverfehler) geführt haben. Dies kann auf einen internen DAX-Servicefehler oder auf einen HTTP 500 in der SystemErrors Metrik der zugrunde liegenden Tabelle hinweisen.
Feststellen, ob Benutzerfehler aufgetreten sind?	Überwachen Sie `ErrorRequestCount` , um festzustellen, ob Anforderungen zu einem HTTP 400-Fehler (Clientfehler) geführt haben. Wenn Sie feststellen, dass die Zahl der Fehler zunimmt, sollten Sie möglicherweise untersuchen, ob Sie korrekte Clientanfragen senden, und dies sicherstellen.
Feststellen, ob Cache-Fehler aufgetreten sind?	Überwachen Sie `ItemCacheMisses` , um festzustellen, wie häufig ein Element nicht im Cache gefunden wurde. Überwachen Sie `QueryCacheMisses` und `ScanCacheMisses` , um festzustellen, wie häufig eine Abfrage oder ein Scanergebnis nicht im Cache gefunden wurden.
Überwachen von Cache-Zugriffsraten	Verwenden Sie CloudWatch Metric Math, um mithilfe mathematischer Ausdrücke eine Metrik zur Cache-Trefferquote zu definieren. Für den Element-Cache können Sie beispielsweise den Ausdruck m1/SUM([m1, m2])*100 verwenden. Hier sind m1 die `ItemCacheHits` -Metrik und m2 die `ItemCacheMisses` -Metrik für Ihren Cluster. Für die Abfrage- und Scan-Caches

Wie kann ich ...?	Relevante Metriken
	können Sie das gleiche Muster unter Verwendung der entsprechenden Abfrage- und Scan-Cache-Metriken verwenden.

Anzeigen von DAX-Metriken und -Dimensionen

Wenn Sie mit Amazon DynamoDB interagieren, sendet es Metriken und Dimensionen an Amazon. CloudWatch Sie können die folgenden Vorgehensweisen nutzen, um die Metriken für Ihre DynamoDB-Accelerators (DAX) anzuzeigen.

So zeigen Sie Metriken an (Konsole)

Metriken werden zunächst nach dem Service-Namespace und anschließend nach den verschiedenen Dimensionskombinationen in den einzelnen Namespaces gruppiert.

1. Öffnen Sie die CloudWatch Konsole unter. https://console.aws.amazon.com/cloudwatch/
2. Wählen Sie im Navigationsbereich Metriken aus.
3. Wählen Sie den Namespace DAX aus.

So zeigen Sie Metriken an (AWS CLI)

- Geben Sie als Eingabeaufforderung den folgenden Befehl ein.

```
aws cloudwatch list-metrics --namespace "AWS/DAX"
```

DAX-Metriken und -Dimensionen

Die folgenden Abschnitte enthalten die Metriken und Dimensionen, an die DAX sendet CloudWatch.

DAX-Metriken

Die folgenden Metriken stehen in DAX zur Verfügung. DAX sendet Metriken CloudWatch nur dann an, wenn sie einen Wert ungleich Null haben.

> **Note**
>
> CloudWatch aggregiert die folgenden DAX-Metriken in Intervallen von einer Minute:

- CPUUtilization
- CacheMemoryUtilization
- NetworkBytesIn
- NetworkBytesOut
- BaselineNetworkBytesInUtilization
- BaselineNetworkBytesOutUtilization
- NetworkPacketsIn
- NetworkPacketsOut
- GetItemRequestCount
- BatchGetItemRequestCount
- BatchWriteItemRequestCount
- DeleteItemRequestCount
- PutItemRequestCount
- UpdateItemRequestCount
- TransactWriteItemsCount
- TransactGetItemsCount
- ItemCacheHits
- ItemCacheMisses
- QueryCacheHits
- QueryCacheMisses
- ScanCacheHits
- ScanCacheMisses
- TotalRequestCount
- ErrorRequestCount
- FaultRequestCount
- FailedRequestCount
- QueryRequestCount
- ScanRequestCount
- ClientConnections
- EstimatedDbSize

- `EvictedSize`
- `CPUCreditUsage`
- `CPUCreditBalance`
- `CPUSurplusCreditBalance`
- `CPUSurplusCreditsCharged`

Nicht alle Statistiken, wie `Average` oder `Sum` sind für jede Metrik anwendbar. Alle diese Werte sind jedoch über die DAX-Konsole oder mithilfe der CloudWatch Konsole oder AWS SDKs für alle AWS CLI Metriken verfügbar. In der folgenden Tabelle enthält jede Metrik eine Liste gültiger Statistiken, die für diese Metrik gelten.

Metrik	Description
`CPUUtilization`	Prozentsatz der CPU-Auslastung des Knotens oder Clusters. Einheiten: `Percent` Gültige Statistiken: • `Minimum` • `Maximum` • `Average`
`CacheMemoryUtilization`	Prozentsatz des verfügbaren Cachespeichers, der vom Elementcache und im Abfragecache auf dem Knoten oder Cluster verwendet wird. Zwischengespeicherte Daten werden entfernt, bevor die Speicherauslastung 100 % erreicht (siehe `EvictedSize` -Metrik). Wenn `CacheMemoryUtilization` 100 % auf jedem Knoten erreicht, werden Schreibanforderungen gedrosselt und Sie sollten in Erwägung ziehen, zu einem Cluster mit einem größeren Knotentyp zu wechseln. Einheiten: `Percent` Gültige Statistiken:

Metrik	Description
	• `Minimum` • `Maximum` • `Average`
`NetworkBytesIn`	Anzahl der von dem Knoten oder Cluster auf allen Netzwerks chnittstellen empfangenen Byte. Einheiten: `Bytes` Gültige Statistiken: • `Minimum` • `Maximum` • `Average`
`NetworkBytesOut`	Anzahl der vom Knoten oder Cluster auf allen Netzwerks chnittstellen gesendeten Byte. Diese Metrik identifiziert das Volumen des ausgehenden Datenverkehrs in Bezug auf die Anzahl der Bytes auf einem einzelnen Knoten oder Cluster. Einheiten: `Bytes` Gültige Statistiken: • `Minimum` • `Maximum` • `Average`

Metrik	Description
`BaselineNetworkBytesInUtilization`	Der Prozentsatz der verbrauchten Basis-Netzwerkbandbreite zu einem bestimmten Zeitpunkt für eingehenden Datenverkehr. Als Referenz bedeutet 50 %, dass die Hälfte der verfügbaren Netzwerkbandbreite für eingehenden Verkehr genutzt wird. Einheiten: `Percent` Gültige Statistiken: • `Minimum` • `Maximum` • `Average` • `SampleCount` • `Sum`
`BaselineNetworkBytesOutUtilization`	Der Prozentsatz der verbrauchten Basis-Netzwerkbandbreite zu einem bestimmten Zeitpunkt für ausgehenden Datenverkehr. Als Referenz bedeutet 50 %, dass die Hälfte der verfügbaren Netzwerkbandbreite für ausgehenden Verkehr genutzt wird. Einheiten: `Percent` Gültige Statistiken: • `Minimum` • `Maximum` • `Average` • `SampleCount` • `Sum`

Metrik	Description
`NetworkPacketsIn`	Anzahl der vom Knoten oder Cluster auf allen Netzwerks chnittstellen empfangenen Pakete. Einheiten: `Count` Gültige Statistiken: • `Minimum` • `Maximum` • `Average`
`NetworkPacketsOut`	Anzahl der vom Knoten oder Cluster auf allen Netzwerks chnittstellen gesendeten Pakete. Diese Metrik identifiziert das Volumen des ausgehenden Datenverkehrs in Bezug auf die Anzahl der Pakete auf einem einzelnen Knoten oder Cluster. Einheiten: `Count` Gültige Statistiken: • `Minimum` • `Maximum` • `Average`
`GetItemRequestCount`	Anzahl von `GetItem`-Anforderungen, die vom Knoten oder Cluster behandelt werden. Einheiten: `Count` Gültige Statistiken: • `Minimum` • `Maximum` • `Average` • `SampleCount` • `Sum`

Metrik	Description
`BatchGetItemRequestCount`	Anzahl von `BatchGetItem` -Anforderungen, die vom Knoten oder Cluster behandelt werden. Einheiten: `Count` Gültige Statistiken: • `Minimum` • `Maximum` • `Average` • `SampleCount` • `Sum`
`BatchWriteItemRequestCount`	Anzahl von `BatchWriteItem` -Anforderungen, die vom Knoten oder Cluster behandelt werden. Einheiten: `Count` Gültige Statistiken: • `Minimum` • `Maximum` • `Average` • `SampleCount` • `Sum`

Metrik	Description
`DeleteItemRequestCount`	Anzahl von `DeleteItem` -Anforderungen, die vom Knoten oder Cluster behandelt werden. Einheiten: `Count` Gültige Statistiken: • `Minimum` • `Maximum` • `Average` • `SampleCount` • `Sum`
`PutItemRequestCount`	Anzahl von `PutItem`-Anforderungen, die vom Knoten oder Cluster behandelt werden. Einheiten: `Count` Gültige Statistiken: • `Minimum` • `Maximum` • `Average` • `SampleCount` • `Sum`

Metrik	Description
`UpdateItemRequestCount`	Anzahl von `UpdateItem` -Anforderungen, die vom Knoten oder Cluster behandelt werden. Einheiten: `Count` Gültige Statistiken: • `Minimum` • `Maximum` • `Average` • `SampleCount` • `Sum`
`TransactWriteItemsCount`	Anzahl von `TransactWriteItems` -Anforderungen, die vom Knoten oder Cluster behandelt werden. Einheiten: `Count` Gültige Statistiken: • `Minimum` • `Maximum` • `Average` • `SampleCount` • `Sum`

Metrik	Description
`TransactGetItemsCount`	Anzahl von `TransactGetItems` -Anforderungen, die vom Knoten oder Cluster behandelt werden. Einheiten: `Count` Gültige Statistiken: • `Minimum` • `Maximum` • `Average` • `SampleCount` • `Sum`
`ItemCacheHits`	Gibt an, wie oft ein Element vom Knoten oder Cluster aus dem Cache zurückgegeben wurde. Einheiten: `Count` Gültige Statistiken: • `Minimum` • `Maximum` • `Average` • `SampleCount` • `Sum`

Metrik	Description
`ItemCacheMisses`	Die Häufigkeit, mit der sich ein Element nicht im Knoten- oder Cluster-Cache befand und von DynamoDB abgerufen werden musste. Einheiten: `Count` Gültige Statistiken: • `Minimum` • `Maximum` • `Average` • `SampleCount` • `Sum`
`QueryCacheHits`	Gibt an, wie oft ein Abfrageergebnis aus dem Knoten- oder Clustercache zurückgegeben wurde. Einheiten: `Count` Gültige Statistiken: • `Minimum` • `Maximum` • `Average` • `SampleCount` • `Sum`

Metrik	Description
`QueryCacheMisses`	Gibt an, wie oft sich ein Abfrageergebnis nicht im Knoten- oder Clustercache befand und aus DynamoDB abgerufen werden musste. Einheiten: `Count` Gültige Statistiken: • `Minimum` • `Maximum` • `Average` • `SampleCount` • `Sum`
`ScanCacheHits`	Gibt an, wie oft ein Scanergebnis aus dem Knoten- oder Clustercache zurückgegeben wurde. Einheiten: `Count` Gültige Statistiken: • `Minimum` • `Maximum` • `Average` • `SampleCount` • `Sum`

Metrik	Description
`ScanCacheMisses`	Gibt an, wie oft sich ein Scanergebnis nicht im Knoten- oder Clustercache befand und aus DynamoDB abgerufen werden musste. Einheiten: `Count` Gültige Statistiken: • `Minimum` • `Maximum` • `Average` • `SampleCount` • `Sum`
`TotalRequestCount`	Die Gesamtanzahl von Anforderungen, die vom Knoten oder Cluster behandelt werden. Einheiten: `Count` Gültige Statistiken: • `Minimum` • `Maximum` • `Average` • `SampleCount` • `Sum`

Metrik	Description
`ErrorRequestCount`	Die Gesamtanzahl der Anforderungen, die zu einem Benutzerf ehler führten, der vom Knoten oder Cluster gemeldet wurde. Anforderungen, die vom Knoten oder Cluster gedrosselt wurden, sind enthalten. Einheiten: `Count` Gültige Statistiken: • `Minimum` • `Maximum` • `Average` • `SampleCount` • `Sum`
`ThrottledRequestCount`	Die Gesamtanzahl der Anforderungen, die vom Knoten oder Cluster gedrosselt werden. Anforderungen, die von DynamoDB gedrosselt wurden, sind nicht enthalten und können mit DynamoDB-Metriken überwacht werden. Einheiten: `Count` Gültige Statistiken: • `Minimum` • `Maximum` • `Average` • `SampleCount` • `Sum`

Metrik	Description
`FaultRequestCount`	Die Gesamtanzahl der Anforderungen, die zu einem internen Fehler führten, der vom Knoten oder Cluster gemeldet wurde. Einheiten: `Count` Gültige Statistiken: • `Minimum` • `Maximum` • `Average` • `SampleCount` • `Sum`
`FailedRequestCount`	Die Gesamtanzahl der Anforderungen, die zu einem Fehler führten, der vom Knoten oder Cluster gemeldet wurde. Einheiten: `Count` Gültige Statistiken: • `Minimum` • `Maximum` • `Average` • `SampleCount` • `Sum`

Metrik	Description
`QueryRequestCount`	Anzahl von Abfrageanforderungen, die vom Knoten oder Cluster behandelt werden. Einheiten: `Count` Gültige Statistiken: • `Minimum` • `Maximum` • `Average` • `SampleCount` • `Sum`
`ScanRequestCount`	Anzahl von Scan-Anforderungen, die vom Knoten oder Cluster behandelt werden. Einheiten: `Count` Gültige Statistiken: • `Minimum` • `Maximum` • `Average` • `SampleCount` • `Sum`

Metrik	Description
`ClientConnections`	Die Anzahl der gleichzeitigen Verbindungen, die von Clients zum Knoten oder Cluster hergestellt werden. Einheiten: `Count` Gültige Statistiken: • `Minimum` • `Maximum` • `Average` • `SampleCount` • `Sum`
`EstimatedDbSize`	Eine Annäherung an die Datenmenge, die vom Knoten oder Cluster im Elementcache und im Abfragecache zwischeng espeichert wird. Einheiten: `Bytes` Gültige Statistiken: • `Minimum` • `Maximum` • `Average`

Metrik	Description
`EvictedSize`	Die Datenmenge, die vom Knoten oder Cluster geräumt wurde, um Platz für neu angeforderte Daten zu schaffen. Wenn die Fehlrate steigt und Sie sehen, dass diese Metrik ebenfalls wächst, bedeutet dies wahrscheinlich, dass Ihr Working Set gestiegen ist. Sie sollten erwägen, zu einem Cluster mit einem größeren Knotentyp zu wechseln. Einheiten: `Bytes` Gültige Statistiken: • `Minimum` • `Maximum` • `Average` • `Sum`

Metrik	Description
`CPUCreditUsage`	Die Anzahl der vom Knoten für die CPU-Nutzung verbrauchten CPU-Guthaben. Ein CPU-Guthaben entspricht einer vCPU, die eine Minute lang bei 100% Auslastung läuft, oder einer äquivalenten Kombination aus vCPUs, Auslastung und Zeit (z. B. eine vCPU, die zwei Minuten lang mit 50% Auslastung läuft oder zwei v, die zwei Minuten lang mit 25% Auslastung CPUs laufen). Die Metriken für CPU-Guthaben sind nur mit einer fünfminütigen Frequenz verfügbar. Wenn Sie ein größeres Intervall als 5 Minuten angeben, verwenden Sie die Statistik `Sum` anstelle der `Average`. Einheiten: `Credits (vCPU-minutes)` Gültige Statistiken: • `Minimum` • `Maximum` • `Average` • `SampleCount` • `Sum`

Metrik	Description
`CPUCreditBalance`	Die Anzahl verdienter CPU-Guthaben, die ein Knotenang esammelt hat, seit er gestartet wurde. Guthaben werden auf dem Guthaben-Konto angesamme lt, nachdem sie verdient wurden, und davon entfernt, wenn sie verbraucht werden. Der Guthaben-Kontostand hat ein maximales Limit, das anhand der DAX-Knoten-Größe bestimmt wird. Nachdem das Limit erreicht ist, verfallen alle neu verdienten Guthabenpunkte. Die Guthaben in `CPUCreditBalance` sind verfügbar, um die Leistung des Knotens über die CPU-Basisnutzung hinaus zu steigern. Einheiten: `Credits (vCPU-minutes)` Gültige Statistiken: • `Minimum` • `Maximum` • `Average` • `SampleCount` • `Sum`

Metrik	Description
`CPUSurplusCreditBalance`	Die Anzahl überzähliger Guthaben, die von einem DAX-Knoten verbraucht wurden, wenn ihr `CPUCreditBalance` -Wert null ist. Der `CPUSurplusCreditBalance` -Wert wird durch erworbene CPU-Guthaben abgezahlt. Wenn die Anzahl überzähliger Guthaben die Höchstzahl der Guthaben überschreitet, die der Knoten in einem 24-Stunden-Zeitraum verdienen kann, fallen für die verbrauchten überzähligen Guthaben zusätzliche Gebühren an. Einheiten: `Credits (vCPU-minutes)` Gültige Statistiken: • `Minimum` • `Maximum` • `Average` • `SampleCount` • `Sum`

Metrik	Description
`CPUSurplusCreditsCharged`	Die Anzahl verbrauchter überzähliger Guthaben, die nicht durch verdiente CPU-Guthaben zurückgezahlt wurden, und für die deshalb eine zusätzliche Gebühr anfällt. Ausgegebene überschüssige Guthaben werden in Rechnung gestellt, wenn die ausgegebenen überschüssigen Guthaben die maximale Anzahl von Guthaben überschreiten, die der Knoten in einem Zeitraum von 24 Stunden verdienen kann. Über das Maximum hinaus ausgegebene überschüssige Guthaben werden am Ende der Stunde oder bei Beendigung des Knotens verrechnet. Einheiten: `Credits (vCPU-minutes)` Gültige Statistiken: • `Minimum` • `Maximum` • `Average` • `SampleCount` • `Sum`

> **Note**
>
> Die `CPUCreditUsage`-, `CPUCreditBalance`-, `CPUSurplusCreditBalance`- und `CPUSurplusCreditsCharged`-Metriken sind nur für T3-Knoten verfügbar.

Dimensionen für DAX-Metriken

Die Metriken für DAX sind über die Werte für die Kombination von Konto, Cluster-ID oder Cluster-ID und Knoten-ID qualifiziert. Sie können die CloudWatch Konsole verwenden, um DAX-Daten für jede der Dimensionen in der folgenden Tabelle abzurufen.

Dimension	Description
`Account`	Stellt aggregierte Statistiken über alle Knoten in einem Konto bereit.
`ClusterId`	Beschränkt die Daten auf einen Cluster.
`ClusterId, NodeId`	Beschränkt die Daten auf einen Knoten innerhalb eines Clusters.

CloudWatch Alarme zur Überwachung von DAX erstellen

Sie können einen CloudWatch Amazon-Alarm erstellen, der eine Amazon Simple Notification Service (Amazon SNS) -Nachricht sendet, wenn sich der Status des Alarms ändert. Ein Alarm überwacht eine Metrik über einen bestimmten, von Ihnen definierten Zeitraum. Der Alarm führt eine oder mehrere Aktionen durch, basierend auf dem Wert der Metrik im Vergleich zu einem bestimmten Schwellenwert in einer Reihe von Zeiträumen. Die Aktion ist eine Benachrichtigung, die an ein Amazon-SNS-Thema oder eine Auto-Scaling-Richtlinie gesendet wird. Alarme lösen nur Aktionen für anhaltende Statusänderungen aus. CloudWatch Alarme lösen keine Aktionen aus, nur weil sie sich in einem bestimmten Zustand befinden. Der Status muss sich geändert haben und für eine festgelegte Anzahl an Zeiträumen aufrechterhalten worden sein.

Wie kann ich über Abfrage-Cache-Fehler benachrichtigt werden?

1. Erstellen Sie ein Amazon-SNS-Thema, `arn:aws:sns:us-west-2:522194210714:QueryMissAlarm`.

   Weitere Informationen finden Sie unter Amazon Simple Notification Service einrichten im CloudWatch Amazon-Benutzerhandbuch.

2. Erstellen Sie den Alarm.

```
aws cloudwatch put-metric-alarm \
    --alarm-name QueryCacheMissesAlarm  \
    --alarm-description "Alarm over query cache misses" \
    --namespace AWS/DAX \
    --metric-name QueryCacheMisses  \
    --dimensions Name=ClusterID,Value=myCluster \
    --statistic Sum \
    --threshold 8 \
    --comparison-operator GreaterThanOrEqualToThreshold \
    --period 60 \
    --evaluation-periods 1 \
    --alarm-actions arn:aws:sns:us-west-2:522194210714:QueryMissAlarm
```

3. Testen Sie den Alarm.

```
aws cloudwatch set-alarm-state --alarm-name QueryCacheMissesAlarm --state-reason
 "initializing" --state-value OK
```

```
aws cloudwatch set-alarm-state --alarm-name QueryCacheMissesAlarm --state-reason
 "initializing" --state-value ALARM
```

> **Note**
>
> Sie können den Schwellenwert in einen höheren oder niedrigeren Schwellenwert ändern, der für Ihre Anwendung sinnvoll ist. Sie können CloudWatch Metric Math auch verwenden, um eine Metrik zur Cache-Fehlrate zu definieren und einen Alarm für diese Metrik einzurichten.

Wie kann ich benachrichtigt werden, wenn Anforderungen zu internen Fehlern im Cluster führen?

1. Erstellen Sie ein Amazon-SNS-Thema, `arn:aws:sns:us-west-2:123456789012:notify-on-system-errors`.

   Weitere Informationen finden Sie unter Amazon Simple Notification Service einrichten im CloudWatch Amazon-Benutzerhandbuch.

2. Erstellen Sie den Alarm.

```
aws cloudwatch put-metric-alarm \
```

```
    --alarm-name FaultRequestCountAlarm \
    --alarm-description "Alarm when a request causes an internal error" \
    --namespace AWS/DAX \
    --metric-name FaultRequestCount \
    --dimensions Name=ClusterID,Value=myCluster \
    --statistic Sum \
    --threshold 0 \
    --comparison-operator GreaterThanThreshold \
    --period 60 \
    --unit Count \
    --evaluation-periods 1 \
    --alarm-actions arn:aws:sns:us-east-1:123456789012:notify-on-system-errors
```

3. Testen Sie den Alarm.

```
aws cloudwatch set-alarm-state --alarm-name FaultRequestCountAlarm --state-reason
 "initializing" --state-value OK
```

```
aws cloudwatch set-alarm-state --alarm-name FaultRequestCountAlarm --state-reason
 "initializing" --state-value ALARM
```

Produktionsüberwachung

Sie sollten einen Ausgangswert für die normale DAX-Leistung in Ihrer Umgebung festlegen, indem Sie die Leistung zu verschiedenen Zeiten und unter verschiedenen Lastbedingungen messen. Wenn Sie DAX überwachen, sollten Sie in Betracht ziehen, historische Überwachungsdaten zu speichern. Diese Daten bieten eine Basis für den Vergleich mit aktuellen Leistungsdaten, zur Erkennung normaler Leistungsmuster und von Leistungsanomalien sowie zur Entwicklung von Verfahren für den Umgang mit Problemen.

Um einen Ausgangswert zu erstellen, sollten Sie mindestens die folgenden Punkte sowohl während der Lasttests als auch in der Produktion überwachen.

- CPU-Auslastung und gedrosselte Anforderungen, sodass Sie feststellen können, ob Sie möglicherweise einen größeren Knotentyp in Ihrem Cluster verwenden müssen. Die CPU-Auslastung Ihres Clusters ist anhand der `CPUUtilization` CloudWatch Metrik verfügbar. Die Durchschnittsstatistik für diese Metrik gibt einen Überblick über die durchschnittliche CPU-Auslastung aller Knoten in Ihrem Cluster. Für Entscheidungen zur Cluster-Skalierung empfehlen

wir, die maximale Statistik zu verwenden, die die maximale Auslastung über alle Knoten hinweg darstellt.

> **Note**
>
> AWS hat die Granularität der `CPUUtilization` Metrik verbessert. Möglicherweise stellen Sie ab 17.05.2024 bis 22.06.2024 Änderungen an der Metrik fest.

- Die Latenzzeit der Operation (gemessen auf der Client-Seite) sollte konsistent innerhalb der Latenzanforderungen Ihrer Anwendung bleiben.
- Die Fehlerquoten sollten niedrig bleiben, wie aus den `FailedRequestCount` CloudWatch Kennzahlen `ErrorRequestCountFaultRequestCount`, und hervorgeht.
- Netzwerk-Byte-Verbrauch, sodass Sie bestimmen können, ob Sie mehr Knoten oder einen größeren Knotentyp in Ihrem Cluster verwenden sollten. Um den Verbrauch zu überwachen, können Sie Warnmeldungen `BaselineNetworkBytesInUtilization` und verfügbare `BaselineNetworkBytesOutUtilization` Metriken einrichten CloudWatch, die den prozentualen Verbrauch der verfügbaren Netzwerkbandbreite für Ihren Instance-Typ angeben, jeweils für eingehenden und ausgehenden Datenverkehr.
- Zwischenspeichernutzung und bereinigte Größe, damit Sie feststellen können, ob für den Knotentyp des Clusters ausreichend Arbeitsspeicher für den aktiven Datensatz verfügbar ist.

> **Note**
>
> Bei einer großen Anzahl von Cache-Fehlern und Schreibvorgängen kann die Cache-Speicherauslastung um bis zu 100 % ansteigen, was zu Ausfallzeiten bei der Verfügbarkeit führen kann.

- Clientverbindungen, damit Sie nicht erklärte Spitzen in Verbindungen zum Cluster entdecken können.

Protokollierung von DAX-Vorgängen mit AWS CloudTrail

Amazon DynamoDB Accelerator (DAX) ist in einen Service integriert AWS CloudTrail, der eine Aufzeichnung der Aktionen bereitstellt, die von einem Benutzer, einer Rolle oder einem AWS Service in DAX ausgeführt wurden.

Weitere Informationen zu DAX und CloudTrail finden Sie im Abschnitt DynamoDB Accelerator (DAX) unter. Protokollieren von DynamoDB-Vorgängen mithilfe von AWS CloudTrail

DAX-T3/T2-Instances mit Spitzenlastleistung

Mit DAX können Sie zwischen Instances mit fester Leistung (z. B. R4, R5 und R7) und Instances mit Spitzenlastleistung (wie T2 und T3) wählen. Instance mit Spitzenlastleistung bieten ein Basisniveau der CPU-Leistung mit der Möglichkeit, bei Bedarf über die Baseline zu steigen.

Die Baseline-Leistung und die Fähigkeit, darüber zu platzen, werden durch CPU-Credits bestimmt. Instance mit Spitzenlastleistung akkumulieren CPU-Credits kontinuierlich mit einer Rate, die durch die Instanzgröße bestimmt wird, wenn der Workload unter dem Baseline-Schwellenwert liegt. Diese Credits können dann verbraucht werden, wenn der Workload zunimmt. Ein CPU-Guthaben stellt die Leistung eines gesamten CPU-Kerns für eine Minute zur Verfügung.

Viele Workloads benötigen keine gleichbleibend hohen CPUs, profitieren jedoch von vollem Zugriff auf sehr schnelle CPUs, wenn sie diese benötigen. Instance mit Spitzenlastleistung werden speziell für diese Anwendungsfälle entwickelt. Wenn Sie eine gleichbleibend hohe CPU-Leistung für Ihre Datenbank benötigen, empfehlen wir Ihnen, Instanzen mit fester Leistung zu verwenden.

DAX-T2-Instance-Familie

DAX-T2-Instances sind burstbare Allzweck-Performance-Instances, die ein Basisniveau der CPU-Leistung mit der Fähigkeit bieten, über die Baseline zu steigen. T2-Instances sind eine gute Wahl für Test- und Entwicklungs-Workloads, die Preisvorhersagbarkeit erfordern. DAX-T2-Instances sind für den Standardmodus konfiguriert. Wenn die Instance nicht mehr viele angesammelte Guthaben aufweist, wird die CPU-Nutzung schrittweise bis auf Baseline-Ebene gesenkt. Weitere Informationen zum Standardmodus finden Sie unter Standardmodus für Instances mit Spitzenlastleistung im Amazon-EC2-Benutzerhandbuch.

DAX-T3-Instance-Familie

DAX-T3-Instances sind Allzweck-Instance-Typen mit Spitzenlastleistung der nächsten Generation, die eine CPU-Basisleistung mit der Fähigkeit bereitstellen, die CPU-Nutzung jederzeit so lange wie nötig zu steigern. T3-Instances bieten ein ausgewogenes Verhältnis von Rechen-, Speicher- und Netzwerkressourcen und sind ideal für Workloads mit moderater CPU-Nutzung, bei der temporäre Nutzungsspitzen auftreten. DAX-T3-Instances sind für den unbegrenzten Modus konfiguriert, was bedeutet, dass sie gegen eine zusätzliche Gebühr über ein 24-Stunden-Fenster über die

Baseline hinausgehen können. Weitere Informationen zum unbegrenzten Modus finden Sie unter Unbegrenzter Modus für Instances mit Spitzenlastleistung im Amazon-EC2-Benutzerhandbuch.

DAX-T3-Instances können eine hohe CPU-Leistung aufrechterhalten, solange ein Workload dies erfordert. Für die meisten allgemeinen Workloads bieten T3-Instances eine ausreichende Leistung ohne zusätzliche Gebühren. Der stündliche T3-Instance-Preis deckt automatisch alle zwischenzeitlichen Nutzungsspitzen ab, wenn die durchschnittliche CPU-Auslastung einer T3-Instance über ein 24-Stunden-Fenster auf oder unter dem Basiswert liegt.

Beispielsweise ein `dax.t3.small`-Instance erhält kontinuierlich Credits mit einer Rate von 24 CPU-Credits pro Stunde. Diese Funktion bietet eine Basisleistung, die 20% eines CPU-Kerns entspricht (20% × 60 Minuten = 12 Minuten). Wenn die Instance die erhaltenen Credits nicht verwendet, werden sie in ihrem CPU-Guthaben bis zu einem Maximum von 576 CPU-Credits gespeichert. Wenn die `t3.small`-Instance auf mehr als 20% eines Kerns hinausgehen muss, zieht sie aus seinem CPU-Guthaben, um diesen Anstieg automatisch zu bewältigen.

Während DAX-T2-Instances auf die Basisleistung beschränkt sind, sobald der CPU-Guthaben auf Null herabgesetzt wurde, können DAX-T3-Instances über der Basislinie herausgehen, selbst wenn ihr CPU-Guthaben gleich Null ist. Für die überwiegende Mehrheit der Workloads, bei denen die durchschnittliche CPU-Nutzung bei oder unterhalb der Baseline-Leistung liegt, deckt der Baseline-Stundenpreis für `t3.small` alle CPU-Bursts ab. Wenn die Instance zufällig mit einer durchschnittlichen CPU-Auslastung von 25% (5% über dem Basiswert) über einen Zeitraum von 24 Stunden nach der Berechnung des CPU-Guthabens auf Null läuft, werden zusätzliche 11,52 Cent (9,6 Cent/VCPU-Stunde × 1 vCPU × 5% × 24 Stunden) berechnet. Siehe Amazon-DynamoDB-Preise für weitere Informationen zur Preisdetails.

DAX-Zugriffskontrolle

DynamoDB Accelerator (DAX) kann mit DynamoDB zusammenarbeiten, um Anwendungen nahtlos eine Caching-Schicht hinzuzufügen. DAX und DynamoDB verfügen jedoch über separate Zugriffssteuerungsmechanismen. Beide Dienste verwenden AWS Identity and Access Management (IAM), um ihre jeweiligen Sicherheitsrichtlinien zu implementieren, aber die Sicherheitsmodelle für DAX und DynamoDB sind unterschiedlich.

Wir empfehlen dringend, dass Sie beide Sicherheitsmodelle verstehen, damit Sie für Ihre Anwendungen mit DAX ordnungsgemäße Sicherheitsmaßnahmen implementieren können.

In diesem Abschnitt werden die Zugriffskontrollmechanismen von DAX beschrieben und beispielhafte IAM-Richtlinien bereitgestellt, die Sie an Ihre Bedürfnisse anpassen können.

Mit DynamoDB können Sie IAM-Richtlinien erstellen, die Aktionen begrenzen, die ein Benutzer mit individuellen DynamoDB-Ressourcen durchführen kann. Sie können beispielsweise eine Benutzerrolle erstellen, die ausschließlich dem Benutzer erlaubt, schreibgeschützte Aktionen auf einer bestimmten DynamoDB-Tabelle durchzuführen. (Weitere Informationen finden Sie unter Identity and Access Management für Amazon DynamoDB.) Im Vergleich dazu konzentriert sich das DAX-Sicherheitsmodell auf Cluster-Sicherheit und die Fähigkeit des Clusters zum Ausführen von DynamoDB-API-Aktionen in Ihrem Namen.

> **Warning**
>
> Wenn Sie derzeit IAM-Rollen und -Richtlinien zum Einschränken des Zugriffs auf DynamoDB-Tabellendaten nutzen, kann die Verwendung von DAX diese Richtlinien unterlaufen. Beispiel: Ein Benutzer hat Zugriff auf eine DynamoDB-Tabelle über DAX, aber keinen expliziten Zugriff auf dieselbe Tabelle, wenn er direkt auf DynamoDB zugreift. Weitere Informationen finden Sie unter Identity and Access Management für Amazon DynamoDB.
>
> DAX erzwingt keine Separation der Daten in DynamoDB auf Benutzerebene. Stattdessen erben Benutzer die Berechtigungen der IAM-Richtlinie des DAX-Clusters, wenn sie auf diesen Cluster zugreifen. Beim Zugriff auf die DynamoDB-Tabellen über DAX sind daher die einzigen aktiven Zugriffskontrollen die Berechtigungen in der IAM-Richtlinie des DAX-Clusters. Es werden keine weiteren Berechtigungen erkannt.
>
> Wenn Sie Isolation benötigen, empfehlen wir, dass Sie zusätzliche DAX-Cluster erstellen und die IAM-Richtlinie für jeden Cluster entsprechend abgrenzen. Sie können z. B. mehrere DAX-Cluster erstellen und jedem Cluster den Zugriff auf eine einzelne Tabelle erlauben.

IAM-Servicerolle für DAX

Beim Erstellen eines DAX-Clusters, müssen Sie den Cluster mit einer IAM-Rolle verknüpfen. Dies wird als Servicerolle für den Cluster bezeichnet.

Angenommen, Sie möchten einen neuen DAX-Cluster mit dem Namen erstellen. DAXCluster01 Sie könnten eine Servicerolle mit dem Namen DAXServiceRole erstellen und die Rolle zuordnen DAXCluster01. Die Richtlinie für DAXServiceRole würde die DynamoDB-Aktionen definieren, die im Namen der Benutzer, mit denen interagiert, ausgeführt werden DAXCluster01könnten. DAXCluster01

Wenn Sie eine Servicerolle erstellen, müssen Sie eine Vertrauensbeziehung zwischen DAXServiceRole und dem DAX-Service selbst angeben. Eine Vertrauensstellung bestimmt, welche

Entitäten eine Rolle übernehmen und deren Berechtigungen nutzen können. Im Folgenden finden Sie ein Beispiel für ein Dokument zur Vertrauensstellung für DAXServiceRole:

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "Service": "dax.amazonaws.com"
            },
            "Action": "sts:AssumeRole"
        }
    ]
}
```

Diese Vertrauensstellung ermöglicht es einem DAX-Cluster, DAXServiceeine Rolle zu übernehmen und DynamoDB-API-Aufrufe in Ihrem Namen durchzuführen.

Die zulässigen DynamoDB-API-Aktionen werden in einem IAM-Richtliniendokument beschrieben, das Sie an Role anhängen. DAXService Dies ist ein Beispiel für ein Richtliniendokument.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "DaxAccessPolicy",
            "Effect": "Allow",
            "Action": [
                "dynamodb:DescribeTable",
                "dynamodb:PutItem",
                "dynamodb:GetItem",
                "dynamodb:UpdateItem",
                "dynamodb:DeleteItem",
                "dynamodb:Query",
                "dynamodb:Scan",
```

```
                "dynamodb:BatchGetItem",
                "dynamodb:BatchWriteItem",
                "dynamodb:ConditionCheckItem"
            ],
            "Resource": [
                "arn:aws:dynamodb:us-west-2:123456789012:table/Books"
            ]
        }
    ]
}
```

Diese Richtlinie erlaubt DAX die Durchführung aller DynamoDB-API-Aktionen an einer DynamoDB-Tabelle. Die `dynamodb:DescribeTable`-Aktion ist erforderlich, damit DAX Metadaten über die Tabelle beibehält, und die anderen sind Lese- und Schreibvorgänge, die für Elemente in der Tabelle ausgeführt werden. Die Tabelle mit dem Namen `Books` befindet sich in der Region us-west-2 und ist im Besitz der AWS-Konto-ID `123456789012`.

> **Note**
>
> DAX unterstützt Mechanismen, um das Confused-Deputy-Problem beim dienstübergreifenden Zugriff zu verhindern. Weitere Informationen finden Sie unter Das Problem des verwirrten Stellvertreters im IAM-Benutzerhandbuch.

IAM-Richtlinie, um DAX-Cluster-Zugriff zu gewähren

Nach dem Erstellen eines DAX-Clusters müssen Sie einem Benutzer Berechtigungen erteilen, damit er auf den DAX-Cluster zugreifen kann.

Nehmen wir beispielsweise an, Sie möchten einem Benutzer namens Alice DAXCluster01Zugriff gewähren. Sie würden zunächst eine IAM-Richtlinie (AliceAccessPolicy) erstellen, die die DAX-Cluster und DAX-API-Aktionen definiert, auf die der Empfänger zugreifen kann. Sie können anschließend den Zugriff gewähren, indem Sie diese Richtlinie der Benutzerin Alice zuweisen.

Das folgende Richtliniendokument gewährt dem Empfänger vollen Zugriff auf DAXCluster01.

JSON

```
{
```

```
    "Version":"2012-10-17",
    "Statement": [
        {
            "Action": [
                "dax:*"
            ],
            "Effect": "Allow",
            "Resource": [
                "arn:aws:dax:us-west-2:123456789012:cache/DAXCluster01"
            ]
        }
    ]
}
```

Das Richtliniendokument erlaubt den Zugriff auf den DAX-Cluster, erteilt jedoch keine DynamoDB-Berechtigungen. (Die DynamoDB-Berechtigungen werden von der DAX-Servicerolle zugewiesen.)

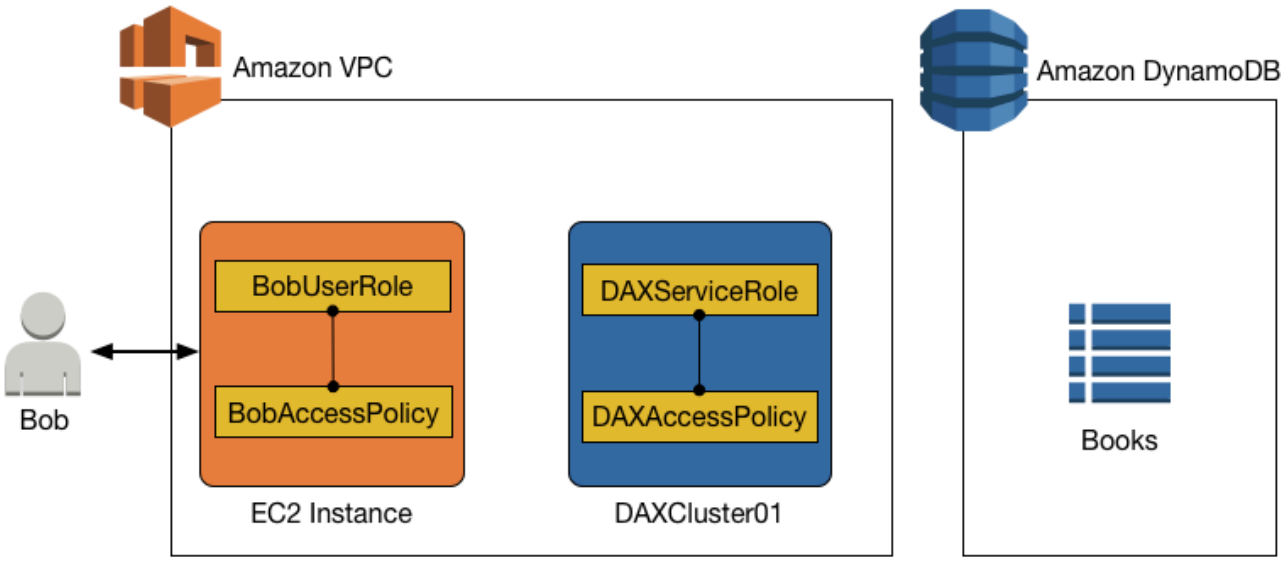
Sie würden für die Benutzerin Alice zunächst eine `AliceAccessPolicy` mit dem zuvor gezeigten Richtliniendokument erstellen. Anschließend würden Sie die Richtlinie Alice zuweisen.

> **Note**
>
> Anstatt die Richtlinie einem Benutzer zuzuweisen, könnten Sie sie einer IAM-Rolle zuweisen. Auf diese Weise hätten alle Benutzer, die diese Rolle übernehmen, die Berechtigungen, die Sie in der Richtlinie festgelegt haben.

Die Benutzerrichtlinie bestimmt in Verbindung mit der DAX-Servicerolle die DynamoDB-Ressourcen und API-Aktionen, auf die der Empfänger über DAX zugreifen kann.

Fallstudie: Zugreifen auf DynamoDB und DAX

Das folgende Szenario kann helfen, das Verständnis der IAM-Richtlinien für die Verwendung mit DAX zu vertiefen. (Wir werden uns für den Rest dieses Abschnitts auf dieses Szenario beziehen.) Das folgende Diagramm zeigt einen allgemeinen Überblick des Szenarios.

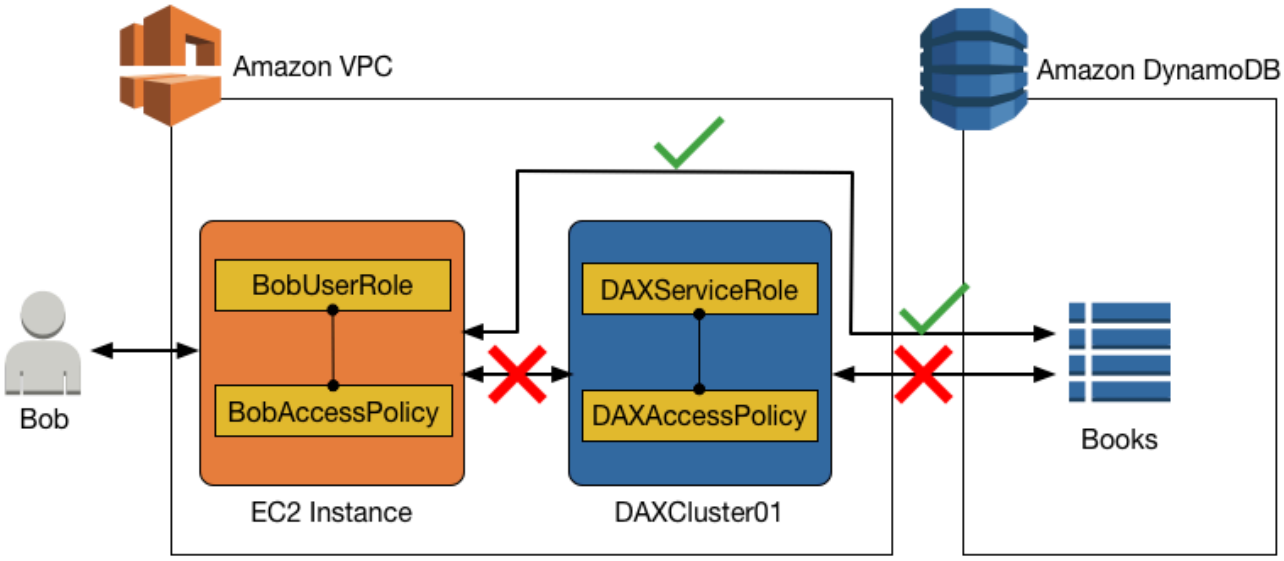


In diesem Szenario gibt es die folgenden Entitäten:

- Ein Benutzer (Bob).
- Eine IAM-Rolle (`BobUserRole`). Bob übernimmt diese Rolle zur Laufzeit.
- Eine IAM-Richtlinie (`BobAccessPolicy`). Diese Richtlinie ist an `BobUserRole` angehängt. `BobAccessPolicy` definiert die Ressourcen DynamoDB und DAX, auf die `BobUserRole` zugreifen darf.
- Ein DAX-Cluster (`DAXCluster01`).
- Eine IAM-Servicerolle (`DAXServiceRole`). Diese Rolle erlaubt `DAXCluster01` den Zugriff auf DynamoDB.
- Eine IAM-Richtlinie (`DAXAccessPolicy`). Diese Richtlinie ist beigefügt`DAXServiceRole`. `DAXAccessPolicy`definiert die DynamoDB APIs und die Ressourcen, auf die `DAXCluster01` zugegriffen werden darf.
- Eine DynamoDB-Tabelle (`Books`)

Die Kombination der Richtlinienanweisungen in `BobAccessPolicy` und `DAXAccessPolicy` bestimmt, was Bob mit der Tabelle `Books` machen kann. Beispielsweise kann Bob `Books` direkt (über den DynamoDB-Endpunkt), indirekt (über den DAX-Cluster) oder beides zugreifen. Bob könnte auch Daten von `Books` lesen, Daten in `Books` schreiben oder beides.

Zugriff auf DynamoDB, aber kein Zugriff mit DAX



Es ist möglich, direkten Zugriff auf eine DynamoDB-Tabelle zuzulassen, während ein indirekter Zugriff mit einem DAX-Cluster verhindert wird. Für direkten Zugriff auf DynamoDB werden die Berechtigungen für `BobUserRole` durch `BobAccessPolicy` bestimmt (welche an die Rolle angehängt sind).

Lesezugriff auf (nur) DynamoDB

Bob kann auf DynamoDB mit `BobUserRole` zugreifen. Die dieser Rolle (`BobAccessPolicy`) zugeordnete IAM-Richtlinie bestimmt, auf welche DynamoDB-Tabellen zugegriffen `BobUserRole` werden kann und welche APIs davon `BobUserRole` aufgerufen werden können.

Berücksichtigen Sie das folgende Richtliniendokument für `BobAccessPolicy`.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "DynamoDBAccessStmt",
            "Effect": "Allow",
```

```
            "Action": [
                "dynamodb:GetItem",
                "dynamodb:BatchGetItem",
                "dynamodb:Query",
                "dynamodb:Scan"
            ],
            "Resource": "arn:aws:dynamodb:us-west-2:123456789012:table/Books"
        }
    ]
}
```

Wenn dieses Dokument an `BobAccessPolicy` angehängt wird, ermöglicht es `BobUserRole` den Zugriff auf den DynamoDB-Endpunkt und das Ausführen von schreibgeschützten Operationen mit der `Books`-Tabelle.

DAX erscheint nicht in dieser Richtlinie, sodass ein Zugriff über DAX verweigert wird.

Lese- und Schreibzugriff auf (nur) DynamoDB

Wenn read/write Zugriff auf DynamoDB `BobUserRole` erforderlich ist, würde die folgende Richtlinie funktionieren.

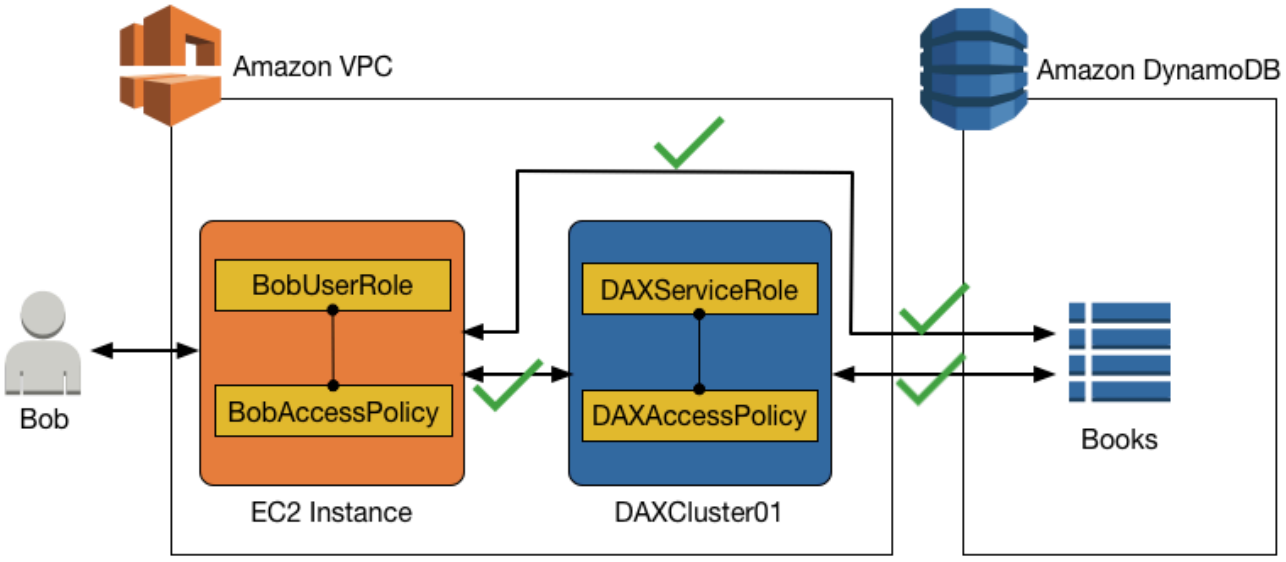
JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "DynamoDBAccessStmt",
            "Effect": "Allow",
            "Action": [
                "dynamodb:GetItem",
                "dynamodb:BatchGetItem",
                "dynamodb:Query",
                "dynamodb:Scan",
                "dynamodb:PutItem",
                "dynamodb:UpdateItem",
                "dynamodb:DeleteItem",
                "dynamodb:BatchWriteItem",
                "dynamodb:ConditionCheckItem"
            ],
```

```
            "Resource": "arn:aws:dynamodb:us-west-2:123456789012:table/Books"
        }
    ]
}
```

Erneut erscheint DAX nicht in dieser Richtlinie, sodass ein Zugriff über DAX verweigert wird.

Zugriff auf DynamoDB und DAX



Um einen Zugriff auf einen DAX-Cluster zu erlauben, müssen Sie DAX-spezifische Aktionen in einer IAM-Richtlinie angeben.

Die folgenden DAX-spezifischen Aktionen entsprechen ihren Gegenstücken mit vergleichbaren Namen in der DynamoDB-API:

- `dax:GetItem`
- `dax:BatchGetItem`
- `dax:Query`
- `dax:Scan`
- `dax:PutItem`
- `dax:UpdateItem`

- `dax:DeleteItem`
- `dax:BatchWriteItem`
- `dax:ConditionCheckItem`

Das Gleiche gilt für den `dax:EnclosingOperation`-Bedingungsschlüssel.

Schreibgeschützter Zugriff auf DynamoDB und schreibgeschützter Zugriff auf DAX

In diesem Beispiel gehen wir davon aus, dass Bob Lesezugriff auf die `Books`-Tabelle aus DynamoDB und DAX benötigt. Die folgende Richtlinie (`BobUserRole` zugeordnet) gewährt diesen Zugriff.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "DAXAccessStmt",
            "Effect": "Allow",
            "Action": [
                "dax:GetItem",
                "dax:BatchGetItem",
                "dax:Query",
                "dax:Scan"
            ],
            "Resource": "arn:aws:dax:us-west-2:123456789012:cache/DAXCluster01"
        },
        {
            "Sid": "DynamoDBAccessStmt",
            "Effect": "Allow",
            "Action": [
                "dynamodb:GetItem",
                "dynamodb:BatchGetItem",
                "dynamodb:Query",
                "dynamodb:Scan"
            ],
            "Resource": "arn:aws:dynamodb:us-west-2:123456789012:table/Books"
        }
    ]
}
```

Die Richtlinie enthält eine Anweisung für DAX-Zugriff (`DAXAccessStmt`) und eine weitere Anweisung für Dynamo DBaccess (). `DynamoDBAccessStmt` Diese Anweisungen würden Bob gestatten, `GetItem`-, `BatchGetItem`-, `Query`- und `Scan`-Anforderungen an den `DAXCluster01` zu senden.

Die Servicerolle für `DAXCluster01` würde jedoch auch Lesezugriff auf die `Books`-Tabelle in DynamoDB erfordern. Die folgende, an `DAXServiceRole` angehängte IAM-Richtlinie würde diese Anforderung erfüllen.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "DynamoDBAccessStmt",
            "Effect": "Allow",
            "Action": [
                "dynamodb:GetItem",
                "dynamodb:BatchGetItem",
                "dynamodb:Query",
                "dynamodb:Scan"
            ],
            "Resource": "arn:aws:dynamodb:us-west-2:123456789012:table/Books"
        }
    ]
}
```

Lese-/Schreibzugriff auf DynamoDB und schreibgeschützt mit DAX

Für eine bestimmte Benutzerrolle können Sie read/write Zugriff auf eine DynamoDB-Tabelle gewähren und gleichzeitig den schreibgeschützten Zugriff über DAX zulassen.

Für Bob müsste die IAM-Richtlinie für `BobUserRole` DynamoDB-Lese- und Schreibaktionen für die `Books`-Tabelle zulassen und gleichzeitig schreibgeschützte Aktionen über `DAXCluster01` unterstützen.

Das folgende Richtliniendokument für `BobUserRole` gewährt diesen Zugriff.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "DAXAccessStmt",
            "Effect": "Allow",
            "Action": [
                "dax:GetItem",
                "dax:BatchGetItem",
                "dax:Query",
                "dax:Scan"
            ],
            "Resource": "arn:aws:dax:us-west-2:123456789012:cache/DAXCluster01"
        },
        {
            "Sid": "DynamoDBAccessStmt",
            "Effect": "Allow",
            "Action": [
                "dynamodb:GetItem",
                "dynamodb:BatchGetItem",
                "dynamodb:Query",
                "dynamodb:Scan",
                "dynamodb:PutItem",
                "dynamodb:UpdateItem",
                "dynamodb:DeleteItem",
                "dynamodb:BatchWriteItem",
                "dynamodb:DescribeTable",
                "dynamodb:ConditionCheckItem"
            ],
            "Resource": "arn:aws:dynamodb:us-west-2:123456789012:table/Books"
        }
    ]
}
```

Zudem würde `DAXServiceRole` eine IAM-Richtlinie erforderlich machen, die `DAXCluster01` schreibgeschützte Aktionen mit der `Books`-Tabelle erlaubt.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "DynamoDBAccessStmt",
            "Effect": "Allow",
            "Action": [
                "dynamodb:GetItem",
                "dynamodb:BatchGetItem",
                "dynamodb:Query",
                "dynamodb:Scan",
                "dynamodb:DescribeTable"
           ],
            "Resource": "arn:aws:dynamodb:us-west-2:123456789012:table/Books"
        }
    ]
}
```

Read/write access to DynamoDB and read/writeZugriff auf DAX

Nehmen wir nun an, Bob benötigte read/write Zugriff auf die `Books` Tabelle, direkt von DynamoDB oder indirekt von. `DAXCluster01` Das folgende Richtliniendokument, das `BobAccessPolicy` zugeordnet ist, gewährt diesen Zugriff.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "DAXAccessStmt",
            "Effect": "Allow",
            "Action": [
                "dax:GetItem",
                "dax:BatchGetItem",
                "dax:Query",
                "dax:Scan",
                "dax:PutItem",
```

```
                "dax:UpdateItem",
                "dax:DeleteItem",
                "dax:BatchWriteItem",
                "dax:ConditionCheckItem"
            ],
            "Resource": "arn:aws:dax:us-west-2:123456789012:cache/DAXCluster01"
        },
        {
            "Sid": "DynamoDBAccessStmt",
            "Effect": "Allow",
            "Action": [
                "dynamodb:GetItem",
                "dynamodb:BatchGetItem",
                "dynamodb:Query",
                "dynamodb:Scan",
                "dynamodb:PutItem",
                "dynamodb:UpdateItem",
                "dynamodb:DeleteItem",
                "dynamodb:BatchWriteItem",
                "dynamodb:DescribeTable",
                "dynamodb:ConditionCheckItem"
            ],
            "Resource": "arn:aws:dynamodb:us-west-2:123456789012:table/Books"
        }
    ]
}
```

Darüber hinaus `DAXServiceRole` wäre eine IAM-Richtlinie erforderlich, die es ermöglicht, read/write Aktionen `DAXCluster01` an der Tabelle auszuführen. `Books`

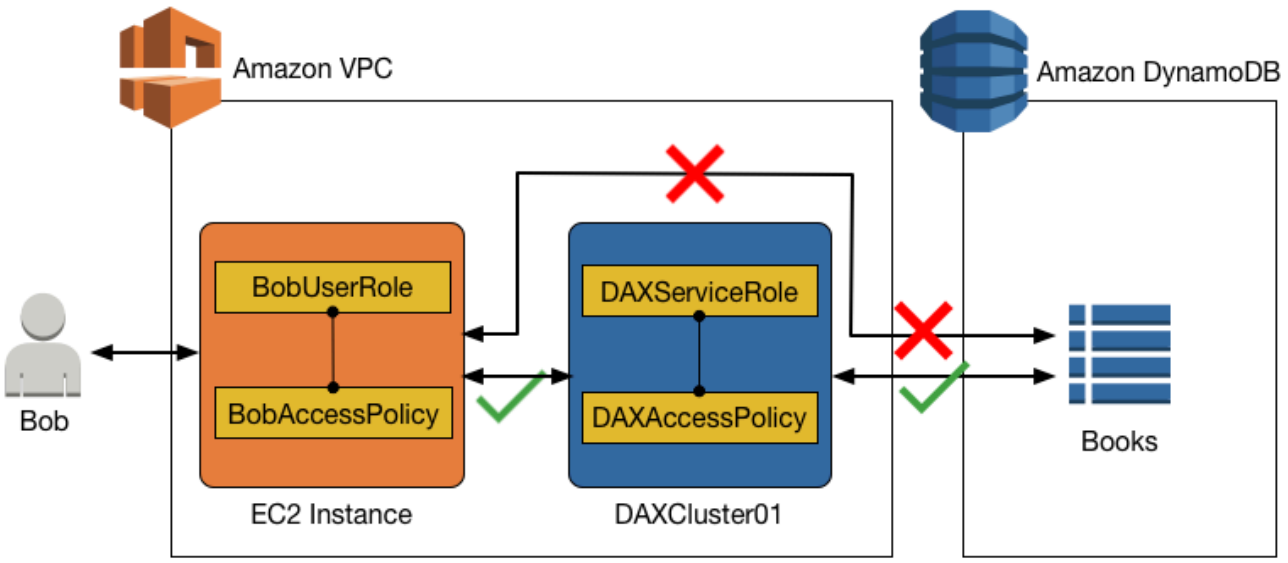
JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "DynamoDBAccessStmt",
            "Effect": "Allow",
            "Action": [
                "dynamodb:GetItem",
                "dynamodb:BatchGetItem",
```

```
                "dynamodb:Query",
                "dynamodb:Scan",
                "dynamodb:PutItem",
                "dynamodb:UpdateItem",
                "dynamodb:DeleteItem",
                "dynamodb:BatchWriteItem",
                "dynamodb:DescribeTable"
            ],
            "Resource": "arn:aws:dynamodb:us-west-2:123456789012:table/Books"
        }
    ]
}
```

Zugriff auf DynamoDB via DAX, aber kein direkter Zugriff auf DynamoDB

In diesem Szenario kann Bob über DAX auf die `Books`-Tabelle zugreifen, hat aber keinen direkten Zugriff auf die `Books`-Tabelle in DynamoDB. Wenn Bob also Zugriff auf DAX erhält, erhält er auch Zugriff auf eine DynamoDB-Tabelle, auf die er sonst möglicherweise nicht zugreifen könnte. Wenn Sie eine IAM-Richtlinie für die DAX-Servicerolle konfigurieren, denken Sie daran, dass jeder Benutzer, dem über die Benutzerzugriffsrichtlinie Zugriff auf den DAX-Cluster gewährt wird, Zugriff auf die in dieser Richtlinie angegebenen Tabellen erhält. In diesem Fall erhält `BobAccessPolicy` Zugriff auf die in `DAXAccessPolicy` angegebenen Tabellen.



Wenn Sie derzeit IAM-Rollen und -Richtlinien verwenden, um den Zugriff auf DynamoDB-Tabellen und -Daten einzuschränken, kann die Verwendung von DAX diese Richtlinien untergraben. In der folgenden Richtlinie hat Bob zugriff auf eine DynamoDB-Tabelle über DAX, aber keinen expliziten direkten Zugriff auf dieselbe Tabelle in DynamoDB.

Das folgende Richtliniendokument (`BobAccessPolicy`), das an `BobUserRole` angehängt ist, würde diesen Zugriff gewähren.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "DAXAccessStmt",
            "Effect": "Allow",
            "Action": [
                "dax:GetItem",
                "dax:BatchGetItem",
                "dax:Query",
                "dax:Scan",
                "dax:PutItem",
                "dax:UpdateItem",
                "dax:DeleteItem",
                "dax:BatchWriteItem",
                "dax:ConditionCheckItem"
            ],
            "Resource": "arn:aws:dax:us-west-2:123456789012:cache/DAXCluster01"
        }
    ]
}
```

In dieser Zugriffsrichtlinie gibt es keine Berechtigungen, um direkt auf DynamoDB zuzugreifen.

Zusammen mit `BobAccessPolicy` gibt `BobUserRole` der Folgenden `DAXAccessPolicy` den Zugriff auf die DynamoDB-Tabelle `Books`, auch wenn `BobUserRole` nicht direkt auf die `Books`-Tabelle zugegriffen werden kann.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "DynamoDBAccessStmt",
            "Effect": "Allow",
            "Action": [
                "dynamodb:GetItem",
                "dynamodb:BatchGetItem",
                "dynamodb:Query",
                "dynamodb:Scan",
                "dynamodb:PutItem",
                "dynamodb:UpdateItem",
                "dynamodb:DeleteItem",
                "dynamodb:BatchWriteItem",
                "dynamodb:DescribeTable",
                "dynamodb:ConditionCheckItem"
            ],
            "Resource": "arn:aws:dynamodb:us-west-2:123456789012:table/Books"
        }
    ]
}
```

Wie dieses Beispiel zeigt, müssen Sie bei der Konfiguration der Zugriffskontrolle für die Benutzerzugriffsrichtlinie und die DAX-Clusterzugriffsrichtlinie den end-to-end Zugriff vollständig verstehen, um sicherzustellen, dass das Prinzip der geringsten Rechte eingehalten wird. Stellen Sie zudem sicher, dass, wenn Sie einem Benutzer Zugriff auf einen DAX-Cluster gewähren, zuvor festgelegte Zugriffskontrollrichtlinien nicht unterlaufen werden.

DAX-Verschlüsselung im Ruhezustand

Die Amazon-DynamoDB-Accelerator-(DAX)-Funktion zur Verschlüsselung von Daten im Ruhezustand stellt eine zusätzliche Sicherheitsebene für die Daten bereit, indem die Daten vor unautorisiertem Zugriff auf den zugrunde liegenden Speicher geschützt werden. Richtlinien der Organisation, Vorschriften der Branche oder Behörde und Compliance-Anforderungen schreiben möglicherweise zum Schutz Ihrer Daten die Verschlüsselung im Ruhezustand vor. Sie können

mithilfe der Verschlüsselung den Datenschutz Ihrer in der Cloud bereitgestellten Anwendungen erhöhen.

Bei der Verschlüsselung im Ruhezustand werden die von DAX auf der Festplatte persistenten Daten mit dem 256-Bit Advanced Encryption Standard, auch bekannt als AES-256-Verschlüsselung, verschlüsselt. DAX schreibt Daten auf den Datenträger als Teil der Weitergabe von Änderungen vom primären Knoten an Read Replicas.

Die DAX-Verschlüsselung im Ruhezustand wird automatisch in AWS Key Management Service (AWS KMS) integriert, um den Standardschlüssel für den einzelnen Dienst zu verwalten, der zur Verschlüsselung Ihrer Cluster verwendet wird. Wenn bei der Erstellung Ihres verschlüsselten DAX-Clusters kein Dienst-Standardschlüssel vorhanden ist, AWS KMS wird automatisch ein neuer AWS verwalteter Schlüssel für Sie erstellt. Dieser Schlüssel wird für verschlüsselte Cluster verwendet, die in future erstellt werden. AWS KMS kombiniert sichere, hochverfügbare Hardware und Software, um ein für die Cloud skaliertes Schlüsselverwaltungssystem bereitzustellen.

Nachdem Sie die Daten verschlüsselt haben, übernimmt DAX die Entschlüsselung Ihrer Daten auf transparente Art und Weise und mit minimaler Auswirkung auf die Leistung. An Ihren Anwendungen sind keine Änderungen zur Nutzung der Verschlüsselung erforderlich.

> **Note**
>
> DAX ruft nicht AWS KMS für jeden einzelnen DAX-Vorgang auf. DAX verwendet den Schlüssel nur beim Starten des Clusters. Auch wenn der Zugriff verweigert wird, kann DAX auf die Daten zugreifen, bis der Cluster heruntergefahren wurde. Vom Kunden angegebene AWS KMS Schlüssel werden nicht unterstützt.

Die DAX-Verschlüsselung von Daten im Ruhezustand ist für die folgenden Cluster-Knotentypen verfügbar.

Familie	Knotentyp
Speicheroptimiert (R4, R5 und R7)	dax.r4.large dax.r4.xlarge dax.r4.2xlarge dax.r4.4xlarge

Familie	Knotentyp
	dax.r4.8xlarge dax.r4.16xlarge dax.r5.large dax.r5.xlarge dax.r5.2xlarge dax.r5.4xlarge dax.r5.8xlarge dax.r5.12xlarge dax.r5.16xlarge dax.r5.24xlarge dax.r7i.large dax.r7i.xlarge dax.r7i.2xlarge dax.r7i.4xlarge dax.r7i.8xlarge dax.r7i.12xlarge dax.r7i.16xlarge dax.r7i.24xlarge
Allgemein (T2)	dax.t2.small dax.t2.medium

Familie	Knotentyp
Allgemein (T3)	dax.t3.small dax.t3.medium

> **Important**
>
> Die DAX-Verschlüsselung von Daten im Ruhezustand wird für `dax.r3.*`-Knotentypen nicht unterstützt.

Sie können die Verschlüsselung von Daten im Ruhezustand nicht aktivieren oder deaktivieren, nachdem ein Cluster erstellt wurde. Sie müssen den Cluster neu erstellen, um Verschlüsselung von Daten im Ruhezustand zu aktivieren, wenn die Funktion bei der ursprünglichen Erstellung nicht aktiviert wurde.

Die DAX-Verschlüsselung im Ruhezustand wird ohne zusätzliche Kosten angeboten (es fallen Gebühren für die Nutzung des AWS KMS Verschlüsselungsschlüssels an). Vollständige Informationen zu Preisen finden Sie unter Amazon-DynamoDB-Preise.

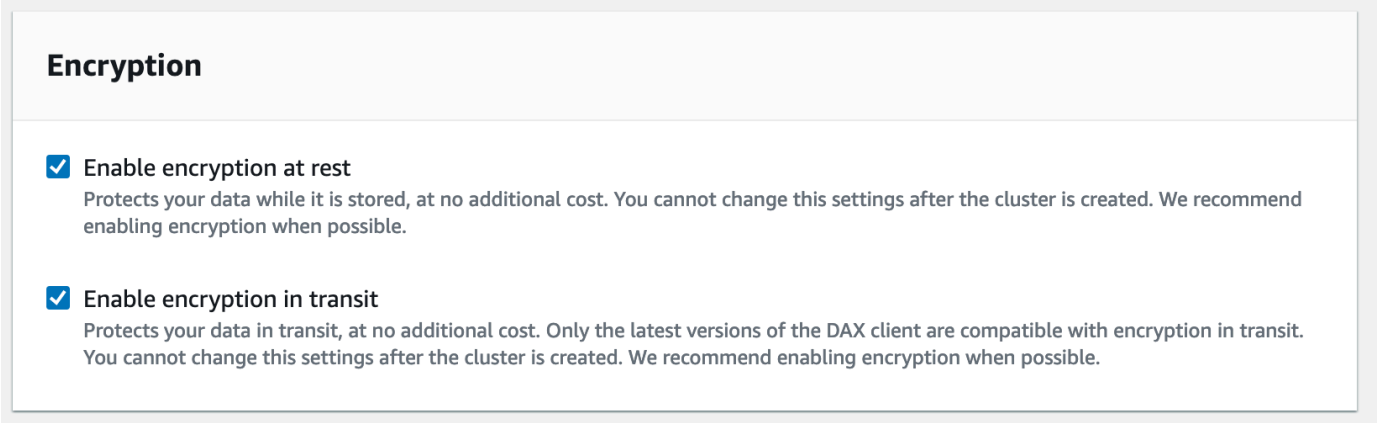
Aktivierung der Verschlüsselung im Ruhezustand mit dem AWS-Managementkonsole

Gehen Sie wie folgt vor, um die DAX-Verschlüsselung von Daten im Ruhezustand mit der Konsole zu aktivieren.

Um die DAX-Verschlüsselung im Ruhezustand zu aktivieren,

1. Melden Sie sich bei der an AWS-Managementkonsole und öffnen Sie die DynamoDB-Konsole unter. https://console.aws.amazon.com/dynamodb/
2. Wählen Sie im Navigationsbereich auf der linken Seite der Konsole unter DAX die Option Clusters (Cluster).
3. Wählen Sie Cluster erstellen.
4. Geben Sie unter Cluster name (Cluster-Name) einen Kurznamen für den Cluster ein. Wählen Sie einen Node type (Knotentyp) für alle Knoten im Cluster und geben Sie als Cluster-Größe **3** Knoten an.

5. Unter Encryption (Verschlüsselung) muss Enable encryption (Verschlüsselung aktivieren) ausgewählt sein.



6. Wählen Sie nach Auswahl von IAM-Rolle, Subnetzgruppe, Sicherheitsgruppen und Cluster-Einstellungen die Option Launch cluster (Cluster starten).

Sie können ermitteln, ob der Cluster verschlüsselt ist, indem Sie die Cluster-Details im Bereich Clusters (Cluster) prüfen. Die Verschlüsselung sollte ENABLED (AKTIVIERT) sein.

DAX-Verschlüsselung während der Übertragung

Amazon DynamoDB Accelerator (DAX) unterstützt die Verschlüsselung bei der Übertragung von Daten zwischen Ihrer Anwendung und Ihrem DAX-Cluster, sodass Sie DAX in Anwendungen mit strengen Verschlüsselungsanforderungen verwenden können.

Unabhängig davon, ob Sie die Verschlüsselung bei der Übertragung wählen oder nicht, bleibt der Datenverkehr zwischen Ihrer Anwendung und Ihrem DAX-Cluster in Ihrer Amazon VPC. Dieser Datenverkehr wird an Elastic-Network-Schnittstellen mit privaten IP-Adressen in Ihrer VPC weitergeleitet, die an die Knoten des Clusters angeschlossen sind. Mit Ihrer VPC als Vertrauensgrenze haben Sie erhebliche Kontrolle über die Sicherheit Ihrer Daten durch die Verwendung von Standardwerkzeugen wie Sicherheitsgruppen, Subnetzsegmentierung mit Netzwerk-ACLs und VPC-Ablaufverfolgung. Die DAX-Verschlüsselung bei der Übertragung erhöht diese Vertraulichkeitsstufe und stellt sicher, dass alle Anforderungen und Antworten zwischen der Anwendung und dem Cluster durch TLS (Transport Level Security) verschlüsselt werden und Verbindungen mit dem Cluster durch Überprüfung eines x509-Cluster-Zertifikats authentifiziert werden können. Daten, die von DAX auf die Festplatte geschrieben werden, können auch verschlüsselt werden, wenn Sie beim Erstellen Ihres DAX-Clusters die Verschlüsselung im Ruhezustand wählen.

Die Verschlüsselung bei der Übertragung mit DAX ist einfach. Wählen Sie einfach diese Option aus, wenn Sie einen neuen Cluster erstellen, und verwenden Sie eine aktuelle Version eines der DAX-Clients in Ihrer Anwendung. Cluster, die Verschlüsselung während der Übertragung verwenden, unterstützen keinen unverschlüsselten Datenverkehr. Daher besteht keine Möglichkeit, Ihre Anwendung falsch zu konfigurieren und die Verschlüsselung zu umgehen. Der DAX-Client verwendet das x509-Zertifikat des Clusters, um die Identität des Clusters zu authentifizieren, wenn er Verbindungen herstellt, und stellt sicher, dass Ihre DAX-Anforderungen an die gewünschte Stelle gehen. Alle Methoden zum Erstellen von DAX-Clustern unterstützen die Verschlüsselung bei der Übertragung: die AWS-Managementkonsole, AWS CLI, alle SDKs und CloudFormation.

Verschlüsselung bei der Übertragung kann auf einem vorhandenen DAX-Cluster nicht aktiviert werden. Um die Verschlüsselung bei der Übertragung in einer vorhandenen DAX-Anwendung zu verwenden, erstellen Sie einen neuen Cluster mit aktivierter Verschlüsselung im Transit, verlagern Sie den Datenverkehr Ihrer Anwendung und löschen Sie dann den alten Cluster.

Verwendung von serviceverknüpften IAM-Rollen für DAX

Amazon DynamoDB Accelerator (DAX) verwendet AWS Identity and Access Management (IAM) serviceverknüpfte Rollen. Eine serviceverknüpfte Rolle ist ein spezieller Typ einer IAM-Rolle, die direkt mit DAX verknüpft ist. Servicebezogene Rollen sind von DAX vordefiniert und beinhalten alle Berechtigungen, die der Service benötigt, um andere AWS Dienste in Ihrem Namen aufzurufen.

Eine serviceverknüpfte Rolle vereinfacht das Einrichten von DAX, da Sie die erforderlichen Berechtigungen nicht manuell hinzufügen müssen. DAX definiert die Berechtigungen seiner serviceverknüpften Rollen. Sofern keine andere Konfiguration festgelegt wurde, kann nur DAX die Rollen übernehmen. Die definierten Berechtigungen umfassen die Vertrauens- und Berechtigungsrichtlinie. Diese Berechtigungsrichtlinie kann an keine andere IAM-Entität angefügt werden.

Sie können die Rollen nur nach dem Löschen der zugehörigen Ressourcen löschen. Dies schützt Ihre DAX-Ressourcen, da Sie nicht versehentlich die Berechtigung für den Zugriff auf die Ressourcen entfernen können.

Informationen zu anderen Services, die serviceverlinkte Rollen unterstützen, finden Sie unter AWS -Services, die mit IAM funktionieren im IAM-Benutzerhandbuch. Suchen Sie nach den Services, für die Ja in der Spalte Serviceverknüpfte Rollen angegeben ist. Wählen Sie den Link Ja, um die Dokumentation zu serviceverknüpften Rollen für diesen Service anzuzeigen.

Themen

- Berechtigungen von serviceverknüpften Rollen für DAX
- Erstellen einer serviceverknüpften Rolle für DAX
- Bearbeiten einer serviceverknüpften Rolle für DAX
- Löschen einer serviceverknüpften Rolle für DAX

Berechtigungen von serviceverknüpften Rollen für DAX

DAX verwendet die serviceverknüpfte Rolle namens `AWSServiceRoleForDAX`. Diese Rolle ermöglicht es DAX, Services im Namen Ihres DAX-Clusters aufzurufen.

> **Important**
>
> Mit der serviceverknüpften Rolle `AWSServiceRoleForDAX` ist es einfacher für Sie, einen DAX-Cluster einzurichten und zu verwalten. Sie müssen jedoch weiterhin jedem Cluster Zugriff auf DynamoDB gewähren, bevor Sie ihn verwenden können. Weitere Informationen finden Sie unter DAX-Zugriffskontrolle.

Die serviceverknüpfte Rolle `AWSServiceRoleForDAX` vertraut darauf, dass die folgenden Services die Rolle annehmen:

- `dax.amazonaws.com`

Mit der Rollenberechtigungsrichtlinie kann DAX die folgenden Aktionen für die angegebenen Ressourcen ausführen:

- Aktionen auf `ec2`:
  - `AuthorizeSecurityGroupIngress`
  - `CreateNetworkInterface`
  - `CreateSecurityGroup`
  - `DeleteNetworkInterface`
  - `DeleteSecurityGroup`
  - `DescribeAvailabilityZones`
  - `DescribeNetworkInterfaces`

- `DescribeSecurityGroups`
- `DescribeSubnets`
- `DescribeVpcs`
- `ModifyNetworkInterfaceAttribute`
- `RevokeSecurityGroupIngress`

Sie müssen Berechtigungen konfigurieren, damit eine juristische Stelle von IAM (z. B. Benutzer, Gruppe oder Rolle) eine serviceverknüpfte Rolle erstellen, bearbeiten oder löschen kann. Weitere Informationen finden Sie unter serviceverknüpfte Rollenberechtigungen im IAM-Benutzerhandbuch.

Um es einer IAM-Entität zu ermöglichen, dienstbezogene AWSService RoleFor DAX-Rollen zu erstellen

Fügen Sie den Berechtigungen für diese IAM-Entität die folgende Richtlinienanweisung hinzu.

```
{
    "Effect": "Allow",
    "Action": [
        "iam:CreateServiceLinkedRole"
    ],
    "Resource": "*",
    "Condition": {"StringLike": {"iam:AWSServiceName": "dax.amazonaws.com"}}
}
```

Erstellen einer serviceverknüpften Rolle für DAX

Sie müssen eine serviceverknüpfte Rolle nicht manuell erstellen. Wenn Sie einen Cluster erstellen, erstellt DAX die dienstverknüpfte Rolle für Sie.

> **Important**
>
> Wenn Sie den DAX-Service vor dem 28. Februar 2018 verwendet haben, als er mit der Unterstützung servicebezogener Rollen begann, hat DAX die `AWSServiceRoleForDAX`-Rolle in Ihrem Konto erstellt. Weitere Informationen finden Sie unter Eine neue Rolle wurde in „Mein AWS Konto“ angezeigt im IAM-Benutzerhandbuch.

Wenn Sie diese serviceverknüpfte Rolle löschen und dann erneut erstellen müssen, können Sie die Rolle in Ihrem Konto mit demselben Verfahren neu anlegen. DAX erstellt die serviceverknüpfte Rolle erneut für Sie, wenn Sie eine Instance oder einen Cluster erstellen.

Bearbeiten einer serviceverknüpften Rolle für DAX

DAX verhindert die Bearbeitung der `AWSServiceRoleForDAX` serviceverknüpften Rolle. Da möglicherweise verschiedene Entitäten auf die Rolle verweisen, kann der Rollenname nach dem Erstellen einer serviceverknüpften Rolle nicht mehr geändert werden. Sie können jedoch die Beschreibung der Rolle mit IAM bearbeiten. Weitere Informationen finden Sie unter Bearbeiten einer serviceverknüpften Rolle im IAM-Benutzerhandbuch.

Löschen einer serviceverknüpften Rolle für DAX

Wenn Sie ein Feature oder einen Dienst, die bzw. der eine serviceverknüpften Rolle erfordert, nicht mehr benötigen, sollten Sie diese Rolle löschen. Auf diese Weise haben Sie keine ungenutzte juristische Stelle, die nicht aktiv überwacht oder verwaltet wird. Sie müssen jedoch alle Ihre DAX-Cluster löschen, bevor Sie die serviceverknüpfte Rolle löschen können.

Bereinigen einer serviceverknüpften Rolle

Bevor Sie mit IAM eine serviceverknüpfte Rolle löschen können, müssen Sie sich zunächst vergewissern, dass die Rolle über keine aktiven Sitzungen verfügt, und alle Ressourcen entfernen, die von der Rolle verwendet werden.

So überprüfen Sie in der IAM-Konsole, ob die serviceverknüpfte Rolle über eine aktive Sitzung verfügt

1. Melden Sie sich bei der an AWS-Managementkonsole und öffnen Sie die IAM-Konsole unter. https://console.aws.amazon.com/iam/
2. Wählen Sie im Navigationsbereich der IAM Console Roles (Rollen) aus. Wählen Sie dann den Namen (nicht das Kontrollkästchen) der AWSServiceRoleForDAX-Rolle aus.
3. Wählen Sie auf der Seite Summary für die ausgewählte Rolle die Registerkarte Access Advisor.
4. Überprüfen Sie auf der Registerkarte Access Advisor die jüngsten Aktivitäten für die serviceverknüpfte Rolle.

> **Note**
>
> Wenn Sie sich nicht sicher sind, ob DAX die Rolle `AWSServiceRoleForDAX` verwendet, können Sie versuchen, die Rolle zu löschen. Wenn der Service die Rolle verwendet, schlägt der Löschvorgang fehl und Sie können die Regionen anzeigen, in denen die Rolle verwendet wird. Wenn die Rolle verwendet wird, müssen Sie Ihre DAX-Cluster löschen, bevor Sie die Rolle löschen können. Die Sitzung für eine serviceverknüpfte Rolle können Sie nicht widerrufen.

Wenn Sie die Rolle `AWSServiceRoleForDAX` entfernen wollen, müssen Sie zunächst alle DAX-Cluster löschen.

Löschen Ihrer kompletten DAX-Cluster

Verwenden Sie eine dieser Verfahren, um alle Ihre DAX-Cluster zu löschen.

So löschen Sie einen DAX-Cluster (Konsole)

1. Öffnen Sie die DynamoDB-Konsole unter. https://console.aws.amazon.com/dynamodb/
2. Klicken Sie im Navigationsbereich unter DAX auf Cluster.
3. Wählen Sie Aktionen und anschließend Löschen aus.
4. Wählen Sie im Dialogfeld Löschbestätigung für den Cluster) die Option Löschen aus.

So löschen Sie einen DAX-Cluster (AWS CLI)

Sehen Sie Cluster löschen in der AWS CLI -Befehlsreferenz.

So löschen Sie einen DAX-Cluster (API)

Weitere Informationen finden Sie DeleteClusterin der Amazon DynamoDB DynamoDB-API-Referenz.

Löschen der serviceverknüpften Rolle

So löschen Sie die serviceverknüpfte Rolle mit IAM

Sie können die IAM-Konsole, die IAM-CLI oder die IAM-API verwenden, um die `AWSServiceRoleForDAX`-serviceverknüpfte Rolle zu löschen. Weitere Informationen finden Sie unter Löschen einer serviceverknüpften Rolle im IAM-Leitfaden.

AWS Kontenübergreifender Zugriff auf DAX

Stellen Sie sich vor, Sie haben einen DynamoDB Accelerator (DAX) -Cluster, der in einem AWS Konto (Konto A) läuft, und der DAX-Cluster muss von einer Amazon Elastic Compute Cloud (Amazon EC2) -Instance in einem anderen AWS Konto (Konto B) aus zugänglich sein. In diesem Tutorial erreichen Sie dies, indem Sie eine EC2-Instance in Konto B mit einer IAM-Rolle in Konto B starten. Anschließend verwenden Sie temporäre Sicherheitsanmeldeinformationen von der EC2-Instance, um eine IAM-Rolle in Konto A zu übernehmen. Schließlich verwenden Sie die temporären Sicherheitsanmeldeinformationen von der übernommenen IAM-Rolle in Konto A, um Anwendungsaufrufe über eine VPC-Peering-Verbindung an den DAX-Cluster in Konto A auszugeben. Um diese Aufgaben ausführen zu können, benötigen Sie Administratorzugriff in beiden AWS Konten.

> **Important**
>
> Es ist nicht möglich, dass ein DAX-Cluster von einem anderen Konto aus auf eine DynamoDB-Tabelle zugreift.

Themen

- IAM-einrichten
- Einrichten einer VPC
- Ändern des DAX-Clients, um den kontoübergreifenden Zugriff zu erlauben

IAM-einrichten

1. Erstellen Sie die Textdatei `AssumeDaxRoleTrust.json` mit dem folgenden Inhalt, der es Amazon EC2 ermöglicht, in Ihrem Auftrag zu arbeiten.

   JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "Service": "ec2.amazonaws.com"
```

```
            },
            "Action": "sts:AssumeRole"
        }
    ]
}
```

2. Erstellen Sie in Konto B eine Rolle, die von Amazon EC2 beim Starten von Instances verwendet werden kann.

```
aws iam create-role \
    --role-name AssumeDaxRole \
    --assume-role-policy-document file://AssumeDaxRoleTrust.json
```

3. Erstellen Sie eine Textdatei `AssumeDaxRolePolicy.json` mit dem folgenden Inhalt, die es Code ermöglicht, der auf der EC2-Instance in Konto B ausgeführt wird, eine IAM-Rolle in Konto A anzunehmen. *`accountA`* Ersetzen Sie sie durch die tatsächliche ID von Konto A.

   JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "sts:AssumeRole",
            "Resource": "arn:aws:iam::111122223333:role/DaxCrossAccountRole"
        }
    ]
}
```

4. Fügen Sie diese Richtlinie der gerade erstellten Rolle hinzu.

```
aws iam put-role-policy \
    --role-name AssumeDaxRole \
    --policy-name AssumeDaxRolePolicy \
    --policy-document file://AssumeDaxRolePolicy.json
```

5. Erstellen Sie ein Instance-Profil, damit Instances die Rolle verwenden können.

```
aws iam create-instance-profile \
    --instance-profile-name AssumeDaxInstanceProfile
```

6. Ordnen Sie die Rolle dem Instance-Profil zu.

```
aws iam add-role-to-instance-profile \
    --instance-profile-name AssumeDaxInstanceProfile \
    --role-name AssumeDaxRole
```

7. Erstellen Sie die Textdatei `DaxCrossAccountRoleTrust.json` mit dem folgenden Inhalt, der es Konto B gestattet, eine Rolle von Kontos A zu übernehmen. *`accountB`*Durch die tatsächliche ID von Konto B ersetzen.

   JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "AWS": "arn:aws:iam::111122223333:role/AssumeDaxRole"
            },
            "Action": "sts:AssumeRole"
        }
    ]
}
```

8. Erstellen Sie in Konto A die Rolle, die Konto B übernehmen kann.

```
aws iam create-role \
    --role-name DaxCrossAccountRole \
    --assume-role-policy-document file://DaxCrossAccountRoleTrust.json
```

9. Erstellen Sie eine Textdatei mit dem Namen `DaxCrossAccountPolicy.json`, die den Zugriff auf den DAX-Cluster ermöglicht. *`dax-cluster-arn`*Ersetzen Sie es durch den richtigen Amazon-Ressourcennamen (ARN) Ihres DAX-Clusters.

   JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
```

```
        {
            "Effect": "Allow",
            "Action": [
                "dax:GetItem",
                "dax:BatchGetItem",
                "dax:Query",
                "dax:Scan",
                "dax:PutItem",
                "dax:UpdateItem",
                "dax:DeleteItem",
                "dax:BatchWriteItem",
                "dax:ConditionCheckItem"
            ],
            "Resource": "arn:aws:dax:us-east-1:111122223333:cache/dax-
cluster-name"
        }
    ]
}
```

10. Fügen Sie in Konto A die Richtlinie zur Rolle hinzu.

```
aws iam put-role-policy \
    --role-name DaxCrossAccountRole \
    --policy-name DaxCrossAccountPolicy \
    --policy-document file://DaxCrossAccountPolicy.json
```

Einrichten einer VPC

1. Suchen Sie die Subnetzgruppe des DAX-Clusters von Konto A. *cluster-name*Ersetzen Sie es durch den Namen des DAX-Clusters, auf den Konto B zugreifen muss.

```
aws dax describe-clusters \
    --cluster-name cluster-name
    --query 'Clusters[0].SubnetGroup'
```

2. Suchen Sie damit *subnet-group* die VPC des Clusters.

```
aws dax describe-subnet-groups \
    --subnet-group-name subnet-group \
    --query 'SubnetGroups[0].VpcId'
```

3. Suchen Sie damit *vpc-id* den CIDR der VPC.

```
aws ec2 describe-vpcs \
    --vpc vpc-id \
    --query 'Vpcs[0].CidrBlock'
```

4. Erstellen Sie in Konto B eine VPC mit einer anderen, nicht überlappenden CIDR als der, die im im vorherigen Schritt gefunden wurde. Erstellen Sie dann mindestens ein Subnetz. Sie können entweder den VPC-Erstellungsassistenten in AWS-Managementkonsole oder in verwenden. AWS CLI

5. Fordern Sie in Konto B eine Peering-Verbindung mit der VPC von Konto A an, wie unter Erstellen und Akzeptieren einer VPC-Peering-Verbindung beschrieben. Akzeptieren Sie die Verbindung in Konto A.

6. Suchen Sie in Konto B die Routingtabelle der neuen VPC. *vpc-id*Ersetzen Sie durch die ID der VPC, die Sie in Konto B erstellt haben.

```
aws ec2 describe-route-tables \
    --filters 'Name=vpc-id,Values=vpc-id' \
    --query 'RouteTables[0].RouteTableId'
```

7. Fügen Sie eine Route hinzu, um Datenverkehr für die CIDR von Konto A an die VPC-Peering-Verbindung zu senden. Denken Sie daran, jeden Wert *user input placeholder* durch die richtigen Werte für Ihre Konten zu ersetzen.

```
aws ec2 create-route \
    --route-table-id accountB-route-table-id \
    --destination-cidr accountA-vpc-cidr \
    --vpc-peering-connection-id peering-connection-id
```

8. Suchen Sie von Konto A aus nach der Routing-Tabelle des DAX-Clusters, indem *vpc-id* Sie die zuvor gefundene Tabelle verwenden.

```
aws ec2 describe-route-tables \
    --filters 'Name=vpc-id, Values=accountA-vpc-id' \
    --query 'RouteTables[0].RouteTableId'
```

9. Fügen Sie in Konto A eine Route hinzu, um Datenverkehr für die CIDR von Konto B an die VPC-Peering-Verbindung zu senden. Ersetzen Sie jeden *user input placeholder* Wert durch die richtigen Werte für Ihre Konten.

```
aws ec2 create-route \
    --route-table-id accountA-route-table-id \
    --destination-cidr accountB-vpc-cidr \
    --vpc-peering-connection-id peering-connection-id
```

10. Starten Sie in Konto B eine EC2-Instance in der zuvor von Ihnen erstellten VPC. Ordnen Sie ihr das `AssumeDaxInstanceProfile` zu. Sie können entweder den Startassistenten im AWS-Managementkonsole oder im verwenden AWS CLI. Notieren Sie sich die Sicherheitsgruppe der Instance.
11. Suchen Sie in Konto A die Sicherheitsgruppe, die vom DAX-Cluster verwendet wird. Denken Sie daran, es *`cluster-name`* durch den Namen Ihres DAX-Clusters zu ersetzen.

```
aws dax describe-clusters \
    --cluster-name cluster-name \
    --query 'Clusters[0].SecurityGroups[0].SecurityGroupIdentifier'
```

12. Aktualisieren Sie die Sicherheitsgruppe des DAX-Clusters, um eingehenden Datenverkehr von der Sicherheitsgruppe der EC2-Instance zuzulassen, die Sie in Konto B erstellt haben. Denken Sie daran, die durch die *`user input placeholders`* richtigen Werte für Ihre Konten zu ersetzen.

```
aws ec2 authorize-security-group-ingress \
    --group-id accountA-security-group-id \
    --protocol tcp \
    --port 8111 \
    --source-group accountB-security-group-id \
    --group-owner accountB-id
```

An dieser Stelle kann eine Anwendung auf der EC2-Instance von Konto B mithilfe des Instance-Profils die Rolle `arn:aws:iam::`*`accountA-id`*`:role/DaxCrossAccountRole` übernehmen und den DAX-Cluster nutzen.

Ändern des DAX-Clients, um den kontoübergreifenden Zugriff zu erlauben

> **Note**
>
> AWS -Security-Token-Service (AWS STS) Anmeldeinformationen sind temporäre Anmeldeinformationen. Einige Clients nehmen automatisch Aktualisierungen vor, während

andere zusätzliche Logik benötigen, um die Anmeldeinformationen zu aktualisieren. Wir empfehlen Ihnen, die Anleitung der entsprechenden Dokumentation zu befolgen.

Java

Dieser Abschnitt unterstützt Sie beim Ändern Ihres vorhandenen DAX-Clientcodes, um kontoübergreifenden DAX-Zugriff zu ermöglichen. Wenn Sie noch keinen DAX-Clientcode haben, finden Sie Beispiele von funktionierendem Code im Tutorial Java und DAX.

1. Fügen Sie die folgenden Importe hinzu.

```
import com.amazonaws.auth.STSAssumeRoleSessionCredentialsProvider;
import com.amazonaws.services.securitytoken.AWSSecurityTokenService;
import
 com.amazonaws.services.securitytoken.AWSSecurityTokenServiceClientBuilder;
```

2. Rufen Sie einen Anbieter für Anmeldeinformationen von ab AWS STS und erstellen Sie ein DAX-Client-Objekt. Denken Sie daran, jeden Wert *user input placeholder* durch die richtigen Werte für Ihre Konten zu ersetzen.

```
AWSSecurityTokenService awsSecurityTokenService =
 AWSSecurityTokenServiceClientBuilder
     .standard()
     .withRegion(region)
     .build();

STSAssumeRoleSessionCredentialsProvider credentials =  new
 STSAssumeRoleSessionCredentialsProvider.Builder("arn:aws:iam::accountA:role/
RoleName", "TryDax")
     .withStsClient(awsSecurityTokenService)
     .build();

DynamoDB client = AmazonDaxClientBuilder.standard()
    .withRegion(region)
    .withEndpointConfiguration(dax_endpoint)
    .withCredentials(credentials)
    .build();
```

.NET

Dieser Abschnitt unterstützt Sie beim Ändern Ihres vorhandenen DAX-Clientcodes, um kontoübergreifenden DAX-Zugriff zu ermöglichen. Wenn Sie noch keinen DAX-Clientcode haben, finden Sie Beispiele von funktionierendem Code im Tutorial .NET und DAX.

1. Füge das hinzu AWSSDK. SecurityToken NuGet Paket zur Lösung.

```
<PackageReference Include="AWSSDK.SecurityToken" Version="latest version" />
```

2. Verwenden Sie die Pakete `SecurityToken` und `SecurityToken.Model`.

```
using Amazon.SecurityToken;
using Amazon.SecurityToken.Model;
```

3. Fordern Sie temporäre Anmeldeinformationen von `AmazonSimpleTokenService` an und erstellen Sie ein `ClusterDaxClient`-Objekt. Denken Sie daran, jeden Wert *`user input placeholder`* durch die richtigen Werte für Ihre Konten zu ersetzen.

```
IAmazonSecurityTokenService sts = new AmazonSecurityTokenServiceClient();

var assumeRoleResponse = sts.AssumeRole(new AssumeRoleRequest
{
    RoleArn = "arn:aws:iam::accountA:role/RoleName",
                RoleSessionName = "TryDax"
});

Credentials credentials = assumeRoleResponse.Credentials;

var clientConfig = new DaxClientConfig(dax_endpoint, port)
{
    AwsCredentials = assumeRoleResponse.Credentials

};

var client = new ClusterDaxClient(clientConfig);
```

Go

Dieser Abschnitt unterstützt Sie beim Ändern Ihres vorhandenen DAX-Clientcodes, um kontoübergreifenden DAX-Zugriff zu ermöglichen. Falls Sie noch keinen DAX-Client-Code haben, finden Sie funktionierende Codebeispiele unter GitHub.

1. Importieren Sie die Pakete AWS STS und die Sitzungspakete.

```
import (
    "github.com/aws/aws-sdk-go/aws/session"
    "github.com/aws/aws-sdk-go/service/sts"
    "github.com/aws/aws-sdk-go/aws/credentials/stscreds"
)
```

2. Fordern Sie temporäre Anmeldeinformationen von `AmazonSimpleTokenService` an und erstellen Sie ein DAX-Clientobjekt. Denken Sie daran, jeden Wert *`user input placeholder`* durch die richtigen Werte für Ihre Konten zu ersetzen.

```
sess, err := session.NewSession(&aws.Config{
    Region: aws.String(region)},
)
if err != nil {
    return nil, err
}

stsClient := sts.New(sess)
arp := &stscreds.AssumeRoleProvider{
                Duration:     900 * time.Second,
                ExpiryWindow: 10 * time.Second,
                RoleARN:      "arn:aws:iam::accountA:role/role_name",
                Client:       stsClient,
                RoleSessionName: "session_name",
        }cfg := dax.DefaultConfig()

cfg.HostPorts = []string{dax_endpoint}
cfg.Region = region
cfg.Credentials = credentials.NewCredentials(arp)
daxClient := dax.New(cfg)
```

Python

Dieser Abschnitt unterstützt Sie beim Ändern Ihres vorhandenen DAX-Clientcodes, um kontoübergreifenden DAX-Zugriff zu ermöglichen. Wenn Sie noch keinen DAX-Clientcode haben, finden Sie Beispiele von funktionierendem Code im Tutorial Python und DAX.

1. Importieren Sie `boto3`.

```
import boto3
```

2. Fordern Sie temporäre Anmeldeinformationen von `sts` an und erstellen Sie ein `AmazonDaxClient`-Objekt. Denken Sie daran, jeden Wert *`user input placeholder`* durch die richtigen Werte für Ihre Konten zu ersetzen.

```
sts = boto3.client('sts')
stsresponse =
 sts.assume_role(RoleArn='arn:aws:iam::accountA:role/
RoleName',RoleSessionName='tryDax')
credentials = botocore.session.get_session()['Credentials']

dax = amazondax.AmazonDaxClient(session, region_name=region,
 endpoints=[dax_endpoint], aws_access_key_id=credentials['AccessKeyId'],
 aws_secret_access_key=credentials['SecretAccessKey'],
 aws_session_token=credentials['SessionToken'])
client = dax
```

Node.js

Dieser Abschnitt unterstützt Sie beim Ändern Ihres vorhandenen DAX-Clientcodes, um kontoübergreifenden DAX-Zugriff zu ermöglichen. Wenn Sie noch keinen DAX-Clientcode haben, finden Sie Beispiele von funktionierendem Code im Tutorial Node.js und DAX. Denken Sie daran, jeden Wert *`user input placeholder`* durch die richtigen Werte für Ihre Konten zu ersetzen.

```
const AmazonDaxClient = require('amazon-dax-client');
const AWS = require('aws-sdk');
const region = 'region';
const endpoints = [daxEndpoint1, ...];

const getCredentials = async() => {
  return new Promise((resolve, reject) => {
    const sts = new AWS.STS();
```

```
    const roleParams = {
      RoleArn: 'arn:aws:iam::accountA:role/RoleName',
      RoleSessionName: 'tryDax',
    };
    sts.assumeRole(roleParams, (err, session) => {
      if(err) {
        reject(err);
      } else {
        resolve({
          accessKeyId: session.Credentials.AccessKeyId,
          secretAccessKey: session.Credentials.SecretAccessKey,
          sessionToken: session.Credentials.SessionToken,
        });
      }
    });
  });
};

const createDaxClient = async() => {
  const credentials = await getCredentials();
  const daxClient = new AmazonDaxClient({endpoints: endpoints, region: region,
 accessKeyId: credentials.accessKeyId, secretAccessKey: credentials.secretAccessKey,
 sessionToken: credentials.sessionToken});
  return new AWS.DynamoDB.DocumentClient({service: daxClient});
};

createDaxClient().then((client) => {
  client.get(...);
  ...
}).catch((error) => {
  console.log('Caught an error: ' + error);
});
```

DAX-Clustergrößenleitfaden

In diesem Handbuch finden Sie Hinweise zur Auswahl einer geeigneten Amazon-DynamoDB-Accelerator-(DAX)-Clustergröße und des Knotentyps für Ihre Anwendung. Diese Anweisungen führen Sie durch die Schritte zum Schätzen des DAX-Datenverkehrs Ihrer Anwendung, zur Auswahl einer Clusterkonfiguration und zum Testen dieser Anwendung.

Wenn Sie über einen vorhandenen DAX-Cluster verfügen und prüfen möchten, ob er die entsprechende Anzahl und Größe von Knoten hat, lesen Sie bitte Skalieren eines DAX-Clusters.

Themen

- -Übersicht
- Schätzung des Datenverkehrs
- Lasttest

-Übersicht

Es ist wichtig, den DAX-Cluster entsprechend für Ihre Workload zu skalieren, unabhängig davon, ob Sie einen neuen Cluster erstellen oder einen vorhandenen Cluster verwalten. Wenn die Zeit vergeht und sich der Workload Ihrer Anwendung ändert, sollten Sie Ihre Skalierungsentscheidungen regelmäßig überprüfen, um sicherzustellen, dass sie weiterhin angemessen sind.

Der Prozess führt in der Regel folgende Schritte aus:

1. Schätzung des Datenverkehrs. In diesem Schritt machen Sie Vorhersagen über das Datenvolumen, das Ihre Anwendung an DAX senden wird, die Art des Datenverkehrs (Lese- vs. Schreibvorgänge) und die erwartete Cache-Trefferrate.
2. Durchführung eines Lasttests. In diesem Schritt erstellen Sie einen Cluster und senden Datenverkehr an ihn und spiegeln dabei Ihre Schätzungen aus dem vorherigen Schritt. Wiederholen Sie diesen Schritt, bis Sie eine geeignete Clusterkonfiguration gefunden haben.
3. Produktionsüberwachung. Während Ihre Anwendung DAX in der Produktion verwendet, sollten Sie den Cluster überwachen, um kontinuierlich zu überprüfen, ob er immer noch korrekt skaliert ist, wenn sich Ihre Workload im Laufe der Zeit ändert.

Schätzung des Datenverkehrs

Es gibt drei Hauptfaktoren, die eine typische DAX-Workload charakterisieren:

- Cache-Trefferrate
- Lesekapazitätseinheiten (RCUs) pro Sekunde
- Schreibkapazitätseinheiten (WCUs) pro Sekunde

Schätzen der Cache-Zugriffsrate

Wenn Sie bereits über einen DAX-Cluster verfügen, können Sie anhand der CloudWatch Metriken `ItemCacheHits` und `ItemCacheMisses` Amazon die Cache-Trefferquote ermitteln. Die Cache-Zugriffsrate ist gleich `ItemCacheHits` / (`ItemCacheHits` + `ItemCacheMisses`). Wenn Ihr Workload `Query`- oder `Scan`-Operationen beinhaltet, sollten Sie sich auch die Metriken `QueryCacheHits`, `QueryCacheMisses`, `ScanCacheHits` und `ScanCacheMisses` ansehen. Die Cache-Trefferraten variieren von Anwendung zu Anwendung und werden stark von der TTL-Einstellung (Time to Live) des Clusters beeinflusst. Typische Trefferraten für Anwendungen, die DAX verwenden, liegen bei 85-95 Prozent.

Schätzen von Lese- und Schreibkapazitätseinheiten

Wenn Sie bereits über DynamoDB-Tabellen für Ihre Anwendung verfügen, sehen Sie sich die Metriken `ConsumedReadCapacityUnits` und `ConsumedWriteCapacityUnits` CloudWatch an. Verwenden Sie die Statistik `Sum` und dividieren Sie durch die Anzahl der Sekunden im Zeitraum.

Wenn Sie auch bereits über einen DAX-Cluster verfügen, denken Sie daran, dass die DynamoDB-Metrik `ConsumedReadCapacityUnits` nur Cache-Fehlschläge berücksichtigt. Um eine Vorstellung von den Lesekapazitätseinheiten pro Sekunde zu erhalten, die von Ihrem DAX-Cluster verarbeitet werden, teilen Sie die Zahl durch Ihre Cache-Fehlerrate (d. h. 1 - Cache-Trefferrate).

Wenn Sie noch keine DynamoDB-Tabelle haben, lesen Sie die Dokumentation zu Lese- und Schreibkapazitätseinheiten, um den Datenverkehr basierend auf der geschätzten Anforderungsrate Ihrer Anwendung, der Anzahl der Elemente, auf die pro Anforderung zugegriffen wird, und der Elementgröße zu schätzen.

Planen Sie bei der Erstellung von Datenverkehrsschätzungen das zukünftige Wachstum sowie die erwarteten und unerwarteten Spitzen, um sicherzustellen, dass Ihr Cluster genügend Spielraum für den Datenverkehr hat.

Lasttest

Der nächste Schritt nach der Schätzung des Datenverkehrs besteht darin, die Clusterkonfiguration unter Last zu testen.

1. Für den ersten Lasttest empfehlen wir, dass Sie mit dem `dax.r4.large`-Knotentyp beginnen, dem speicheroptimierten Knotentyp mit einer Leistung mit den niedrigsten Fixkosten.
2. Ein fehlertoleranter Cluster erfordert mindestens drei Knoten, verteilt auf drei Availability Zones. Wenn in diesem Fall eine Availability Zone nicht mehr verfügbar ist, wird die effektive Anzahl von

Availability Zones um ein Drittel reduziert. Für den ersten Lasttest empfehlen wir, mit einem Cluster mit zwei Knoten zu beginnen, der den Ausfall einer Availability Zone in einem Cluster mit drei Knoten simuliert.

3. Leiten Sie für die Dauer des Lasttests anhaltenden Datenverkehr (wie im vorherigen Schritt geschätzt) zu Ihrem Testcluster.
4. Überwachen Sie die Leistung des Clusters während des Lasttests.

Idealerweise sollte das Datenverkehrsprofil, das Sie während des Lasttests steuern, dem realen Datenverkehr Ihrer Anwendung so ähnlich wie möglich sein. Dazu gehören die Aufteilung der Vorgänge (z. B. 70 Prozent `GetItem`, 25 Prozent `Query` und 5 Prozent `PutItem`), die Anforderungsrate für jeden Vorgang, die Anzahl der Elemente, auf die pro Anforderung zugegriffen wird, und die Verteilung der Elementgrößen. Um eine Cache-Zugriffsrate zu erreichen, die der erwarteten Cache-Zugriffsrate Ihrer Anwendung entspricht, achten Sie genau auf die Verteilung der Schlüssel im Testdatenverkehr.

> **Note**
>
> Seien Sie vorsichtig, wenn Sie die Last der T2-Knotentypen (`dax.t2.small` und `dax.t2.medium`) testen. T2-Knotentypen bieten eine dynamische CPU-Leistung, die je nach CPU-Guthaben des Knotens im Laufe der Zeit variiert. Ein DAX-Cluster, der auf T2-Knoten ausgeführt wird, scheint normal zu funktionieren, aber wenn einer der Knoten die Basisleistung seiner Instance überschreitet, gibt der Knoten sein aufgelaufenes CPU-Guthaben aus. Wenn der Gutschriftensaldo niedrig ist, wird die Leistung schrittweise auf das Basisleistungsniveau gesenkt.

Überwachen Sie Ihren DAX-Cluster während des Lasttests, um festzustellen, ob der Knotentyp, den Sie für den Lasttest verwenden, der richtige Knotentyp für Sie ist. Darüber hinaus sollten Sie während eines Lasttests Ihre Anforderungsrate und die Cache-Zugriffsrate überwachen, um sicherzustellen, dass Ihre Testinfrastruktur tatsächlich den von Ihnen beabsichtigten Datenverkehr steuert.

Sie sollten auf den Netzwerk-Byte-Verbrauch Ihres ausgewählten Cluster-Instance-Typs achten. Eine Überschreitung der verfügbaren Basisbandbreite für eine Amazon-EC2-Instance weist darauf hin, dass Ihr Cluster die Workload Ihrer Anwendung möglicherweise nicht unterstützt und skaliert werden muss.

Wenn Auslastungstests darauf hindeuten, dass die ausgewählte Clusterkonfiguration die Workload-Auslastung Ihrer Anwendung nicht aufrechterhalten kann, wird empfohlen, zu einem größeren Knotentyp zu wechseln, insbesondere wenn Sie eine hohe CPU-Auslastung auf dem Primärknoten im Cluster, hohe Bereinigungsraten oder eine hohe Cachespeicherauslastung feststellen. Wenn die Zugriffsraten konstant hoch sind und das Verhältnis zwischen Lese- und Schreibdatenverkehr hoch ist, sollten Sie dem Cluster weitere Knoten hinzufügen. Weitere Hinweise zum Verwenden eines größeren Knotentyps (vertikale Skalierung) oder zum Hinzufügen weiterer Knoten (horizontale Skalierung) finden Sie unter Skalieren eines DAX-Clusters.

Sie sollten den Lasttest wiederholen, nachdem Sie Änderungen an der Clusterkonfiguration vorgenommen haben.

Datenmodellierung für DynamoDB-Tabellen

Bevor wir uns mit der Datenmodellierung befassen, ist es wichtig, einige Grundlagen von DynamoDB zu klären. DynamoDB ist eine NoSQL-Schlüsselwert-Datenbank, die ein flexibles Schema ermöglicht. Der Satz von Datenattributen kann, abgesehen von den Schlüsselattributen für jedes Element, entweder einheitlich oder diskret sein. Das DynamoDB-Schlüsselschema hat entweder die Form eines einfachen Primärschlüssels, bei dem ein Partitionsschlüssel ein Element eindeutig identifiziert, oder es hat die Form eines zusammengesetzten Primärschlüssels, bei dem eine Kombination aus Partitionsschlüssel und Sortierschlüssel ein Element eindeutig definiert. Der Partitionsschlüssel wird gehasht, um den physischen Speicherort der Daten zu ermitteln und sie abzurufen. Daher ist es wichtig, ein Attribut mit hoher Kardinalität und horizontal skalierbarem Attribut als Partitionsschlüssel auszuwählen, um eine gleichmäßige Verteilung der Daten zu gewährleisten. Das Sortierschlüsselattribut ist im Schlüsselschema optional. Ein Sortierschlüssel ermöglicht die Modellierung von 1:n-Beziehungen und die Erstellung von Elementsammlungen in DynamoDB. Sortierschlüssel werden auch als Bereichsschlüssel bezeichnet. Sie werden verwendet, um Elemente in einer Artikelsammlung zu sortieren und ermöglichen auch flexible bereichsbasierte Operationen.

Weitere Informationen und bewährte Methoden zum DynamoDB-Schlüsselschema finden Sie im Folgenden:

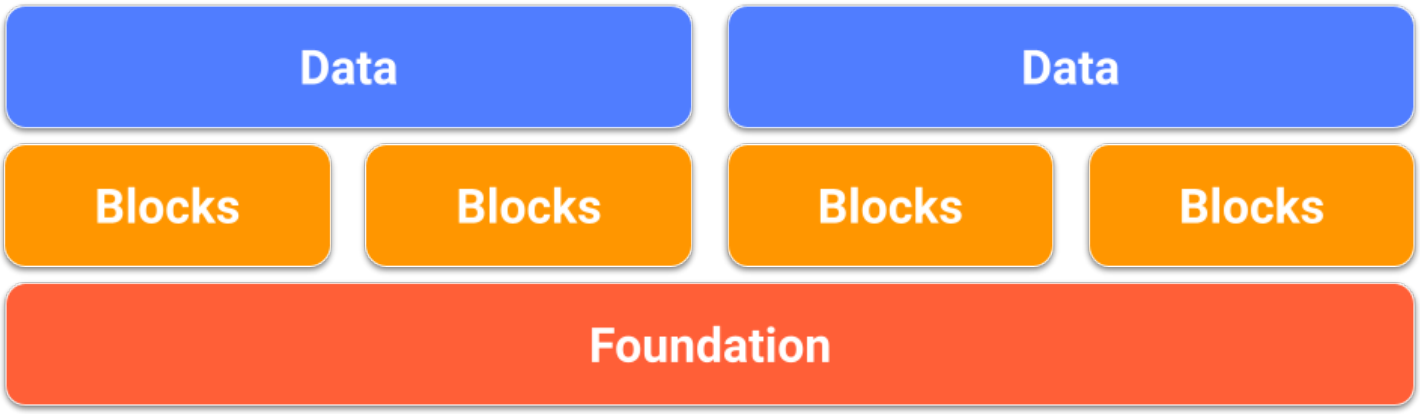
- the section called “Partitionen und Datenverteilung in DynamoDB”
- the section called “Entwerfen von Partitionsschlüsseln”
- the section called “Sortierschlüsselentwurf”
- Auswahl des richtigen DynamoDB-Partitionsschlüssels

Sekundäre Indizes werden häufig benötigt, um zusätzliche Abfragemuster in DynamoDB zu unterstützen. Sekundäre Indizes sind Schattentabellen, in denen dieselben Daten über ein anderes Schlüsselschema organisiert sind als in der Basistabelle. Ein lokaler sekundärer Index (LSI) verwendet denselben Partitionsschlüssel wie die Basistabelle und ermöglicht die Verwendung eines alternativen Sortierschlüssels, mit dem er die Kapazität der Basistabelle gemeinsam nutzen kann. Ein globaler sekundärer Index (GSI) kann einen anderen Partitionsschlüssel sowie ein anderes Sortierschlüsselattribut als die Basistabelle haben, was bedeutet, dass das Durchsatzmanagement für einen GSI unabhängig von der Basistabelle ist.

Weitere Informationen zu Sekundärindizes und bewährte Methoden finden Sie im Folgenden:

- the section called “Arbeiten mit Indizes”
- the section called “Sekundäre Indizes”

Schauen wir uns die Datenmodellierung nun etwas genauer an. Das Entwerfen eines flexiblen und hochgradig optimierten Schemas für DynamoDB oder eine beliebige NoSQL-Datenbank kann eine anspruchsvolle Fertigkeit sein. Dieses Modul soll Sie bei der Entwicklung eines mentalen Flussdiagramms für das Design eines Schemas unterstützen, das Sie vom Anwendungsfall zur Produktion führt. Wir beginnen mit einer Einführung in die grundlegende Wahl eines Designs: ein Design mit einer einzelnen Tabelle oder ein Design mit mehreren Tabellen. Anschließend überprüfen wir die Vielzahl von Designmustern (Bausteinen), die verwendet werden können, um verschiedene Organisations- oder Leistungsergebnisse für Ihre Anwendung zu erzielen. Schließlich bieten wir eine Vielzahl an vollständigen Schemadesign-Paketen für verschiedene Anwendungsfälle und Branchen an.



Themen

- Elementsammlungen — wie man one-to-many Beziehungen in DynamoDB modelliert
- Grundlagen der Datenmodellierung in DynamoDB
- Bausteine der Datenmodellierung in DynamoDB
- Pakete für das Schemadesign für die Datenmodellierung in DynamoDB
- Bewährte Methoden für die Modellierung relationaler Daten in DynamoDB

Elementsammlungen — wie man one-to-many Beziehungen in DynamoDB modelliert

In DynamoDB stellt eine Elementauflistung eine Gruppe von Elementen dar, die denselben Partitionsschlüsselwert haben, was bedeutet, dass die Elemente verwandt sind. Elementsammlungen sind der wichtigste Mechanismus zum Modellieren von one-to-many Beziehungen in DynamoDB. Elementauflistungen können nur für Tabellen oder Indizes vorhanden sein, die für die Verwendung eines zusammengesetzten Primärschlüssels konfiguriert sind.

> **Note**
>
> Elementauflistungen können entweder in einer Basistabelle oder einem sekundären Index vorhanden sein. Weitere Informationen darüber, wie Elementauflistungen mit Indizes interagieren, finden Sie unter Elementauflistungen in lokalen sekundären Indizes.

Betrachten Sie die folgende Tabelle, die drei verschiedene Benutzer und ihr In-Game-Inventar zeigt:

Primary key		Attributes		
Partition key: PK	**Sort key: SK**			
account1234	inventory::armor	data		
		{"armor": [{"name" : "Pauldron of the Paladin", "type" : "chest", "gear score" : 545 }, { "name" : "Greaves of the Ranger", "type" : "sword", "gear score" : 382}]}		
	inventory::weapons	data		
		{"weapons" : [{ "name" : "Sword of the Ancients", "type" : "sword", "gear score" : 320}]}		
	login-data	pw	state	last-login
		d1e8a70b5ccab1dc2f56bbf7e99f064a660c08e361a35751b9c483c88943d082	Active	1649276737
account1387	info	data		
		{"email" : "bot123@gmail.com"}		
	inventory::armor	data		
		{"armor": [{"name" : "Pauldron of the Paladin", "type" : "chest", "gear score" : 545 }, { "name" : "Greaves of the Ranger", "type" : "sword", "gear score" : 382}]}		
	login-data	pw	state	last-login
		k2g8jk0m5ppab1dc2f56bbf7e99f064a660c08e361a35751b9c464r23943i082	Banned	1649456737
account1138	info	data		
		{"email" : "luh-3417@gmail.com" }		
	login-data	pw	state	last-login
		88A41A9A62B11CCC8C120861928765A3EA41DEB9EAFE261D90F619473B89A2D4	Active	14275516966

Für einige Elemente in jeder Auflistung ist der Sortierschlüssel eine Verkettung, die aus Informationen besteht, die zum Gruppieren von Daten verwendet werden, z. B. `inventory::armor`, `inventory::weapon` oder `info`. Jede Elementauflistung kann eine andere Kombination

dieser Attribute als Sortierschlüssel haben. Benutzer `account1234` verfügt über ein `inventory::weapons`-Element, Benutzer `account1387` hingegen nicht (weil er noch keins gefunden hat). Benutzer `account1138` verwendet nur zwei Elemente für seinen Sortierschlüssel (da er noch kein Inventar hat), während die anderen Benutzer drei verwenden.

Mit DynamoDB können Sie Elemente aus diesen Elementauflistungen selektiv abrufen, um folgende Vorgänge auszuführen:

- Abrufen aller Elemente von einem bestimmten Benutzer
- Abrufen nur eines Elements von einem bestimmten Benutzer
- Abrufen aller Elemente eines bestimmten Typs, der zu einem bestimmten Benutzer gehört

Beschleunigen von Abfragen durch Organisieren der Daten mithilfe von Elementauflistungen

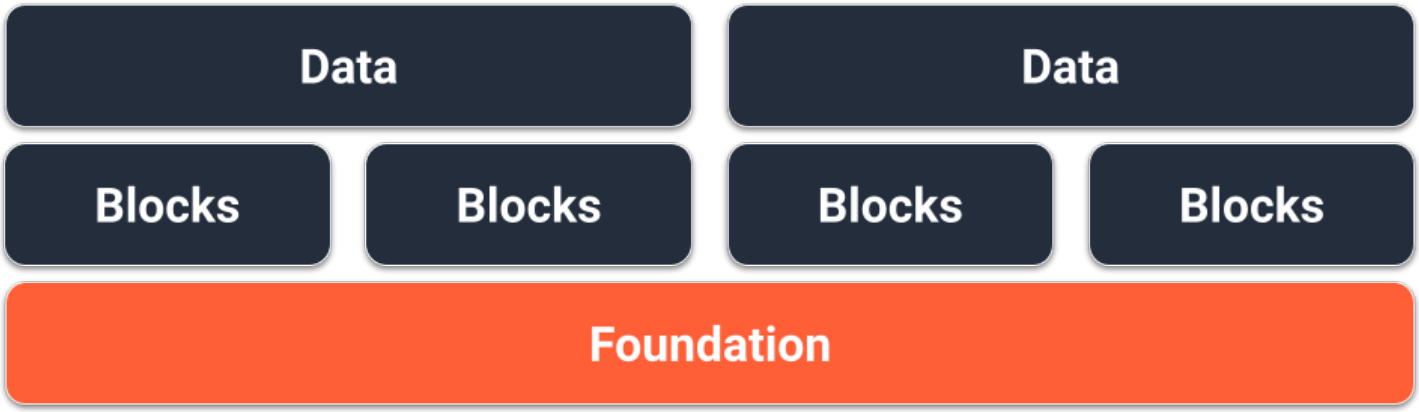
In diesem Beispiel stellt jedes der Elemente in diesen drei Elementauflistungen einen Spieler und das von uns ausgewählte Datenmodell dar, basierend auf den Zugriffsmustern des Spiels und des Spielers. Welche Daten benötigt das Spiel? Wann benötigt es diese Daten? Wie oft benötigt es die Daten? Wie hoch sind die Kosten dafür? Diese Entscheidungen zur Datenmodellierung wurden auf der Grundlage der Antworten auf diese Fragen getroffen.

In diesem Spiel wird dem Spieler eine Seite für sein Waffeninventar und eine andere Seite für sein Rüstungsinventar präsentiert. Wenn der Spieler sein Inventar öffnet, werden zuerst Waffen angezeigt, da diese Seite extrem schnell geladen werden soll, während nachfolgende Inventarseiten später geladen werden können. Da jeder dieser Elementtypen ziemlich groß sein kann, da der Spieler mehr Elemente im Spiel erwirbt, haben wir uns dazu entschieden, jede Inventarseite als eigenes Element in der Elementauflistung des Spielers in der Datenbank darzustellen.

Im folgenden Abschnitt erfahren Sie mehr darüber, wie Sie mithilfe der `Query`-Operation mit Elementauflistungen interagieren können.

Grundlagen der Datenmodellierung in DynamoDB

In diesem Abschnitt werden zunächst die beiden Arten des Tabellendesigns behandelt: das Design mit einer einzelnen und das Design mit mehreren Tabellen.



Fundament für das Design mit einer einzelnen Tabelle

Eine Option für das Fundament unseres DynamoDB-Schemas ist das Design mit einer einzelnen Tabelle. Das Design mit einer einzelnen Tabelle ist ein Muster, mit dem Sie mehrere Datentypen (Entitäten) in einer einzigen DynamoDB-Tabelle speichern können. Ziel ist es, die Datenzugriffsmuster zu optimieren, die Leistung zu verbessern und die Kosten zu senken, indem die Notwendigkeit entfällt, mehrere Tabellen und komplexe Beziehungen zwischen diesen zu verwalten. Dies ist möglich, weil DynamoDB Elemente mit dem gleichen Partitionsschlüssel (als Elementauflistung bezeichnet) auf der-/denselben Partition(en) speichert. In diesem Design werden verschiedene Datentypen als Elemente in derselben Tabelle gespeichert und jedes Element wird durch einen eindeutigen Sortierschlüssel identifiziert.

Primary key		Attributes	
Partition key: PK	Sort key: SK		
UserID	Address#USA#CA#LA#90029	data	GSI-SK
		"Street Address"	Default
	Cart#ACTIVE#Coffee	data	GSI-SK
		CoffeeSKU	2019-11-27T103324
	Cart#ACTIVE#Spice	data	GSI-SK
		SpiceSKU	2019-11-28T091245
	Cart#SAVED#Cocoa	data	GSI-SK
		CocoaSKU	2019-11-28T125642
	OrderHistory#OrderUID	data	GSI-SK
		{Order:DataMap}	2019-10-08T132612
	ProfileName	data	
		"Paul Atreides"	
	Store#StoreUID	data	GSI-SK
		Los Angeles	Active

Vorteile

- Datenlokalität zur Unterstützung von Abfragen für mehrere Entitätstypen in einem einzigen Datenbankaufruf
- Reduzierung der finanziellen Gesamtkosten und der Latenzkosten für Lesevorgänge:
  - Eine einzelne Abfrage für zwei Elemente mit einer Gesamtgröße von weniger als 4 KB entspricht einem letztendlich konsistenten Lesevorgang mit 0,5 RCU.
  - Zwei Abfragen für zwei Elemente mit einer Gesamtgröße von weniger als 4 KB entsprechen einem letztendlich konsistenten Lesevorgang mit 1 RCU (jeweils 0,5 RCU).
  - Die Zeit für die Rückgabe von zwei separaten Datenbankaufrufen ist im Durchschnitt höher als bei einem einzelnen Aufruf.
- Reduzierung der Anzahl an zu verwaltenden Tabellen:
  - Berechtigungen müssen nicht für mehrere IAM-Rollen oder -Richtlinien verwaltet werden.
  - Die Kapazitätsverwaltung für die Tabelle wird über alle Einheiten hinweg gemittelt, was in der Regel ein besser vorhersehbares Nutzungsmuster zur Folge hat.

  - Für die Überwachung sind weniger Alarme erforderlich.
  - Vom Kunden verwaltete Verschlüsselungsschlüssel müssen nur für eine Tabelle rotiert werden.
- Reibungsloser Datenverkehr zur Tabelle:
  - Durch die Zusammenfassung mehrerer Nutzungsmuster in derselben Tabelle ist die Gesamtnutzung in der Regel reibungsloser (so wie die Leistung eines Aktienindex tendenziell reibungsloser ist als bei einer einzelnen Aktie). Dies eignet sich besser, um mit Tabellen im Modus bereitgestellter Kapazität eine höhere Auslastung zu erzielen.

Nachteile

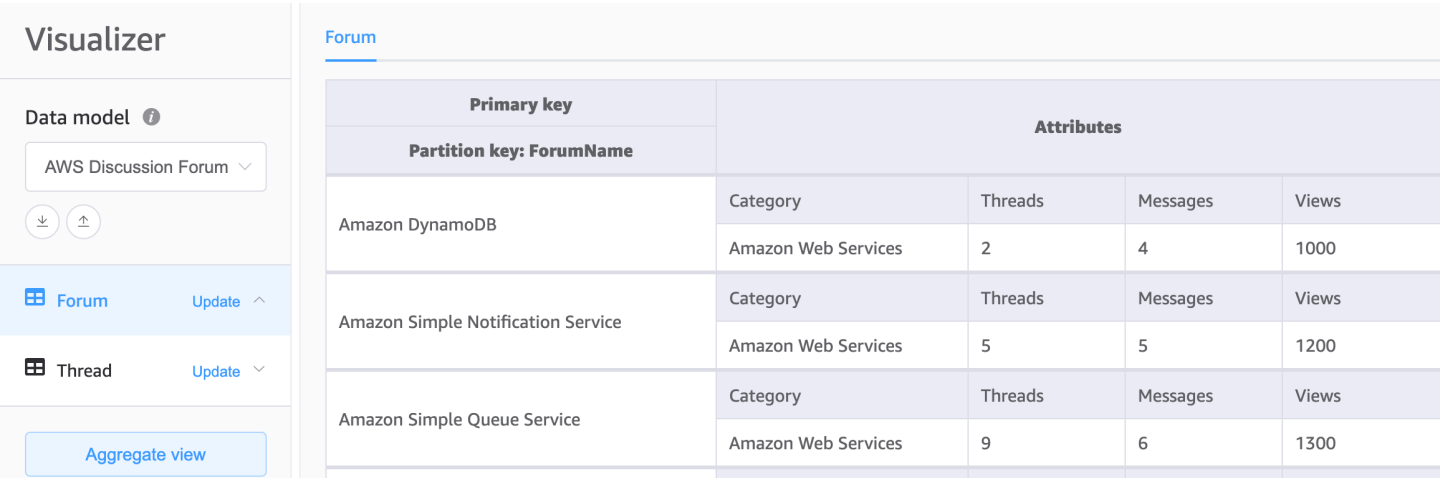
- Die Lernkurve kann aufgrund des paradoxen Designs im Vergleich zu relationalen Datenbanken steil sein.
- Die Datenanforderungen müssen für alle Entitätstypen konsistent sein.
  - Für Backups gilt das „alles oder nichts“-Prinzip. Wenn einige Daten nicht geschäftskritisch sind, sollten Sie erwägen, diese in einer separaten Tabelle zu speichern.
  - Die Tabellenverschlüsselung ist für alle Elemente gleich. Im Fall von Anwendungen für mehrere Mandanten mit individuellen Mandanten-Verschlüsselungsanforderungen wäre eine clientseitige Verschlüsselung erforderlich.
  - Im Fall von Tabellen mit einer Mischung aus historischen Daten und Betriebsdaten wird die Aktivierung der Speicherklasse Infrequent Access nicht so vorteilhaft sein. Weitere Informationen finden Sie unter DynamoDB-Tabellenklassen.
- Alle geänderten Daten werden an DynamoDB Streams weitergegeben, auch wenn nur eine Teilmenge von Entitäten verarbeitet werden muss.
  - Dank der Lambda-Ereignisfilter hat dies keine Auswirkungen auf Ihre Abrechnung, wenn Sie Lambda verwenden. Es ist jedoch mit zusätzlichen Kosten verbunden, wenn Sie die Kinesis Consumer Library verwenden.
- Bei Verwendung von GraphQL wird es schwieriger sein, das Design mit einer einzelnen Tabelle zu implementieren.
- Wenn Sie übergeordnete SDK-Clients wie `DynamoDBMapper` oder Enhanced Client von Java verwenden, kann es schwieriger sein, Ergebnisse zu verarbeiten, da Elemente in derselben Antwort möglicherweise verschiedenen Klassen zugeordnet sind.

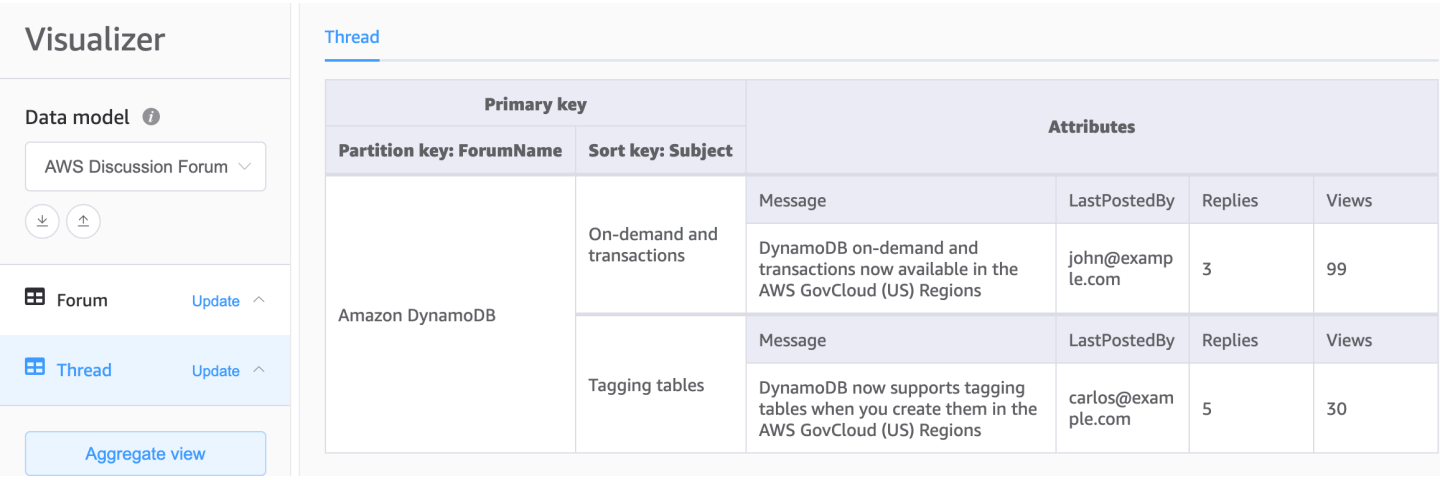
Wann sollte dies verwendet werden?

Das Design mit einer einzigen Tabelle eignet sich gut für Anwendungen, die häufig mehrere Entitätstypen zusammen abfragen oder Beziehungen zwischen verschiedenen Datentypen aufrechterhalten müssen. Es ist besonders effektiv, wenn Ihre Zugriffsmuster von der Datenlokalität profitieren und wenn Sie den Aufwand für die Verwaltung mehrerer Tabellen minimieren möchten.

Fundament für das Design mit mehreren Tabellen

Die zweite Option für das Fundament unseres DynamoDB-Schemas ist das Design mit mehreren Tabellen. Das Design mit mehreren Tabellen ist ein Muster, das eher einem herkömmlichen Datenbankdesign ähnelt, bei dem Sie in jeder DynamoDB-Tabelle einen einzigen Datentyp (Entität) speichern. Die Daten in jeder Tabelle werden weiterhin nach Partitionsschlüsseln organisiert, sodass die Leistung innerhalb eines einzelnen Entitätstyps im Hinblick auf Skalierbarkeit und Leistung optimiert wird. Abfragen über mehrere Tabellen müssen jedoch unabhängig voneinander durchgeführt werden.





Vorteile

- Einfacheres Design, wenn Sie die Arbeit mit dem Design mit einer einzelnen Tabelle nicht gewohnt sind
- Einfachere Implementierung von GraphQL-Resolvern, da jeder Resolver einer einzelnen Entität (Tabelle) zugeordnet wird
- Ermöglicht eindeutige Datenanforderungen für verschiedene Entitätstypen:
  - Für die einzelnen Tabellen, die geschäftskritisch sind, können Backups erstellt werden.
  - Die Tabellenverschlüsselung kann für jede Tabelle verwaltet werden. Im Fall von Anwendungen für mehrere Mandanten mit individuellen Mandanten-Verschlüsselungsanforderungen ermöglichen separate Mandantentabellen, dass jeder Kunde seinen eigenen Verschlüsselungsschlüssel hat.
  - Die Speicherklasse Infrequent Access kann nur für Tabellen mit historischen Daten aktiviert werden, um vollumfänglich vom Vorteil der Kosteneinsparungen zu profitieren. Weitere Informationen finden Sie unter DynamoDB-Tabellenklassen.
- Jede Tabelle hat ihren eigenen Änderungsdatenstrom, sodass für jeden Elementtyp eine eigene Lambda-Funktion entworfen werden kann, anstatt einen einzelnen monolithischen Prozessor verwenden zu müssen.

Nachteile

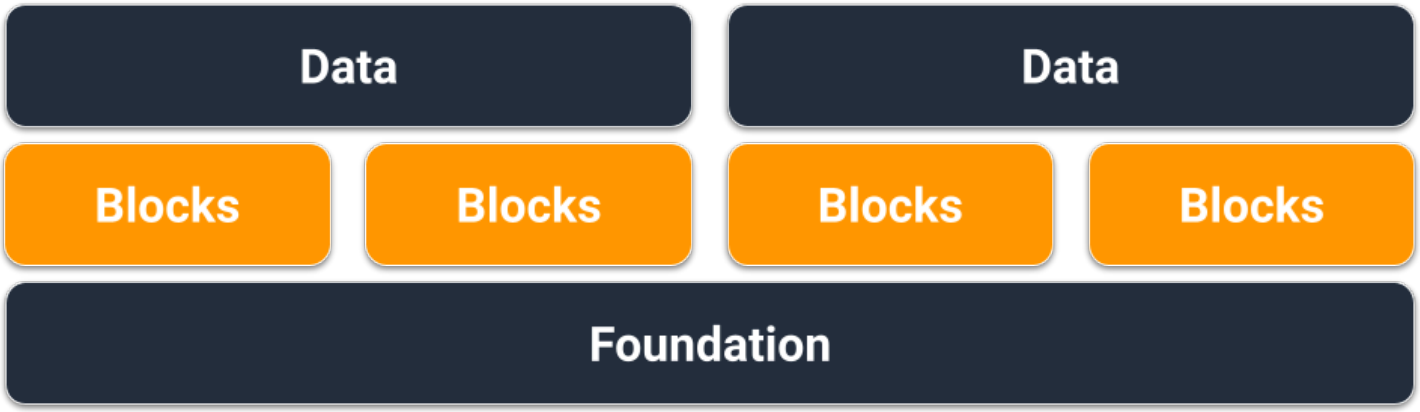
- Für Zugriffsmuster, die Daten aus mehreren Tabellen erfordern, sind mehrere Lesevorgänge aus DynamoDB erforderlich, und Daten müssen sich möglicherweise processed/joined im Client-Code befinden.
- Für den Betrieb und die Überwachung mehrerer Tabellen sind mehr CloudWatch Alarme erforderlich, und jede Tabelle muss unabhängig skaliert werden
- Die Berechtigungen jeder Tabelle müssen separat verwaltet werden. Wenn in Zukunft Tabellen hinzugefügt werden, müssen alle erforderlichen IAM-Rollen oder -Richtlinien geändert werden.

Wann sollte dies verwendet werden?

Wenn die Zugriffsmuster Ihrer Anwendung nicht mehrere Entitäten oder Tabellen zusammen abfragen müssen, ist der Entwurf mehrerer Tabellen ein guter und ausreichender Ansatz.

Bausteine der Datenmodellierung in DynamoDB

In diesem Abschnitt werden die Bausteine behandelt, damit Sie Designmuster erhalten, die Sie für Ihre Anwendung verwenden können.



Themen

- Baustein für zusammengesetzte Sortierschlüssel
- Baustein für Mehrmandantenfähigkeit
- Baustein für Sparse Index
- Baustein für Time to Live
- Baustein für Time to Live für die Archivierung
- Baustein für vertikale Partitionierung
- Baustein für Schreib-Sharding

Baustein für zusammengesetzte Sortierschlüssel

Bei NoSQL denken viele vielleicht an eine nicht relationale Datenbank. Letztendlich spricht jedoch nichts dagegen, Beziehungen in ein DynamoDB-Schema einzubringen, diese sehen nur anders aus als relationale Datenbanken und ihre Fremdschlüssel. Eines der wichtigsten Muster, die wir verwenden können, um eine logische Hierarchie unserer Daten in DynamoDB zu entwickeln, ist ein zusammengesetzter Sortierschlüssel. Die gebräuchlichste Gestaltungsmethode besteht darin, jede Hierarchieebene (übergeordnete Ebene > untergeordnete Ebene > zweite untergeordnete Ebene) durch einen Hashtag zu trennen. Beispiel, `PARENT#CHILD#GRANDCHILD#ETC`.

Primary key	
Partition key: PK	**Sort key: SK**
UserID	CART#ACTIVE#Apples
	CART#ACTIVE#Bananas
	CART#SAVED#Oranges
	CART#SAVED#Pears
	WISH#VEGGIES#Carrots

Während ein Partitionsschlüssel in DynamoDB für die Datenabfrage immer den genauen Wert benötigt, können wir eine Teilbedingung von links nach rechts auf den Sortierschlüssel anwenden, ähnlich wie beim Durchlaufen einer Binärstruktur.

Im obigen Beispiel haben wir einen E-Commerce-Shop mit einem Warenkorb, der über alle Benutzersitzungen hinweg verwaltet werden muss. Bei der Anmeldung möchte der Benutzer möglicherweise den gesamten Warenkorb einschließlich der für später gespeicherten Elemente sehen. An der Kasse sollten jedoch nur Elemente im aktiven Warenkorb zum Kauf geladen werden. Da beide `KeyConditions` explizit nach CART-Sortierschlüsseln fragen, werden die zusätzlichen Wunschlistendaten beim Lesen von DynamoDB einfach ignoriert. Zwar sind gespeicherte und aktive Elemente Teil desselben Warenkorbs, in verschiedenen Teilen der Anwendung müssen sie jedoch unterschiedlich behandelt werden. Die Anwendung einer `KeyCondition` auf das Präfix des Sortierschlüssels ist daher die beste Methode, um nur die Daten abzurufen, die für den jeweiligen Anwendungsteil benötigt werden.

Hauptmerkmale dieses Bausteins

- Zusammengehörige Elemente werden lokal zueinander gespeichert, um einen effektiven Datenzugriff zu ermöglichen.
- Mithilfe von `KeyCondition` Ausdrücken können Teilmengen der Hierarchie selektiv abgerufen werden, sodass sie nicht verschwendet werden RCUs
- Verschiedene Teile der Anwendung können ihre Elemente unter einem bestimmten Präfix speichern, um das Überschreiben von Elementen oder widersprüchliche Schreibvorgänge zu verhindern.

Baustein für Mehrmandantenfähigkeit

Viele Kunden verwenden DynamoDB für das Hosting von Daten für ihre mandantenfähigen Anwendungen. Für solche Szenarien möchten wir das Schema so gestalten, dass alle Daten eines einzelnen Mandanten in einer eigenen logischen Partition der Tabelle gespeichert werden. Dabei wird das Konzept der Elementauflistung genutzt, einem Begriff, der alle Elemente in einer DynamoDB-Tabelle mit demselben Partitionsschlüssel bezeichnet. Weitere Informationen über den Umgang von DynamoDB mit Mehrmandantenfähigkeit finden Sie unter Mehrmandantenfähigkeit in DynamoDB.

Primary key		Attributes
Partition key: PK	**Sort key: SK**	
UserOne	PhotoID1	ImageURL
		https://s3.amazonaws.com/[BUCKET-NAME]/[FILE-NAME].[FILE-TYPE]
	PhotoID2	ImageURL
		https://s3.amazonaws.com/[BUCKET-NAME]/[FILE-NAME].[FILE-TYPE]
UserTwo	PhotoID3	ImageURL
		https://s3.amazonaws.com/[BUCKET-NAME]/[FILE-NAME].[FILE-TYPE]
	PhotoID4	ImageURL
		https://s3.amazonaws.com/[BUCKET-NAME]/[FILE-NAME].[FILE-TYPE]
UserThree	PhotoID5	ImageURL
		https://s3.amazonaws.com/[BUCKET-NAME]/[FILE-NAME].[FILE-TYPE]

In diesem Beispiel betreiben wir eine Website für das Hosting von Fotos mit potenziell Tausenden von Benutzern. Jeder Benutzer lädt zunächst nur Fotos in sein eigenes Profil hoch, standardmäßig ist es den Benutzern jedoch nicht gestattet, die Fotos anderer Benutzer zu sehen. Idealerweise würde der Autorisierung jedes Benutzeraufrufs an Ihre API eine zusätzliche Isolationsstufe hinzugefügt, um sicherzustellen, dass die Benutzer nur Daten von ihrer eigenen Partition anfordern. Auf Schemaebene sind eindeutige Partitionsschlüssel jedoch angemessen.

Hauptmerkmale dieses Bausteins

- Die Datenmenge, die ein Benutzer oder Mandant lesen kann, kann nur so groß wie die Gesamtmenge der Elemente in seiner Partition sein.
- Die Entfernung der Daten eines Mandanten aufgrund einer Kontoschließung oder einer Aufforderung zur Einhaltung der Vorschriften kann diskret und kostengünstig erfolgen. Hierfür muss nur eine Abfrage ausgeführt werden, bei der der Partitionsschlüssel der betreffenden Mandanten-

ID entspricht, und dann muss für jeden zurückgegebenen Primärschlüssel eine `DeleteItem`-Operation durchgeführt werden.

> **Note**
>
> Bei der Entwicklung wurde die Mehrmandantenfähigkeit berücksichtigt. Sie können verschiedene Anbieter von Verschlüsselungsschlüsseln in einer einzigen Tabelle verwenden, um Daten sicher zu isolieren. Mit dem AWS Datenbankverschlüsselungs-SDK für Amazon DynamoDB können Sie eine clientseitige Verschlüsselung in Ihre DynamoDB-Workloads einbeziehen. Sie können eine Verschlüsselung auf Attributebene durchführen, sodass Sie bestimmte Attributwerte verschlüsseln können, bevor Sie sie in Ihrer DynamoDB-Tabelle speichern, und nach verschlüsselten Attributen suchen können, ohne zuvor die gesamte Datenbank zu entschlüsseln.

Baustein für Sparse Index

Manchmal erfordert ein Zugriffsmuster die Suche nach Elementen, die mit einem seltenen Element übereinstimmen, oder nach einem Element, das einen Status erhält (der eine eskalierte Antwort erfordert). Anstatt regelmäßig den gesamten Datensatz nach diesen Elementen abzufragen, können wir die Tatsache nutzen, dass globale sekundäre Indizes (GSI) nur spärlich mit Daten gefüllt sind. Dies bedeutet, dass nur die Elemente in der Basistabelle, die die im Index definierten Attribute aufweisen, in den Index repliziert werden.

Primary key		Attributes		
Partition key: DeviceID	**Sort key: State#Date**			
d#12345	NORMAL#2020-04-24T14:55:00	Operator	Date	
		Liz	2020-04-24	
	WARNING1#2020-04-24T14:45:00	Operator	Date	
		Liz	2020-04-24	
	WARNING1#2020-04-24T14:50:00	Operator	Date	
		Liz	2020-04-24	
d#54321	NORMAL#2020-04-11T06:00:00	Operator	Date	
		Liz	2020-04-11	
	NORMAL#2020-04-11T09:30:00	Operator	Date	
		Sue	2020-04-11	
	WARNING2#2020-04-11T09:25:00	Operator	Date	
		Sue	2020-04-11	
	WARNING3#2020-04-11T05:55:00	Operator	Date	
		Liz	2020-04-11	
d#11223	WARNING4#2020-04-27T16:10:00	Operator	Date	
		Sue	2020-04-27	
	WARNING4#2020-04-27T16:15:00	Operator	Date	EscalatedTo
		Sue	2020-04-27	Sara

Primary key		Attributes	
Partition key: EscalatedTo	Sort key: State#Date		
Sara	WARNING4#2020-04-27T16:15:00	DeviceID	Operator
		d#11223	Sue

In diesem Beispiel sehen wir einen IoT-Anwendungsfall, bei dem jedes Gerät regelmäßig einen Status zurückmeldet. In den meisten Fällen erwarten wir die Meldung, dass alles in Ordnung ist. Gelegentlich kann es jedoch zu einem Fehler kommen, der an einen Reparaturtechniker weitergeleitet werden muss. Bei Berichten mit einer Eskalation wird dem Element das Attribut `EscalatedTo` hinzugefügt, dieses ist ansonsten aber nicht vorhanden. Der GSI in diesem Beispiel ist nach `EscalatedTo` partitioniert. Da der GSI Schlüssel aus der Basistabelle übernimmt, können wir immer noch sehen, welche DeviceID den Fehler zu welcher Uhrzeit gemeldet hat.

Lesevorgänge sind in DynamoDB zwar günstiger als Schreibvorgänge, Sparse Indexes stellen jedoch ein sehr leistungsfähiges Tool für Anwendungsfälle dar, in denen Instances eines bestimmten Elementtyps selten vorkommen, Lesevorgänge, um diese zu finden, jedoch häufig stattfinden.

Hauptmerkmale dieses Bausteins

- Die Schreib- und Speicherkosten für den globalen Index mit geringer Dichte fallen nur für Elemente an, die dem Schlüsselmuster entsprechen, sodass die Kosten für den globalen Index erheblich niedriger sein können als bei anderen GSIs, auf die alle Elemente repliziert werden
- Es kann immer noch ein zusammengesetzter Sortierschlüssel verwendet werden, um die Elemente, die der gewünschten Abfrage entsprechen, weiter einzugrenzen. So könnte beispielsweise ein Zeitstempel für den Sortierschlüssel verwendet werden, um nur die in den letzten X Minuten gemeldeten Fehler anzuzeigen (`SK > 5 minutes ago, ScanIndexForward: False`).

Baustein für Time to Live

Bei den meisten Daten ist es für eine gewisse Zeitspanne sinnvoll, sie in einem primären Datenspeicher aufzubewahren. Um den Ablauf von Daten in DynamoDB zu erleichtern, gibt es eine Funktion namens Time to Live (TTL). Mit der TTL-Funktion können Sie auf Tabellenebene ein bestimmtes Attribut definieren, das auf Elemente mit einem Epochenzeitstempel (der in der Vergangenheit liegt) überwacht werden muss. Auf diese Weise können Sie abgelaufene Datensätze kostenlos aus der Tabelle löschen.

> **Note**
>
> Wenn Sie Version 2019.11.21 der globalen Tabellen (aktuell) und außerdem die Time-to-Live-Funktion verwenden, repliziert DynamoDB TTL-Löschungen in alle Replikattabellen. Die anfängliche TTL-Löschung verbraucht keine Schreibkapazität in der Region, in der die TTL abläuft. Die in die Replikattabelle(n) replizierte TTL-Löschung verbraucht jedoch in jeder Region mit einem Replikat replizierte Schreibkapazität. Dafür werden die entsprechenden Gebühren berechnet.

Primary key		Attributes	
Partition key: PK	**Sort key: MessageTimestamp**		
UserID	2030-06-30T12:12:12	TTL	Message
		1909570332	Hello
	2030-06-30T12:17:22	TTL	Message
		1909570647	DynamoDB
	2030-06-30T12:22:27	TTL	Message
		1909570947	TTL

Mit der Anwendung in diesem Beispiel kann ein Benutzer kurzlebige Nachrichten erstellen. Wenn eine Nachricht in DynamoDB erstellt wird, wird das TTL-Attribut vom Anwendungscode auf ein Datum gesetzt, das sieben Tage in der Zukunft liegt. In etwa sieben Tagen wird DynamoDB feststellen, dass der Epochenzeitstempel dieser Elemente in der Vergangenheit liegt, und die Elemente löschen.

Da die von TTL durchgeführten Löschungen kostenlos sind, ist die Verwendung dieser Funktion zum Entfernen von historischen Daten aus der Tabelle dringend zu empfehlen. Dadurch reduzieren sich die monatlichen Speicherkosten insgesamt und wahrscheinlich auch die Kosten für Lesevorgänge der Benutzer, da bei den Abfragen weniger Daten abgerufen werden müssen. TTL ist zwar auf Tabellenebene aktiviert, es ist jedoch Ihre Entscheidung, für welche Elemente oder Entitäten Sie ein TTL-Attribut erstellen möchten und wie weit in der Zukunft Sie den Epochenzeitstempel festlegen möchten.

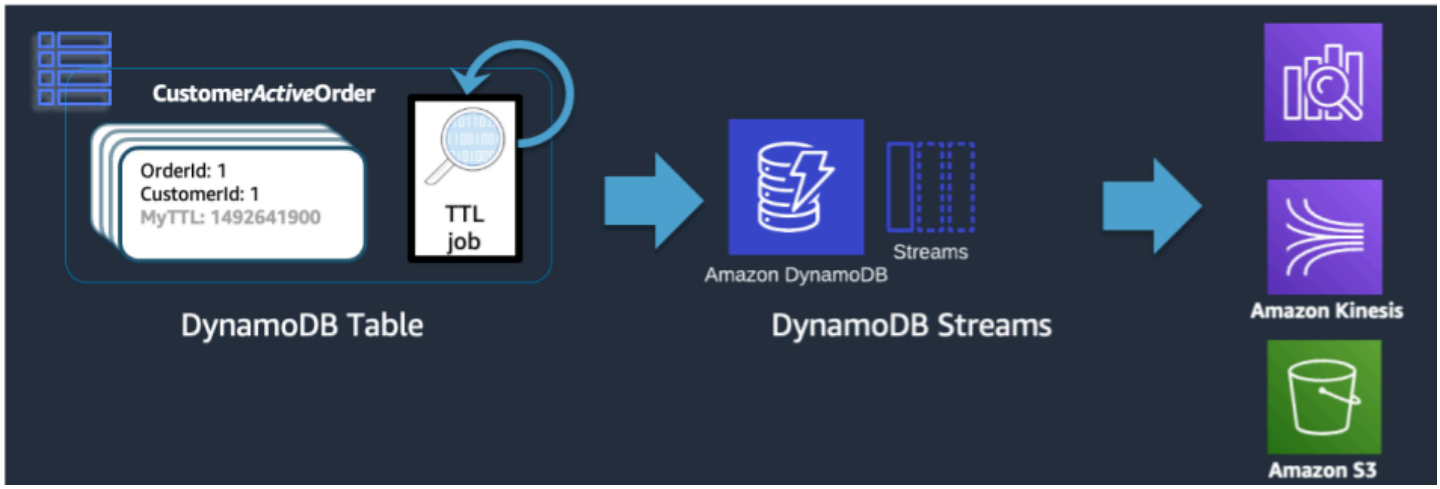
Hauptmerkmale dieses Bausteins

- TTL-Löschungen werden im Hintergrund ohne Auswirkungen auf die Tabellenleistung ausgeführt.

- TTL ist ein asynchroner Prozess, der ungefähr alle sechs Stunden ausgeführt wird. Es kann jedoch mehr als 48 Stunden dauern, bis ein abgelaufener Datensatz gelöscht wird.
  - Verlassen Sie sich nicht auf TTL-Löschungen für Anwendungsfälle wie Sperrdatensätze oder die Statusverwaltung, wenn veraltete Daten in weniger als 48 Stunden bereinigt werden müssen.
- Sie können dem TTL-Attribut einen gültigen Attributnamen geben, der Wert muss jedoch ein numerischer Wert sein.

Baustein für Time to Live für die Archivierung

TTL ist zwar ein effektives Tool zum Löschen älterer Daten aus DynamoDB, in vielen Anwendungsfällen müssen die Daten jedoch über ihre Zeit im primären Datenspeicher hinausgehend archiviert werden. In diesem Fall können wir die zeitgesteuerte Löschung von Datensätzen durch TTL nutzen, um abgelaufene Datensätze in einen langfristigen Datenspeicher zu verschieben.



Wenn DynamoDB eine TTL-Löschung durchführt, wird dies weiterhin als `Delete`-Ereignis in den DynamoDB-Stream übertragen. Wenn DynamoDB TTL jedoch den Löschvorgang durchführt, enthält der Stream-Datensatz das Attribut `principal:dynamodb`. Wenn wir einen Lambda-Abonnenten für den DynamoDB-Stream verwenden, können wir einen Event-Filter nur für das DynamoDB-Prinzipalattribut anwenden und wissen, dass alle Datensätze, die diesem Filter entsprechen, in einen Archivspeicher wie Amazon Glacier übertragen werden müssen.

Hauptmerkmale dieses Bausteins

- Sobald die Lesevorgänge von DynamoDB mit niedriger Latenz für die historischen Elemente nicht mehr benötigt werden, kann die Migration zu einem kälteren Speicherservice wie Amazon Glacier

die Speicherkosten erheblich senken und gleichzeitig die Datenkonformitätsanforderungen Ihres Anwendungsfalls erfüllen.

- Wenn die Daten in Amazon S3 gespeichert werden, können kostengünstige Analysetools wie Amazon Athena oder Redshift Spectrum für historische Datenanalysen verwendet werden.

Baustein für vertikale Partitionierung

Benutzer, die mit einer Dokumentmodelldatenbank vertraut sind, werden es gewohnt sein, alle zusammengehörigen Daten in einem einzigen JSON-Dokument zu speichern. DynamoDB unterstützt zwar JSON-Datentypen, nicht jedoch die Ausführung von `KeyConditions` für verschachteltes JSON. Da `KeyConditions` sie bestimmen, wie viele Daten von der Festplatte gelesen werden und wie viele Daten RCUs eine Abfrage tatsächlich verbraucht, kann dies zu großen Ineffizienzen führen. Zur Optimierung der Schreib- und Lesevorgänge von DynamoDB empfehlen wir, die einzelnen Entitäten des Dokuments in DynamoDB-Elemente aufzuteilen, was auch als vertikale Partitionierung bezeichnet wird.

```
{
        ”UserProfile” : {
                "FirstName": "Paul",
                "LastName": "Atreides",
                "DateJoined":"1965-08-01"},
        ”Store” : {
                ”store_id”: ”STOREUID”,
                 "city": "Los Angeles",
                 "zip_code":"90029”}
        "ShoppingCart" : [
                {"Spice":
                        { "SKU": "SpiceSKU",
                          "CategoryID":"FictionalSpice",
                          "DateAddded " :"2019-06-11"}},
                {"EspressoBeans":
                        { "SKU": "CaffeineSKU",
                          "CategoryID":"FOODANDDRINK",
                          "DateAddded " :"2019-06-10"}}],
        "ShippingAddress" : {
                 "street_address": "1234 Arrakis Dr",
                 "city": "Los Angeles",
                 "zip_code":"90029",
                 "status": "default"}
        ”OrderHistory#OrderUID” : {
                ”ProductA”: ”SKU_A”,
                ”ProductB”: ”SKU_B”,
                "DateOrdered":”2018-09-28"}
}
```

Primary key		Attributes	
Partition key: PK	Sort key: SK		
UserID	Address#USA#CA#LA#90029	data	GSI-SK
		"Street Address"	Default
	Cart#ACTIVE#Coffee	data	GSI-SK
		CoffeeSKU	2019-11-27T103324
	Cart#ACTIVE#Spice	data	GSI-SK
		SpiceSKU	2019-11-28T091245
	Cart#SAVED#Cocoa	data	GSI-SK
		CocoaSKU	2019-11-28T125642
	OrderHistory#OrderUID	data	GSI-SK
		{Order:DataMap}	2019-10-08T132612
	ProfileName	data	
		"Paul Atreides"	
	Store#StoreUID	data	GSI-SK
		Los Angeles	Active

Die vertikale Partitionierung, wie oben dargestellt, ist ein wichtiges Beispiel für das Design einer einzelnen Tabelle in Aktion, kann aber bei Bedarf auch für mehrere Tabellen implementiert werden. Da DynamoDB Schreibvorgänge in 1-KB-Schritten abrechnet, sollten Sie das Dokument idealerweise so partitionieren, dass Elemente mit weniger als 1 KB entstehen.

Hauptmerkmale dieses Bausteins

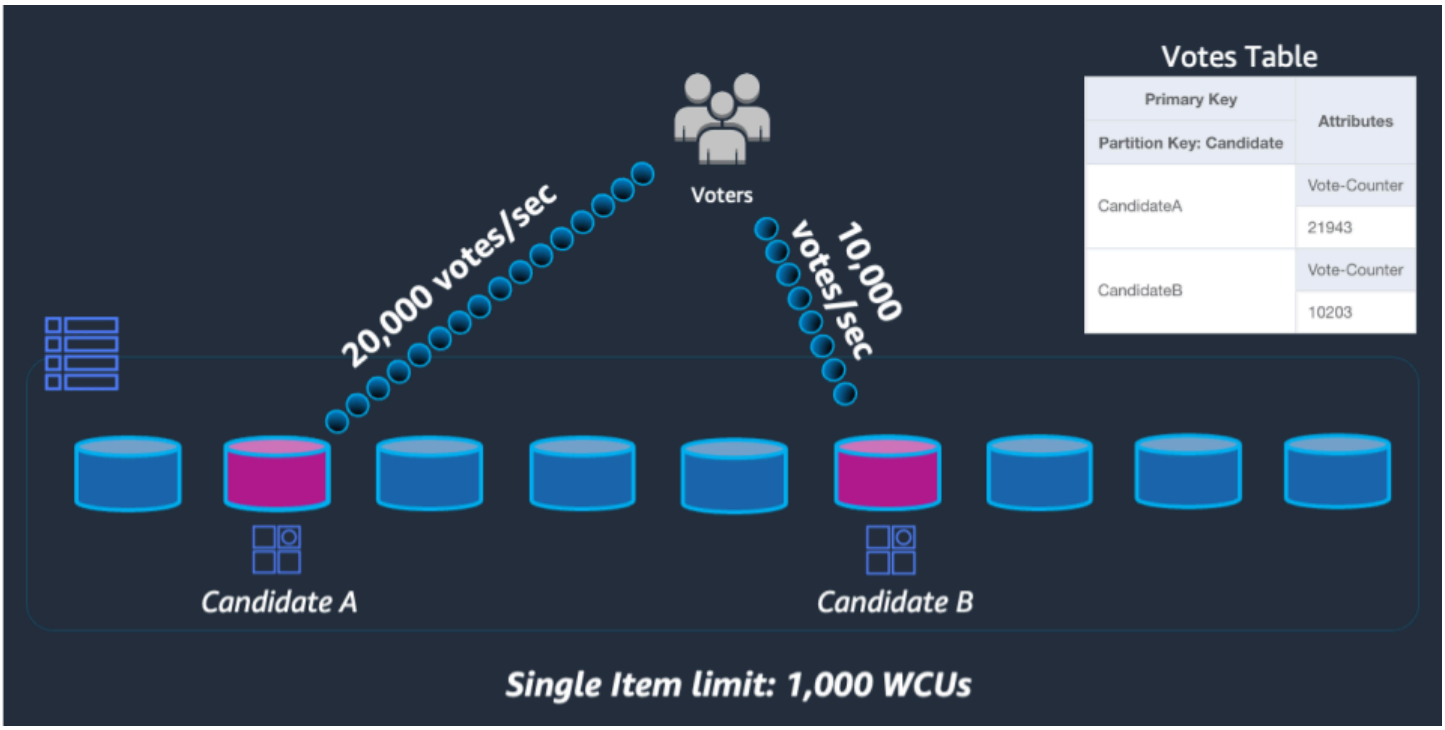
- Eine Hierarchie von Datenbeziehungen wird über Sortierschlüsselpräfixe aufrechterhalten, sodass die einzelne Dokumentstruktur bei Bedarf clientseitig neu aufgebaut werden kann.
- Einzelne Komponenten der Datenstruktur können unabhängig voneinander aktualisiert werden, sodass kleine Elementaktualisierungen nur 1 WCU umfassen.
- Bei Verwendung des Sortierschlüssels `BeginsWith` kann die Anwendung ähnliche Daten in einer einzigen Abfrage abrufen. So können die Lesekosten aggregiert werden, um die Gesamtkosten/Latenz zu verringern.
- Große Dokumente können leicht die maximale Größe von 400 KB für einzelne Elemente in DynamoDB überschreiten. Mithilfe der vertikalen Partitionierung lässt sich dieser Höchstwert umgehen.

Baustein für Schreib-Sharding

Eine der wenigen festen Beschränkungen in DynamoDB bezieht sich auf den Durchsatz, den eine einzelne physische Partition (nicht unbedingt ein einzelner Partitionsschlüssel) pro Sekunde aufrechterhalten kann. Die aktuellen Höchstwerte lauten wie folgt:

- 1 000 WCU (oder 1 000 <= 1 KB geschriebene Elemente pro Sekunde) und 3 000 RCU (oder 3 000 <= 4 KB Lesevorgänge pro Sekunde) Strikt konsistent oder
- 6 000 <= 4 KB Lesevorgänge pro Sekunde Letztendlich konsistent

Falls Anfragen an die Tabelle einen dieser Grenzwerte überschreiten, wird ein Fehler `ThroughputExceededException` an das Client-SDK zurückgesendet. Dies wird allgemein als Drosselung bezeichnet. Für Anwendungsfälle, die über diesen Grenzwert hinausgehende Lesevorgänge erfordern, ist es meistens am besten, einen Lese-Cache vor DynamoDB zu platzieren. Schreibvorgänge erfordern jedoch ein Design auf Schemaebene, das als Schreib-Sharding bekannt ist.



Primary Key	Attributes	
Partition Key: Candidate		
CandidateA#1	Vote-Counter	Last-Update
	10238	2019-09-30T11:35:53
CandidateA#2	Vote-Counter	Last-Update
	8452	2019-09-30T11:35:53
CandidateA#3	Vote-Counter	Last-Update
	9148	2019-09-30T11:35:53
CandidateA#4	Vote-Counter	Last-Update
	11092	2019-09-30T11:35:53

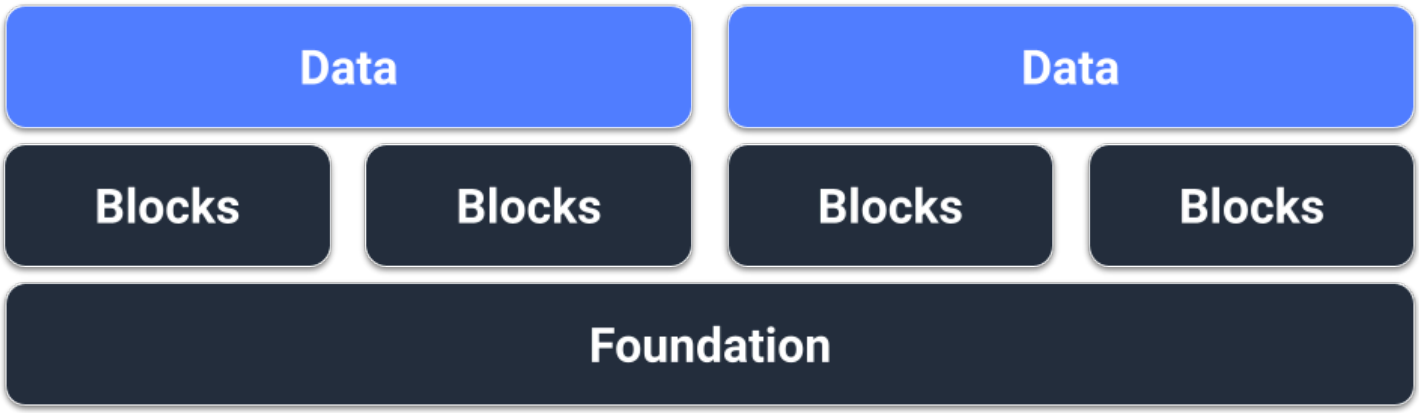
Um dieses Problem zu lösen, fügen wir für jeden Teilnehmer im `UpdateItem`-Code der Anwendung eine zufällige Ganzzahl an das Ende des Partitionsschlüssels an. Der Bereich des Zufallszahlengenerators muss eine Obergrenze haben, die der erwarteten Anzahl von Schreibvorgängen pro Sekunde für einen bestimmten Teilnehmer geteilt durch 1 000 entspricht oder diese übersteigt. Um 20 000 Stimmen pro Sekunde zu unterstützen, würde die Angabe beispielsweise „rand(0,19)“ lauten. Da die Daten nun auf separaten logischen Partitionen gespeichert sind, müssen sie beim Lesen wieder zusammengeführt werden. Da die Gesamtzahl der Stimmen nicht in Echtzeit vorliegen muss, könnte eine Lambda-Funktion, die alle X Minuten alle Abstimmungspartitionen liest, eine gelegentliche Aggregation für jeden Teilnehmer durchführen und für Live-Lesevorgänge in einen einzigen Datensatz für die Gesamtzahl der Stimmen zurückschreiben.

Hauptmerkmale dieses Bausteins

- Für Anwendungsfälle mit nicht vermeidbarem extrem hohem Schreibdurchsatz für einen bestimmten Partitionsschlüssel können Schreibvorgänge künstlich auf mehrere DynamoDB-Partitionen verteilt werden.
- GSIs Bei einem Partitionsschlüssel mit niedriger Kardinalität sollte dieses Muster ebenfalls verwendet werden, da die Drosselung auf einer globalen Datenschnittstelle zu Gegendruck bei Schreibvorgängen in der Basistabelle führt

Pakete für das Schemadesign für die Datenmodellierung in DynamoDB

Erfahren Sie mehr über Schema-Design-Pakete zur Datenmodellierung für DynamoDB, einschließlich Anwendungsfällen, Zugriffsmustern und endgültigen Schemadesigns für soziale Netzwerke, Gaming-Profile, Beschwerdemanagement, wiederkehrende Zahlungen, Gerätestatus und Online-Shops.



Voraussetzungen

Bevor wir versuchen, unser Schema für DynamoDB zu gestalten, müssen wir zunächst einige erforderliche Daten zu dem Anwendungsfall sammeln, den das Schema unterstützen soll. Anders als bei relationalen Datenbanken findet in DynamoDB standardmäßig Sharding statt. Die Daten werden auf mehreren Servern im Hintergrund gespeichert. Daher ist es wichtig, die Datenlokalität zu berücksichtigen. Wir müssen für jedes Schemadesign folgende Liste zusammenstellen:

- Liste der Entitäten (ER-Diagramm)
- Geschätztes Volumen und Durchsatz für jede Entität
- Zugriffsmuster, die unterstützt werden müssen (Abfragen und Schreibvorgänge)
- Anforderungen an die Datenaufbewahrung

Themen

- Schemadesign für soziale Netzwerke in DynamoDB
- Schemadesign für Gaming-Profile in DynamoDB
- Schemadesign des Systems zur Beschwerdeverwaltung in DynamoDB
- Schemadesign für wiederkehrende Zahlungen in DynamoDB

- Überwachung von Gerätestatusaktualisierungen in DynamoDB
- Verwendung von DynamoDB als Datenspeicher für einen Online-Shop

Schemadesign für soziale Netzwerke in DynamoDB

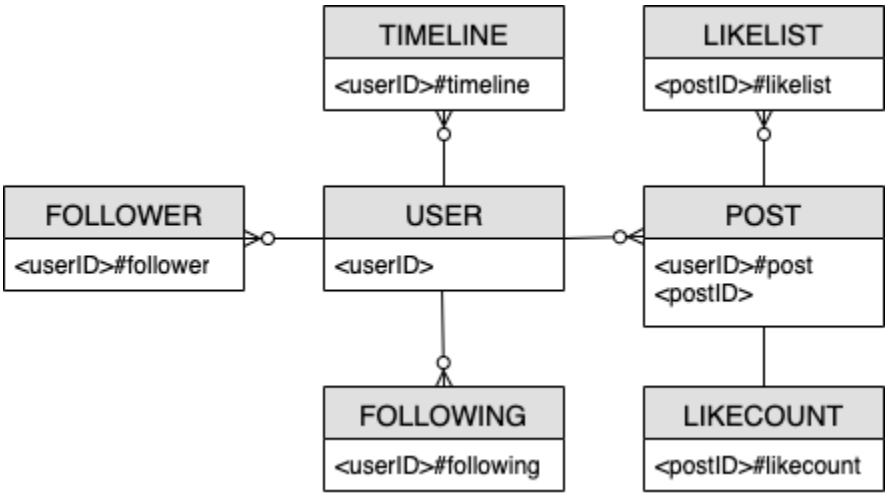
Geschäftlicher Anwendungsfall für soziale Netzwerke

In diesem Anwendungsfall geht es um die Verwendung von DynamoDB als soziales Netzwerk. Ein soziales Netzwerk ist ein Online-Service, über den verschiedene Benutzer miteinander interagieren können. In dem sozialen Netzwerk, das wir gestalten werden, wird den Benutzern eine Timeline mit ihren Beiträgen, ihren Followern, den Personen, denen sie folgen, und deren Beiträgen angezeigt. Die Zugriffsmuster für dieses Schemadesign sind folgende:

- Abrufen von Benutzerinformationen für eine bestimmte userID
- Abrufen der Follower-Liste für eine bestimmte userID
- Abrufen der Liste der gefolgten Personen für eine bestimmte userID
- Abrufen der Beitragsliste für eine bestimmte userID
- Abrufen der Liste der Benutzer, denen der Beitrag gefällt, für eine bestimmte postID
- Abrufen der Anzahl der Likes für eine bestimmte postID
- Abrufen der Timeline für eine bestimmte userID

Diagramm der Entitätsbeziehungen in sozialen Netzwerken

Dies ist das Diagramm der Entitätsbeziehungen (Entity Relationship Diagram, ERD), das wir für das Schemadesign für soziale Netzwerke verwenden werden.



Zugriffsmuster für soziale Netzwerke

Dies sind die Zugriffsmuster, die wir für das Schemadesign für soziale Netzwerke berücksichtigen werden.

- `getUserInfoByUserID`
- `getFollowerListByUserID`
- `getFollowingListByUserID`
- `getPostListByUserID`
- `getUserLikesByPostID`
- `getLikeCountByPostID`
- `getTimelineByUserID`

Entwicklung des Schemadesigns für soziale Netzwerke

DynamoDB ist eine NoSQL-Datenbank. Daher können Sie keine Verknüpfung – eine Operation, bei der Daten aus mehreren Datenbanken kombiniert werden – durchführen. Kunden, die nicht mit DynamoDB vertraut sind, könnten Philosophien der Gestaltung von relationalen Datenbankmanagementsystemen (RDBMS) auf DynamoDB anwenden (z. B. das Erstellen einer Tabelle für jede Entität), obwohl dies nicht notwendig ist. Der Zweck des Einzeltabellendesigns von DynamoDB besteht darin, Daten gemäß dem Zugriffsmuster der Anwendung in einer vorverknüpften Form zu schreiben und dann ohne weitere Berechnungen sofort zu verwenden. Weitere Informationen finden Sie unter Single-table vs. multi-table design in DynamoDB.

Gehen wir nun Schritt für Schritt die Entwicklung unseres Schemadesigns durch, um alle Zugriffsmuster zu berücksichtigen.

Schritt 1: Zugriffsmuster 1 (**`getUserInfoByUserID`**) angehen

Um bestimmte Benutzerinformationen abzurufen, müssen wir eine `Query` für die Basistabelle mit der Schlüsselbedingung `PK=<userID>` durchführen. Mit der Abfrageoperation können Sie die Ergebnisse paginieren, was nützlich sein kann, wenn ein Benutzer viele Follower hat. Weitere Informationen zu Query finden Sie unter Abfragen von Tabellen in DynamoDB.

In unserem Beispiel verfolgen wir zwei Arten von Daten für unseren Benutzer: „count“ und „info“. „count“ gibt an, wie viele Follower ein Benutzer hat, wie vielen Benutzern er folgt und wie viele

Beiträge er erstellt hat. „info“ spiegelt die persönlichen Daten eines Benutzers wie z. B. seinen Namen wider.

Wie wir sehen, werden diese beiden Arten von Daten durch die beiden unten aufgeführten Elemente dargestellt. Das Element, dessen Sortierschlüssel „count“ enthält, wird sich eher ändern als das Element mit „info“. DynamoDB berücksichtigt die Größe des Elements vor und nach der Aktualisierung und der verbrauchte bereitgestellte Durchsatz spiegelt die größere dieser Elementgrößen wider. Auch wenn Sie nur eine Teilmenge der Attribute des Elements aktualisieren, verbraucht `UpdateItem` weiterhin den gesamten bereitgestellten Durchsatz (die größere der Elementgrößen vor und nach der Aktualisierung). Sie können die Elemente mit einer einzigen `Query`-Operation abrufen und `UpdateItem` verwenden, um vorhandene numerische Attribute hinzuzufügen oder davon zu subtrahieren.

Primary key		Attributes			
Partition key: PK	**Sort key: SK**				
u#12345	"count"	follower#	following#	post#	
		3000000000	971	4945	
	"info"	name	content	imageUrl	etc
		hyuklee	My name is Hyuk Lee	s3://....	...

Schritt 2: Zugriffsmuster 2 (**`getFollowerListByUserID`**) angehen

Um eine Liste der Benutzer abzurufen, die einem bestimmten Benutzer folgen, müssen wir eine `Query` für die Basistabelle mit der Schlüsselbedingung `PK=<userID>#follower` durchführen.

Primary key		Attributes			
Partition key: PK	**Sort key: SK**				
u#12345	"count"	follower#	following#	post#	
		3000000000	971	4945	
	"info"	name	content	imageUrl	etc
		hyuklee	My name is Hyuk Lee	s3://....	...
u#12345#follower	u#23456				
	u#34567				
	u#45678				

Schritt 3: Zugriffsmuster 3 (**`getFollowingListByUserID`**) angehen

Um eine Liste der Benutzer abzurufen, denen ein bestimmter Benutzer folgt, müssen wir eine `Query` für die Basistabelle mit der Schlüsselbedingung `PK=<userID>#following` durchführen. Sie können dann eine `TransactWriteItems`-Operation verwenden, um mehrere Anforderungen zusammenzufassen und Folgendes zu tun:

- Benutzer A zur Follower-Liste von Benutzer B hinzufügen und anschließend die Anzahl der Follower von Benutzer B um eins erhöhen
- Benutzer B zur Follower-Liste von Benutzer A hinzufügen und anschließend die Anzahl der Follower von Benutzer A um eins erhöhen

Primary key		Attributes			
Partition key: PK	**Sort key: SK**				
u#12345	"count"	follower#	following#	post#	
		3000000000	971	4945	
	"info"	name	content	imageUrl	etc
		hyuklee	My name is Hyuk Lee	s3://....	...
u#12345#follower	u#23456				
	u#34567				
	u#45678				
u#12345#following	u#56789				
	u#67890				
	u#78912				

Schritt 4: Zugriffsmuster 4 (**`getPostListByUserID`**) angehen

Um eine Liste der Beiträge abzurufen, die von einem bestimmten Benutzer erstellt wurden, müssen wir eine `Query` für die Basistabelle mit der Schlüsselbedingung `PK=<userID>#post` durchführen. Eine wichtige Sache, die hier zu beachten ist, ist, dass der Beitrag eines Benutzers inkrementell sein IDs muss: Der zweite PostID-Wert muss größer sein als der erste PostID-Wert (da Benutzer ihre Beiträge sortiert sehen möchten). Sie können dies tun, indem Sie einen Beitrag auf der IDs Grundlage eines Zeitwerts wie einem Universally Unique Lexicographically Sortable Identifier (ULID) generieren.

Primary key		Attributes			
Partition key: PK	**Sort key: SK**				
u#12345	"count"	follower#	following#	post#	
		3000000000	971	4945	
	"info"	name	content	imageUrl	etc
		hyuklee	My name is Hyuk Lee	s3://....	...
u#12345#follower	u#23456				
	u#34567				
	u#45678				
u#12345#following	u#56789				
	u#67890				
	u#78912				
u#12345#post	p#12345	content	imageUrl	timestamp	
		content for a post	s3://....	1571827560	
	p#23456	content	imageUrl	timestamp	
		content for a post	s3://....	1571827561	

Schritt 5: Zugriffsmuster 5 (**`getUserLikesByPostID`**) angehen

Um eine Liste der Benutzer abzurufen, denen der Beitrag eines bestimmten Benutzers gefallen hat, müssen wir eine `Query` für die Basistabelle mit der Schlüsselbedingung `PK=<postID>#likelist` durchführen. Dies ist dasselbe Muster, das wir zum Abrufen der Listen der Follower und der Personen, denen ein Benutzer folgt, in Zugriffsmuster 2 (`getFollowerListByUserID`) und Zugriffsmuster 3 (`getFollowingListByUserID`) verwendet haben.

Primary key		Attributes			
Partition key: PK	**Sort key: SK**				
u#12345	"count"	follower#	following#	post#	
		3000000000	971	4945	
	"info"	name	content	imageUrl	etc
		hyuklee	My name is Hyuk Lee	s3://....	...
u#12345#follower	u#23456				
	u#34567				
	u#45678				
u#12345#following	u#56789				
	u#67890				
	u#78912				
u#12345#post	p#12345	content	imageUrl	timestamp	
		content for a post	s3://....	1571827560	
	p#23456	content	imageUrl	timestamp	
		content for a post	s3://....	1571827561	
p#12345#likelist	u#23456				
	u#34567				
	u#45678				

Schritt 6: Zugriffsmuster 6 (**`getLikeCountByPostID`**) angehen

Um die Anzahl der Likes für einen bestimmten Beitrag abzurufen, müssen wir eine `GetItem`-Operation für die Basistabelle mit der Schlüsselbedingung `PK=<postID>#likecount` ausführen. Dieses Zugriffsmuster kann zu Drosselungsproblemen führen, wenn ein Benutzer mit vielen Followern (z. B. eine prominente Persönlichkeit) einen Beitrag verfasst, da es zu einer Drosselung kommt, wenn der Durchsatz einer Partition 1 000 WCU pro Sekunde überschreitet. Dieses Problem ist nicht auf DynamoDB zurückzuführen, es tritt nur in DynamoDB auf, da sich DynamoDB am Ende des Software-Stacks befindet.

Sie sollten prüfen, ob es wirklich wichtig ist, dass alle Benutzer die Anzahl der Likes gleichzeitig sehen, oder ob dies schrittweise im Laufe der Zeit geschehen kann. Im Allgemeinen muss die Anzahl der Likes für einen Beitrag nicht sofort zu 100 % korrekt sein. Sie können diese Strategie einführen, indem Sie eine Warteschlange zwischen Ihrer Anwendung und DynamoDB einrichten, damit die Aktualisierungen in regelmäßigen Abständen erfolgen.

Primary key		Attributes			
Partition key: PK	**Sort key: SK**				
u#12345	"count"	follower#	following#	post#	
		3000000000	971	4945	
	"info"	name	content	imageUrl	etc
		hyuklee	My name is Hyuk Lee	s3://....	...
u#12345#follower	u#23456				
	u#34567				
	u#45678				
u#12345#following	u#56789				
	u#67890				
	u#78912				
u#12345#post	p#12345	content	imageUrl	timestamp	
		content for a post	s3://....	1571827560	
	p#23456	content	imageUrl	timestamp	
		content for a post	s3://....	1571827561	
p#12345#likelist	u#23456				
	u#34567				
	u#45678				
p#12345#likecount	"count"	etc			
		100			

Schritt 7: Zugriffsmuster 7 (**`getTimelineByUserID`**) angehen

Um die Timeline für einen bestimmten Benutzer abzurufen, müssen wir eine `Query`-Operation für die Basistabelle mit der Schlüsselbedingung `PK=<userID>#timeline` ausführen. Betrachten wir ein Szenario, in dem die Follower eines Benutzers ihren Beitrag synchron sehen müssen. Jedes Mal, wenn ein Benutzer einen Beitrag schreibt, wird seine Follower-Liste gelesen und seine userID und postID werden langsam in den Timeline-Schlüssel aller seiner Follower eingegeben. Wenn Ihre Anwendung nun gestartet wird, können Sie den Timeline-Schlüssel mit der `Query`-Operation lesen und den Timeline-Bildschirm mit einer Kombination aus userID und postID füllen, indem Sie die `BatchGetItem`-Operation für alle neuen Elemente verwenden. Sie können die Timeline nicht mit einem API-Aufruf lesen, doch diese Lösung ist kostengünstiger, wenn die Beiträge häufig bearbeitet werden könnten.

Auf der Timeline werden die neuesten Beiträge angezeigt, daher benötigen wir eine Möglichkeit, die alten Beiträge zu bereinigen. Anstatt zum Löschen der Beiträge WCU zu verwenden, können Sie dies über das TTL-Feature von DynamoDB kostenlos tun.

Primary key		Attributes			
Partition key: PK	**Sort key: SK**				
u#12345	"count"	follower#	following#	post#	
		3000000000	971	4945	
	"info"	name	content	imageUrl	etc
		hyuklee	My name is Hyuk Lee	s3://....	...
u#12345#follower	u#23456				
	u#34567				
	u#45678				
u#12345#following	u#56789				
	u#67890				
	u#78912				
u#12345#post	p#12345	content	imageUrl	timestamp	
		content for a post	s3://....	1571827560	
	p#23456	content	imageUrl	timestamp	
		content for a post	s3://....	1571827561	
p#12345#likelist	u#23456				
	u#34567				
	u#45678				
p#12345#likecount	"count"	etc			
		100			
u#12345#timeline	p#34567#u#56789	ttl			
		1571827560			
	p#45678#u#67890	ttl			
		1571827560			
	p#56789#u#78901	ttl			
		1571827560			

Alle Zugriffsmuster und wie das Schemadesign sie behandelt, sind in der folgenden Tabelle zusammengefasst:

Zugriffsmuster	Basis-table/ GSI/LSI	Operation	Partitionsschlüsselwert	Sortierschlüsselwert	Sonstige Bedingungen/ Filter
getUserInfoByUserID	Basistabelle	Query	PK= <userID>		

Zugriffsmuster	Basis-table/GSI/LSI	Operation	Partitionsschlüsselwert	Sortierschlüsselwert	Sonstige Bedingungen/Filter
getFollowerListByUserAUSWEIS	Basistabelle	Query	PK=<userID>#follower		
getFollowingListByUserAUSWEIS	Basistabelle	Query	PK=<userID>#following		
getPostListByUserAUSWEIS	Basistabelle	Query	PK=<userID>#post		
getUserLikesByPostAUSWEIS	Basistabelle	Query	PK=<postID>#likelist		
getLikeCountByPostAUSWEIS	Basistabelle	GetItem	PK=<postID>#likecount		
getTimelineByUserID	Basistabelle	Query	PK=<userID>#timeline		

Endgültiges Schema für soziale Netzwerke

Dies ist das endgültige Schemadesign. Informationen zum Herunterladen dieses Schemadesign als JSON-Datei finden Sie unter DynamoDB-Beispiele auf. GitHub

Basistabelle:

Primary key		Attributes			
Partition key: PK	**Sort key: SK**				
u#12345	"count"	follower#	following#	post#	
		3000000000	971	4945	
	"info"	name	content	imageUrl	etc
		hyuklee	My name is Hyuk Lee	s3://....	...
u#12345#follower	u#23456				
	u#34567				
	u#45678				
u#12345#following	u#56789				
	u#67890				
	u#78912				
u#12345#post	p#12345	content	imageUrl	timestamp	
		content for a post	s3://....	1571827560	
	p#23456	content	imageUrl	timestamp	
		content for a post	s3://....	1571827561	
p#12345#likelist	u#23456				
	u#34567				
	u#45678				
p#12345#likecount	"count"	etc			
		100			
u#12345#timeline	p#34567#u#56789	ttl			
		1571827560			
	p#45678#u#67890	ttl			
		1571827560			
	p#56789#u#78901	ttl			
		1571827560			

Verwendung von NoSQL Workbench mit diesem Schemadesign

Sie können dieses endgültige Schema in NoSQL Workbench importieren, um Ihr neues Projekt weiter zu untersuchen und zu bearbeiten. NoSQL Workbench ist ein visuelles Tool, das Features zur Datenmodellierung, Datenvisualisierung und Abfrageentwicklung für DynamoDB bereitstellt. Gehen Sie folgendermaßen vor, um zu beginnen:

1. Laden Sie NoSQL Workbench herunter. Weitere Informationen finden Sie unter the section called “Download”.
2. Laden Sie die oben aufgeführte JSON-Schemadatei herunter, die bereits das NoSQL-Workbench-Modellformat aufweist.

3. Importieren Sie die JSON-Schemadatei in NoSQL Workbench. Weitere Informationen finden Sie unter the section called “Importieren eines vorhandenen Modells”.
4. Nach dem Import in NOSQL Workbench können Sie das Datenmodell bearbeiten. Weitere Informationen finden Sie unter the section called “Bearbeiten eines vorhandenen Modells”.

Schemadesign für Gaming-Profile in DynamoDB

Geschäftlicher Anwendungsfall für Gaming-Profile

In diesem Anwendungsfall geht es um die Verwendung von DynamoDB zum Speichern von Spielerprofilen für ein Gaming-System. Bevor Benutzer (in diesem Fall Spieler) mit vielen modernen Spielen, insbesondere Online-Spielen, interagieren können, müssen sie Profile erstellen. Gaming-Profile beinhalten in der Regel Folgendes:

- Grundlegende Informationen wie beispielsweise den Benutzernamen
- Spieledaten wie beispielsweise Gegenstände und Ausrüstung
- Spieledatensätze wie beispielsweise Aufgaben und Aktivitäten
- Soziale Informationen wie beispielsweise Freundeslisten

Um die differenzierten Anforderungen für den Zugriff auf Datenabfragen für diese Anwendung zu erfüllen, verwenden die Primärschlüssel (Partitionsschlüssel und Sortierschlüssel) generische Namen (PK und SK), sodass sie mit verschiedenen Arten von Werten überlastet werden können, wie wir weiter unten sehen werden.

Die Zugriffsmuster für dieses Schemadesign sind folgende:

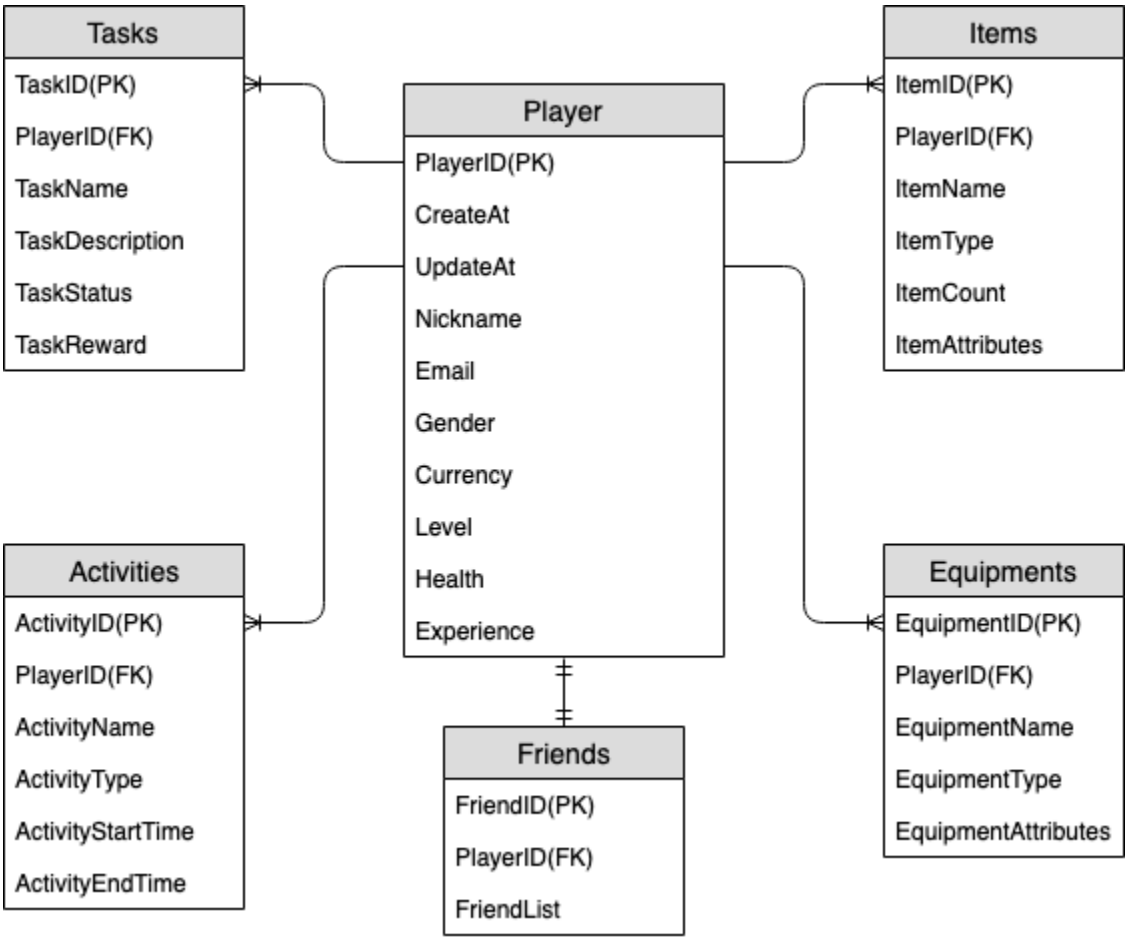
- Abrufen der Freundesliste eines Benutzers
- Abrufen aller Informationen eines Spielers
- Abrufen der Gegenstandsliste eines Benutzers
- Abrufen eines spezifischen Gegenstands aus der Gegenstandsliste des Benutzers
- Aktualisieren des Charakters eines Benutzers
- Aktualisieren der Anzahl an Gegenständen für einen Benutzer

Die Größe des Gaming-Profils variiert je nach Spiel. Wenn Sie große Attributwerte komprimieren, liegen sie möglicherweise innerhalb der Elementeinschränkungen in DynamoDB, sodass Ihre Kosten

reduziert werden. Die Strategie für die Durchsatzverwaltung würde von verschiedenen Faktoren abhängen, z. B. der Anzahl der Spieler, der Anzahl der pro Sekunde gespielten Spiele und der Saisonalität des Workloads. In der Regel sind bei einem neu gestarteten Spiel die Anzahl der Spieler und der Beliebtheitsgrad nicht bekannt, daher beginnen wir mit dem On-Demand-Durchsatzmodus.

Diagramm der Entitätsbeziehungen in Gaming-Profilen

Dies ist das Diagramm der Entitätsbeziehungen (Entity Relationship Diagram, ERD), das wir für das Schemadesign für Gaming-Profile verwenden werden.



Zugriffsmuster für Gaming-Profile

Dies sind die Zugriffsmuster, die wir für das Schemadesign für soziale Netzwerke berücksichtigen werden.

- `getPlayerFriends`
- `getPlayerAllProfile`
- `getPlayerAllItems`

- `getPlayerSpecificItem`
- `updateCharacterAttributes`
- `updateItemCount`

Entwicklung des Schemas für Gaming-Profile

Aus dem obigen ERD können wir ersehen, dass es sich um einen one-to-many Beziehungstyp der Datenmodellierung handelt. In DynamoDB können one-to-many Datenmodelle in Elementsammlungen organisiert werden, was sich von herkömmlichen relationalen Datenbanken unterscheidet, bei denen mehrere Tabellen erstellt und über Fremdschlüssel verknüpft werden. Eine Elementauflistung ist eine Gruppe von Elementen, die den gleichen Partitionsschlüsselwert, aber unterschiedliche Sortierschlüsselwerte aufweisen. Innerhalb einer Elementauflistung hat jedes Element einen eindeutigen Sortierschlüsselwert, der es von anderen Elementen unterscheidet. Vor diesem Hintergrund verwenden wir das folgende Muster für `HASH`- und `RANGE`-Werte für jeden Entitätstyp.

Zu Beginn verwenden wir generische Namen wie `PK` und `SK`, um verschiedene Arten von Entitäten in derselben Tabelle zu speichern und das Modell zukunftssicher zu machen. Zur besseren Lesbarkeit können wir Präfixe zur Angabe des Datentyps oder ein beliebiges Attribut mit der Bezeichnung `Entity_type` oder `Type` hinzufügen. Im aktuellen Beispiel verwenden wir eine Zeichenfolge, die mit `player` beginnt, um `player_ID` als `PK` zu speichern. Wir verwenden `entity name#` als Präfix von `SK` und fügen das Attribut `Type` hinzu, um anzugeben, um welchen Entitätstyp es sich bei diesen Daten handelt. Dadurch können wir in Zukunft die Speicherung weiterer Entitätstypen unterstützen und fortschrittliche Technologien wie GSI-Überladung und Sparse GSI verwenden, um mehr Zugriffsmuster zu erfüllen.

Beginnen wir mit der Implementierung der Zugriffsmuster. Zugriffsmuster wie das Hinzufügen von Spielern und das Hinzufügen von Ausrüstung können über die Operation `PutItem` realisiert werden, sodass wir sie ignorieren können. In diesem Dokument konzentrieren wir uns auf die oben aufgeführten typischen Zugriffsmuster.

Schritt 1: Zugriffsmuster 1 (**`getPlayerFriends`**) angehen

In diesem Schritt gehen wir das Zugriffsmuster 1 (`getPlayerFriends`) an. In unserem aktuellen Design ist die Freundschaft einfach und die Anzahl der Freunde im Spiel ist gering. Der Einfachheit halber verwenden wir einen Listendatentyp, um Freundeslisten zu speichern (1:1-Modellierung). In diesem Design verwenden wir `GetItem`, um dieses Zugriffsmuster zu erfüllen. In

der Operation `GetItem` geben wir explizit den Partitions- und den Sortierschlüsselwert an, um ein spezifisches Element abzurufen.

Wenn ein Spiel jedoch eine große Anzahl von Freunden hat und die Beziehungen zwischen ihnen komplex sind (z. B. bidirektionale Freundschaften mit einer Einladungs- und einer Annahmekomponente), wäre es notwendig, eine many-to-many Beziehung zu verwenden, um jeden Freund einzeln zu speichern, um auf eine unbegrenzte Freundeslistengröße skalieren zu können. Und wenn die Änderung der Freundschaft beinhaltet, mehrere Objekte gleichzeitig zu bearbeiten, können DynamoDB-Transaktionen verwendet werden, um mehrere Aktionen zu gruppieren und sie als einzelne all-or-nothing `TransactWriteItems`Oder-Operation einzureichen. `TransactGetItems`

Primary key		Attributes	
Partition key: PK	**Sort key: SK**		
player001	FRIENDS#player001	Type	FriendList
		Friends	[{"M": {"FriendId": {"S":"player002"},"FriendName": {"S":"Alice"}}}, {"M":{"FriendId": {"S":"player003"},"FriendName": {"S":"Bob"}}}]

Schritt 2: Zugriffsmuster 2 (**`getPlayerAllProfile`**), 3 (**`getPlayerAllItems`**) und 4 (**`getPlayerSpecificItem`**) angehen

In diesem Schritt gehen wir die Zugriffsmuster 2 (`getPlayerAllProfile`), 3 (`getPlayerAllItems`) und 4 (`getPlayerSpecificItem`) an. Was diese drei Zugriffsmuster gemeinsam haben, ist eine Bereichsabfrage, die die Operation `Query` verwendet. Je nach Umfang der Abfrage werden Schlüsselbedingung und Filterausdrücke verwendet, die in der praktischen Entwicklung häufig verwendet werden.

In der Abfrage-Operation geben wir einen einzelnen Wert für den Partitionsschlüssel an und rufen alle Elemente mit diesem Partitionsschlüsselwert ab. Das Zugriffsmuster 2 (`getPlayerAllProfile`) wird auf diese Weise implementiert. Optional können wir einen Bedingungsausdruck für den Sortierschlüssel hinzufügen – eine Zeichenfolge, die bestimmt, welche Elemente aus der Tabelle gelesen werden sollen. Das Zugriffsmuster 3 (`getPlayerAllItems`) wird implementiert, indem die Schlüsselbedingung sort key begins_with `ITEMS#` hinzugefügt wird. Zum Vereinfachen der Entwicklung der Anwendungsseite können wir außerdem Filterausdrücke verwenden, um das Zugriffsmuster 4 (`getPlayerSpecificItem`) zu implementieren.

Dies ist ein Pseudocode-Beispiel, das einen Filterausdruck verwendet, der Elemente der Kategorie `Weapon` filtert:

```
filterExpression: "ItemType = :itemType"
expressionAttributeValues: {":itemType": "Weapon"}
```

Primary key		Attributes				
Partition key: PK	**Sort key: SK**					
player001	ITEMS#001	Type	ItemName	ItemType	ItemCount	ItemAttributes
		Item	Health Potion	Consumable	5	{"M":{"HP": {"N":"50"}}}
	ITEMS#002	Type	ItemName	ItemType	ItemCount	ItemAttributes
		Item	Armor of the Knight	Armor	1	{"M":{"DEF": {"N":"100"}}}
		Type	ItemName	ItemType	ItemCount	ItemAttributes
	ITEMS#003	Item	Sword of the Dragon	Weapon	1	{"M":{"ATK": {"N":"100"},"DEF" :{"N":"50"}}}

> **Note**
>
> Ein Filterausdruck wird angewendet, nachdem eine Abfrage abgeschlossen ist und bevor die Ergebnisse zurückgegeben werden. Folglich verbraucht eine Abfrage unabhängig davon, ob ein Filterausdruck vorhanden ist oder nicht, gleich viel Lesekapazität.

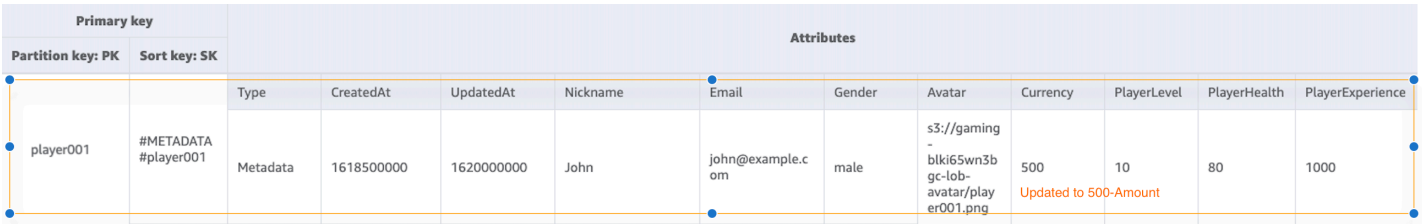
Wenn das Zugriffsmuster darin besteht, einen großen Datensatz abzufragen und eine große Datenmenge herauszufiltern, um nur eine kleine Teilmenge der Daten zu behalten, sollten der DynamoDB-Partitionsschlüssel und -Sortierschlüssel effektiver gestaltet werden. Wenn es im obigen Beispiel zum Abrufen eines bestimmten `ItemType` beispielsweise viele Gegenstände für jeden Spieler gibt und die Abfrage eines bestimmten `ItemType` ein typisches Zugriffsmuster ist, wäre es effizienter, `ItemType` als zusammengesetzten Schlüssel in den `SK` aufzunehmen. Das Datenmodell würde dann wie folgt aussehen: `ITEMS#ItemType#ItemId`.

Schritt 3: Zugriffsmuster 5 (**`updateCharacterAttributes`**) und 6 (**`updateItemCount`**) angehen

In diesem Schritt gehen wir die Zugriffsmuster 5 (`updateCharacterAttributes`) und 6 (`updateItemCount`) an. Wenn der Spieler den Charakter modifizieren – z. B. die Währung reduzieren oder die Menge einer bestimmten Waffe in seinen Gegenständen ändern – muss,

verwenden Sie `UpdateItem`, um diese Zugriffsmuster zu implementieren. Wenn wir die Währung eines Spielers aktualisieren und gleichzeitig sicherstellen möchten, dass sie einen Mindestbetrag niemals unterschreitet, können wir the section called “CLI-Beispiel” hinzufügen, um das Guthaben nur dann zu reduzieren, wenn es größer oder gleich dem Mindestbetrag ist. Dies ist ein Pseudocode-Beispiel:

```
UpdateExpression: "SET currency = currency - :amount"
ConditionExpression: "currency >= :minAmount"
```



Primary key		Attributes									
Partition key: PK	Sort key: SK	Type	CreatedAt	UpdatedAt	Nickname	Email	Gender	Avatar	Currency	PlayerLevel	PlayerHealth	PlayerExperience
player001	#METADATA #player001	Metadata	1618500000	1620000000	John	john@example.com	male	s3://gaming-blki65wn3bgc-lob-avatar/player001.png	500 Updated to 500-Amount	10	80	1000

Wenn wir bei der Entwicklung mit DynamoDB unteilbare Zähler zum Verringern des Inventars verwenden, können wir die Idempotenz sicherstellen, indem wir die optimistische Sperre verwenden. Dies ist ein Pseudocode-Beispiel für unteilbare Zähler:

```
UpdateExpression: "SET ItemCount = ItemCount - :incr"
expression-attribute-values: '{":incr":{"N":"1"}}'
```

Primary key		Attributes				
Partition key: PK	Sort key: SK	Type	ItemName	ItemType	ItemCount	ItemAttributes
player001	ITEMS#001	Item	Health Potion	Consumable	5 Updated to 4	{"M":{"HP": {"N":"50"}}}

Darüber hinaus muss in einem Szenario, in dem der Spieler einen Gegenstand mit einer Währung kauft, der gesamte Vorgang die Währung abziehen und gleichzeitig einen Gegenstand hinzufügen. Wir können DynamoDB-Transaktionen verwenden, um mehrere Aktionen zu gruppieren und sie als einzelne all-or-nothing `TransactWriteItems` `TransactGetItems` ODER-Operation einzureichen. `TransactWriteItems`ist eine synchrone und idempotente Schreiboperation, die bis zu 100 Schreibaktionen in einer einzigen Operation gruppiert. all-or-nothing Die Aktionen werden atomarisch ausgeführt, d. h. entweder sind alle von ihnen oder keine von ihnen erfolgreich. Mit Transaktionen lässt sich das Risiko beseitigen, dass eine Währung dupliziert wird oder verschwindet. Weitere Informationen zu Transaktionen finden Sie unter DynamoDB-Transaktionen-Beispiel.

Alle Zugriffsmuster und wie das Schemadesign sie behandelt, sind in der folgenden Tabelle zusammengefasst:

Zugriffsmuster	Basis-table/GSI/LSI	Operation	Partitionsschlüsselwert	Sortierschlüsselwert	Sonstige Bedingungen/Filter
getPlayerFriends	Basistabelle	GetItem	PK=PlayerID	SK=“FRIENDS#playerID”	
getPlayerAll-Profil	Basistabelle	Query	PK=PlayerID		
getPlayerAllArtikel	Basistabelle	Query	PK=PlayerID	SK begins_with “ITEMS#”	
getPlayerSpecificArtikel	Basistabelle	Query	PK=PlayerID	SK begins_with “ITEMS#”	Filterausdruck: "ItemType=:itemType“ expressionAttributeValues: {„:itemType“: „Waffe“}
updateCharacterAttributes	Basistabelle	UpdateItem	PK=PlayerID	SK=“#METADATA#playerID”	UpdateExpression: „SET-Währung = Währung -:Betrag“: „Währung >=:Mindestbetrag“ ConditionExpression

Zugriffsmuster	Basis-table/GSI/LSI	Operation	Partitionsschlüsselwert	Sortierschlüsselwert	Sonstige Bedingungen/Filter
updateItemCount	Basistabelle	UpdateItem	PK=PlayerID	SK =“ITEMS#ItemID”	Ausdruck aktualisieren: „SET ItemCount = ItemCount -:incr“: '{“ :incr“ expression-attribute-values: {"N“ :"1"}}'

Endgültiges Schema des Gaming-Profils

Dies ist das endgültige Schemadesign. Informationen zum Herunterladen dieses Schemadesign als JSON-Datei finden Sie unter DynamoDB-Beispiele auf. GitHub

Basistabelle:

Primary key		Attributes										
Partition key: PK	Sort key: SK											
player001	#METADATA #player001	Type	CreatedAt	UpdatedAt	Nickname	Email	Gender	Avatar	Currency	PlayerLevel	PlayerHealth	PlayerExperience
		Metadata	1618500000	1620000000	John	john@example.com	male	s3://gaming-blki65wn3bgc-lob-avatar/player001.png	500	10	80	1000
	ACTIVITY#001	Type	ActivityEndTime	ActivityName	ActivityReward	ActivityStartTime	ActivityType					
		Activity	1647475199	Hunting Trip	{"M":{"Gold":{"N":"50"},"XP":{"N":"200"}}}	1647388800	Hunting					
	ACTIVITY#002	Type	ActivityEndTime	ActivityName	ActivityReward	ActivityStartTime	ActivityType					
		Activity	1647647999	Mining Adventure	{"M":{"Gold":{"N":"1000"},"XP":{"N":"500"}}}	1647561600	Mining					
	ACTIVITY#003	Type	ActivityEndTime	ActivityName	ActivityReward	ActivityStartTime	ActivityType					
		Activity	1647820799	Arena Challenge	{"M":{"Gold":{"N":"2000"},"XP":{"N":"1000"}}}	1647734400	Arena					
	EQUIPMENTS#001	Type	EquipmentName	EquipmentType	EquipmentAttributes							
		Equipment	Sword of the Dragon	Weapon	{"M":{"ATK":{"N":"100"},"DEF":{"N":"50"}}}							
	EQUIPMENTS#001EQUIPMENTS#002	Type	EquipmentName	EquipmentType	EquipmentAttributes							
		Equipment	Armor of the Knight	Armor	{"M":{"DEF":{"N":"100"}}}							
	EQUIPMENTS#003	Type	EquipmentName	EquipmentType	EquipmentAttributes							
		Equipment	Ring of the Mage	Accessory	{"M":{"SP":{"N":"50"}}}							
	FRIENDS#player001	Type	FriendList									
		Friends	[{"M":{"FriendId":{"S":"player002"},"FriendName":{"S":"Alice"}}},{"M":{"FriendId":{"S":"player003"},"FriendName":{"S":"Bob"}}}]									
	ITEMS#001	Type	ItemName	ItemType	ItemCount	ItemAttributes						
		Item	Health Potion	Consumable	5	{"M":{"HP":{"N":"50"}}}						
	ITEMS#002	Type	ItemName	ItemType	ItemCount	ItemAttributes						
		Item	Armor of the Knight	Armor	1	{"M":{"DEF":{"N":"100"}}}						
	ITEMS#003	Type	ItemName	ItemType	ItemCount	ItemAttributes						
		Item	Sword of the Dragon	Weapon	1	{"M":{"ATK":{"N":"100"},"DEF":{"N":"50"}}}						
	TASK#001	Type	TaskName	TaskDescription	TaskStatus	TaskReward						
		Task	Find the Lost TreasureFind the Lost Treasure	Get clues from a lost adventurer and find the lost treasure.	InProgress	{"M":{"Gold":{"N":"100"},"XP":{"N":"50"}}}						
	TASK#002	Type	TaskName	TaskDescription	TaskStatus	TaskReward						
		Task	Defeat Magic Monsters	Go to the Magic Forest and defeat three magic monsters.	Completed	{"M":{"Gold":{"N":"200"},"XP":{"N":"100"}}}						
	TASK#003	Type	TaskName	TaskDescription	TaskStatus	TaskReward						
		Task	Rescue the Princess	Go to the Demon King's Castle and rescue the princess who is being held captive by the Demon King.	Available	{"M":{"Gold":{"N":"500"},"XP":{"N":"200"}}}						

Verwendung von NoSQL Workbench mit diesem Schemadesign

Sie können dieses endgültige Schema in NoSQL Workbench importieren, um Ihr neues Projekt weiter zu untersuchen und zu bearbeiten. NoSQL Workbench ist ein visuelles Tool, das Features zur Datenmodellierung, Datenvisualisierung und Abfrageentwicklung für DynamoDB bereitstellt. Gehen Sie folgendermaßen vor, um zu beginnen:

1. Laden Sie NoSQL Workbench herunter. Weitere Informationen finden Sie unter the section called “Download”.
2. Laden Sie die oben aufgeführte JSON-Schemadatei herunter, die bereits das NoSQL-Workbench-Modellformat aufweist.
3. Importieren Sie die JSON-Schemadatei in NoSQL Workbench. Weitere Informationen finden Sie unter the section called “Importieren eines vorhandenen Modells”.
4. Nach dem Import in NOSQL Workbench können Sie das Datenmodell bearbeiten. Weitere Informationen finden Sie unter the section called “Bearbeiten eines vorhandenen Modells”.

Schemadesign des Systems zur Beschwerdeverwaltung in DynamoDB

Anwendungsfall eines Systems zur Beschwerdeverwaltung für Unternehmen

DynamoDB ist eine Datenbank, die sich gut für den Anwendungsfall eines Systems zur Beschwerdeverwaltung (oder eines Kontaktzentrums) eignet, da es sich bei den meisten damit verbundenen Zugriffsmustern um Transaktionsabfragen handelt, die auf Schlüsselwerten basieren. Die typischen Zugriffsmuster in diesem Szenario wären:

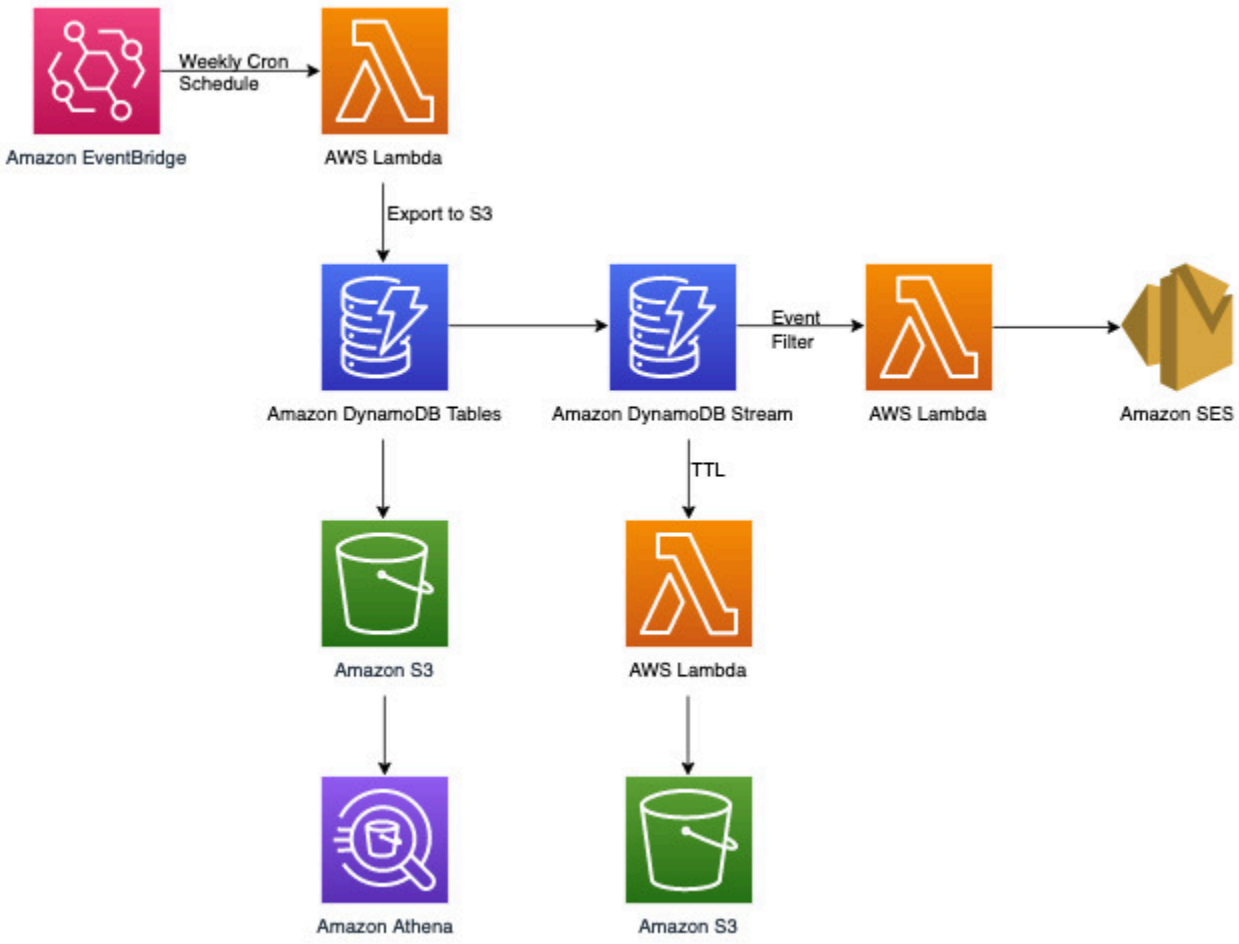
- Beschwerden erstellen und aktualisieren
- Eine Beschwerde weiterleiten
- Kommentare zu einer Beschwerde erstellen und lesen
- Alle Beschwerden eines Kunden abrufen
- Alle Kommentare eines Kundendienstmitarbeiters und alle Eskalationen abrufen

Einige Kommentare können Anlagen enthalten, in denen die Beschwerde oder Lösung beschrieben wird. Dies sind zwar alles wichtige Zugriffsmuster, es können jedoch zusätzliche Anforderungen gelten, z. B. das Versenden von Benachrichtigungen, wenn ein neuer Kommentar zu einer Beschwerde hinzugefügt wird, oder die Durchführung analytischer Abfragen, um die Verteilung der Beschwerden nach Schweregrad (oder Leistung des Kundendienstmitarbeiters) pro Woche

zu ermitteln. Eine weitere Anforderung im Zusammenhang mit dem Lebenszyklusmanagement oder der Einhaltung von Vorschriften wäre die Archivierung der Beschwerdedaten nach dreijähriger Protokollierung der Beschwerde.

Architekturdiagramm des Systems zur Beschwerdeverwaltung

Das folgende Diagramm ist ein Architekturdiagramm des Beschwerdemanagementsystems. Dieses Diagramm zeigt die verschiedenen AWS-Service Integrationen, die das Beschwerdemanagementsystem verwendet.



Abgesehen von den transaktionalen Zugriffsmustern mit Schlüsselwerten, die wir später im Abschnitt zur DynamoDB-Datenmodellierung behandeln, gibt es drei Anforderungen, die nicht transaktional sind. Das obige Architekturdiagramm kann in die folgenden drei Workflows unterteilt werden:

1. Eine Benachrichtigung senden, wenn ein neuer Kommentar zu einer Beschwerde hinzugefügt wird
2. Analytische Abfragen für wöchentliche Daten durchführen

3. Daten archivieren, die älter als drei Jahre sind

Schauen wir sie uns jeweils genauer an.

Eine Benachrichtigung senden, wenn ein neuer Kommentar zu einer Beschwerde hinzugefügt wird

Wir können diese Anforderung mit folgendem Workflow erfüllen:

DynamoDB-Streams ist ein Mechanismus zur Erfassung von Änderungsdaten, mit dem alle Schreibaktivitäten in Ihren DynamoDB-Tabellen aufgezeichnet werden. Sie können Lambda-Funktionen so konfigurieren, dass sie bei einigen oder allen dieser Änderungen ausgelöst werden. Ein Ereignisfilter kann auf Lambda-Triggern konfiguriert werden, um Ereignisse herauszufiltern, die für den Anwendungsfall nicht relevant sind. In diesem Fall können wir einen Filter verwenden, um Lambda nur dann auszulösen, wenn ein neuer Kommentar ausgelöst wird, und um eine Benachrichtigung an die entsprechenden E-Mail-IDs zu senden, die von AWS -Manager für Secrets oder einem anderen Speicher für Anmeldeinformationen abgerufen werden können.

Analytische Abfragen für wöchentliche Daten durchführen

DynamoDB eignet sich für Workloads, die sich hauptsächlich auf die Online-Transaktionsverarbeitung (OLTP) konzentrieren. Für die anderen 10- bis 20-prozentigen Zugriffsmuster mit analytischen Anforderungen können die Daten mit dem verwalteten Feature Nach Amazon S3 exportieren ohne Auswirkung auf den Live-Datenverkehr auf der DynamoDB-Tabelle nach S3 exportiert werden. Schauen Sie sich diesen Workflow unten an:

Amazon EventBridge kann verwendet werden, um AWS Lambda nach Zeitplan auszulösen. Sie können damit einen Cron-Ausdruck für den regelmäßigen Lambda-Aufruf konfigurieren. Lambda kann den `ExportToS3`API-Aufruf aufrufen DynamoDB-Daten in S3 speichern. Auf diese S3-Daten kann dann eine SQL-Engine wie Amazon Athena zugreifen, um analytische Abfragen für DynamoDB-Daten auszuführen, ohne den Workload der Live-Transaktion der Tabelle zu beeinträchtigen. Eine Athena-Beispielabfrage zur Ermittlung der Anzahl der Beschwerden pro Schweregrad würde wie folgt aussehen:

```
SELECT Item.severity.S as "Severity", COUNT(Item) as "Count"
FROM "complaint_management"."data"
WHERE NOT Item.severity.S = ''
GROUP BY Item.severity.S ;
```

Diese führt zu folgendem Athena-Abrageergebnis:

Results (3)

Search rows

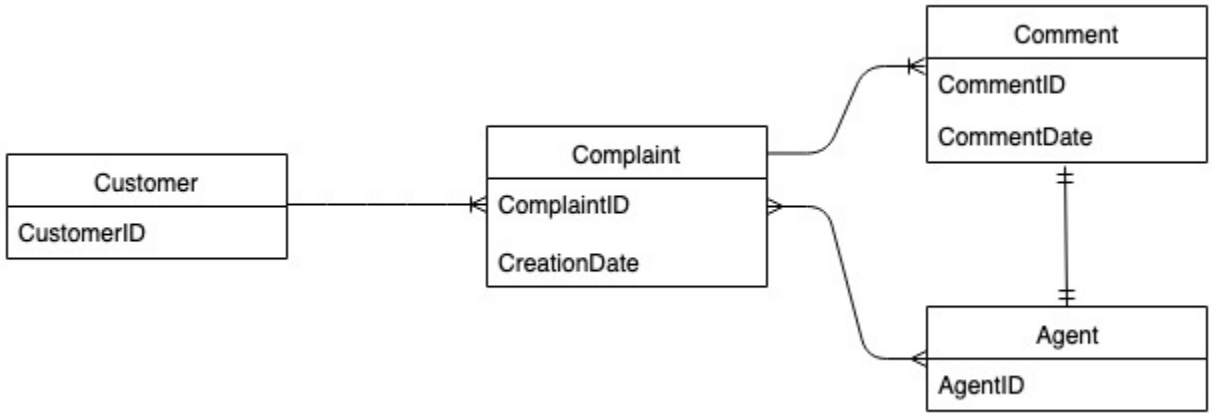
#	Severity	Count
1	P3	1
2	P2	2
3	P1	1

Daten archivieren, die älter als drei Jahre sind

Sie können mit dem DynamoDB-Feature Time to Live (TTL) veraltete Daten ohne Zusatzkosten (außer im Fall von globalen Tabellenreplikaten für die Version 2019.11.21 (aktuell), bei der TTL-Löschungen, die in andere Regionen repliziert wurden, Schreibkapazität verbrauchen) aus Ihrer DynamoDB-Tabelle löschen. Diese Daten werden in DynamoDB Streams angezeigt und können für eine Archivierung in Amazon S3 genutzt werden. Der Arbeitsablauf für diese Anforderung sieht wie folgt aus:

Beziehungsdiagramm der Entitäten des Systems zur Beschwerdeverwaltung

Dies ist das Diagramm der Entitätsbeziehungen (Entity Relationship Diagram, ERD), das wir für das Schemadesign für ein System zur Beschwerdeverwaltung verwenden werden.



Zugriffsmuster für das System zur Beschwerdeverwaltung

Dies sind die Zugriffsmuster, die wir für das Schemadesign für die Beschwerdeverwaltung berücksichtigen werden.

1. createComplaint
2. updateComplaint
3. updateSeveritybyComplaintID

4. getComplaintByID der Beschwerde
5. addCommentByBeschwerde-ID
6. getAllCommentsByComplaintAusweis
7. getLatestCommentByComplaintAUSWEIS
8. AComplaintbyIDAndKundenbeschwerde-ID abrufen
9. getAllComplaintsByCustomerID
10. escalateComplaintByID der Beschwerde
11. getAllEscalatedBeschwerden
12. getEscalatedComplaintsByAgentID (Reihenfolge vom neuesten zum ältesten)
13. getCommentsByAgentID (zwischen zwei Daten)

Schemadesignentwicklung des Systems zur Beschwerdeverwaltung

Da es sich um ein System zur Beschwerdeverwaltung handelt, drehen sich die meisten Zugriffsmuster um eine Beschwerde als primäre Entität. Da die `ComplaintID` eine hohe Priorität aufweist, ist eine gleichmäßige Verteilung der Daten auf den zugrunde liegenden Partitionen gewährleistet. Sie ist auch das häufigste Suchkriterium für unsere identifizierten Zugriffsmuster. Deshalb ist `ComplaintID` ein guter Kandidat für den Partitionsschlüssel in diesem Datensatz.

Schritt 1: Zugriffsmuster 1 (**`createComplaint`**), 2 (**`updateComplaint`**), 3 (**`updateSeveritybyComplaintID`**) und 4 (**`getComplaintByComplaintID`**) angehen

Wir können einen generischen Sortierschlüssel mit dem Wert „Metadaten“ (oder „AA“) verwenden, um beschwerdespezifische Informationen zu speichern, z. B.`CustomerID`, `State`, `Severity` und `CreationDate`. Wir verwenden Singleton-Operationen mit `PK=ComplaintID` und `SK="metadata"`, um Folgendes zu tun:

1. `PutItem`, um eine neue Beschwerde zu erstellen
2. `UpdateItem`, um den Schweregrad oder andere Felder in den Metadaten der Beschwerde zu aktualisieren
3. `GetItem`, um Metadaten für die Beschwerde abzurufen

Primary key		Attributes				
Partition key: PK	Sort key: SK					
Complaint1321	metadata	customer_id	current_state	creation_time	severity	complaint_description
		custXYZ	assigned	2023-05-10T15:58:00	P2	<Complaint Description>

Schritt 2: Zugriffsmuster 5 (**`addCommentByComplaintID`**) angehen

Dieses Zugriffsmuster erfordert ein one-to-many Beziehungsmodell zwischen einer Beschwerde und Kommentaren zu der Beschwerde. Wir werden hier die vertikale Partitionierungstechnik nutzen, um einen Sortierschlüssel zu verwenden und eine Elementsammlung mit verschiedenen Datentypen zu erstellen. Wenn wir uns die Zugriffsmuster 6 (`getAllCommentsByComplaintID`) und 7 (`getLatestCommentByComplaintID`) ansehen, wissen wir, dass Kommentare nach Zeit sortiert werden müssen. Es können auch mehrere Kommentare gleichzeitig eingehen, sodass wir die Technik zusammengesetzter Sortierschlüssel zum Anfügenvon Zeit und `CommentID` im Sortierschlüsselattribut verwenden können.

Andere Optionen für den Umgang mit solchen möglichen Kommentarkollisionen sind die Erhöhung der Granularität für den Zeitstempel oder das Hinzufügen einer inkrementellen Zahl als Suffix, anstatt der `Comment_ID`. In diesem Fall stellen wir dem Sortierschlüsselwert für Elemente, die Kommentaren entsprechen, „comm#“ voran, um bereichsbasierte Operationen zu ermöglichen.

Wir müssen auch sicherstellen, dass `currentState` in den Beschwerdemetadaten den Status angeben, wenn ein neuer Kommentar hinzugefügt wird. Das Hinzufügen eines Kommentars kann darauf hinweisen, dass die Beschwerde einem Kundendienstmitarbeiter zugewiesen wurde oder dass sie gelöst wurde usw. Um das Hinzufügen von Kommentaren und die Aktualisierung des aktuellen Status in den Metadaten der Beschwerde in gewisser all-or-nothing Weise zu bündeln, werden wir die TransactWriteItemsAPI verwenden. Der sich daraus ergebende Tabellenstatus sieht nun wie folgt aus:

Primary key		Attributes				
Partition key: PK	Sort key: SK					
Complaint1321	comm#2023-05-10T16:00:00#comm3	comm_id	comm_date	complaint_state	comm_text	agentID
		comm3	2023-05-10T16:00:00	investigating	<Comment text>	AgentB
	metadata	customer_id	current_state	creation_time	severity	complaint_description
		custXYZ	investigating	2023-05-10T15:58:00	P2	<Complaint Description>

Fügen wir der Tabelle weitere Daten und auch `ComplaintID` als separates Feld aus unseremPK hinzu, um das Modell zukunftssicher zu machen, falls wir zusätzliche Indizes für `ComplaintID` benötigen. Beachten Sie auch, dass einige Kommentare möglicherweise Anlagen

enthalten, die wir in Amazon Simple Storage Service speichern und nur deren Referenzen oder URLs in DynamoDB speichern. Es ist eine bewährte Methode, die Transaktionsdatenbank so schlank wie möglich zu halten, um Kosten und Leistung zu optimieren. Die Daten sehen jetzt wie folgt aus:

Primary key		Attributes					
Partition key: PK	**Sort key: SK**						
Complaint123	comm#2023-04-30T12:00:24#comm1	comm_id	comm_date	complaint_state	comm_text	agentID	
		comm1	2023-04-30T12:00:24	investigating	<comm text>	AgentA	
	comm#2023-04-30T12:35:54#comm2	comm_id	comm_date	complaint_state	comm_text	attachments	agentID
		comm2	2023-04-30T12:35:54	resolved	<comm text>	["s3://URL_for_attachment1","s3://URL_for_attachment2"]	AgentA
	metadata	customer_id	complaint_id	current_state	creation_time	severity	complaint_description
		custABC	Complaint123	resolved	2023-04-30T12:00:00	P2	<description text>
Complaint1321	comm#2023-05-10T16:00:00#comm3	comm_id	comm_date	complaint_state	comm_text	agentID	
		comm3	2023-05-10T16:00:00	investigating	<comm text>	AgentB	
	metadata	customer_id	complaint_id	current_state	creation_time	severity	complaint_description
		custXYZ	Complaint1321	investigating	2023-05-10T15:58:00	P2	<descr_text>

Schritt 3: Zugriffsmuster 6 (**`getAllCommentsByComplaintID`**) und 7 (**`getLatestCommentByComplaintID`**) angehen

Wenn Sie alle Kommentare zu einer Beschwerde abrufen möchten, können wir die `query`-Operation mit der `begins_with`-Bedingung für den Sortierschlüssel verwenden. Anstatt zusätzliche Lesekapazität zum Lesen des Metadateneintrags zu verbrauchen und dann den Aufwand für das Filtern der relevanten Ergebnisse aufzuwenden, hilft uns eine solche Sortierschlüsselbedingung dabei, nur die erforderlichen Elemente zu lesen. Zum Beispiel würde eine Abfrageoperation mit `PK=Complaint123` und `SK begins_with comm#` Folgendes zurückgeben und dabei den Metadateneintrag überspringen:

Primary key		Attributes					
Partition key: PK	**Sort key: SK**						
Complaint123	comm#2023-04-30T12:00:24#comm1	comm_id	comm_date	complaint_state	comm_text	agentID	
		comm1	2023-04-30T12:00:24	investigating	<comm text>	AgentA	
	comm#2023-04-30T12:35:54#comm2	comm_id	comm_date	complaint_state	comm_text	attachments	agentID
		comm2	2023-04-30T12:35:54	resolved	<comm text>	["s3://URL_for_attachment1","s3://URL_for_attachment2"]	AgentA
	metadata	customer_id	complaint_id	current_state	creation_time	severity	complaint_description
		custABC	Complaint123	resolved	2023-04-30T12:00:00	P2	<description text>
Complaint1321	comm#2023-05-10T16:00:00#comm3	comm_id	comm_date	complaint_state	comm_text	agentID	
		comm3	2023-05-10T16:00:00	investigating	<comm text>	AgentB	
	metadata	customer_id	complaint_id	current_state	creation_time	severity	complaint_description
		custXYZ	Complaint1321	investigating	2023-05-10T15:58:00	P2	<descr_text>

Da wir den letzten Kommentar für eine Beschwerde in Muster 7 benötigen (`getLatestCommentByComplaintID`), verwenden wir zwei zusätzliche Abfrageparameter:

1. `ScanIndexForward` sollte auf False festgelegt werden, um die Ergebnisse in absteigender Reihenfolge zu sortieren
2. `Limit` sollte auf 1 festgelegt werden, um den neuesten (nur einen) Kommentar abzurufen

Ähnlich wie bei Zugriffsmuster 6 (`getAllCommentsByComplaintID`), überspringen wir die Metadateneingabe mit `begins_with comm#` als Sortierschlüsselbedingung. Jetzt können Sie das Zugriffsmuster 7 für dieses Design ausführen, indem Sie die Abfrageoperation mit `PK=Complaint123` und `SK=begins_with comm#`, `ScanIndexForward=False` und `Limit` verwenden 1. Das folgende anvisierte Element wird als Ergebnis zurückgegeben:

Partition key: PK	Sort key: SK	Attributes					
	comm#2023-04-30T12:00:24#comm1	comm_id	comm_date	complaint_state	comm_text	agentID	
		comm1	2023-04-30T12:00:24	investigating	<comm text>	AgentA	
Complaint123	comm#2023-04-30T12:35:54#comm2	comm_id	comm_date	complaint_state	comm_text	attachments	agentID
		comm2	2023-04-30T12:35:54	resolved	<comm text>	["s3://URL_for_attachment1","s3://URL_for_attachment2"]	AgentA
	metadata	customer_id	complaint_id	current_state	creation_time	severity	complaint_description
		custABC	Complaint123	resolved	2023-04-30T12:00:00	P2	<description text>
Complaint1321	comm#2023-05-10T16:00:00#comm3	comm_id	comm_date	complaint_state	comm_text	agentID	
		comm3	2023-05-10T16:00:00	investigating	<comm text>	AgentB	
	metadata	customer_id	complaint_id	current_state	creation_time	severity	complaint_description
		custXYZ	Complaint1321	investigating	2023-05-10T15:58:00	P2	<descr_text>

Fügen wir der Tabelle weitere Dummy-Daten hinzu.

Primary key		Attributes					
Partition key: PK	**Sort key: SK**						
Complaint123	comm#2023-04-30T12:00:24#comm1	comm_id	comm_date	complaint_state	comm_text	agentID	
		comm1	2023-04-30T12:00:24	investigating	<comm text>	AgentA	
	comm#2023-04-30T12:35:54#comm2	comm_id	comm_date	complaint_state	comm_text	attachments	agentID
		comm2	2023-04-30T12:35:54	resolved	<comm text>	["s3://URL_for_attachment1","s3://URL_for_attachment2"]	AgentA
	metadata	customer_id	complaint_id	current_state	creation_time	severity	complaint_description
		custABC	Complaint123	resolved	2023-04-30T12:00:00	P2	<description text>
Complaint1321	comm#2023-05-10T16:00:00#comm3	comm_id	comm_date	complaint_state	comm_text	agentID	
		comm3	2023-05-10T16:00:00	investigating	<comm text>	AgentB	
	metadata	customer_id	complaint_id	current_state	creation_time	severity	complaint_description
		custXYZ	Complaint1321	investigating	2023-05-10T15:58:00	P2	<descr_text>
Complaint1444	comm#2022-12-31T19:32:00#comm4	comm_id	comm_date	complaint_state	comm_text		
		comm4	2022-12-31T19:32:00	waiting	<comm text>		
	comm#2022-12-31T19:40:00#comm5	comm_id	comm_date	complaint_state	comm_text	attachments	agentID
		comm5	2022-12-31T19:40:00	assigned	<comm text>	["s3://URL_for_attachment1"]	AgentC
	metadata	customer_id	complaint_id	current_state	creation_time	severity	complaint_description
		custXY32	Complaint1444	assigned	2022-12-31T19:39:57	P1	<description text>
Complaint0987	metadata	customer_id	complaint_id	current_state	creation_time	severity	complaint_description
		custXYZ	Complaint0987	assigned	2023-06-10T12:30:08	P3	<description text>

Schritt 4: Zugriffsmuster 8 (**`getAComplaintbyCustomerIDAndComplaintID`**) und 9 (**`getAllComplaintsByCustomerID`**) angehen

Zugriffsmuster 8 (`getAComplaintbyCustomerIDAndComplaintID`) und 9 (`getAllComplaintsByCustomerID`) führen ein neues Suchkriterium ein: `CustomerID`. Das Abrufen aus der vorhandenen Tabelle erfordert eine teure `Scan`, um alle Daten zu lesen und dann relevante Elemente für die fragliche `CustomerID` zu filtern. Wir können diese Suche effizienter

gestalten, indem wir einen globalen sekundärer Index (GSI) mit dem Partitionsschlüssel `CustomerID` erstellen. Das one-to-many Verhältnis zwischen Kunde und Beschwerden sowie das Zugriffsmuster 9 (`getAllComplaintsByCustomerID`) zu berücksichtigen, `ComplaintID` wäre der richtige Kandidat für den Sortierschlüssel.

Die Daten im GSI würden so aussehen:

Primary key		Attributes					
Partition key: customer_id	**Sort key: complaint_id**						
custABC	Complaint123	PK	SK	current_state	creation_time	severity	complaint_description
		Complaint123	metadata	resolved	2023-04-30T12:00:00	P2	<description text>
custXY32	Complaint1444	PK	SK	current_state	creation_time	severity	complaint_description
		Complaint1444	metadata	assigned	2022-12-31T19:39:57	P1	<description text>
custXYZ	Complaint0987	PK	SK	current_state	creation_time	severity	complaint_description
		Complaint0987	metadata	assigned	2023-06-10T12:30:08	P3	<description text>
	Complaint1321	PK	SK	current_state	creation_time	severity	complaint_description
		Complaint1321	metadata	investigating	2023-05-10T15:58:00	P2	<descr_text>

Eine Beispielabfrage auf diesem GSI für Zugriffsmuster 8 (`getAComplaintbyCustomerIDAndComplaintID`) wäre: `customer_id=custXYZ`, `sort key=Complaint1321`. Das Ergebnis wäre:

Primary key		Attributes					
Partition key: customer_id	**Sort key: complaint_id**						
custABC	Complaint123	PK	SK	current_state	creation_time	severity	complaint_description
		Complaint123	metadata	resolved	2023-04-30T12:00:00	P2	<description text>
custXY32	Complaint1444	PK	SK	current_state	creation_time	severity	complaint_description
		Complaint1444	metadata	assigned	2022-12-31T19:39:57	P1	<description text>
	Complaint0987	PK	SK	current_state	creation_time	severity	complaint_description
		Complaint0987	metadata	assigned	2023-06-10T12:30:08	P3	<description text>
custXYZ							
	Complaint1321	PK	SK	current_state	creation_time	severity	complaint_description
		Complaint1321	metadata	investigating	2023-05-10T15:58:00	P2	<descr_text>

Bei einem Abruf aller Beschwerden eines Kunden für das Zugriffsmuster 9 (`getAllComplaintsByCustomerID`) würde die Anfrage an den GSI `customer_id=custXYZ`als Bedingung für den Partitionsschlüssel lauten. Das Ergebnis wäre:

Primary key		Attributes					
Partition key: customer_id	**Sort key: complaint_id**						
custABC	Complaint123	PK	SK	current_state	creation_time	severity	complaint_description
		Complaint123	metadata	resolved	2023-04-30T12:00:00	P2	<description text>
custXY32	Complaint1444	PK	SK	current_state	creation_time	severity	complaint_description
		Complaint1444	metadata	assigned	2022-12-31T19:39:57	P1	<description text>
custXYZ	Complaint0987	PK	SK	current_state	creation_time	severity	complaint_description
		Complaint0987	metadata	assigned	2023-06-10T12:30:08	P3	<description text>
	Complaint1321	PK	SK	current_state	creation_time	severity	complaint_description
		Complaint1321	metadata	investigating	2023-05-10T15:58:00	P2	<descr_text>

Schritt 5: Zugriffsmuster 10 (**`escalateComplaintByComplaintID`**) angehen

Dieser Zugriff führt den Eskalationsaspekt ein. Für die Eskalation einer Beschwerde können wir `UpdateItem` verwenden, um Attribute zum vorhandenen Metadatenelement für Beschwerden hinzuzufügen, z. B. `escalated_to` und `escalation_time`. DynamoDB bietet ein flexibles Schemadesign, was bedeutet, dass eine Reihe von Nicht-Schlüsselattributen für verschiedene Elemente einheitlich oder diskret sein kann. Ein Beispiel finden Sie unten:

`UpdateItem with PK=Complaint1444, SK=metadata`

Complaint1444	comm#2022-12-31T19:32:00#comm4	comm_id	comm_date	complaint_state	comm_text			
		comm4	2022-12-31T19:32:00	waiting	<comm text>			
	comm#2022-12-31T19:40:00#comm5	comm_id	comm_date	complaint_state	comm_text	attachments	agentID	
		comm5	2022-12-31T19:40:00	assigned	<comm text>	["s3://URL_for_attachment1"]	AgentC	
	metadata	customer_id	complaint_id	current_state	creation_time	severity	complaint_description	escalated_to	escalation_time
		custXY32	Complaint1444	assigned	2022-12-31T19:39:57	P1	<description text>	AgentB	2023-01-03T04:00:07

Schritt 6: Zugriffsmuster 11 (**`getAllEscalatedComplaints`**) und 12 (**`getEscalatedComplaintsByAgentID`**) angehen

Es wird erwartet, dass nur eine Handvoll Beschwerden aus dem gesamten Datensatz eskaliert werden. Daher würde die Erstellung eines Index für die eskalationsbezogenen Attribute zu effizienten Suchvorgängen sowie zu einem kostengünstigem GSI-Speicher führen. Wir erreichen dies, indem wir die Technik spärlicher Index nutzen. Der GSI mit dem Partitionsschlüssel `escalated_to` und dem Sortierschlüssel `escalation_time` würde so aussehen:

Primary key		Attributes							
Partition key: escalated_to	**Sort key: escalation_time**								
AgentB	2023-01-03T04:00:07	PK	SK	customer_id	complaint_id	current_state	creation_time	severity	complaint_description
		Complaint1444	metadata	custXY32	Complaint1444	assigned	2022-12-31T19:39:57	P1	<description text>
	2023-05-15T14:00:00	PK	SK	customer_id	complaint_id	current_state	creation_time	severity	complaint_description
		Complaint1321	metadata	custXYZ	Complaint1321	investigating	2023-05-10T15:58:00	P2	<descr_text>

Um alle eskalierten Beschwerden für das Zugriffsmuster 11 (`getAllEscalatedComplaints`) abzurufen, scannen wir einfach diesen GSI. Beachten Sie, dass dieser Scan aufgrund der Größe des GSI performant und kosteneffizient ist. Für den Abruf von weitergeleiteten Beschwerden für einen bestimmten Kundendienstmitarbeiter (Zugriffsmuster 12) (`getEscalatedComplaintsByAgentID`)) ist der Partitionsschlüssel `escalated_to=agentID` und für eine Reihenfolge vom neuesten zum ältesten Element legen wir `ScanIndexForward` auf `False` fest.

Schritt 7: Zugriffsmuster 13 (**`getCommentsByAgentID`**) angehen

Für das letzte Zugriffsmuster müssen wir eine Suche nach einer neuen Dimension durchführen: `AgentID`. Wir benötigen auch eine zeitabhängige Reihenfolge, um Kommentare zwischen zwei Datumsangaben zu lesen. Also erstellen wir einen GSI mit dem Partitionsschlüssel `agent_id` und dem Sortierschlüssel `comm_date`. Die Daten in diesem GSI sehen wie folgt aus:

Primary key		Attributes					
Partition key: agentID	**Sort key: comm_date**						
AgentA	2023-04-30T12:00:24	PK	SK	comm_id	complaint_state	comm_text	
		Complaint123	comm#2023-04-30T12:00:24#comm1	comm1	investigating	<comm text>	
	2023-04-30T12:35:54	PK	SK	comm_id	complaint_state	comm_text	attachments
		Complaint123	comm#2023-04-30T12:35:54#comm2	comm2	resolved	<comm text>	["s3://URL_for_attachment1","s3://URL_for_attachment2"]
AgentB	2023-05-10T16:00:00	PK	SK	comm_id	complaint_state	comm_text	
		Complaint1321	comm#2023-05-10T16:00:00#comm3	comm3	investigating	<comm text>	
AgentC	2022-12-31T19:40:00	PK	SK	comm_id	complaint_state	comm_text	attachments
		Complaint1444	comm#2022-12-31T19:40:00#comm5	comm5	assigned	<comm text>	["s3://URL_for_attachment1"]

Eine Beispielabfrage zu diesem GSI ist `partition key agentID=AgentA` und `sort key=comm_date between (2023-04-30T12:30:00, 2023-05-01T09:00:00)`, dessen Ergebnis wie folgt lautet:

Primary key		Attributes					
Partition key: agentID	Sort key: comm_date						
AgentA	2023-04-30T12:00:24	PK	SK	comm_id	complaint_state	comm_text	
		Complaint123	comm#2023-04-30T12:00:24#comm1	comm1	investigating	<comm text>	
	2023-04-30T12:35:54	PK	SK	comm_id	complaint_state	comm_text	attachments
		Complaint123	comm#2023-04-30T12:35:54#comm2	comm2	resolved	<comm text>	["s3://URL_for_attachment1","s3://URL_for_attachment2"]
AgentB	2023-05-10T16:00:00	PK	SK	comm_id	complaint_state	comm_text	
		Complaint1321	comm#2023-05-10T16:00:00#comm3	comm3	investigating	<comm text>	
AgentC	2022-12-31T19:40:00	PK	SK	comm_id	complaint_state	comm_text	attachments
		Complaint1444	comm#2022-12-31T19:40:00#comm5	comm5	assigned	<comm text>	["s3://URL_for_attachment1"]

Alle Zugriffsmuster und wie das Schemadesign sie behandelt, sind in der folgenden Tabelle zusammengefasst:

Zugriffsmuster	Basis-table/ GSI/LSI	Operation	Partitionsschlüsselwert	Sortierschlüsselwert	Sonstige Bedingungen/ Filter
createComplaint	Basistabelle	PutItem	PK=complaint_id	SK=metadata	
updateComplaint	Basistabelle	UpdateItem	PK=complaint_id	SK=metadata	
updateSeveritybyComplaintID	Basistabelle	UpdateItem	PK=complaint_id	SK=metadata	
getComplaintByID der Beschwerde	Basistabelle	GetItem	PK=complaint_id	SK=metadata	

Zugriffsmuster	Basis-table/GSI/LSI	Operation	Partitionsschlüsselwert	Sortierschlüsselwert	Sonstige Bedingungen/Filter
addCommentByBeschwerde-ID	Basistabelle	TransactWriteItems	PK=complaint_id	SK=metadata, SK=comm#comm_date#comm_id	
getAllCommentsByComplaintAusweis	Basistabelle	Query	PK=complaint_id	SK begins_with "child#"	
getLatestCommentByComplaint AUSWEIS	Basistabelle	Query	PK=complaint_id	SK begins_with "child#"	scan_index_forward=False, Limit 1
AComplaintbyIDAndKundenbeschwerde-ID abrufen	Customer_complaint_GSI	Query	customer_id=customer_id	complaint_id = complaint_id	
getAllComplaintsByCustomerID	Customer_complaint_GSI	Query	customer_id=customer_id	–	
escalateComplaintByID der Beschwerde	Basistabelle	UpdateItem	PK=complaint_id	SK=metadata	
getAllEscalatedBeschwerden	Escalations_GSI	Scan	–	–	

Zugriffsmuster	Basis-table/GSI/LSI	Operation	Partitionsschlüsselwert	Sortierschlüsselwert	Sonstige Bedingungen/Filter
getEscalatedComplaintsByAgentID (Reihenfolge vom neuesten zum ältesten)	Escalations_GSI	Query	escalated_to=agent_id	–	scan_index_forward=False
getCommentsByAgentID (zwischen zwei Daten)	Agents_Comments_GSI	Query	agent_id=agent_id	SK zwischen (date1, date2)	

Endgültiges Schema des Systems zur Beschwerdeverwaltung

Dies sind die endgültigen Schemadesigns. Informationen zum Herunterladen dieses Schemadesign als JSON-Datei finden Sie unter DynamoDB-Beispiele auf. GitHub

Basistabelle

Primary key		Attributes							
Partition key: PK	**Sort key: SK**								
Complaint123	comm#2023-04-30T12:00:24#comm1	comm_id	comm_date	complaint_state	comm_text	agentID			
		comm1	2023-04-30T12:00:24	investigating	<comm text>	AgentA			
	comm#2023-04-30T12:35:54#comm2	comm_id	comm_date	complaint_state	comm_text	attachments	agentID		
		comm2	2023-04-30T12:35:54	resolved	<comm text>	["s3://URL_for_attachment1","s3://URL_for_attachment2"]	AgentA		
	metadata	customer_id	complaint_id	current_state	creation_time	severity	complaint_description		
		custABC	Complaint123	resolved	2023-04-30T12:00:00	P2	<description text>		
Complaint1321	comm#2023-05-10T16:00:00#comm3	comm_id	comm_date	complaint_state	comm_text	agentID			
		comm3	2023-05-10T16:00:00	investigating	<comm text>	AgentB			
	metadata	customer_id	complaint_id	current_state	creation_time	severity	complaint_description	escalated_to	escalation_time
		custXYZ	Complaint1321	investigating	2023-05-10T15:58:00	P2	<descr_text>	AgentB	2023-05-15T14:00:00
Complaint1444	comm#2022-12-31T19:32:00#comm4	comm_id	comm_date	complaint_state	comm_text				
		comm4	2022-12-31T19:32:00	waiting	<comm text>				
	comm#2022-12-31T19:40:00#comm5	comm_id	comm_date	complaint_state	comm_text	attachments	agentID		
		comm5	2022-12-31T19:40:00	assigned	<comm text>	["s3://URL_for_attachment1"]	AgentC		
	metadata	customer_id	complaint_id	current_state	creation_time	severity	complaint_description	escalated_to	escalation_time
		custXY32	Complaint1444	assigned	2022-12-31T19:39:57	P1	<description text>	AgentB	2023-01-03T04:00:07
Complaint0987	metadata	customer_id	complaint_id	current_state	creation_time	severity	complaint_description		
		custXYZ	Complaint0987	assigned	2023-06-10T12:30:08	P3	<description text>		

Customer_Complaint_GSI

Primary key		Attributes							
Partition key: customer_id	**Sort key: complaint_id**								
custABC	Complaint123	PK	SK	current_state	creation_time	severity	complaint_description		
		Complaint123	metadata	resolved	2023-04-30T12:00:00	P2	<description text>		
custXY32	Complaint1444	PK	SK	current_state	creation_time	severity	complaint_description	escalated_to	escalation_time
		Complaint1444	metadata	assigned	2022-12-31T19:39:57	P1	<description text>	AgentB	2023-01-03T04:00:07
custXYZ	Complaint0987	PK	SK	current_state	creation_time	severity	complaint_description		
		Complaint0987	metadata	assigned	2023-06-10T12:30:08	P3	<description text>		
	Complaint1321	PK	SK	current_state	creation_time	severity	complaint_description	escalated_to	escalation_time
		Complaint1321	metadata	investigating	2023-05-10T15:58:00	P2	<descr_text>	AgentB	2023-05-15T14:00:00

Escalations_GSI

Primary key		Attributes							
Partition key: escalated_to	Sort key: escalation_time								
AgentB	2023-01-03T04:00:07	PK	SK	customer_id	complaint_id	current_state	creation_time	severity	complaint_description
		Complaint1444	metadata	custXY32	Complaint1444	assigned	2022-12-31T19:39:57	P1	<description text>
	2023-05-15T14:00:00	PK	SK	customer_id	complaint_id	current_state	creation_time	severity	complaint_description
		Complaint1321	metadata	custXYZ	Complaint1321	investigating	2023-05-10T15:58:00	P2	<descr_text>

Agents_Comments_GSI

Primary key		Attributes					
Partition key: agentID	Sort key: comm_date						
AgentA	2023-04-30T12:00:24	PK	SK	comm_id	complaint_state	comm_text	
		Complaint123	comm#2023-04-30T12:00:24#comm1	comm1	investigating	<comm text>	
	2023-04-30T12:35:54	PK	SK	comm_id	complaint_state	comm_text	attachments
		Complaint123	comm#2023-04-30T12:35:54#comm2	comm2	resolved	<comm text>	["s3://URL_for_attachment1","s3://URL_for_attachment2"]
AgentB	2023-05-10T16:00:00	PK	SK	comm_id	complaint_state	comm_text	
		Complaint1321	comm#2023-05-10T16:00:00#comm3	comm3	investigating	<comm text>	
AgentC	2022-12-31T19:40:00	PK	SK	comm_id	complaint_state	comm_text	attachments
		Complaint1444	comm#2022-12-31T19:40:00#comm5	comm5	assigned	<comm text>	["s3://URL_for_attachment1"]

Verwendung von NoSQL Workbench mit diesem Schemadesign

Sie können dieses endgültige Schema in NoSQL Workbench importieren, um Ihr neues Projekt weiter zu untersuchen und zu bearbeiten. NoSQL Workbench ist ein visuelles Tool, das Features zur Datenmodellierung, Datenvisualisierung und Abfrageentwicklung für DynamoDB bereitstellt. Gehen Sie folgendermaßen vor, um zu beginnen:

1. Laden Sie NoSQL Workbench herunter. Weitere Informationen finden Sie unter the section called “Download”.
2. Laden Sie die oben aufgeführte JSON-Schemadatei herunter, die bereits das NoSQL-Workbench-Modellformat aufweist.
3. Importieren Sie die JSON-Schemadatei in NoSQL Workbench. Weitere Informationen finden Sie unter the section called “Importieren eines vorhandenen Modells”.
4. Nach dem Import in NOSQL Workbench können Sie das Datenmodell bearbeiten. Weitere Informationen finden Sie unter the section called “Bearbeiten eines vorhandenen Modells”.

Schemadesign für wiederkehrende Zahlungen in DynamoDB

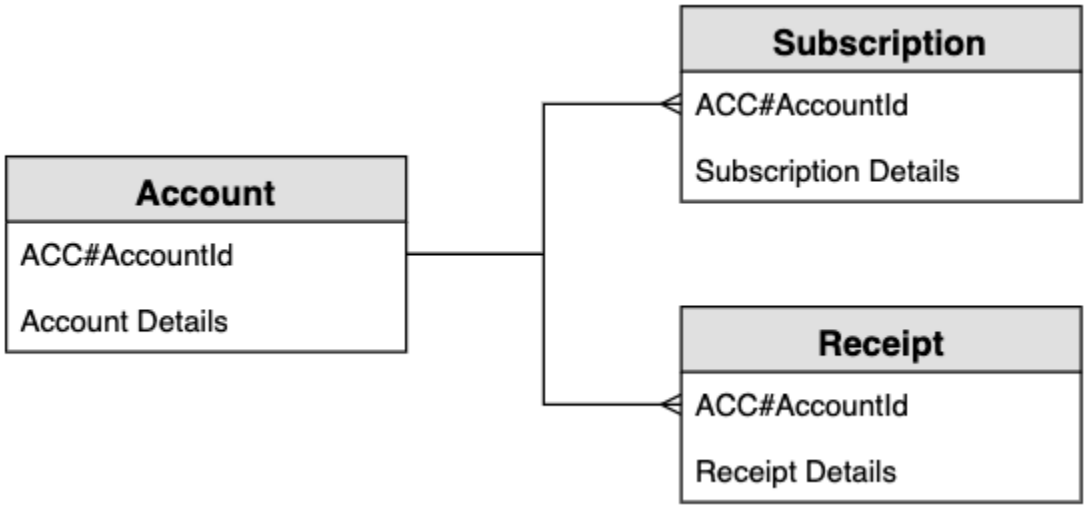
Geschäftsszenario für wiederkehrende Zahlungen

In diesem Anwendungsfall geht es um die Verwendung von DynamoDB zur Implementierung eines Systems für wiederkehrende Zahlungen. Das Datenmodell hat die folgenden Entitäten: Konten, Abonnements und Belege. Zu den Besonderheiten unseres Anwendungsfalls gehört Folgendes:

- Jedes Konto kann mehrere Abonnements haben
- Der Abonnement hat ein `NextPaymentDate`, an dem die nächste Zahlung bearbeitet werden muss und ein `NextReminderDate`, an dem eine E-Mail-Erinnerung an den Kunden gesendet wird
- Es gibt ein Element für Abonnement, das gespeichert und aktualisiert wird, wenn die Zahlung verarbeitet wurde (die durchschnittliche Artikelgröße liegt bei etwa 1 KB und der Durchsatz hängt von der Anzahl der Kontenund Abonnements ab)
- Der Zahlungsabwickler erstellt als Teil des Prozesses auch einen Beleg, der mithilfe eines TTL-Attributs in der Tabelle gespeichert ist und nach einer bestimmten Zeit ablaufen soll

Beziehungsdiagramm für Unternehmen mit wiederkehrenden Zahlungen

Dies ist das Diagramm der Entitätsbeziehungen (Entity Relationship Diagram, ERD), das wir für das Schemadesign für der wiederkehrende Zahlungen verwenden werden.



Zugriffsmuster für das System für wiederkehrende Zahlungen

Dies sind die Zugriffsmuster, die wir für das Schemadesign für das System für wiederkehrende Zahlungen berücksichtigen werden.

1. `createSubscription`
2. `createReceipt`
3. `updateSubscription`
4. `getDueRemindersByDate`
5. `getDuePaymentsByDate`
6. `getSubscriptionsByAccount`
7. `getReceiptsByAccount`

Schemadesign für wiederkehrende Zahlungen

Die generischen Namen `PK` und `SK` werden für Schlüsselattribute verwendet, um das Speichern verschiedener Entitätstypen in derselben Tabelle zu ermöglichen, z. B. die Konto-, Abonnement- und Empfangsentitäten. Der Benutzer erstellt zunächst ein Abonnement. In diesem Fall verpflichtet sich der Benutzer, jeden Monat am selben Tag einen Betrag als Gegenleistung für ein Produkt zu bezahlen. Sie können entscheiden, an welchem Tag des Monats die Zahlung bearbeitet werden soll. Es gibt auch eine Erinnerung, die vor der Bearbeitung der Zahlung gesendet wird. Die Anwendung funktioniert mit zwei Batch-Jobs, die jeden Tag ausgeführt werden: Ein Batch-Job sendet Erinnerungen, die an diesem Tag fällig sind, und der andere Batch-Job verarbeitet alle an diesem Tag fälligen Zahlungen.

Schritt 1: Zugriffsmuster 1 (**`createSubscription`**) angehen

Zugriffsmuster 1 (`createSubscription`) wird verwendet, um das Abonnement zunächst zu erstellen, und die Details
einschließlich `SKU`, `NextPaymentDate`, `NextReminderDate` und `PaymentDetails` werden festgelegt. In diesem Schritt wird der Status der Tabelle für nur ein Konto mit einem Abonnement angezeigt. Die Artikelsammlung kann mehrere Abonnements enthalten, es handelt sich also um eine one-to-many Beziehung.

Primary key		Attributes									
Partition key: PK	Sort key: SK	Email	PaymentDay	PaymentAmount	LastPaymentDate	NextPaymentDate	LastReminderDate	NextReminderDate	SKU	PaymentDetails	CreatedDate
ACC#123	SUB#123#SKU#999	s@s.com	28	12.99	1970-01-01T00:00:00.000Z	2023-05-28	1970-01-01T00:00:00.000Z	2023-05-21	999	{"default-card":{"S":"1234123412341234"},"default-address":{"S":"12 Bridge Street, Birmingham, B12 7ST"}}	2023-05-18T09:41:25.856Z

Schritt 2: Zugriffsmuster 2 (**`createReceipt`**) und 3 (**`updateSubscription`**) angehen

Zugriffsmuster 2 (`createReceipt`) wird verwendet, um die Belegposition zu erstellen. Nachdem die Zahlung jeden Monat bearbeitet wurde, schreibt der Zahlungsabwickler einen Beleg zurück in die Basistabelle. Die Artikelsammlung kann mehrere Belege enthalten, es handelt sich also um eine one-to-many Beziehung. Der Zahlungsabwickler aktualisiert auch das zu aktualisierende Abonnementelement (Zugriff auf Muster 3 (`updateSubscription`)) für den `NextReminderDate` oder den `NextPaymentDate` für den nächsten Monat.

Primary key		Attributes									
Partition key: PK	Sort key: SK										
ACC#123	REC#12023-05-28T14:15:39.24#SKU#999	Email	SKU	ProcessedDate	ProcessedAmount	TTL					
		s@s.com	999	2023-05-28T14:15:39.247Z	12.99	1700318200					
	SUB#123#SKU#999	Email	PaymentDay	PaymentAmount	LastPaymentDate	NextPaymentDate	LastReminderDate	NextReminderDate	SKU	PaymentDetails	CreatedDate
		s@s.com	28	12.99	2023-05-18T14:15:39.247Z	2023-06-28	2023-05-21T14:15:39.247Z	2023-06-21	999	{"default-card":{"S":"1234123412341234"},"default-address":{"S":"12 Bridge Street, Birmingham, B12 7ST"}}	2023-05-18T09:41:25.856Z

Schritt 3: Zugriffsmuster 4 (**`getDueRemindersByDate`**) angehen

Die Anwendung verarbeitet Zahlungserinnerungen für den aktuellen Tag stapelweise. Daher muss die Anwendung in einer anderen Dimension auf die Abonnements zugreifen: Datum statt Konto. Dies ist ein guter Anwendungsfall für einen globalen sekundären Index (GSI). In diesem Schritt fügen wir den Index `GSI-1` hinzu. Dabei wird das `NextReminderDate` als GSI-Partitionsschlüssel genutzt. Wir müssen nicht alle Artikel replizieren. Dieses GSI ist ein spärlicher Index und die Belegpositionen werden nicht repliziert. Wir müssen auch nicht alle Attribute projizieren – wir müssen nur eine Teilmenge der Attribute einbeziehen. Das folgende Bild zeigt das Schema von `GSI-1` und enthält die erforderlichen Informationen, damit die Anwendung die Erinnerungs-E-Mail senden kann.

Primary key		Attributes				
Partition key: NextReminderDate	Sort key: LastReminderDate	SK	PK	SKU	Email	NextPaymentDate
2023-06-21	2023-05-21T14:15:39.247Z	SUB#123#SKU#999	ACC#123	999	s@s.com	2023-06-28

Schritt 4: Zugriffsmuster 5 (**`getDuePaymentsByDate`**) angehen

Die Anwendung verarbeitet Zahlungserinnerungen für den aktuellen Tag auf dieselbe weise in Stapeln wie Erinnerungen. In diesem Schritt fügen wir `GSI-2` hinzu. Dabei wird

das `NextPaymentDate` als GSI-Partitionsschlüssel genutzt. Wir müssen nicht alle Artikel replizieren. Dieses GSI ist ein spärlicher Index und die Belegpositionen werden nicht repliziert. Das folgende Bild zeigt das Schema von `GSI-2`.

Primary key		Attributes						
Partition key: NextPaymentDate	Sort key: LastPaymentDate	PK	SK	Email	PaymentDay	PaymentAmount	SKU	PaymentDetails
2023-06-28	2023-05-18T14:15:39.247Z	ACC#123	SUB#123#SKU#999	s@s.com	28	12.99	999	{"default-card":{"S":"1234123412341234"},"default-address":{"S":"12 Bridge Street, Birmingham, B12 7ST"}}

Schritt 5: Zugriffsmuster 6 (**`getSubscriptionsByAccount`**) und 7 (**`getReceiptsByAccount`**) angehen

Die Anwendung kann alle Abonnements für ein Konto mithilfe einer Abfrage in der Basistabelle abrufen, die auf die Konto-ID abzielt (der PK) und den Bereichsoperator verwendet, um alle Artikel abzurufen, bei denen der SK mit „SUB#“ beginnt. Die Anwendung kann dieselbe Abfragestruktur auch verwenden, um alle Belege abzurufen, indem sie einen Bereichsoperator verwendet, um alle Artikel abzurufen, bei denen der SK mit „REC#“ beginnt. Dadurch können wir die Zugriffsmuster 6 (`getSubscriptionsByAccount`) und 7 (`getReceiptsByAccount`) erfüllen. Die Anwendung verwendet diese Zugriffsmuster, damit der Benutzer seine aktuellen Abonnements und seine früheren Einnahmen für die letzten sechs Monate einsehen kann. In diesem Schritt wird das Tabellenschema nicht geändert. Im Folgenden sehen Sie, wie wir nur auf die Abonnementelemente im Zugriffsmuster 6 abzielen (`getSubscriptionsByAccount`).

Primary key		Attributes									
Partition key: PK	Sort key: SK										
	REC#12023-05-28T14:15:39.24#SKU#999	Email	SKU	ProcessedDate	ProcessedAmount	TTL					
		s@s.com	999	2023-05-28T14:15:39.247Z	12.99	1700318200					
ACC#123	SUB#123#SKU#999	Email	PaymentDay	PaymentAmount	LastPaymentDate	NextPaymentDate	LastReminderDate	NextReminderDate	SKU	PaymentDetails	CreatedDate
		s@s.com	28	12.99	2023-05-18T14:15:39.247Z	2023-06-28	2023-05-21T14:15:39.247Z	2023-06-21	999	{"default-card":{"S":"1234123412341234"},"default-address":{"S":"12 Bridge Street, Birmingham, B12 7ST"}}	2023-05-18T09:41:25.856Z

Alle Zugriffsmuster und wie das Schemadesign sie behandelt, sind in der folgenden Tabelle zusammengefasst:

Zugriffsmuster	Basis-table/GSI/LSI	Operation	Partitionsschlüsselwert	Sortierschlüsselwert
createSubscription	Basistabelle	PutItem	ACC #account_id	SUB#<SUBID>#SKU<SKUID>

Zugriffsmuster	Basis-table/GSI/LSI	Operation	Partitionsschlüsselwert	Sortierschlüsselwert
createReceipt	Basistabelle	PutItem	ACC#account_id	REC#< > #SKU RecieptDate <SKUID>
updateSubscription	Basistabelle	UpdateItem	ACC #account_id	SUB#<SUBID>#SKU<SKUID>
getDueRemindersByDate	GSI-1	Query	<NextReminderDate>	
getDuePaymentsByDate	GSI-2	Query	<NextPaymentDate>	
getSubscriptionsByKonto	Basistabelle	Query	ACC#account_id	SK begins_with “SUB#”
getReceiptsByKonto	Basistabelle	Query	ACC#account_id	SK begins_with “REC#”

Endgültiges Schema für wiederkehrende Zahlungen

Dies sind die endgültigen Schemadesigns. Informationen zum Herunterladen dieses Schemadesign als JSON-Datei finden Sie unter DynamoDB-Beispiele auf. GitHub

Basistabelle

Primary key		Attributes									
Partition key: PK	Sort key: SK										
ACC#123	REC#12023-05-28T14:15:39.24#SKU#999	Email	SKU	ProcessedDate	ProcessedAmount	TTL					
		s@s.com	999	2023-05-28T14:15:39.247Z	12.99	1700318200					
	SUB#123#SKU#999	Email	PaymentDay	PaymentAmount	LastPaymentDate	NextPaymentDate	LastReminderDate	NextReminderDate	SKU	PaymentDetails	CreatedDate
		s@s.com	28	12.99	2023-05-18T14:15:39.247Z	2023-06-28	2023-05-21T14:15:39.247Z	2023-06-21	999	{"default-card":{"S":"1234123412341234"},"default-address":{"S":"12 Bridge Street, Birmingham, B12 7ST"}}	2023-05-18T09:41:25.856Z

GSI-1

Primary key		Attributes				
Partition key: NextReminderDate	Sort key: LastReminderDate					
		SK	PK	SKU	Email	NextPaymentDate
2023-06-21	2023-05-21T14:15:39.247Z	SUB#123#SKU#999	ACC#123	999	s@s.com	2023-06-28

GSI-2

Primary key		Attributes						
Partition key: NextPaymentDate	Sort key: LastPaymentDate							
		PK	SK	Email	PaymentDay	PaymentAmount	SKU	PaymentDetails
2023-06-28	2023-05-18T14:15:39.247Z	ACC#123	SUB#123#SKU#999	s@s.com	28	12.99	999	{"default-card":{"S":"1234123412341234"},"default-address":{"S":"12 Bridge Street, Birmingham, B12 7ST"}}

Verwendung von NoSQL Workbench mit diesem Schemadesign

Sie können dieses endgültige Schema in NoSQL Workbench importieren, um Ihr neues Projekt weiter zu untersuchen und zu bearbeiten. NoSQL Workbench ist ein visuelles Tool, das Features zur Datenmodellierung, Datenvisualisierung und Abfrageentwicklung für DynamoDB bereitstellt. Gehen Sie folgendermaßen vor, um zu beginnen:

1. Laden Sie NoSQL Workbench herunter. Weitere Informationen finden Sie unter the section called “Download”.
2. Laden Sie die oben aufgeführte JSON-Schemadatei herunter, die bereits das NoSQL-Workbench-Modellformat aufweist.
3. Importieren Sie die JSON-Schemadatei in NoSQL Workbench. Weitere Informationen finden Sie unter the section called “Importieren eines vorhandenen Modells”.
4. Nach dem Import in NOSQL Workbench können Sie das Datenmodell bearbeiten. Weitere Informationen finden Sie unter the section called “Bearbeiten eines vorhandenen Modells”.

Überwachung von Gerätestatusaktualisierungen in DynamoDB

In diesem Anwendungsfall geht es um die Verwendung von DynamoDB zur Überwachung von Gerätestatusaktualisierungen (oder -änderungen) in DynamoDB.

Anwendungsfall

Bei IoT-Anwendungsfällen (z. B. Smart Factory) müssen viele Geräte von den Betreibern überwacht werden und ihr Status oder ihre Protokolle werden regelmäßig an ein Überwachungssystem gesendet. Wenn bei einem Gerät ein Problem auftritt, ändert sich sein Status von normal in warning. Je nach Schweregrad und Art des abnormalen Verhaltens des Geräts gibt es unterschiedliche

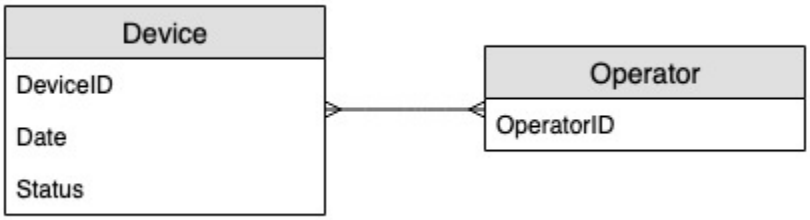
Protokollstufen oder Status. Das System fordert dann einen Betreiber auf, das Gerät zu überprüfen, und dieser leitet das Problem gegebenenfalls an einen Supervisor weiter.

Zu den typischen Zugriffsmustern für dieses System gehören:

- Protokolleintrag für ein Gerät erstellen
- Alle Protokolle für einen bestimmten Gerätestatus abrufen, wobei die neuesten Protokolle zuerst angezeigt werden
- Alle Protokolle für einen bestimmten Betreiber und einen bestimmten Zeitraum abrufen
- Alle eskalierten Protokolle für einen bestimmten Supervisor abrufen
- Alle eskalierten Protokolle mit einem bestimmten Gerätestatus für einen bestimmten Supervisor abrufen
- Alle eskalierten Protokolle mit einem bestimmten Gerätestatus für einen bestimmten Supervisor und ein bestimmtes Datum abrufen

Diagramm der Entitätsbeziehungen

Wir nutzen dieses Diagramm der Entitätsbeziehungen zur Überwachung von Gerätestatusaktualisierungen.



Zugriffsmuster

Diese Zugriffsmuster ziehen wir für die Überwachung von Gerätestatusaktualisierungen in Betracht.

1. `createLogEntryForSpecificDevice`
2. `getLogsForSpecificDevice`
3. `getWarningLogsForSpecificDevice`
4. `getLogsForOperatorBetweenTwoDates`
5. `getEscalatedLogsForSupervisor`

6. `getEscalatedLogsWithSpecificStatusForSupervisor`
7. `getEscalatedLogsWithSpecificStatusForSupervisorForDate`

Entwicklung des Schemadesigns

Schritt 1: Zugriffsmuster 1 (**`createLogEntryForSpecificDevice`**) und 2 (**`getLogsForSpecificDevice`**) angehen

Die Skalierungseinheit für ein Nachverfolgungssystem für Geräte wären individuelle Geräte. In diesem System wird ein Gerät durch eine `deviceID` eindeutig identifiziert. Dadurch wäre die `deviceID` gut für den Partitionsschlüssel geeignet. Jedes Gerät sendet in regelmäßigen Abständen (etwa alle fünf Minuten) Informationen an das Nachverfolgungssystem. Diese Reihenfolge macht das Datum zu einem logischen Sortierkriterium und damit zum Sortierschlüssel. In diesem Fall sähen die Beispieldaten in etwa so aus:

Primary key		Attributes
Partition key: DeviceID	Sort key: State#Date	
d#12345	2020-04-24T14:40:00	State
		WARNING1
	2020-04-24T14:45:00	State
		WARNING1
	2020-04-24T14:50:00	State
		WARNING1
	2020-04-24T14:55:00	State
		NORMAL
d#54321	2020-04-11T05:50:00	State
		WARNING3
	2020-04-11T05:55:00	State
		WARNING3
	2020-04-11T06:00:00	State
		NORMAL
	2020-04-11T09:25:00	State
		WARNING2
	2020-04-11T09:30:00	State
		NORMAL
d#11223	2020-04-27T16:10:00	State
		WARNING4
	2020-04-27T16:15:00	State
		WARNING4

Um Protokolleinträge für ein bestimmtes Gerät abzurufen, können wir einen Query-Vorgang mit dem Partitionsschlüssel `DeviceID="d#12345"` durchführen.

Schritt 2: Zugriffsmuster 3 (**`getWarningLogsForSpecificDevice`**) angehen

Da `State` kein Schlüsselattribut ist, wäre ein Filterausdruck erforderlich, um Zugriffsmuster 3 mit dem aktuellen Schema anzugehen. In DynamoDB werden Filterausdrücke angewendet, nachdem Daten mithilfe von Schlüsselbedingungsausdrücken gelesen wurden. Wenn wir beispielsweise Warnprotokolle für `d#12345` abrufen wollten, würde der Abfragevorgang mit dem Partitionsschlüssel `DeviceID="d#12345"` vier Elemente aus der obigen Tabelle lesen und dann das Element ohne den Status warning herausfiltern. Zur Anwendung im großen Maßstab ist dieser Ansatz nicht effizient. Ein Filterausdruck eignet sich, um abgefragte Elemente auszuschließen, wenn der Anteil der

ausgeschlossenen Elemente gering ist oder die Abfrage selten ausgeführt wird. In Fällen, in denen viele Elemente aus einer Tabelle abgerufen und der Großteil der Elemente herausgefiltert werden, sollten wir jedoch unser Tabellendesign erweitern, damit es effizienter funktioniert.

Ändern wir nun unseren Ansatz für dieses Zugriffsmuster, indem wir zusammengesetzte Sortierschlüssel nutzen. Sie können Beispieldaten aus DeviceStateLog_3.json importieren, wo der Sortierschlüssel geändert wird. `State#Date` Dieser Sortierschlüssel ist eine Zusammensetzung aus den Attributen `State`, `#` und `Date`. In diesem Beispiel dient # als Trennzeichen. Die Daten sehen jetzt in etwa so aus:

Primary key	
Partition key: DeviceID	**Sort key: State#Date**
d#12345	NORMAL#2020-04-24T14:55:00
	WARNING1#2020-04-24T14:40:00
	WARNING1#2020-04-24T14:45:00
	WARNING1#2020-04-24T14:50:00

Die Abfrage wird mit diesem Schema zielgerichteter und eignet sich besser, um nur Warnprotokolle für ein Gerät abzurufen. Die Schlüsselbedingung für die Abfrage verwendet den Partitionsschlüssel `DeviceID="d#12345"`und den Sortierschlüssel `State#Date begins_with “WARNING”`. Diese Abfrage liest nur die drei relevanten Elemente mit Status warning.

Schritt 3: Zugriffsmuster 4 (**`getLogsForOperatorBetweenTwoDates`**) angehen

Sie können DeviceStateLog_4.json D importieren, wo das `Operator` Attribut der `DeviceStateLog` Tabelle mit Beispieldaten hinzugefügt wurde.

Primary key		Attributes		
Partition key: DeviceID	**Sort key: State#Date**			
d#12345	NORMAL#2020-04-24T14:55:00	Operator	Date	State
		Liz	2020-04-24T14:55:00	NORMAL
	WARNING1#2020-04-24T14:40:00	Operator	Date	State
		Liz	2020-04-24T14:40:00	WARNING1
	WARNING1#2020-04-24T14:45:00	Operator	Date	State
		Liz	2020-04-24T14:45:00	WARNING1
	WARNING1#2020-04-24T14:50:00	Operator	Date	State
		Liz	2020-04-24T14:50:00	WARNING1
d#54321	NORMAL#2020-04-11T06:00:00	Operator	Date	State
		Liz	2020-04-11T06:00:00	NORMAL
	NORMAL#2020-04-11T09:30:00	Operator	Date	State
		Sue	2020-04-11T09:30:00	NORMAL
	WARNING2#2020-04-11T09:25:00	Operator	Date	State
		Sue	2020-04-11T09:25:00	WARNING2
	WARNING3#2020-04-11T05:50:00	Operator	Date	State
		Sue	2020-04-11T05:50:00	WARNING3
	WARNING3#2020-04-11T05:55:00	Operator	Date	State
		Liz	2020-04-11T05:55:00	WARNING3
d#11223	WARNING4#2020-04-27T16:10:00	Operator	Date	State
		Sue	2020-04-27T16:10:00	WARNING4
	WARNING4#2020-04-27T16:15:00	Operator	Date	State
		Sue	2020-04-27T16:15:00	WARNING4

Da `Operator` derzeit kein Partitionsschlüssel ist, gibt es keine Möglichkeit, in dieser Tabelle eine direkte Schlüssel-Wert-Suche auf der Grundlage von `OperatorID` durchzuführen. Wir müssen

eine neue Elementauflistung mit einem globalen sekundären Index für `OperatorID` erstellen. Das Zugriffsmuster erfordert eine Suche auf der Grundlage von Daten, also ist Date (Datum) das Sortierschlüsselattribut für den globalen sekundären Index (GSI). So sieht der GSI jetzt aus:

Primary key		Attributes		
Partition key: Operator	**Sort key: Date**			
Liz	2020-04-11T05:55:00	DeviceID	State#Date	State
		d#54321	WARNING3#2020-04-11T05:55:00	WARNING3
	2020-04-11T06:00:00	DeviceID	State#Date	State
		d#54321	NORMAL#2020-04-11T06:00:00	NORMAL
	2020-04-24T14:40:00	DeviceID	State#Date	State
		d#12345	WARNING1#2020-04-24T14:40:00	WARNING1
	2020-04-24T14:45:00	DeviceID	State#Date	State
		d#12345	WARNING1#2020-04-24T14:45:00	WARNING1
	2020-04-24T14:50:00	DeviceID	State#Date	State
		d#12345	WARNING1#2020-04-24T14:50:00	WARNING1
	2020-04-24T14:55:00	DeviceID	State#Date	State
		d#12345	NORMAL#2020-04-24T14:55:00	NORMAL
Sue	2020-04-11T05:50:00	DeviceID	State#Date	State
		d#54321	WARNING3#2020-04-11T05:50:00	WARNING3
	2020-04-11T09:25:00	DeviceID	State#Date	State
		d#54321	WARNING2#2020-04-11T09:25:00	WARNING2
	2020-04-11T09:30:00	DeviceID	State#Date	State
		d#54321	NORMAL#2020-04-11T09:30:00	NORMAL
	2020-04-27T16:10:00	DeviceID	State#Date	State
		d#11223	WARNING4#2020-04-27T16:10:00	WARNING4
	2020-04-27T16:15:00	DeviceID	State#Date	State
		d#11223	WARNING4#2020-04-27T16:15:00	WARNING4

Für Zugriffsmuster 4 (`getLogsForOperatorBetweenTwoDates`) können Sie diesen GSI mit dem Partitionsschlüssel `OperatorID=Liz` und Sortierschlüssel `Date` zwischen `2020-04-11T05:58:00` und `2020-04-24T14:50:00` abfragen.

Primary key		Attributes		
Partition key: Operator	**Sort key: Date**			
	2020-04-11T05:55:00	DeviceID	State#Date	State
		d#54321	WARNING3#2020-04-11T05:55:00	WARNING3
Liz	2020-04-11T06:00:00	DeviceID	State#Date	State
		d#54321	NORMAL#2020-04-11T06:00:00	NORMAL
	2020-04-24T14:40:00	DeviceID	State#Date	State
		d#12345	WARNING1#2020-04-24T14:40:00	WARNING1
	2020-04-24T14:45:00	DeviceID	State#Date	State
		d#12345	WARNING1#2020-04-24T14:45:00	WARNING1
	2020-04-24T14:50:00	DeviceID	State#Date	State
		d#12345	WARNING1#2020-04-24T14:50:00	WARNING1
	2020-04-24T14:55:00	DeviceID	State#Date	State
		d#12345	NORMAL#2020-04-24T14:55:00	NORMAL
Sue	2020-04-11T05:50:00	DeviceID	State#Date	State
		d#54321	WARNING3#2020-04-11T05:50:00	WARNING3
	2020-04-11T09:25:00	DeviceID	State#Date	State
		d#54321	WARNING2#2020-04-11T09:25:00	WARNING2
	2020-04-11T09:30:00	DeviceID	State#Date	State
		d#54321	NORMAL#2020-04-11T09:30:00	NORMAL
	2020-04-27T16:10:00	DeviceID	State#Date	State
		d#11223	WARNING4#2020-04-27T16:10:00	WARNING4
	2020-04-27T16:15:00	DeviceID	State#Date	State
		d#11223	WARNING4#2020-04-27T16:15:00	WARNING4

Schritt 4: Zugriffsmuster 5 (**`getEscalatedLogsForSupervisor`**), 6 (**`getEscalatedLogsWithSpecificStatusForSupervisor`**) und 7 (**`getEscalatedLogsWithSpecificStatusForSupervisorForDate`**) angehen

Wir verwenden einen spärlichen Index, um diese Zugriffsmuster anzugehen.

Globale sekundäre Indizes sind standardmäßig spärlich, sodass nur Elemente aus der Basistabelle, die Primärschlüsselattribute des Indexes enthalten, tatsächlich im Index erscheinen. Das ist eine weitere Möglichkeit, Elemente auszuschließen, die für das modellierte Zugriffsmuster nicht relevant sind.

Sie können DeviceStateLog_6.json importieren, wo das `EscalatedTo` Attribut der `DeviceStateLog` Tabelle mit Beispieldaten hinzugefügt wurde. Wie bereits erwähnt, werden nicht alle Protokolle an einen Supervisor eskaliert.

Primary key		Attributes			
Partition key: DeviceID	**Sort key: State#Date**				
d#12345	NORMAL#2020-04-24T14:55:00	Operator	Date	State	
		Liz	2020-04-24T14:55:00	NORMAL	
	WARNING1#2020-04-24T14:40:00	Operator	Date	State	
		Liz	2020-04-24T14:40:00	WARNING1	
	WARNING1#2020-04-24T14:45:00	Operator	Date	State	
		Liz	2020-04-24T14:45:00	WARNING1	
	WARNING1#2020-04-24T14:50:00	Operator	Date	State	
		Liz	2020-04-24T14:50:00	WARNING1	
d#54321	NORMAL#2020-04-11T06:00:00	Operator	Date	State	
		Liz	2020-04-11T06:00:00	NORMAL	
	NORMAL#2020-04-11T09:30:00	Operator	Date	State	
		Sue	2020-04-11T09:30:00	NORMAL	
	WARNING2#2020-04-11T09:25:00	Operator	Date	State	
		Sue	2020-04-11T09:25:00	WARNING2	
	WARNING3#2020-04-11T05:50:00	Operator	Date	State	
		Sue	2020-04-11T05:50:00	WARNING3	
	WARNING3#2020-04-11T05:55:00	Operator	Date	State	
		Liz	2020-04-11T05:55:00	WARNING3	
d#11223	WARNING4#2020-04-27T16:10:00	Operator	Date	State	
		Sue	2020-04-27T16:10:00	WARNING4	
	WARNING4#2020-04-27T16:15:00	Operator	Date	State	EscalatedTo
		Sue	2020-04-27T16:15:00	WARNING4	Sara

Sie können jetzt einen neuen GSI erstellen, bei dem `EscalatedTo` der Partitionsschlüssel und `State#Date` der Sortierschlüssel ist. Beachten Sie, dass nur Elemente, die `EscalatedTo`- und `State#Date`-Attribute erhalten, im Index erscheinen.

Primary key		Attributes			
Partition key: EscalatedTo	Sort key: State#Date				
		DeviceID	Operator	Date	State
Sara	WARNING4#2020-04-27T16:15:00	d#11223	Sue	2020-04-27T16:15:00	WARNING4

Die übrigen Zugriffsmuster sind wie folgt zusammengefasst:

Alle Zugriffsmuster und wie das Schemadesign sie behandelt, sind in der folgenden Tabelle zusammengefasst:

Zugriffsmuster	Basis-table/ GSI/LSI	Operation	Partitionsschlüsselwert	Sortierschlüsselwert	Sonstige Bedingungen/ Filter
createLogEntryForSpecificDevice	Basistabelle	PutItem	DeviceID=deviceId	State#Date=state#date	
getLogsForSpecificDevice	Basistabelle	Query	DeviceID=deviceId	State#Date begins_with "state1#"	ScanIndexForward = Falsch
getWarningLogsForSpecificDevice	Basistabelle	Query	DeviceID=deviceId	State#Date begins_with "WARNING"	
getLogsForOperatorBetweenTwoDates	GSI-1	Query	Operator=operatorName	Datum zwischen Datum1 und Datum2	
getEscalatedLogsForSupervisor	GSI-2	Query	EscalatedTo=Name des Vorgesetzten		

Zugriffsmuster	Basis-table/GSI/LSI	Operation	Partitionsschlüsselwert	Sortierschlüsselwert	Sonstige Bedingungen/Filter
getEscalatedLogsWithSpecificStatusForSupervisor	GSI-2	Query	EscalatedTo=Name des Vorgesetzten	State#Date begins_with "state1#"	
getEscalatedLogsWithSpecificStatusForSupervisorForDate	GSI-2	Query	EscalatedTo=Name des Vorgesetzten	State#Date begins_with "state1#date1"	

Endgültiges Schema

Dies sind die endgültigen Schemadesigns. Informationen zum Herunterladen dieses Schemadesign als JSON-Datei finden Sie unter DynamoDB-Beispiele auf. GitHub

Basistabelle

Primary key		Attributes			
Partition key: DeviceID	**Sort key: State#Date**				
d#12345	NORMAL#2020-04-24T14:55:00	Operator	Date	State	
		Liz	2020-04-24T14:55:00	NORMAL	
	WARNING1#2020-04-24T14:40:00	Operator	Date	State	
		Liz	2020-04-24T14:40:00	WARNING1	
	WARNING1#2020-04-24T14:45:00	Operator	Date	State	
		Liz	2020-04-24T14:45:00	WARNING1	
	WARNING1#2020-04-24T14:50:00	Operator	Date	State	
		Liz	2020-04-24T14:50:00	WARNING1	
d#54321	NORMAL#2020-04-11T06:00:00	Operator	Date	State	
		Liz	2020-04-11T06:00:00	NORMAL	
	NORMAL#2020-04-11T09:30:00	Operator	Date	State	
		Sue	2020-04-11T09:30:00	NORMAL	
	WARNING2#2020-04-11T09:25:00	Operator	Date	State	
		Sue	2020-04-11T09:25:00	WARNING2	
	WARNING3#2020-04-11T05:50:00	Operator	Date	State	
		Sue	2020-04-11T05:50:00	WARNING3	
	WARNING3#2020-04-11T05:55:00	Operator	Date	State	
		Liz	2020-04-11T05:55:00	WARNING3	
d#11223	WARNING4#2020-04-27T16:10:00	Operator	Date	State	
		Sue	2020-04-27T16:10:00	WARNING4	
	WARNING4#2020-04-27T16:15:00	Operator	Date	State	EscalatedTo
		Sue	2020-04-27T16:15:00	WARNING4	Sara

GSI-1

Primary key		Attributes		
Partition key: Operator	**Sort key: Date**			
Liz	2020-04-11T05:55:00	DeviceID	State#Date	State
		d#54321	WARNING3#2020-04-11T05:55:00	WARNING3
	2020-04-11T06:00:00	DeviceID	State#Date	State
		d#54321	NORMAL#2020-04-11T06:00:00	NORMAL
	2020-04-24T14:40:00	DeviceID	State#Date	State
		d#12345	WARNING1#2020-04-24T14:40:00	WARNING1
	2020-04-24T14:45:00	DeviceID	State#Date	State
		d#12345	WARNING1#2020-04-24T14:45:00	WARNING1
	2020-04-24T14:50:00	DeviceID	State#Date	State
		d#12345	WARNING1#2020-04-24T14:50:00	WARNING1
	2020-04-24T14:55:00	DeviceID	State#Date	State
		d#12345	NORMAL#2020-04-24T14:55:00	NORMAL
Sue	2020-04-11T05:50:00	DeviceID	State#Date	State
		d#54321	WARNING3#2020-04-11T05:50:00	WARNING3
	2020-04-11T09:25:00	DeviceID	State#Date	State
		d#54321	WARNING2#2020-04-11T09:25:00	WARNING2
	2020-04-11T09:30:00	DeviceID	State#Date	State
		d#54321	NORMAL#2020-04-11T09:30:00	NORMAL
	2020-04-27T16:10:00	DeviceID	State#Date	State
		d#11223	WARNING4#2020-04-27T16:10:00	WARNING4
	2020-04-27T16:15:00	DeviceID	State#Date	State
		d#11223	WARNING4#2020-04-27T16:15:00	WARNING4

GSI-2

Primary key		Attributes			
Partition key: EscalatedTo	Sort key: State#Date				
Sara	WARNING4#2020-04-27T16:15:00	DeviceID	Operator	Date	State
		d#11223	Sue	2020-04-27T16:15:00	WARNING4

Verwendung von NoSQL Workbench mit diesem Schemadesign

Sie können dieses endgültige Schema in NoSQL Workbench importieren, um Ihr neues Projekt weiter zu untersuchen und zu bearbeiten. NoSQL Workbench ist ein visuelles Tool, das Features zur Datenmodellierung, Datenvisualisierung und Abfrageentwicklung für DynamoDB bereitstellt. Gehen Sie folgendermaßen vor, um zu beginnen:

1. Laden Sie NoSQL Workbench herunter. Weitere Informationen finden Sie unter the section called “Download”.
2. Laden Sie die oben aufgeführte JSON-Schemadatei herunter, die bereits das NoSQL-Workbench-Modellformat aufweist.
3. Importieren Sie die JSON-Schemadatei in NoSQL Workbench. Weitere Informationen finden Sie unter the section called “Importieren eines vorhandenen Modells”.
4. Nach dem Import in NOSQL Workbench können Sie das Datenmodell bearbeiten. Weitere Informationen finden Sie unter the section called “Bearbeiten eines vorhandenen Modells”.

Verwendung von DynamoDB als Datenspeicher für einen Online-Shop

In diesem Anwendungsfall geht es um die Verwendung von DynamoDB als Datenspeicher für einen Online-Shop (E-Store).

Anwendungsfall

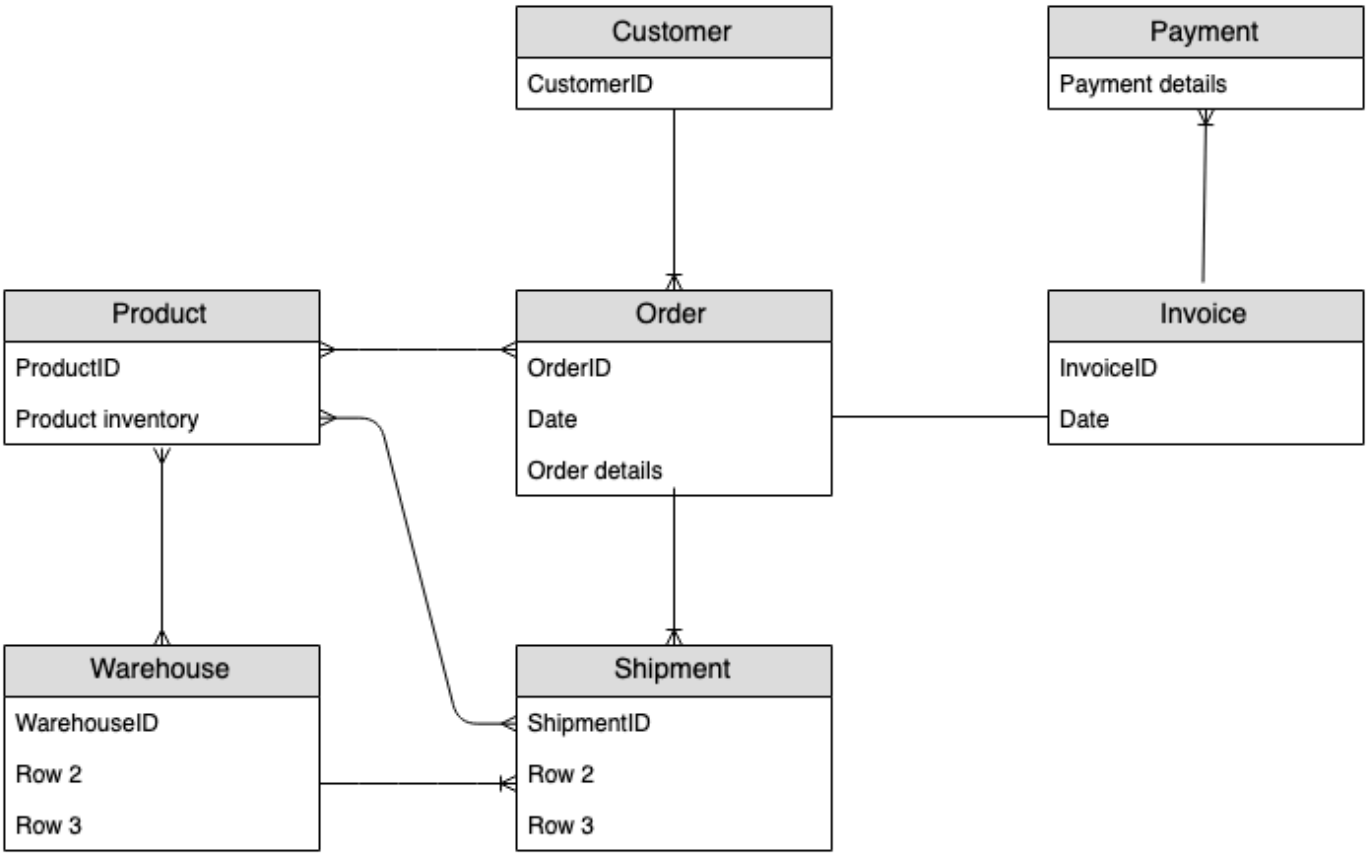
In einem Online-Shop können Benutzer verschiedene Produkte durchsuchen und diese schließlich kaufen. Basierend auf der generierten Rechnung kann ein Kunde mit einem Rabattcode oder einer Geschenkkarte bezahlen und dann den Restbetrag mit einer Kreditkarte begleichen. Die gekauften Produkte werden von einem der Warenlager abgeholt und an die angegebene Adresse versandt. Zu den typischen Zugriffsmustern für einen Online-Shop gehören folgende:

- Kunden für eine bestimmte CustomerId abrufen

- Produkt für eine bestimmte ProductId abrufen
- Warenlager für eine bestimmte WarehouseId abrufen
- Produktbestand für alle Warenlager nach ProductId abrufen
- Bestellung für eine bestimmte OrderId abrufen
- Alle Produkte für eine bestimmte OrderId abrufen
- Rechnung für eine bestimmte OrderId abrufen
- Alle Lieferungen für eine bestimmte OrderId abrufen
- Alle Bestellungen für eine bestimmte ProductId für einen bestimmten Zeitraum abrufen
- Rechnung für eine bestimmte InvoiceId abrufen
- Alle Zahlungen für eine bestimmte InvoiceId abrufen
- Versanddetails für eine bestimmte ShipmentId abrufen
- Alle Lieferungen für eine bestimmte WarehouseId abrufen
- Bestand aller Produkte für eine bestimmte WarehouseId abrufen
- Alle Rechnungen für eine bestimmte CustomerId für einen bestimmten Zeitraum abrufen
- Alle von einer bestimmten CustomerId bestellten Produkte für einen bestimmten Zeitraum abrufen

Diagramm der Entitätsbeziehungen

Dieses Diagramm der Entitätsbeziehungen (Entity Relationship Diagram, ERD) verwenden wir, um DynamoDB als Datenspeicher für einen Online-Shop zu verwenden.



Zugriffsmuster

Das sind die Zugriffsmuster, die wir in Betracht ziehen, wenn wir DynamoDB als Datenspeicher für einen Online-Shop verwenden.

1. `getCustomerByCustomerId`
2. `getProductByProductId`
3. `getWarehouseByWarehouseId`
4. `getProductInventoryByProductId`
5. `getOrderDetailsByOrderId`
6. `getProductByOrderId`
7. `getInvoiceByOrderId`
8. `getShipmentByOrderId`
9. `getOrderByProductIdForDateRange`
10. `getInvoiceByInvoiceId`

11getPaymentByInvoiceId

12getShipmentDetailsByShipmentId

13getShipmentByWarehouseId

14getProductInventoryByWarehouseId

15getInvoiceByCustomerIdForDateRange

16getProductsByCustomerIdForDateRange

Entwicklung des Schemadesigns

Verwenden SieNoSQL-Workbench für DynamoDB, import AnOnlineShop_1.json, um ein neues Datenmodell namens AnOnlineShop und eine neue Tabelle namens zu erstellen. OnlineShop Beachten Sie, dass wir für den Partitionsschlüssel und den Sortierschlüssel die generischen Namen PK und SK verwenden. Diese Methode wird verwendet, um verschiedene Arten von Entitäten in derselben Tabelle zu speichern.

Schritt 1: Zugriffsmuster 1 (**`getCustomerByCustomerId`**) angehen

Importieren Sie AnOnlineShop_2.json, um das Zugriffsmuster 1 () zu verarbeiten. getCustomerByCustomerId Manche Entitäten haben keine Beziehungen zu anderen Entitäten, daher verwenden wir für sie den gleichen Wert von PK und SK. Beachten Sie in den Beispieldaten, dass die Schlüssel das Präfix c# verwenden, um die customerId von anderen Entitäten zu unterscheiden, die später hinzugefügt werden. Diese Vorgehensweise wird auch für andere Entitäten genutzt.

Um dieses Zugriffsmuster anzugehen, kann ein GetItem-Vorgang mit PK=customerId und SK=customerId verwendet werden.

Schritt 2: Zugriffsmuster 2 (**`getProductByProductId`**) angehen

Importieren Sie AnOnlineShop_3.json, um das Zugriffsmuster 2 (getProductByProductId) für die Entität zu adressieren. product Den Produktentitäten wird das Präfix p# vorangestellt und dasselbe Sortierschlüsselattribut wurde zum Speichern der customerID und der productID verwendet. Die generische Benennung und vertikale Partitionierung ermöglichen es uns, solche Elementauflistungen zu erstellen, um ein effektives Einzeltabellendesign zu erhalten.

Um dieses Zugriffsmuster anzugehen, kann ein GetItem-Vorgang mit PK=productId und SK=productId verwendet werden.

Schritt 3: Zugriffsmuster 3 (**`getWarehouseByWarehouseId`**) angehen

Importieren Sie AnOnlineShop_4.json, um das Zugriffsmuster 3 (`getWarehouseByWarehouseId`) für die Entität zu adressieren. `warehouse` Derzeit werden die Entitäten `customer`, `product` und `warehouse` zur selben Tabelle hinzugefügt. Sie unterscheiden sich durch Präfixe und das Attribut `EntityType`. Ein Typ-Attribut (oder eine Präfixbenennung) verbessert die Lesbarkeit des Modells. Die Lesbarkeit wäre beeinträchtigt, wenn wir einfach alphanumerische Zahlen IDs für verschiedene Entitäten im selben Attribut speichern würden. Ohne diese Identifikatoren wäre es schwierig, eine Entität von der anderen zu unterscheiden.

Um dieses Zugriffsmuster anzugehen, kann ein `GetItem`-Vorgang mit `PK=warehouseId` und `SK=warehouseId` verwendet werden.

Basistabelle:

Primary key		Attributes		
Partition key: PK	Sort key: SK			
c#12345	c#12345	EntityType	Email	Name
		customer	samaneh@example.com	Samaneh Utter
p#12345	p#12345	EntityType	Detail	Price
		product	{"Name":{"S":"Options Open"},"Description":{"S":"The latest album"}}	100
w#12345	w#12345	EntityType	Address	
		warehouse	{"Country":{"S":"Sweden"},"County":{"S":"Vastra Gotaland"},"City":{"S":"Goteborg"},"Street":{"S":"MainStreet"},"Number":{"S":"20"},"ZipCode":{"S":"41111"}}	

Schritt 4: Zugriffsmuster 4 (**`getProductInventoryByProductId`**) angehen

Importieren Sie AnOnlineShop_5.json, um das Zugriffsmuster 4 () zu adressieren. `getProductInventoryByProductId` `warehouseItem`Die Entität wird verwendet, um die Anzahl der Produkte in jedem Lager zu verfolgen. Dieses Element wird normalerweise aktualisiert, wenn ein Produkt einem Warenlager hinzugefügt oder entnommen wird. Wie aus der ERD hervorgeht, besteht ein many-to-many Zusammenhang zwischen `product` und`warehouse`. Hier wird die one-to-many Beziehung von `product` bis `warehouse` modelliert als`warehouseItem`. Später `product` wird auch die one-to-many Beziehung von `warehouse` bis modelliert.

Das Zugriffsmuster 4 kann mit der Abfrage von `PK=ProductId` und `SK begins_with “w#“` angegangen werden.

Weitere Informationen zu `begins_with()` und weitere Ausdrücke, die auf Sortierschlüssel angewendet werden können, finden Sie unter Schlüsselbedingungsausdrücke.

Basistabelle:

Primary key		Attributes		
Partition key: PK	**Sort key: SK**			
c#12345	c#12345	EntityType	Email	Name
		customer	samaneh@example.com	Samaneh Utter
	p#12345	EntityType	Detail	Price
		product	{"Name":{"S":"Options Open"},"Description":{"S":"The latest album"}}	100
p#12345	w#12345	EntityType	Quantity	
		warehouseItem	50	
w#12345	w#12345	EntityType	Address	
		warehouse	{"Country":{"S":"Sweden"},"County":{"S":"Vastra Gotaland"},"City":{"S":"Goteborg"},"Street":{"S":"MainStreet"},"Number":{"S":"20"},"ZipCode":{"S":"41111"}}	

Schritt 5: Zugriffsmuster 5 (**`getOrderDetailsByOrderId`**) und 6 (**`getProductByOrderId`**) angehen

Fügen Sie der Tabelle weitere `warehouse` Elemente `customerproduct`, und hinzu, indem Sie AnOnlineShop_6.json importieren. Importieren Sie dann AnOnlineShop_7.json, um eine Elementsammlung zu erstellen`order`, die die Zugriffsmuster 5 (`getOrderDetailsByOrderId`) und 6 () adressieren kann. `getProductByOrderId` Sie können die one-to-many Beziehung zwischen OrderItem-Entitäten `order` und deren `product` Modellierung als OrderItem-Entitäten sehen.

Um das Zugriffsmuster 5 (`getOrderDetailsByOrderId`) anzugehen, fragen Sie die Tabelle mit `PK=orderId` ab. Dadurch erhalten Sie alle Informationen zur Bestellung, einschließlich der `customerId`und der bestellten Produkte.

Basistabelle:

Primary key		Attributes		
Partition key: PK	Sort key: SK			
o#12345	c#12345	EntityType	Date	
		order	2020-06-21T19:10:00	
	p#12345	EntityType	Price	Quantity
		orderItem	100	2
	p#99887	EntityType	Price	Quantity
		orderItem	40	5

Um das Zugriffsmuster 6 (`getProductByOrderId`) anzugehen, müssen wir nur Produkte in einer `order` lesen. Fragen Sie dazu die Tabelle mit `PK=orderId` und `SK begins_with "p#"` ab.

Basistabelle:

Primary key		Attributes		
Partition key: PK	Sort key: SK			
	c#12345	EntityType	Date	
		order	2020-06-21T19:10:00	
o#12345	p#12345	EntityType	Price	Quantity
		orderItem	100	2
	p#99887	EntityType	Price	Quantity
		orderItem	40	5

Schritt 6: Zugriffsmuster 7 (**`getInvoiceByOrderId`**) angehen

Importieren Sie AnOnlineShop_8.json, um der Bestellartikelsammlung eine `invoice` Entität hinzuzufügen, die das Zugriffsmuster 7 () verarbeitet. `getInvoiceByOrderId` Um dieses Zugriffsmuster anzugehen, können Sie einen Abfragevorgang mit `PK=orderId` und `SK begins_with "i#"` verwenden.

Basistabelle:

Primary key		Attributes		
Partition key: PK	Sort key: SK			
o#12345	c#12345	EntityType	Date	
		order	2020-06-21T19:10:00	
	i#55443	EntityType	Amount	Date
		invoice	400	2020-06-21T19:18:00
	p#12345	EntityType	Price	Quantity
		orderItem	100	2
	p#99887	EntityType	Price	Quantity
		orderItem	40	5

Schritt 7: Zugriffsmuster 8 (**`getShipmentByOrderId`**) angehen

Importieren Sie AnOnlineShop_9.json, um `shipment` Entitäten zur Bestellartikelsammlung hinzuzufügen, um das Zugriffsmuster 8 zu adressieren (). `getShipmentByOrderId` Wir erweitern dasselbe vertikal partitionierte Modell, indem wir dem Einzeltabellendesign weitere Typen von Entitäten hinzufügen. Beachten Sie, dass die Elementauflistung order die verschiedenen Beziehungen enthält, die eine Entität des Typs `order` zu den Entitäten des Typs `shipment`, `orderItem` und `invoice` hat.

Um Lieferungen nach `orderId` abzurufen, können Sie einen Abfragevorgang mit `PK=orderId`und `SK begins_with “sh#”` durchführen.

Basistabelle:

Primary key		Attributes				
Partition key: PK	**Sort key: SK**					
o#12345	c#12345	EntityType	Date			
		order	2020-06-21T19:10:00			
	i#55443	EntityType	Amount	Date		
		invoice	400	2020-06-21T19:18:00		
	p#12345	EntityType	Price	Quantity		
		orderItem	100	2		
	p#99887	EntityType	Price	Quantity		
		orderItem	40	5		
	sh#88899	EntityType	Address	Type	Date	Warehouseld
		shipment	{"Country":{"S":"Sweden"},"County":{"S":"Vastra Gotaland"},"City":{"S":"Goteborg"},"Street":{"S":"Slanbarsvagen"},"Number":{"S":"34"},"ZipCode":{"S":"41787"}}	Express	2020-06-22T08:20:00	w#12376
	sh#98765	EntityType	Address	Type	Date	Warehouseld
		shipment	{"Country":{"S":"Sweden"},"County":{"S":"Vastra Gotaland"},"City":{"S":"Goteborg"},"Street":{"S":"Slanbarsvagen"},"Number":{"S":"34"},"ZipCode":{"S":"41787"}}	Express	2020-06-22T10:20:00	w#12345

Schritt 8: Zugriffsmuster 9 (**`getOrderByProductIdForDateRange`**) angehen

Wir haben im vorherigen Schritt die Elementauflistung order erstellt. Dieses Zugriffsmuster hat neue Suchdimensionen (`ProductID` und `Date`), weshalb Sie die gesamte Tabelle scannen und relevante Datensätze herausfiltern müssen, um die anvisierten Elemente abzurufen. Um dieses Zugriffsmuster anzugehen, müssen wir einen globalen sekundären Index (GSI) erstellen. Importieren Sie AnOnlineShop_10.json, um mithilfe der GSI eine neue Artikelsammlung zu erstellen, die es ermöglicht, *`orderItem`* Daten aus mehreren Bestellartikelsammlungen abzurufen. Die Daten verfügen jetzt über `GSI1-PK` und `GSI1-SK`, den zukünftigen Partitionsschlüssel und Sortierschlüssel von `GSI1`.

DynamoDB fügt automatisch Elemente, die die Schlüsselattribute eines GSI enthalten, aus der Tabelle in den GSI ein. Zusätzliche manuelle Einfügungen in den GSI sind nicht notwendig.

Um das Zugriffsmuster 9 anzugehen, führen Sie eine Abfrage an `GSI1` mit `GSI1-PK=productId` und `GSI1SK between (date1, date2)` durch.

Basistabelle:

Primary key		Attributes				
Partition key: PK	**Sort key: SK**					
o#12345	p#12345	EntityType	GSI1-PK	GSI1-SK	Price	Quantity
		orderItem	p#12345	2020-06-21T19:18:00	100	2
	p#99887	EntityType	GSI1-PK	GSI1-SK	Price	Quantity
		orderItem	p#99887	2020-06-21T19:20:00	40	5

GSI1:

Primary key		Attributes				
Partition key: GSI1-PK	**Sort key: GSI1-SK**					
p#12345	2020-06-21T19:18:00	PK	SK	EntityType	Quantity	Price
		o#12345	p#12345	orderItem	2	100
p#99887	2020-06-21T19:20:00	PK	SK	EntityType	Quantity	Price
		o#12345	p#99887	orderItem	5	40

Schritt 9: Zugriffsmuster 10 (**`getInvoiceByInvoiceId`**) und 11 (**`getPaymentByInvoiceId`**) angehen

Importieren Sie AnOnlineShop_11.json, um die Zugriffsmuster 10 (`getInvoiceByInvoiceId`) und 11 (`getPaymentByInvoiceId`) zu adressieren, die sich beide auf. `invoice` Obwohl es sich um zwei unterschiedliche Zugriffsmuster handelt, werden sie mit derselben Schlüsselbedingung realisiert. `Payments` sind als Attribut mit dem Datentyp „Karte“ auf der Entität `invoice` definiert.

> **Note**
>
> `GSI1-PK` und `GSI1-SK` sind überladen und speichern Informationen über verschiedene Entitäten, sodass mehrere Zugriffsmuster vom selben GSI aus abgedeckt werden können.

Weitere Informationen zur GSI-Überladung finden Sie unter Überladen globaler sekundärer Indizes in DynamoDB.

Um die Zugriffsmuster 10 und 11 anzugehen, fragen Sie `GSI1` mit `GSI1-PK=invoiceId` und `GSI1-SK=invoiceId` ab.

GSI1:

Primary key		Attributes							
Partition key: GSI1-PK	Sort key: GSI1-SK	PK	SK	EntityType	GSI2-PK	GSI2-SK	Detail	Amount	Date
i#55443	i#55443	o#12345	i#55443	invoice	c#12345	i#2020-06-21T19:18:00	{"Payments":{"L":[{"M":{"Type":{"S":"GiftCard"},"Amount":{"N":"100"},"Data":{"S":"GiftCard data here..."}}},{"M":{"Type":{"S":"MasterCard"},"Amount":{"N":"300"},"Data":{"S":"Payment data here..."}}}]}}	400	2020-06-21T19:18:00

Schritt 10: Zugriffsmuster 12 (**`getShipmentDetailsByShipmentId`**) und 13 (**`getShipmentByWarehouseId`**) angehen

Importieren Sie AnOnlineShop_12.json, um die Zugriffsmuster 12 (`getShipmentDetailsByShipmentId`) und 13 () zu adressieren. `getShipmentByWarehouseId`

Beachten Sie, dass `shipmentItem`-Entitäten zur Elementauflistung order in der Basistabelle hinzugefügt werden, damit alle Details zu einer Bestellung in einem einzigen Abfragevorgang abgerufen werden können.

Basistabelle:

Primary key		Attributes							
Partition key: PK	Sort key: SK								
o#12345	sh#88899	EntityType	GSI1-PK	GSI1-SK	GSI2-PK	GSI2-SK	Address	Type	Date
		shipment	sh#88899	sh#88899	w#12376	sh#88899	{"Country": {"S":"Sweden"},"County": {"S":"Vastra Gotaland"},"City": {"S":"Goteborg"},"Street": {"S":"Slanbarsvagen"},"Number": {"S":"34"},"ZipCode": {"S":"41787"}}	Express	2020-06-22T08:20:00
	sh#98765	EntityType	GSI1-PK	GSI1-SK	GSI2-PK	GSI2-SK	Address	Type	Date
		shipment	sh#98765	sh#98765	w#12345	sh#98765	{"Country": {"S":"Sweden"},"County": {"S":"Vastra Gotaland"},"City": {"S":"Goteborg"},"Street": {"S":"Slanbarsvagen"},"Number": {"S":"34"},"ZipCode": {"S":"41787"}}	Express	2020-06-22T10:20:00
	shp#12345	EntityType	GSI1-PK	GSI1-SK	Quantity				
		shipmentItem	sh#98765	p#99887	3				
	shp#54321	EntityType	GSI1-PK	GSI1-SK	Quantity				
		shipmentItem	sh#88899	p#99887	2				
	shp#55555	EntityType	GSI1-PK	GSI1-SK	Quantity				
		shipmentItem	sh#98765	p#12345	2				

Die `GSI1` Partitions- und Sortierschlüssel wurden bereits verwendet, um eine one-to-many Beziehung zwischen `shipment` und zu modellieren. `shipmentItem` Um das Zugriffsmuster 12 (`getShipmentDetailsByShipmentId`) anzugehen, fragen Sie `GSI1` mit `GSI1-PK=shipmentId` und `GSI1-SK=shipmentId` ab.

GSI1:

Primary key		Attributes							
Partition key: GSI1-PK	Sort key: GSI1-SK								
	p#99887	PK	SK	EntityType	Quantity				
		o#12345	shp#54321	shipmentItem	2				
sh#88899	sh#88899	PK	SK	EntityType	GSI2-PK	GSI2-SK	Address	Type	Date
		o#12345	sh#88899	shipment	w#12376	sh#88899	{"Country":{"S":"Sweden"},"County":{"S":"Vastra Gotaland"},"City":{"S":"Goteborg"},"Street":{"S":"Slanbarsvagen"},"Number":{"S":"34"},"ZipCode":{"S":"41787"}}	Express	2020-06-22T08:20:00
	p#12345	PK	SK	EntityType	Quantity				
		o#12345	shp#55555	shipmentItem	2				
	p#99887	PK	SK	EntityType	Quantity				
		o#12345	shp#12345	shipmentItem	3				
sh#98765	sh#98765	PK	SK	EntityType	GSI2-PK	GSI2-SK	Address	Type	Date
		o#12345	sh#98765	shipment	w#12345	sh#98765	{"Country":{"S":"Sweden"},"County":{"S":"Vastra Gotaland"},"City":{"S":"Goteborg"},"Street":{"S":"Slanbarsvagen"},"Number":{"S":"34"},"ZipCode":{"S":"41787"}}	Express	2020-06-22T10:20:00

Wir müssen eine weitere GSI (`GSI2`) erstellen, um die neue one-to-many Beziehung zwischen `warehouse` und `shipment` für das Zugriffsmuster 13 () zu modellieren.

`getShipmentByWarehouseId` Um dieses Zugriffsmuster anzugehen, fragen Sie `GSI2` mit `GSI2-PK=warehouseId` und `GSI2-SK begins_with “sh#”` ab.

GSI2:

Primary key		Attributes							
Partition key: GSI2-PK	Sort key: GSI2-SK								
		PK	SK	EntityType	GSI1-PK	GSI1-SK	Address	Type	Date
w#12376	sh#88899	o#12345	sh#88899	shipment	sh#88899	sh#88899	{"Country": {"S":"Sweden"},"County": {"S":"Vastra Gotaland"},"City": {"S":"Goteborg"},"Street": {"S":"Slanbarsvagen"},"Number": {"S":"34"},"ZipCode": {"S":"41787"}}	Express	2020-06-22T08:20:00
		PK	SK	EntityType	GSI1-PK	GSI1-SK	Address	Type	Date
w#12345	sh#98765	o#12345	sh#98765	shipment	sh#98765	sh#98765	{"Country": {"S":"Sweden"},"County": {"S":"Vastra Gotaland"},"City": {"S":"Goteborg"},"Street": {"S":"Slanbarsvagen"},"Number": {"S":"34"},"ZipCode": {"S":"41787"}}	Express	2020-06-22T10:20:00

Schritt 11: Zugriffsmuster 14 (**`getProductInventoryByWarehouseId`**), 15 (**`getInvoiceByCustomerIdForDateRange`**) und 16 (**`getProductsByCustomerIdForDateRange`**) angehen

Importieren Sie AnOnlineShop_13.json, um Daten hinzuzufügen, die sich auf die nächsten Zugriffsmuster beziehen. Um das Zugriffsmuster 14 (`getProductInventoryByWarehouseId`) anzugehen, fragen Sie `GSI2` mit `GSI2-PK=warehouseId` und `GSI2-SK begins_with “p#”` ab.

GSI2:

Primary key		Attributes							
Partition key: GSI2-PK	**Sort key: GSI2-SK**								
w#12345	p#12345	PK	SK	EntityType	Quantity				
		p#12345	w#12345	warehouseItem	50				
	p#99887	PK	SK	EntityType	Quantity				
		p#99887	w#12345	warehouseItem	4				
c#12345	i#2020-06-21T19:18:00	PK	SK	EntityType	GSI1-PK	GSI1-SK	Detail	Amount	Date
		o#12345	i#55443	invoice	i#55443	i#55443	{"Payments":{"L":[{"M":{"Type":{"S":"GiftCard"},"Amount":{"N":"100"},"Data":{"S":"GiftCard data here..."}}},{"M":{"Type":{"S":"MasterCard"},"Amount":{"N":"300"},"Data":{"S":"Payment data here..."}}}]}}	400	2020-06-21T19:18:00
	p#2020-06-21T19:18:00	PK	SK	EntityType	GSI1-PK	GSI1-SK	Price	Quantity	
		o#12345	p#12345	orderItem	p#12345	2020-06-21T19:18:00	100	2	
	p#2020-06-21T19:20:00	PK	SK	EntityType	GSI1-PK	GSI1-SK	Price	Quantity	
		o#12345	p#99887	orderItem	p#99887	2020-06-21T19:20:00	40	5	

Um das Zugriffsmuster 15 (`getInvoiceByCustomerIdForDateRange`) anzugehen, fragen Sie `GSI2` mit `GSI2-PK=customerId` und `GSI2-SK between (i#date1, i#date2)` ab.

GSI2:

Primary key		Attributes							
Partition key: GSI2-PK	**Sort key: GSI2-SK**								
w#12345	p#12345	PK	SK	EntityType	Quantity				
		p#12345	w#12345	warehouseItem	50				
	p#99887	PK	SK	EntityType	Quantity				
		p#99887	w#12345	warehouseItem	4				
c#12345	i#2020-06-21T19:18:00	PK	SK	EntityType	GSI1-PK	GSI1-SK	Detail	Amount	Date
		o#12345	i#55443	invoice	i#55443	i#55443	{"Payments":{"L":[{"M":{"Type":{"S":"GiftCard"},"Amount":{"N":"100"},"Data":{"S":"GiftCard data here..."}}},{"M":{"Type":{"S":"MasterCard"},"Amount":{"N":"300"},"Data":{"S":"Payment data here..."}}}]}}	400	2020-06-21T19:18:00
	p#2020-06-21T19:18:00	PK	SK	EntityType	GSI1-PK	GSI1-SK	Price	Quantity	
		o#12345	p#12345	orderItem	p#12345	2020-06-21T19:18:00	100	2	
	p#2020-06-21T19:20:00	PK	SK	EntityType	GSI1-PK	GSI1-SK	Price	Quantity	
		o#12345	p#99887	orderItem	p#99887	2020-06-21T19:20:00	40	5	

Um das Zugriffsmuster 16 (`getProductsByCustomerIdForDateRange`) anzugehen, fragen Sie `GSI2` mit `GSI2-PK=customerId` und `GSI2-SK between (p#date1, p#date2)` ab.

GSI2:

Primary key		Attributes							
Partition key: GSI2-PK	**Sort key: GSI2-SK**								
w#12345	p#12345	PK	SK	EntityType	Quantity				
		p#12345	w#12345	warehouseItem	50				
	p#99887	PK	SK	EntityType	Quantity				
		p#99887	w#12345	warehouseItem	4				
c#12345	i#2020-06-21T19:18:00	PK	SK	EntityType	GSI1-PK	GSI1-SK	Detail	Amount	Date
		o#12345	i#55443	invoice	i#55443	i#55443	{"Payments":{"L":[{"M":{"Type":{"S":"GiftCard"},"Amount":{"N":"100"},"Data":{"S":"GiftCard data here..."}}},{"M":{"Type":{"S":"MasterCard"},"Amount":{"N":"300"},"Data":{"S":"Payment data here..."}}}]}}	400	2020-06-21T19:18:00
	p#2020-06-21T19:18:00	PK	SK	EntityType	GSI1-PK	GSI1-SK	Price	Quantity	
		o#12345	p#12345	orderItem	p#12345	2020-06-21T19:18:00	100	2	
	p#2020-06-21T19:20:00	PK	SK	EntityType	GSI1-PK	GSI1-SK	Price	Quantity	
		o#12345	p#99887	orderItem	p#99887	2020-06-21T19:20:00	40	5	

Note

In NoSQL-Workbench stehen Facets für die verschiedenen Datenzugriffsmuster einer Anwendung für DynamoDB. Facets bieten Ihnen die Möglichkeit, eine Teilmenge der Daten in einer Tabelle anzuzeigen, ohne Datensätze sehen zu müssen, die den Einschränkungen des Facets nicht entsprechen. Facets gelten als visuelles Datenmodellierungswerkzeug und existieren nicht als brauchbares Konstrukt in DynamoDB, da sie eine reine Hilfe zur Modellierung von Zugriffsmustern darstellen.

Importieren Sie AnOnlineShop_facets.json, um die Facetten für diesen Anwendungsfall zu sehen.

Alle Zugriffsmuster und wie das Schemadesign sie behandelt, sind in der folgenden Tabelle zusammengefasst:

Zugriffsmuster	Basis-table/GSI/LSI	Operation	Partitionsschlüsselwert	Sortierschlüsselwert
getCustomerByCustomerId	Basistabelle	GetItem	PK=customerId	SK=customerId
getProductByProductId	Basistabelle	GetItem	PK=productId	SK=productId
getWarehouseByWarehouseId	Basistabelle	GetItem	PK=warehouseId	SK=warehouseId
getProductInventoryByProductId	Basistabelle	Query	PK=productId	SK begins_with "w#"
getOrderDetailsByOrderId	Basistabelle	Query	PK=orderId	
getProductByOrderId	Basistabelle	Query	PK=orderId	SK begins_with "p#"
getInvoiceByOrderId	Basistabelle	Query	PK=orderId	SK begins_with "i#"
getShipmentByOrderId	Basistabelle	Query	PK=orderId	SK begins_with "sh#"
getOrderByProductIdForDateRange	GSI1	Query	PK=productId	SK zwischen Datum1 und Datum2

Zugriffsmuster	Basis-table/GSI/LSI	Operation	Partitionsschlüsselwert	Sortierschlüsselwert
getInvoiceByInvoiceId	GSI1	Query	PK=invoiceId	SK=invoiceId
getPaymentByInvoiceId	GSI1	Query	PK=invoiceId	SK=invoiceId
getShipmentDetailsByShipmentId	GSI1	Query	PK=shipmentId	SK=shipmentId
getShipmentByWarehouseId	GSI2	Query	PK=warehouseId	SK begins_with "sh#"
getProductInventoryByWarehouseId	GSI2	Query	PK=warehouseId	SK begins_with "p#"
getInvoiceByCustomerIdForDateRange	GSI2	Query	PK=customerId	SKU zwischen i#date1 und i#date2
getProductsByCustomerIdForDateRange	GSI2	Query	PK=customerId	SK zwischen p#date1 und p#date2

Endgültiges Schema des Online-Shops

Dies sind die endgültigen Schemadesigns. Informationen zum Herunterladen dieses Schemadesign als JSON-Datei finden Sie unter DynamoDB-Entwurfsmuster auf. GitHub

Basistabelle

Primary key		Attributes			
Partition key: PK	**Sort key: SK**				
c#12345	c#12345	EntityType	Email	Name	
		customer	samaneh@example.com	Samaneh	
c#23456	c#23456	EntityType	Email	Name	
		customer	kathleen@example.com	Kathleen	
c#54321	c#54321	EntityType	Email	Name	
		customer	henrik@example.com	Henrik	
p#12345	p#12345	EntityType	Detail	Price	
		product	{"Name":{"S":"Options Open"},"Description":{"S":"The latest album"}}	100	
	w#12345	EntityType	GSI2-PK	GSI2-SK	Quantity
		warehouseItem	w#12345	p#12345	50
p#99887	p#99887	EntityType	Detail	Price	
		product	{"Name":{"S":"The Book"},"Description":{"S":"The best book ever"}}	40	
	w#12345	EntityType	GSI2-PK	GSI2-SK	Quantity
		warehouseItem	w#12345	p#99887	4
	w#12376	EntityType	Quantity		
		warehouseItem	4		
w#12345	w#12345	EntityType	Address		
		warehouse	{"Country":{"S":"Sweden"},"County":{"S":"Vastra Gotaland"},"City":{"S":"Goteborg"},"Street":{"S":"MainStreet"},"Number":{"S":"20"},"ZipCode":{"S":"41111"}}		
w#12376	w#12376	EntityType	Address		
		warehouse	{"Country":{"S":"Sweden"},"County":		

GSI1

Primary key		Attributes							
Partition key: GSI1-PK	**Sort key: GSI1-SK**								
p#12345	2020-06-21T19:18:00	PK	SK	EntityType	GSI2-PK	GSI2-SK	Price	Quantity	
		o#12345	p#12345	orderItem	c#12345	2020-06-21T19:18:00	100	2	
p#99887	2020-06-21T19:20:00	PK	SK	EntityType	GSI2-PK	GSI2-SK	Price	Quantity	
		o#12345	p#99887	orderItem	c#12345	2020-06-21T19:20:00	40	5	
i#55443	i#55443	PK	SK	EntityType	GSI2-PK	GSI2-SK	Detail	Amount	Date
		o#12345	i#55443	invoice	c#12345	2020-06-21T19:18:00	{"Payments":{"L":[{"M":{"Type":{"S":"GiftCard"},"Amount":{"N":"100"},"Data":{"S":"GiftCard data here..."}}},{"M":{"Type":{"S":"MasterCard"},"Amount":{"N":"300"},"Data":{"S":"Payment data here..."}}}]}}	400	2020-06-21T19:18:00
sh#88899	p#99887	PK	SK	EntityType	Quantity				
		o#12345	shp#54321	shipmentItem	2				
	sh#88899	PK	SK	EntityType	GSI2-PK	GSI2-SK	Address	Type	Date
		o#12345	sh#88899	shipment	w#12376	sh#88899	{"Country":{"S":"Sweden"},"County":{"S":"Vastra Gotaland"},"City":{"S":"Goteborg"},"Street":{"S":"Slanbarsvagen"},"Number":{"S":"34"},"ZipCode":{"S":"41787"}}	Express	2020-06-22T08:20:00
sh#98765	p#12345	PK	SK	EntityType	Quantity				
		o#12345	shp#55555	shipmentItem	2				
	p#99887	PK	SK	EntityType	Quantity				
		o#12345	shp#12345	shipmentItem	3				
	sh#98765	PK	SK	EntityType	GSI2-PK	GSI2-SK	Address	Type	Date
		o#12345	sh#98765	shipment	w#12345	sh#98765	{"Country":{"S":"Sweden"},"County":{"S":"Vastra Gotaland"},"City":{"S":"Goteborg"},"Street":{"S":"Slanbarsvagen"},"Number":{"S":"34"},"ZipCode":{"S":"41787"}}	Express	2020-06-22T10:20:00

GSI2

Primary key		Attributes							
Partition key: GSI2-PK	**Sort key: GSI2-SK**								
w#12345	p#12345	PK	SK	EntityType	Quantity				
		p#12345	w#12345	warehouseItem	50				
	p#99887	PK	SK	EntityType	Quantity				
		p#99887	w#12345	warehouseItem	4				
	sh#98765	PK	SK	EntityType	GSI1-PK	GSI1-SK	Address	Type	Date
		o#12345	sh#98765	shipment	sh#98765	sh#98765	{"Country":{"S":"Sweden"},"County":{"S":"Vastra Gotaland"},"City":{"S":"Goteborg"},"Street":{"S":"Slanbarsvagen"},"Number":{"S":"34"},"ZipCode":{"S":"41787"}}	Express	2020-06-22T10:20:00
c#12345	2020-06-21T19:18:00	PK	SK	EntityType	GSI1-PK	GSI1-SK	Price	Quantity	
		o#12345	p#12345	orderItem	p#12345	2020-06-21T19:18:00	100	2	
	2020-06-21T19:18:00	PK	SK	EntityType	GSI1-PK	GSI1-SK	Detail	Amount	Date
		o#12345	i#55443	invoice	i#55443	i#55443	{"Payments":{"L":[{"M":{"Type":{"S":"GiftCard"},"Amount":{"N":"100"},"Data":{"S":"GiftCard data here..."}}},{"M":{"Type":{"S":"MasterCard"},"Amount":{"N":"300"},"Data":{"S":"Payment data here..."}}}]}}	400	2020-06-21T19:18:00
	2020-06-21T19:20:00	PK	SK	EntityType	GSI1-PK	GSI1-SK	Price	Quantity	
		o#12345	p#99887	orderItem	p#99887	2020-06-21T19:20:00	40	5	
w#12376	sh#88899	PK	SK	EntityType	GSI1-PK	GSI1-SK	Address	Type	Date
		o#12345	sh#88899	shipment	sh#88899	sh#88899	{"Country":{"S":"Sweden"},"County":{"S":"Vastra Gotaland"},"City":{"S":"Goteborg"},"Street":{"S":"Slanbarsvagen"},"Number":{"S":"34"},"ZipCode":{"S":"41787"}}	Express	2020-06-22T08:20:00

Verwendung von NoSQL Workbench mit diesem Schemadesign

Sie können dieses endgültige Schema in NoSQL Workbench importieren, um Ihr neues Projekt weiter zu untersuchen und zu bearbeiten. NoSQL Workbench ist ein visuelles Tool, das Features zur Datenmodellierung, Datenvisualisierung und Abfrageentwicklung für DynamoDB bereitstellt. Gehen Sie folgendermaßen vor, um zu beginnen:

1. Laden Sie NoSQL Workbench herunter. Weitere Informationen finden Sie unter the section called “Download”.
2. Laden Sie die oben aufgeführte JSON-Schemadatei herunter, die bereits das NoSQL-Workbench-Modellformat aufweist.
3. Importieren Sie die JSON-Schemadatei in NoSQL Workbench. Weitere Informationen finden Sie unter the section called “Importieren eines vorhandenen Modells”.
4. Nach dem Import in NOSQL Workbench können Sie das Datenmodell bearbeiten. Weitere Informationen finden Sie unter the section called “Bearbeiten eines vorhandenen Modells”.

Bewährte Methoden für die Modellierung relationaler Daten in DynamoDB

Dieser Abschnitt erläutert bewährte Methoden für die Modellierung relationaler Daten in Amazon DynamoDB. Zunächst stellen wir traditionelle Konzepte der Datenmodellierung vor. Anschließend beschreiben wir die Vorteile der Verwendung von DynamoDB gegenüber herkömmlichen relationalen Datenbankmanagementsystemen – wie JOIN-Operationen überflüssig werden und der Overhead reduziert wird.

Anschließend erläutern wir, wie Sie eine DynamoDB-Tabelle entwerfen, die effizient skaliert werden kann. Schließlich geben wir ein Beispiel für die Modellierung relationaler Daten in DynamoDB.

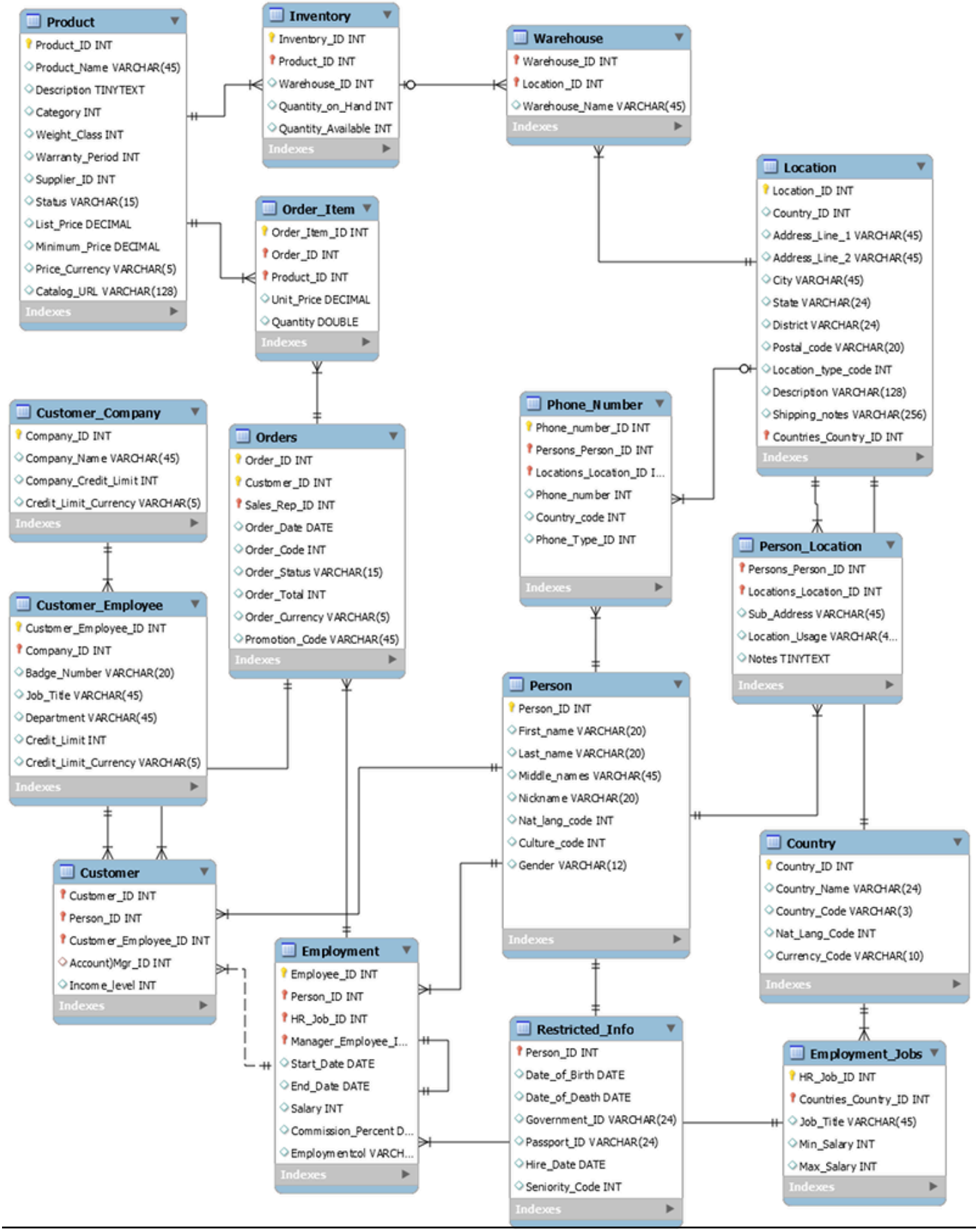
Themen

- Traditionelle relationale Datenbankmodelle
- So macht DynamoDB JOIN-Operationen überflüssig
- So verringern DynamoDB-Transaktionen den Aufwand für den Schreibprozess
- Erste Schritte für die Modellierung relationaler Daten in DynamoDB
- Beispiel für die Modellierung relationaler Daten in DynamoDB

Traditionelle relationale Datenbankmodelle

Herkömmliche relationale Datenbankmanagementsysteme (RDBMS) speichern Daten in einer normalisierten relationalen Struktur. Das Ziel des relationalen Datenmodells besteht darin, die Duplizierung von Daten (durch Normalisierung) zu reduzieren, um die referentielle Integrität zu unterstützen und Datenanomalien zu verringern.

Das folgende Schema ist ein Beispiel für ein relationales Datenmodell für eine generische Auftragserfassungsanwendung. Die Anwendung unterstützt ein Personalschema, das die betrieblichen und geschäftlichen Unterstützungssysteme eines imaginären Fertigungsunternehmens unterstützt.


Product
Product_ID INT
Product_Name VARCHAR(45)
Description TINYTEXT
Category INT
Weight_Class INT
Warranty_Period INT
Supplier_ID INT
Status VARCHAR(15)
List_Price DECIMAL
Minimum_Price DECIMAL
Price_Currency VARCHAR(5)
Catalog_URL VARCHAR(128)
Indexes
Inventory
Inventory_ID INT
Product_ID INT
Warehouse_ID INT
Quantity_on_Hand INT
Quantity_Available INT
Indexes
Warehouse
Warehouse_ID INT
Location_ID INT
Warehouse_Name VARCHAR(45)
Indexes
Location
Location_ID INT
Country_ID INT
Address_Line_1 VARCHAR(45)
Address_Line_2 VARCHAR(45)
City VARCHAR(45)
State VARCHAR(24)
District VARCHAR(24)
Postal_code VARCHAR(20)
Location_type_code INT
Description VARCHAR(128)
Shipping_notes VARCHAR(256)
Countries_Country_ID INT
Indexes
Order_Item
Order_Item_ID INT
Order_ID INT
Product_ID INT
Unit_Price DECIMAL
Quantity DOUBLE
Indexes
Customer_Company
Company_ID INT
Company_Name VARCHAR(45)
Company_Credit_Limit INT
Credit_Limit_Currency VARCHAR(5)
Indexes
Orders
Order_ID INT
Customer_ID INT
Sales_Rep_ID INT
Order_Date DATE
Order_Code INT
Order_Status VARCHAR(15)
Order_Total INT
Order_Currency VARCHAR(5)
Promotion_Code VARCHAR(45)
Indexes
Phone_Number
Phone_number_ID INT
Persons_Person_ID INT
Locations_Location_ID I...
Phone_number INT
Country_code INT
Phone_Type_ID INT
Indexes
Person_Location
Persons_Person_ID INT
Locations_Location_ID INT
Sub_Address VARCHAR(45)
Location_Usage VARCHAR(4...
Notes TINYTEXT
Indexes
Customer_Employee
Customer_Employee_ID INT
Company_ID INT
Badge_Number VARCHAR(20)
Job_Title VARCHAR(45)
Department VARCHAR(45)
Credit_Limit INT
Credit_Limit_Currency VARCHAR(5)
Indexes
Person
Person_ID INT
First_name VARCHAR(20)
Last_name VARCHAR(20)
Middle_names VARCHAR(45)
Nickname VARCHAR(20)
Nat_lang_code INT
Culture_code INT
Gender VARCHAR(12)
Indexes
Country
Country_ID INT
Country_Name VARCHAR(24)
Country_Code VARCHAR(3)
Nat_Lang_Code INT
Currency_Code VARCHAR(10)
Indexes
Customer
Customer_ID INT
Person_ID INT
Customer_Employee_ID INT
Account)Mgr_ID INT
Income_level INT
Indexes
Employment
Employee_ID INT
Person_ID INT
HR_Job_ID INT
Manager_Employee_I...
Start_Date DATE
End_Date DATE
Salary INT
Commission_Percent D...
Employmentcol VARCH...
Indexes
Restricted_Info
Person_ID INT
Date_of_Birth DATE
Date_of_Death DATE
Government_ID VARCHAR(24)
Passport_ID VARCHAR(24)
Hire_Date DATE
Seniority_Code INT
Indexes
Employment_Jobs
HR_Job_ID INT
Countries_Country_ID INT
Job_Title VARCHAR(45)
Min_Salary INT
Max_Salary INT
Indexes

Als nicht-relationaler Datenbankservice bietet DynamoDB viele Vorteile gegenüber herkömmlichen relationalen Datenbankmanagementsystemen.

So macht DynamoDB JOIN-Operationen überflüssig

Ein RDBMS verwendet eine strukturierte Abfragesprache (SQL), um Daten an die Anwendung zurückzugeben. Aufgrund der Normalisierung des Datenmodells erfordern solche Abfragen in der Regel die Verwendung des `JOIN`-Operators, um Daten aus einer oder mehreren Tabellen zu kombinieren.

Um beispielsweise eine Liste von Bestellpositionen zu generieren, die nach der Menge der Artikel in allen Lagern sortiert ist, die alle Bestellpositionen liefern können, könnten Sie die folgende SQL-Abfrage für das vorangehende Schema ausgeben.

```
SELECT * FROM Orders
  INNER JOIN Order_Items ON Orders.Order_ID = Order_Items.Order_ID
  INNER JOIN Products ON Products.Product_ID = Order_Items.Product_ID
  INNER JOIN Inventories ON Products.Product_ID = Inventories.Product_ID
  ORDER BY Quantity_on_Hand DESC
```

SQL-Abfragen dieser Art können eine flexible API für den Zugriff auf Daten bereitstellen. Sie erfordern jedoch einen erheblichen Verarbeitungsaufwand. Jeder Join in der Abfrage erhöht die Laufzeitkomplexität der Abfrage, da die Daten für jede Tabelle zwischengespeichert und dann zusammengestellt werden müssen, um die Ergebnismenge zurückzugeben.

Weitere Faktoren, die sich auf die Dauer der Ausführung der Abfragen auswirken können, sind die Größe der Tabellen und die Frage, ob die zu verknüpfenden Spalten Indizes haben. Die vorangehende Abfrage initiiert komplexe Abfragen für mehrere Tabellen und sortiert anschließend die Ergebnismenge.

Bei der NoSQL-Datenmodellierung geht es im Wesentlichen darum, keine `JOIN`s mehr zu benötigen. Aus diesem Grund haben wir DynamoDB zur Unterstützung von Amazon.com entwickelt und aus diesem Grund bietet DynamoDB in jeder Größenordnung konsistente Leistung. Angesichts der Laufzeitkomplexität von SQL-Abfragen und `JOIN`s der RDBMS-Leistung ist die Leistung im großen Maßstab nicht konstant. Dies führt zu Leistungsproblemen, wenn Kundenanwendungen wachsen.

Durch die Normalisierung von Daten verringert sich zwar die Menge der auf der Festplatte gespeicherten Daten, die am stärksten eingeschränkten Ressourcen mit Auswirkungen auf die Leistung sind jedoch die CPU-Zeit und die Netzwerklatenz.

DynamoDB wurde entwickelt, um beide Einschränkungen zu minimieren, indem `JOINs` wegfallen (und die Denormalisierung von Daten gefördert wird) und die Datenbankarchitektur optimiert wird, um eine Anwendungsabfrage mit einer einzigen Anforderung an ein Element vollständig zu beantworten. Diese Eigenschaften ermöglichen es DynamoDB, in jeder Größenordnung eine Leistung im einstelligen Millisekundenbereich zu bieten. Dies liegt daran, dass die Laufzeitkomplexität für DynamoDB-Operationen unabhängig von der Datengröße bei häufigen Zugriffsmustern konstant ist.

So verringern DynamoDB-Transaktionen den Aufwand für den Schreibprozess

Auch die Verwendung von Transaktionen zum Schreiben in ein normalisiertes Schema kann ein RDBMS verlangsamen. Wie in dem Beispiel dargestellt, müssen die relationalen Datenstrukturen, die von der Mehrzahl der OLTP-Anwendungen (Online Transaction Processing, Online-Transaktionsverarbeitung) verwendet werden, unterteilt und auf mehrere logische Tabellen verteilt werden, wenn sie in einem RDBMS gespeichert werden.

Daher ist ein ACID-kompatibles Framework notwendig, um Race Conditions und Datenintegritätsprobleme zu vermeiden, die auftreten könnten, wenn eine Anwendung versucht, ein Objekt zu lesen, das gerade geschrieben wird. Ein solches Transaktions-Framework kann in Verbindung mit einem relationalen Schema den Schreibprozess deutlich aufwendiger machen.

Die Implementierung von Transaktionen in DynamoDB verhindert Skalierungsprobleme, die bei einem RDBMS typischerweise auftreten. DynamoDB tut dies, indem eine Transaktion als einzelner API-Aufruf ausgegeben und die Anzahl der Elemente begrenzt wird, auf die in dieser einzelnen Transaktion zugegriffen werden kann. Lang andauernde Transaktionen können zu Betriebsproblemen führen, da die Daten für lange Zeit oder dauerhaft gesperrt werden, weil die Transaktion nie abgeschlossen wird.

Um solche Probleme in DynamoDB zu vermeiden, wurden Transaktionen mit zwei verschiedenen API-Operationen implementiert: `TransactWriteItems` und `TransactGetItems`. Diese API-Operationen haben keine Anfangs- und Endsemantik, wie in einem RDBMS üblich. Darüber hinaus gilt in DynamoDB ein Zugriffslimit von 100 Elementen innerhalb einer Transaktion, ebenfalls um lang andauernde Transaktionen zu verhindern. Weitere Informationen zu DynamoDB-Transaktionen finden Sie unter Arbeiten mit Transaktionen.

Aus diesen Gründen ist die Nutzung eines NoSQL-Systems in der Regel technisch und wirtschaftlich sinnvoll, wenn Ihr Unternehmen eine Reaktion mit geringer Latenz auf Abfragen mit hohem

Datenverkehr benötigt. Amazon DynamoDB hilft bei der Lösung von Problemen, die die Skalierbarkeit des relationalen Systems einschränken, indem sie diese vermeiden.

Die Leistung eines RDBMS ist in der Regel aus den folgenden Gründen nicht gut skalierbar:

- Es verwendet kostspielige Joins, um die gewünschten Ansichten von Abfrageergebnissen neu zusammenzustellen.
- Es standardisiert Daten und speichert sie in mehreren Tabellen, die mehrere Abfragen erfordern, bei denen Daten auf den Datenträger geschrieben werden.
- In der Regel fällt der Leistungsaufwand für ein ACID-kompatibles Transaktionssystem an.

DynamoDB kann aus den folgenden Gründen gut skaliert werden:

- Die Flexibilität des Schemas ermöglicht DynamoDB die Speicherung komplexer hierarchischer Daten innerhalb eines einzelnen Elements.
- Das Design mit zusammengesetzten Schlüsseln ermöglicht das Speichern verwandter Elemente nahe beieinander in derselben Tabelle.
- Transaktionen werden in einem einzigen Vorgang ausgeführt. Die Anzahl der Elemente, auf die zugegriffen werden kann, ist auf 100 begrenzt, um lange laufende Operationen zu vermeiden.

Abfragen des Datenspeichers werden sehr viel einfacher und können häufig im folgenden Format erfolgen:

```
SELECT * FROM Table_X WHERE Attribute_Y = "somevalue"
```

DynamoDB erfordert im Vergleich zum RDBMS aus dem vorhergehenden Beispiel deutlich weniger Aufwand, um die angeforderten Daten zu erhalten.

Erste Schritte für die Modellierung relationaler Daten in DynamoDB

> **Note**
>
> Das NoSQL-Design erfordert einen anderen Ansatz als das RDBMS-Design. Sie können ein standardisiertes Datenmodell für ein RDBMS entwickeln, ohne sich Gedanken über Zugriffsmuster machen zu müssen. Anschließend können Sie es erweitern, wenn neue Fragen und Abfrageanforderungen entstehen. Im Fall von Amazon DynamoDB sollten Sie jedoch nicht mit der Entwicklung Ihres Schemas beginnen, bis Sie die Fragen kennen, die

es beantworten können muss. Es ist daher äußerst wichtig, die Business-Probleme und Anwendungsfälle vor der Entwicklung des Schemas zu kennen.

Um mit dem Entwerfen einer DynamoDB-Tabelle zu beginnen, die effizient skaliert werden kann, müssen Sie zunächst mehrere Schritte ausführen, um die Zugriffsmuster zu identifizieren, die von den Betriebs- und Geschäftsunterstützungssystemen (OSS/BSS) benötigt werden, die unterstützt werden müssen:

- Im Fall neuer Anwendungen sollten Sie Berichte von Benutzern zu Aktivitäten und Zielen prüfen. Dokumentieren Sie die von Ihnen identifizierten Anwendungsfälle und analysieren Sie die von diesen benötigten Zugriffsmuster.
- Analysieren Sie im Fall vorhandener Anwendungen die Abfrageprotokolle, um festzustellen, wie das System zurzeit verwendet wird und was die wichtigsten Zugriffsmuster sind.

Nach dem Abschluss dieses Vorgangs sollten Sie über eine Liste verfügen, die ungefähr wie die folgende aussieht.

Zugriffsmuster für die Anwendung zur Auftragserfassung

Muster #	Beschreibung des Zugriffsmusters
1	Suchen von Mitarbeiterdetails nach Mitarbeiter-ID
2	Mitarbeiterdetails nach Mitarbeiternamen abfragen
3	Finden Sie die Telefonnummer (n) eines Mitarbeiters
4	Finden Sie die Telefonnummer (n) eines Kunden
5	Bestellungen für Kunden innerhalb des Datumsbereichs abrufen
6	Alle offenen Bestellungen innerhalb des Datumsbereichs anzeigen
7	Alle kürzlich eingestellten Mitarbeiter anzeigen
8	Finden Sie alle Mitarbeiter im Warehouse
9	Holen Sie sich alle Artikel auf Bestellung für das Produkt

Muster #	Beschreibung des Zugriffsmusters
10	Besorgen Sie sich Inventare für das Produkt in allen Lagern
11	Holen Sie sich Kunden nach Kundenbetreuern
12	Bestellungen nach Kundenbetreuern abrufen
13	Holen Sie sich Mitarbeiter mit Berufsbezeichnung
14	Holen Sie sich den Lagerbestand nach Produkt und Lager
15	Holen Sie sich den gesamten Produktbestand

In einer echten Anwendung könnte Ihre Liste sehr viel länger sein. Diese Liste ist jedoch beispielhaft für die Komplexität der Abfragemuster, denen Sie in einer Produktionsumgebung begegnen können.

Ein moderner Ansatz für das DynamoDB-Schemadesign verwendet aggregationsorientierte Prinzipien, bei denen Daten auf der Grundlage von Zugriffsmustern und nicht auf starren Entitätsgrenzen gruppiert werden. Bei diesem Ansatz werden mehrere Entwurfsmuster berücksichtigt:

- Einzeltabellendesign — Verwendung zusammengesetzter Sortierschlüssel, überladener globaler Sekundärindizes und Adjazenzlistenmuster zum Speichern mehrerer Entitätstypen in einer Tabelle
- Design mit mehreren Tabellen — Verwendung separater Tabellen für Entitäten mit unabhängigen Betriebsmerkmalen und geringer Zugriffskorrelation, strategisch wichtig für entitätsübergreifende Abfragen GSIs
- Aggregiertes Design — Einbetten verwandter Daten, wenn sie immer gemeinsam abgerufen werden (Bestellung + OrderItems), oder Verwendung von Artikelsammlungen zur Identifizierung von Beziehungen (Produkt + Inventar)

Die Wahl zwischen diesen Ansätzen hängt von Ihren spezifischen Zugriffsmustern, Datenmerkmalen und betrieblichen Anforderungen ab. Sie können diese Elemente verwenden, um die Daten so zu strukturieren, dass eine Anwendung mittels einer einzigen Abfrage für eine Tabelle oder einen Index alle für ein bestimmtes Zugriffsmuster benötigten Informationen abrufen kann.

> **Note**
>
> Die Wahl zwischen Einzeltabellendesign und Mehrtabellendesign hängt von Ihren spezifischen Anforderungen ab. Das Design mit einer Tabelle eignet sich gut, wenn Entitäten eine hohe Zugriffskorrelation und ähnliche Betriebsmerkmale aufweisen. Das Design mit mehreren Tabellen wird bevorzugt, wenn Entitäten unabhängige betriebliche Anforderungen haben, unterschiedliche Zugriffsmuster haben oder wenn Sie klare betriebliche Grenzen benötigen. Das Beispiel in diesem Leitfaden veranschaulicht einen Ansatz mit mehreren Tabellen mit strategischer Aggregation und Denormalisierung.

Informationen zur Verwendung von NoSQL Workbench für DynamoDB zur Visualisierung Ihres Partitionsschlüsseldesigns finden Sie unter Erstellen von Datenmodellen mit NoSQL Workbench.

Beispiel für die Modellierung relationaler Daten in DynamoDB

In diesem Beispiel wird die Modellierung relationaler Daten in Amazon DynamoDB beschrieben. Das DynamoDB-Tabellendesign entspricht dem Schema für die Eingabe relationaler Bestellungen, das unter gezeigt wird. Relationale Modellierung Bei diesem Design werden mehrere spezialisierte Tabellen anstelle einer einzigen Adjazenzliste verwendet, wodurch klare betriebliche Grenzen geschaffen werden und gleichzeitig strategisch darauf geachtet wird, dass alle Zugriffsmuster effizient bedient GSIs werden.

Der Entwurfsansatz verwendet aggregationsorientierte Prinzipien, bei denen Daten auf der Grundlage von Zugriffsmustern und nicht anhand starrer Entitätsgrenzen gruppiert werden. Zu den wichtigsten Entwurfsentscheidungen gehören die Verwendung separater Tabellen für Entitäten mit geringer Zugriffskorrelation, das Einbetten verwandter Daten, wenn sie immer gemeinsam abgerufen werden, und die Verwendung von Elementsammlungen zur Identifizierung von Beziehungen.

Die folgenden Tabellen und die zugehörigen Indizes unterstützen das Schema der Eingabe in relationaler Reihenfolge:

Tabellendesign für Mitarbeiter

In der Tabelle „Employee“ werden Mitarbeiterinformationen in einer einzigen Einheit pro Element gespeichert. Sie ist für die direkte Suche nach Mitarbeitern optimiert und unterstützt mehrere Abfragemuster im strategischen GSIs Bereich. Diese Tabelle veranschaulicht das Prinzip, separate Tabellen für Entitäten mit unabhängigen betrieblichen Merkmalen und geringer objektübergreifender Zugriffskorrelation zu entwerfen.

Die Tabelle verwendet einen einfachen Partitionsschlüssel (employee_id) ohne Sortierschlüssel, da jeder Mitarbeiter eine eigene Entität ist. Vier GSIs ermöglichen effizientes Abfragen anhand verschiedener Attribute:

- EmployeeByName GSI — Verwendet die INCLUDE-Projektion mit allen Mitarbeiterattributen, um das vollständige Abrufen von Mitarbeiterdetails anhand des Namens zu unterstützen, wobei potenzielle doppelte Namen mit employee_id als Sortierschlüssel behandelt werden
- EmployeeByWarehouse GSI — Verwendet die INCLUDE-Projektion nur mit wichtigen Attributen (name, job_title, hire_date), um die Lagerkosten zu minimieren und gleichzeitig lagerbasierte Abfragen zu unterstützen
- EmployeeByJobTitle GSI — Ermöglicht rollenbasierte Abfragen mit INCLUDE-Projektion für Berichte und Organisationsanalysen
- EmployeeByHireDate GSI — Verwendet den statischen Partitionsschlüsselwert „EMPLOYEE“ mit hire_date als Sortierschlüssel, um effiziente Abfragen nach Datumsbereichen für kürzlich eingestellte Mitarbeiter zu ermöglichen. Da Mitarbeiter in der additions/updates Regel weniger als 1.000 WCU haben, kann eine einzige Partition die Schreiblast bewältigen, ohne dass es zu Problemen mit der Partition kommt

Tabelle „Mitarbeiter“ — Struktur der Basistabelle

employee_id (PK)	Name	telefonnummern	Lager-ID	job_title	Einstellungsdatum	Entitätstyp
emp_001	John Smith	["+1-555-0101"]	wh_sea	Manager	15.03.2024	MITARBEITER
emp_002	Jane Doe	["+1-555-0102", „+1-555-0103"]	wh_sea	Assoziieren	2025-01-10	MITARBEITER
emp_003	Bob Wilson	["+1-555-0104"]	wh_pdx	Assoziieren	2025-06-20	MITARBEITER
emp_004	Alice Braun	["+1-555-0105"]	wh_pdx	Supervisor	05.11.2023	MITARBEITER

employee_id (PK)	Name	telefonnummern	Lager-ID	job_title	Einstellungsdatum	Entitätstyp
emp_005	Charlie Davis	["+1-555-0106"]	wh_sea	Assoziieren	2025-12-01	MITARBEITER

EmployeeByName GSI — Unterstützung von Abfragen nach Mitarbeiternamen

Name (GSI-PK)	Mitarbeiter-ID (GSI-SK)	telefonnummern	Lager-ID	job_title	Einstellungsdatum
Alice Braun	emp_004	["+1-555-0105"]	wh_pdx	Supervisor	05.11.2023
Bob Wilson	emp_003	["+1-555-0104"]	wh_pdx	Assoziieren	2025-06-20
Charlie Davis	emp_005	["+1-555-0106"]	wh_sea	Assoziieren	2025-12-01
Jane Doe	emp_002	["+1-555-0102", „+1-555-0103"]	wh_sea	Assoziieren	2025-01-10
John Smith	emp_001	["+1-555-0101"]	wh_sea	Manager	15.03.2024

EmployeeByWarehouse GSI — Unterstützung von Lagerabfragen

Lagernummer (GSI-PK)	Mitarbeiter-ID (GSI-SK)	Name	job_title	Einstellungsdatum
wh_pdx	emp_003	Bob Wilson	Assoziieren	2025-06-20
wh_pdx	emp_004	Alice Braun	Supervisor	05.11.2023-23
wh_sea	emp_001	John Smith	Manager	15.03.2024

Lagernummer (GSI-PK)	Mitarbeiter-ID (GSI-SK)	Name	job_title	Einstellungsdatum
wh_sea	emp_002	Jane Doe	Assoziieren	2025-01-10
wh_sea	emp_005	Charlie Davis	Assoziieren	2025-12-01

EmployeeByJobTitle GSI — Unterstützung bei Fragen nach Berufsbezeichnungen

Berufsbezeichnung (GSI-PK)	Mitarbeiter-ID (GSI-SK)	Name	Lager-ID	Einstellungsdatum
Assoziieren	emp_002	Jane Doe	wh_sea	2025-01-10
Assoziieren	emp_003	Bob Wilson	wh_pdx	20.06.2025
Assoziieren	emp_005	Charlie Davis	wh_sea	2025-12-01
Manager	emp_001	John Smith	wh_sea	15.03.2024
Supervisor	emp_004	Alice Braun	wh_pdx	05.11.2023

EmployeeByHireDate GSI — Unterstützung bei aktuellen Einstellungsanfragen

entity_type (GSI-PK)	Einstellungsdatum (GSI-SK)	Mitarbeiter_ID	Name	Lager-ID
MITARBEITER	2023-11-05	emp_004	Alice Braun	wh_pdx
MITARBEITER	2024-03-15	emp_001	John Smith	wh_sea
MITARBEITER	2025-01-10	emp_002	Jane Doe	wh_sea
MITARBEITER	2025-06-20	emp_003	Bob Wilson	wh_pdx
MITARBEITER	2025-12-01	emp_005	Charlie Davis	wh_sea

Tischdesign für Kunden

In der Kundentabelle werden Kundeninformationen mit strategischer Denormalisierung von account_rep_id gespeichert, um effiziente Abfragen von Kundenbetreuern zu ermöglichen. Bei dieser Entwurfswahl wird geringer Speicheraufwand gegen die Abfrageleistung abgewogen, sodass keine Verknüpfungen zwischen Kunden- und Kundenbetreuerdaten erforderlich sind.

Die Tabelle unterstützt mehrere Telefonnummern pro Kunde mithilfe eines Listenattributs, was die Schemaflexibilität von DynamoDB demonstriert. Die einheitliche GSI ermöglicht Workflows für Kundenbetreuer:

- CustomerByAccountRep GSI — Nutzt die INCLUDE-Projektion mit Namens- und E-Mail-Attributen, um die Kundenverwaltung von Kundenbetreuern zu unterstützen, ohne dass vollständige Kundendatensätze abgerufen werden müssen

Kundentabelle — Struktur der Basistabelle

customer_id (PK)	Name	telefonnummern	E-Mail	Kontorep_ID
cust_001	Acme Corporation	["+1-555-1001"]	contact@acme.com	rep_001
cust_002	TechStart Inc	["+1-555-1002", „+1-555-1003"]	info@techstart.com	rep_001
cust_003	Globale Händler	["+1-555-1004"]	sales@globaltraders.com	rep_002
cust_004	BuildRight LLC	["+1-555-1005"]	orders@buildright.com	rep_002
cust_005	FastShip Co	["+1-555-1006"]	support@fastship.com	rep_003

CustomerByAccountRep GSI — Unterstützung von Anfragen von Kundenbetreuern

account_rep_id (GSI-PK)	Kunden-ID (GSI-SK)	Name	E-Mail
rep_001	cust_001	Acme Corporation	contact@acme.com
rep_001	cust_002	TechStart Inc	info@techstart.com
rep_002	cust_003	Globale Händler	sales@globaltraders.com
rep_002	cust_004	BuildRight LLC	orders@buildright.com
rep_003	cust_005	FastShip Co	support@fastship.com

Tischdesign bestellen

Die Bestelltabelle verwendet eine vertikale Partitionierung mit separaten Elementen für Auftragskopfe und Auftragspositionen. Dieses Design ermöglicht effiziente produktbasierte Abfragen, wobei alle Bestellkomponenten für einen effizienten Zugriff innerhalb derselben Partition beibehalten werden. Jede Bestellung besteht aus mehreren Artikeln:

- Bestellkopf — Enthält Bestellmetadaten mit pk=Order_ID, SK=Order_ID
- Bestellartikel — Einzelne Einzelposten mit pk=Order_ID, sk=Product_ID, wodurch direkte Produktanfragen ermöglicht werden

> **Note**
>
> Bei diesem Ansatz zur vertikalen Partitionierung wird die Einfachheit eingebetteter Bestellpositionen gegen eine verbesserte Abfrageflexibilität eingespielt. Jeder Bestellartikel wird zu einem separaten DynamoDB-Artikel, was effiziente produktbasierte Abfragen ermöglicht und gleichzeitig alle Bestelldaten innerhalb derselben Partition aufbewahrt, sodass sie effizient in einer einzigen Anfrage abgerufen werden können.

Die Tabelle beinhaltet die strategische Denormalisierung von account_rep_id (dupliziert aus der Kundentabelle), um direkte Anfragen von Kundenbetreuern zu ermöglichen, ohne dass Kundenanfragen erforderlich sind. Für Schreibszenarien mit hohem Durchsatz enthalten OPEN-Bestellungen Status- und Shard-Attribute, um Schreib-Sharding über mehrere Partitionen hinweg zu ermöglichen.

Vier GSIs unterstützen unterschiedliche Abfragemuster mit optimierten Prognosen:

- OrderByCustomerDate GSI — Nutzt die INCLUDE-Projektion mit Bestellzusammenfassung und Artikeldetails, um die Bestellhistorie von Kunden mit Filterung nach Datumsbereichen zu unterstützen
- OpenOrdersByDate GSI (Sparse, Sharded) — Verwendet einen Partitionsschlüssel mit mehreren Attributen (Status + Shard) mit 5 Shards, um 5.000 WPS (Schreibvorgänge pro Sekunde) auf die Partitionen zu verteilen (jeweils 1.000 WPS, was dem DynamoDB-Limit von 1.000 WCU pro Partition entspricht). Indiziert nur OFFENE Bestellungen (20% der Gesamtmenge), was zur Senkung der GSI-Speicherkosten beitragen kann. Erfordert parallel Abfragen über alle 5 Shards mit clientseitiger Ergebniszusammenführung
- OrderByAccountRep GSI — Nutzt die INCLUDE-Projektion mit Attributen für die Auftragsübersicht, um die Workflows von Kundenbetreuern ohne vollständige Bestelldetails zu unterstützen
- ProductInOrders GSI — Diese aus OrderItem Datensätzen (pk=Order_ID, SK=Product_ID) erstellte GSI ermöglicht Abfragen, um alle Bestellungen zu finden, die ein bestimmtes Produkt enthalten. Verwendet die INCLUDE-Projektion mit dem Bestellkontext (customer_id, order_date, Menge) für die Analyse der Produktnachfrage

Bestelltabelle — Struktur der Basistabelle (vertikale Partitionierung)

PK	SK	customer_id	Bestelldatum	Status	Account-Rep_ID	quantity	price	Shard
ord_001	ord_001	cust_001	15.11.2025	CLOSED	rep_001			
ord_001	prod_100					5	25,00	
ord_002	ord_002	cust_001	20.12.2025	OPEN	rep_001			0
ord_002	prod_101					10	15,00	

PK	SK	customer_id	Bestelldatum	Status	Account-Rep_ID	quantity	price	Shard
ord_003	ord_003	cust_002	2026-01-05	OPEN	rep_001			2
ord_003	prod_100					3	25,00	

OrderByCustomerDate GSI — Unterstützung bei Kundenanfragen zu Bestellungen

Kunden-ID (GSI-PK)	Bestelldatum (GSI-SK)	bestellnummer	Status	total_amount	Artikel bestellen	Shard
cust_001	15.11.2025	ord_001	CLOSED	225,00	[{product_id: „prod_100", Menge: 5}]	
cust_001	20.12.2025	ord_002	OPEN	150,00	[{product_id: „prod_101", Menge: 10}]	0
cust_002	2026-01-05	ord_003	OPEN	175,00	[{product_id: „prod_100", Menge: 3}]	2
cust_003	10.10.2025	ord_004	CLOSED	250,00	[{product_id: „prod_101", Menge: 5}]	
cust_004	2026-01-03	ord_005	OPEN	200,00	[{product_id: „prod_100", Menge: 20}]	1

OpenOrdersByDate GSI (Sparse, Sharded) — Unterstützung von offenen Auftragsabfragen mit hohem Durchsatz

Status (GSI-PK-1)	Scherbe (GSI-PK-2)	Bestellda tum (SK)	Bestell-ID	customer_ id	Account_R ep_ID	Artikel bestellen	total_amo unt
OPEN	0	2025-12-2 0	ord_002	cust_001	rep_001	[{product_id: „prod_101", Menge: 10}]	150,00
OPEN	1	2026-01-0 3	ord_005	cust_004	rep_002	[{product_id: „prod_100", Menge: 20}]	200,00
OPEN	2	2026-01-0 5	ord_003	cust_002	rep_001	[{product_id: „prod_100", Menge: 3}]	175,00

OrderByAccountRep GSI — Unterstützung bei Bestellanfragen von Kundenbetreuern

account_r ep_id (GSI-PK)	Bestelldatum (GSI-SK)	bestellnu mmer	customer_id	Status	total_amount
rep_001	15.11.2025	ord_001	cust_001	CLOSED	225,00
rep_001	20.12.2025	ord_002	cust_001	OPEN	150,00
rep_001	2026-01-05	ord_003	cust_002	OPEN	175,00
rep_002	10.10.2025	ord_004	cust_003	CLOSED	250,00
rep_002	2026-01-03	ord_005	cust_004	OPEN	200,00

ProductInOrders GSI — Unterstützung bei Fragen zu Produktbestellungen

Produktnummer (GSI-PK)	Bestellnummer (GSI-SK)	customer_id	Bestelldatum	quantity
prod_100	ord_001	cust_001	15.11.2025	5
prod_100	ord_003	cust_002	2026-01-05	3
prod_101	ord_002	cust_001	20.12.2025	10

Design von Produkttabellen

Die Produkttabelle verwendet das Artikelsammlungsmuster, um sowohl Produktmetadaten als auch Inventardaten in derselben Partition zu speichern. Dieses Design nutzt die identifizierende Beziehung zwischen Produkten und Inventar — Inventar kann ohne ein übergeordnetes Produkt nicht existieren. Durch die Verwendung von pk=Product_ID mit SK=Product_ID für Produktmetadaten und sk=Warehouse_ID für Inventarartikel entfällt die Notwendigkeit einer separaten Inventartabelle und eines globalen Index, wodurch die Kosten um etwa 50% gesenkt werden.

Dieses Muster ermöglicht effiziente Abfragen sowohl für den individuellen Lagerbestand (GetItem mit zusammengesetztem Schlüssel) als auch für den gesamten Lagerbestand eines Produkts (Abfrage nach Partitionsschlüssel). Das Attribut total_inventory im Produktmetadatenelement ermöglicht eine denormalisierte Aggregation für eine schnelle Suche nach dem Gesamtbestand.

Produkttabelle — Struktur der Basistabelle (Muster für die Artikelsammlung)

product_id (PK)	Lager-ID (SK)	product_name	category	Einzelpreis	Inventarmenge	gesamter Lagerbestand
prod_100	prod_100	Widget A	Hardware (Hardware)	25,00		500
prod_100	wh_sea				200	
prod_100	wh_pdx				150	
prod_100	wh_atl				150	

product_id (PK)	Lager-ID (SK)	product_name	category	Einzelpreis	Inventarmenge	gesamter Lagerbestand
prod_101	prod_101	Gerät B	Elektronik	50,00		300
prod_101	wh_sea				100	
prod_101	wh_pdx				200	

Jede Tabelle ist mit spezifischen globalen sekundären Indizes (GSIs) ausgestattet, um die erforderlichen Zugriffsmuster effizient zu unterstützen. Das Design verwendet aggregationsorientierte Prinzipien mit strategischer Denormalisierung und spärlicher Indexierung, um sowohl die Leistung als auch die Kosten zu optimieren.

Zu den wichtigsten Designoptimierungen gehören:

- Sparse GSI — indexiert OpenOrdersByDate nur OFFENE Bestellungen (20% der Gesamtmenge), was zur Senkung der GSI-Lagerkosten beitragen kann
- Muster für die Artikelerfassung — In der Produkttabelle wird der Lagerbestand mit PK=Product_ID, SK=Warehouse_ID gespeichert, um eine separate Inventartabelle zu vermeiden
- Bestellung + OrderItems Aggregation — Dank hundertprozentiger Zugriffskorrelation als Einzelartikel eingebettet
- Strategische Denormalisierung — account_rep_id wurde in der Order-Tabelle für effiziente Abfragen dupliziert

Schließlich können Sie die Zugriffsmuster erneut aufrufen, die zuvor definiert wurden. Die folgende Tabelle zeigt, wie jedes Zugriffsmuster effizient unterstützt wird, wenn das Multi-Tabellen-Design mit Strategic verwendet wird. GSIs Jedes Muster verwendet entweder direkte Schlüsselabfragen oder einzelne GSI-Abfragen, wodurch teure Scans vermieden und eine konsistente Leistung in jeder Größenordnung gewährleistet wird.

S. Nein.	Zugriffsmuster	Abfragebedingungen
1	Suchen von Mitarbeiterdetails nach Mitarbeiter-ID	Tabelle „Mitarbeiter“: GetItem (employee_id="emp_001")

S. Nein.	Zugriffsmuster	Abfragebedingungen
2	Mitarbeiterdetails nach Mitarbeiternamen abfragen	EmployeeByName GSI: Abfrage (Name="John Smith“)
3	Finden Sie die Telefonnummer (n) eines Mitarbeiters	Tabelle „Mitarbeiter“: GetItem (employee_id="emp_001")
4	Finden Sie die Telefonnummer (n) eines Kunden	Kundentabelle: GetItem (customer_id="cust_001")
5	Bestellungen für Kunden innerhalb des Datumsbereichs abrufen	OrderByCustomerDate GSI: Query (customer_id="cust _001", order_date ZWISCHEN „2025-01-01" UND „2025-12-31")
6	Alle offenen Bestellungen innerhalb des Datumsbereichs anzeigen	OpenOrdersByDate GSI: 5 Shards parallel mit PK mit mehreren Attributen abfragen (status="Open“ + shard=0-4), sk=ORDER_ DATE ZWISCHEN „2025-01-01" UND „2025-12-31", Ergebnisse zusammenführen
7	Alle kürzlich eingestellten Mitarbeiter anzeigen	EmployeeByHireDate GSI: Abfrage (entity_type="Empl oyee“, hire_date >= „2025-01-01")
8	Finde alle Mitarbeiter im Warehouse	EmployeeByWarehouse GSI: Abfrage (warehouse_id="wh_ sea“)
9	Alle bestellten Artikel für das Produkt abrufen	ProductInOrders GSI: Abfrage (product_id="prod_100")

S. Nein.	Zugriffsmuster	Abfragebedingungen
10	Besorgen Sie sich Lagerbestände für Produkte in allen Lagerhäusern	Produkttabelle: Abfrage (product_id="prod_100")
11	Holen Sie sich Kunden nach Kundenbetreuern	CustomerByAccountRep GSI: Abfrage (account_rep_id="rep_001")
12	Bestellungen nach Kundenbetreuern abrufen	OrderByAccountRep GSI: Abfrage (account_rep_id="rep_001")
13	Holen Sie sich Mitarbeiter mit Berufsbezeichnung	EmployeeByJobTitle GSI: Abfrage (job_title="Manager“)
14	Inventar nach Produkt und Lager abrufen	Produkttabelle: GetItem (product_id="prod_100", warehouse_id="wh_sea“)
15	Gesamten Produktbestand abrufen	Produkttabelle: GetItem (product_id="prod_100", warehouse_id="prod_100")

Migrieren aus einer relationalen Datenbank zu DynamoDB

Die Migration einer relationalen Datenbank zu DynamoDB erfordert eine sorgfältige Planung, damit ein erfolgreiches Ergebnis sichergestellt wird. In diesem Leitfaden erfahren Sie, wie dieser Prozess funktioniert, welche Tools Ihnen zur Verfügung stehen und wie Sie anschließend mögliche Migrationsstrategien bewerten und eine von ihnen auswählen können, die Ihren Anforderungen entspricht.

Themen

- Gründe für die Migration zu DynamoDB
- Überlegungen zum Migrieren einer relationalen Datenbank zu DynamoDB
- Verstehen, wie eine Migration zu DynamoDB funktioniert
- Tools für die Migration zu DynamoDB
- Auswählen der geeigneten Strategie für die Migration zu DynamoDB
- Durchführen einer Offline-Migration zu DynamoDB
- Durchführen einer Hybridmigration zu DynamoDB
- Durchführen einer Online-Migration zu DynamoDB mit 1:1-Migration aller Tabellen
- Durchführen einer Online-Migration zu DynamoDB mithilfe einer benutzerdefinierten Staging-Tabelle

Gründe für die Migration zu DynamoDB

Die Migration zu Amazon DynamoDB bietet eine Reihe überzeugender Vorteile für Unternehmen und Organisationen. Hier finden Sie einige wichtige Vorteile, die DynamoDB zu einer attraktiven Wahl für die Datenbankmigration machen:

- Skalierbarkeit: DynamoDB ist darauf ausgelegt, massive Workloads zu bewältigen, und lässt sich nahtlos skalieren, um wachsenden Datenmengen und Datenströmen gerecht zu werden. Mit DynamoDB können Sie die Datenbank je nach Bedarf einfach herauf- oder herunterskalieren und so sicherstellen, dass die Anwendungen plötzliche Spitzen im Datenverkehr ohne Leistungseinbußen bewältigen können.
- Leistung: DynamoDB bietet Datenzugriff mit niedriger Latenz, sodass Anwendungen Daten mit außergewöhnlicher Geschwindigkeit abrufen und verarbeiten können. Die verteilte Architektur stellt sicher, dass Lese- und Schreibvorgänge auf mehrere Knoten verteilt werden, sodass selbst bei

hohen Anforderungsraten konsistente Reaktionszeiten im einstelligen Millisekundenbereich erzielt werden.

- Vollständige Verwaltung: DynamoDB ist ein vollständig von AWS verwalteter Service. Das bedeutet, dass es sich um die betrieblichen Aspekte des Datenbankmanagements AWS kümmert, einschließlich Bereitstellung, Konfiguration, Patching, Backups und Skalierung. Auf diese Weise können Sie sich stärker auf die Entwicklung von Anwendungen als auf Datenbankverwaltungsaufgaben konzentrieren.
- Serverless-Architektur: DynamoDB unterstützt ein Serverless-Modell, bekannt als DynamoDB On-Demand, bei dem Sie nur für die tatsächlichen Lese- und Schreibanforderungen zahlen, die die Anwendung sendet, ohne dass vorab Kapazitäten bereitgestellt werden müssen. Dieses pay-per-request Modell bietet Kosteneffizienz und minimalen Betriebsaufwand, da Sie nur für die Ressourcen zahlen, die Sie verbrauchen, ohne dass Kapazitäten bereitgestellt und überwacht werden müssen.
- NoSQL-Flexibilität: Im Gegensatz zu herkömmlichen relationalen Datenbanken folgt DynamoDB einem NoSQL-Datenmodell, das Flexibilität beim Schemadesign bietet. Mit DynamoDB können Sie strukturierte, halbstrukturierte und unstrukturierte Daten speichern, sodass sie sich gut für die Verarbeitung von unterschiedlichen und sich weiterentwickelnden Datentypen eignen. Diese Flexibilität ermöglicht schnellere Entwicklungszyklen und eine einfachere Anpassung an sich ändernde Geschäftsanforderungen.
- Hohe Verfügbarkeit und Beständigkeit: DynamoDB repliziert Daten über mehrere Availability Zones innerhalb einer Region und gewährleistet so eine hohe Verfügbarkeit und Datenbeständigkeit. Replikation, Failover und Wiederherstellung von Daten werden automatisch übernommen, wodurch sich das Risiko von Datenverlusten oder Serviceunterbrechungen minimiert. Für DynamoDB gilt ein SLA mit einer Verfügbarkeit von bis zu 99,999 %.
- Sicherheit und Compliance: DynamoDB lässt sich in AWS Identity and Access Management integrieren und ermöglicht eine differenzierte Zugriffskontrolle. Es bietet Verschlüsselung im Ruhezustand und während der Übertragung und gewährleistet so die Sicherheit der Daten. DynamoDB erfüllt außerdem verschiedene Compliance-Standards, darunter HIPAA, PCI DSS und DSGVO, und entspricht somit den gesetzlichen Anforderungen.
- Integration mit dem AWS Ökosystem: Als Teil des AWS Ökosystems lässt sich DynamoDB nahtlos in andere AWS Dienste wie AWS Lambda CloudFormation, und integrieren. AWS AppSync Diese Integration ermöglicht es Ihnen, Serverless-Architekturen zu erstellen, Infrastruktur als Code zu nutzen und datengesteuerte Echtzeitanwendungen zu entwickeln.

Überlegungen zum Migrieren einer relationalen Datenbank zu DynamoDB

Relationale Datenbanksysteme und NoSQL-Datenbanken weisen unterschiedliche Vor- und Nachteile auf. Diese Unterschiede führen dazu, dass sich die Datenbankdesigns der beiden Systeme voneinander unterscheiden.

Aufgabentyp	Relationale Datenbank	NoSQL-Datenbank
Abfragen der Datenbank	In relationalen Datenbanken können Daten flexibel abgerufen werden, Abfragen sind jedoch vergleichsweise kostspielig und können bei einer großen Zahl von Zugriffen nicht gut skaliert werden (siehe Erste Schritte für die Modellierung relationaler Daten in DynamoDB). Eine relationale Datenbankanwendung kann Geschäftslogik in gespeicherten Prozeduren, SQL-Unterabfragen, Massenaktualisierungsabfragen und Aggregationsabfragen implementieren.	In einer NoSQL-Datenbank wie DynamoDB können Daten effizient mithilfe einer begrenzten Anzahl von Möglichkeiten abgerufen werden, die ansonsten möglicherweise kostspielig und langsam sind. Schreibvorgänge in DynamoDB sind Singletons. Die Geschäftslogik von Anwendungen, die früher in gespeicherten Prozeduren ausgeführt wurden, muss so umgestaltet werden, dass sie außerhalb von DynamoDB in benutzerdefiniertem Code ausgeführt wird, der auf einem Host wie Amazon Amazon EC2 oder ausgeführt wird. AWS Lambda
Entwerfen der Datenbank	Sie entwerfen das Design im Hinblick auf Flexibilität, ohne sich um Implementierungsdetails oder Leistung zu kümmern. Die Optimierung von Abfragen wirkt sich	Sie entwerfen das Schema im Hinblick darauf, die häufigsten und wichtigsten Abfragen so schnell und kostengünstig wie möglich ausführen zu können. Die Datenstrukturen sind an

Aufgabentyp	Relationale Datenbank	NoSQL-Datenbank
	im Allgemeinen nicht auf das Schemadesign aus, eine Standardisierung ist jedoch wichtig.	die spezifischen Anforderungen Ihrer geschäftlichen Anwendungsfälle angepasst.

Die Entwicklung einer NoSQL-Datenbank erfordert einen anderen Ansatz als die Entwicklung eines relationalen Datenbankmanagementsystems (RDBMS). Sie können ein standardisiertes Datenmodell für ein RDBMS entwickeln, ohne sich Gedanken über Zugriffsmuster machen zu müssen. Anschließend können Sie es erweitern, wenn neue Fragen und Abfrageanforderungen entstehen. Sie können jeden einzelnen Typ von Daten in einer eigenen Tabelle organisieren.

Mit dem NoSQL-Design können Sie das Schema für DynamoDB entwerfen, wenn Sie die Fragen kennen, die es beantworten soll. Es ist äußerst wichtig, die geschäftlichen Probleme und die Lese- und Schreibmuster der Anwendung zu kennen. Zudem sollten Sie in einer DynamoDB-Anwendung möglichst wenige Tabellen verwenden. Weniger Tabellen sorgen dafür, dass die Dinge besser skalierbar sind, weniger Berechtigungsmanagement erforderlich sind und der Overhead für Ihre DynamoDB-Anwendung reduziert wird. Dies kann auch dazu beitragen, die Backup-Kosten insgesamt niedrig zu halten.

Die Aufgabe der Modellierung relationaler Daten für DynamoDB und der Erstellung einer neuen Version der Front-End-Anwendung wird in einem eigenständigen Thema behandelt. In diesem Leitfaden wird davon ausgegangen, dass Sie über eine neue Version Ihrer Anwendung verfügen, die für die Verwendung von DynamoDB entwickelt wurde, Sie jedoch noch herausfinden müssen, wie Sie historische Daten beim Cutover am besten migrieren und synchronisieren können.

Größenüberlegungen

Die maximale Größe der einzelnen Elemente (Zeilen), die Sie in einer DynamoDB-Tabelle speichern können, beträgt 400 KB. Weitere Informationen finden Sie unter the section called “Kontingente”. Die Elementgröße wird durch die Gesamtgröße aller Attributnamen und Attributwerte in einem Element bestimmt. Weitere Informationen finden Sie unter the section called “Elementgrößen und -formate”.

Wenn die Anwendung mehr Daten in einem Element speichern muss, als es die Größenbeschränkung von DynamoDB zulässt, teilen Sie das Element in eine Elementauflistung auf, komprimieren Sie die Elementdaten oder speichern Sie das Element als Objekt in Amazon Simple Storage Service (Amazon S3), während Sie die Amazon-S3-Objekt-ID in dem DynamoDB-

Element speichern. Siehe the section called “Große Elemente”. Die Kosten für die Aktualisierung eines Elements basiert auf der Gesamtgröße des Elements. Bei Workloads, in denen vorhandene Elemente häufig aktualisiert werden müssen, kostet die Aktualisierung kleiner Elemente mit einer Größe von 1 oder 2 KB weniger als die Aktualisierung größerer Elemente. Weitere Informationen zu Elementauflistungen finden Sie unter the section called “Arbeiten mit Elementauflistungen”.

Lesen Sie vor der Auswahl der Attribute für die Partitions- und Sortierschlüssel, anderer Tabelleneinstellungen, der Elementgröße und -struktur und der Entscheidung, ob Sekundärindizes erstellt werden sollen, unbedingt die Dokumentation zur DynamoDB-Modellierung sowie den Leitfaden für the section called “Kostenoptimierung”. Testen Sie unbedingt Ihren Migrationsplan, damit Ihre DynamoDB-Lösung kosteneffizient ist und zu den Features und Einschränkungen von DynamoDB passen.

Verstehen, wie eine Migration zu DynamoDB funktioniert

Bevor Sie sich mit den zur Verfügung stehenden Migrationstools befassen, sollten Sie sich darüber Gedanken machen, wie Schreibvorgänge von DynamoDB verarbeitet werden.

Die standardmäßige und am häufigsten verwendete Schreiboperation ist eine einzelne `PutItem`-API-Operation. Sie können eine `PutItem`-Operation in einer Schleife ausführen, um Datensätze zu verarbeiten. DynamoDB unterstützt praktisch unbegrenzt viele gleichzeitige Verbindungen. Wenn Sie also eine umfangreiche Multithread-Laderoutine wie MapReduce oder Spark konfigurieren und ausführen können, ist die Geschwindigkeit von Schreibvorgängen nur durch die Kapazität der Zieltabelle begrenzt (die im Allgemeinen ebenfalls unbegrenzt ist).

Wenn Sie Daten in DynamoDB laden, ist es wichtig, die Schreibgeschwindigkeit des Loaders zu kennen. Wenn die Größe der geladenen Elemente (Zeilen) 1 KB beträgt oder geringer ist, entspricht die Geschwindigkeit einfach die Anzahl der Elemente pro Sekunde. Die Zieltabelle kann dann mit einer ausreichenden Anzahl von WCUs (Write Capacity Units, Schreibkapazitätseinheiten) ausgestattet werden, um diese Rate zu bewältigen. Wenn der Loader die bereitgestellte Kapazität in einer einzelnen Sekunde überschreitet, werden die zusätzlichen Anfragen möglicherweise gedrosselt oder ganz zurückgewiesen. Sie können in den CloudWatch Diagrammen auf der Registerkarte Überwachung der DynamoDB-Konsole nach Drosselungen suchen.

Für den zweiten ausführbaren Vorgang wird eine zugehörige API namens `BatchWriteItem` verwendet. In `BatchWriteItem`können Sie bis zu 25 Schreibanforderungen in einem API-Aufruf kombinieren. Diese werden vom Service empfangen und als separate `PutItem`-Anfragen an

die Tabelle verarbeitet. Derzeit können Sie bei der Auswahl nicht den Vorteil der automatischen Wiederholungsversuche nutzen`BatchWriteItem`, die im AWS SDK enthalten sind, wenn Sie Singleton-Aufrufe mit tätigen. `PutItem` Wenn also Fehler auftreten (z. B. Drosselungsausnahmen), müssen Sie beim Antwortaufruf an `BatchWriteItem` nach der Liste aller fehlgeschlagenen Schreibvorgänge suchen. Weitere Informationen zum Umgang mit Drosselungswarnungen für den Fall, dass diese in den Drosselungstabellen erkannt werden, finden Sie CloudWatch unter. the section called “Drosselung”

Der Import des dritten Datentyps ist mit dem Feature zum Importieren von DynamoDB-Daten aus S3 möglich. Mit diesem Feature können Sie einen großen Datensatz in Amazon S3 bereitstellen und DynamoDB anweisen, die Daten automatisch in eine neue Tabelle zu importieren. Der Import startet nicht sofort und seine Dauer richtet sich nach der Größe des Datensatzes. Dies ist jedoch praktisch, da weder eine ETL-Plattform noch benutzerdefinierter DynamoDB-Code erforderlich sind. DynamoDB lädt die Daten in eine neue Tabelle, die durch den Import erstellt wurde. Es ist derzeit nicht möglich, Daten in eine vorhandene Tabelle zu laden. DynamoDB importiert die Daten unverändert, ohne eine Transformation vorzunehmen. Ähnlich wie bei `PutItem` ist hier ein Upstream-Prozess erforderlich und die Daten werden in dem von Ihnen ausgewählten Format in einen Amazon-S3-Bucket geschrieben.

Tools für die Migration zu DynamoDB

Es gibt mehrere gängige Migrations- und ETL-Tools, mit denen Daten zu DynamoDB migriert werden können.

Amazon stellt eine Vielzahl von Datentools für die Migration bereit, darunter AWS Database Migration Service (DMS), AWS Glue, Amazon EMR und Amazon Managed Streaming für Apache Kafka. Diese Tools können für eine Migration mit Ausfallzeit verwendet werden und sie nutzen die CDC-Features (Change Data Capture, Erfassung von Datenänderungen) von relationalen Datenbanken für die Online-Migration. Bei der Auswahl ist es hilfreich, Fähigkeiten und Erfahrungen Ihrer Organisation mit den einzelnen Tools sowie deren Features, Leistung und Kosten zu berücksichtigen.

Viele Kunden entscheiden sich dafür, eigene Migrationsskripte und -aufträge zu schreiben und erstellen benutzerdefinierte Datentransformationen für den Migrationsprozess. Wenn Sie eine DynamoDB-Tabelle mit starkem Schreibverkehr oder regelmäßigen umfangreichen Massenladeaufgaben betreiben möchten, können Sie den Migrationscode selbst schreiben, um sich mit dem Verhalten von DynamoDB bei hohem Schreibverkehr vertraut zu machen. Szenarien wie der Umgang mit Drosselungen und die effiziente Bereitstellung von Tabellen können so bereits zu Beginn der Testmigration erprobt werden.

Auswählen der geeigneten Strategie für die Migration zu DynamoDB

Eine große relationale Datenbankanwendung kann mehr als einhundert Tabellen umfassen und mehrere verschiedene Anwendungsfunktionen unterstützen. Wenn Sie eine große Migration planen, sollten Sie erwägen, die Anwendung in kleinere Komponenten oder Microservices aufzuteilen und jeweils nur eine kleine Anzahl von Tabellen zu migrieren. Anschließend können Sie in Wellen weitere Komponenten zu DynamoDB migrieren.

Bei der Auswahl einer Migrationsstrategie hängt die Entscheidung für eine Lösung von verschiedenen Faktoren ab. Wir können die Optionen, die uns je nach unseren Anforderungen und Ressourcen zur Verfügung stehen, in einem Entscheidungsbaum veranschaulichen. Die Konzepte werden hier kurz vorgestellt (und an späterer Stelle im Leitfaden ausführlicher behandelt):

- Offline-Migration: Wenn die Anwendung während der Migration eine gewisse Ausfallzeit toleriert, vereinfacht diese den Migrationsprozess.
- Hybridmigration: Bei diesem Ansatz wird der Betrieb während der Migration teilweise aufrechterhalten; so sind z. B. Lese-, aber keine Schreibvorgänge, und Lese- und Einfügevorgänge, aber keine Aktualisierungen und Löschungen möglich.
- Online-Migration: Die Migration von Anwendungen, die ohne Ausfallzeiten weiterlaufen müssen, ist weniger einfach und erfordert möglicherweise eine umfangreiche Planung und benutzerdefinierte Entwicklung. Eine wichtige Entscheidung besteht darin, die Kosten für die Erstellung eines benutzerdefinierten Migrationsprozesses anhand der Kosten abzuwägen, die dem Unternehmen durch ein Ausfallzeitfenster während des Cutovers entstehen.

Bedingung 1	Bedingung 2	Empfehlung
Es ist möglich, die Anwendung während eines Wartungsfensters für einige Zeit herunterzufahren, damit die Datenmigration durchgeführt werden kann. Dies ist eine Offline-Migration.		Verwenden Sie eine Offlinemigration AWS DMS und führen Sie sie mithilfe einer Vollladeaufgabe durch. Strukturieren Sie die Quelldaten gegebenenfalls mit einem SQL `VIEW` vor.

Bedingung 1	Bedingung 2	Empfehlung
Es ist möglich, die Anwendung während der Migration im schreibgeschützten Modus ausführen. Dies ist eine Hybridmigration.		Deaktivieren Sie Schreibvorgänge in der Anwendungs- oder Quelldatenbank. Verwenden AWS DMS und führen Sie eine Offline-Migration mithilfe einer Volllladetask durch.
Es ist möglich, die Anwendung während der Migration mit Lese- und Einfügungsvorgängen neuer Datensätze, aber ohne Aktualisierungen oder Löschungen ausführen. Dies ist eine Hybridmigration.	Sie verfügen über Kenntnisse in der Anwendungsentwicklung und können die vorhandene relationale Datenbankanwendung aktualisieren, um für alle neuen Datensätze duale Schreibvorgänge durchzuführen, auch in DynamoDB.	Verwenden AWS DMS und führen Sie eine Offline-Migration mithilfe einer Volllladetask durch. Stellen Sie gleichzeitig eine Version der vorhandenen Anwendung bereit, die Lesevorgänge und duale Schreibvorgänge ermöglicht.
Es ist eine Migration mit minimalen Ausfallzeiten erforderlich. Dies ist eine Online-Migration.	• Sie migrieren Quelltabellen 1:1 ohne größere Schemaänderungen zu DynamoDB.	Wird verwendet AWS DMS , um eine Online-Datenmigration durchzuführen. Führen Sie eine Massenladeaufgabe gefolgt von einer CDC-Synchronisierungsaufgabe durch.

Bedingung 1	Bedingung 2	Empfehlung
Es ist eine Migration mit minimalen Ausfallzeiten erforderlich. Dies ist eine Online-Migration.	• Sie möchten Quelltabellen nach dem Konzept von gestapelten Schemas oder einzelnen Tabellen zu einer geringeren Anzahl von DynamoDB-Tabellen kombinieren. • Sie haben Kenntnisse in der Entwicklung von Backend-Datenbanken und der SQL-Host weist freie Kapazitäten auf.	Erstellen Sie die NoSQL-fähige Tabelle in der SQL-Datenbank. Füllen und synchronisieren Sie es mithilfe von JOINs, UNIONs VIEWs, Triggern und gespeicherten Prozeduren.
Es ist eine Migration mit minimalen Ausfallzeiten erforderlich. Dies ist eine Online-Migration.	• Sie möchten Quelltabellen nach dem Konzept von einzelnen Tabellen zu einer geringeren Anzahl von DynamoDB-Tabellen kombinieren. Beispiel: • Sie haben keine Kenntnisse in der Entwicklung von Backend-Datenbanken und der SQL-Host weist keine freien Kapazitäten auf.	Ziehen Sie eine Hybrid- oder Offline-Migration in Betracht.

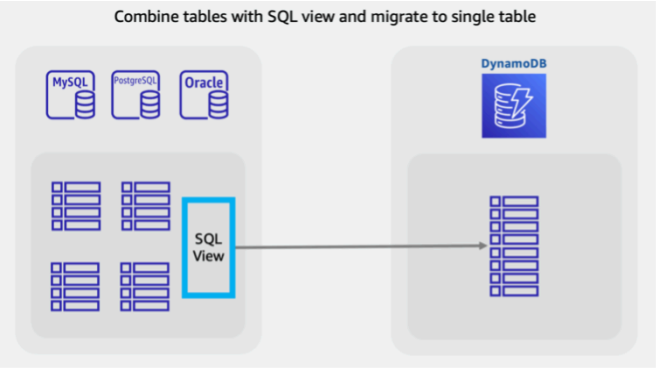
Bedingung 1	Bedingung 2	Empfehlung
Es ist eine Migration mit minimalen Ausfallzeiten erforderlich. Dies ist eine Online-Migration.	Es ist möglich, die Migration historischer Transaktionsdaten zu überspringen oder die Daten in Amazon S3 archivieren, anstatt sie zu migrieren. Es müssen nur einige kleine statische Tabellen migriert werden.	Schreiben Sie ein Skript oder verwenden Sie für die Migration der Tabellen ein beliebiges ETL-Tool. Strukturieren Sie die Quelldaten gegebenenfalls mit einem SQL `VIEW` vor.

Durchführen einer Offline-Migration zu DynamoDB

Offline-Migrationen eignen sich, wenn Sie für die Durchführung der Migration ein Ausfallzeitfenster einplanen können. Bei relationalen Datenbanken kommt es in der Regel jeden Monat zu einer gewissen Ausfallzeit aufgrund von Wartungsarbeiten und Patches oder längeren Ausfallzeiten aufgrund von Hardware-Upgrades oder Upgrades auf Hauptversionen.

Amazon S3 kann während einer Migration als Staging-Bereich verwendet werden. Im CSV- oder DynamoDB-JSON-Format gespeicherte Daten können mit dem Feature zum Importieren von DynamoDB-Daten aus S3 automatisch in eine neue DynamoDB-Tabelle importiert werden.

Sie können Tabellen kombinieren, um eindeutige NoSQL-Zugriffsmuster zu nutzen (z. B. die Umwandlung von vier Legacy-Tabellen in eine einzige DynamoDB-Tabelle). Bei der Anforderung eines Dokuments mit einem einzigen Schlüsselwert oder der Abfrage einer vorgruppierten Elementauflistung ist die Latenz in der Regel besser als einer SQL-Datenbank, in der ein Verbund aus mehreren Tabellen hergestellt wird. Dies macht die Migrationsaufgabe jedoch schwieriger. Eine SQL-Ansicht könnte die Arbeit innerhalb der Quelldatenbank übernehmen und einen einzelnen Datensatz vorbereiten, der alle vier Tabellen in einem Satz darstellt.



In dieser Ansicht können Tabellen per `JOIN` in eine denormalisierte Form umgewandelt oder die Entitäten normalisiert und Tabellen mithilfe eines SQL `UNION` gestapelt werden. Die wichtigsten Entscheidungen im Zusammenhang mit der Umstrukturierung relationaler Daten finden Sie in diesem Video. Bei Offline-Migrationen bietet sich die Verwendung einer Ansicht zum Kombinieren von Tabellen an, um Daten für ein DynamoDB-Einzeltabellenschema zu strukturieren.

Plan

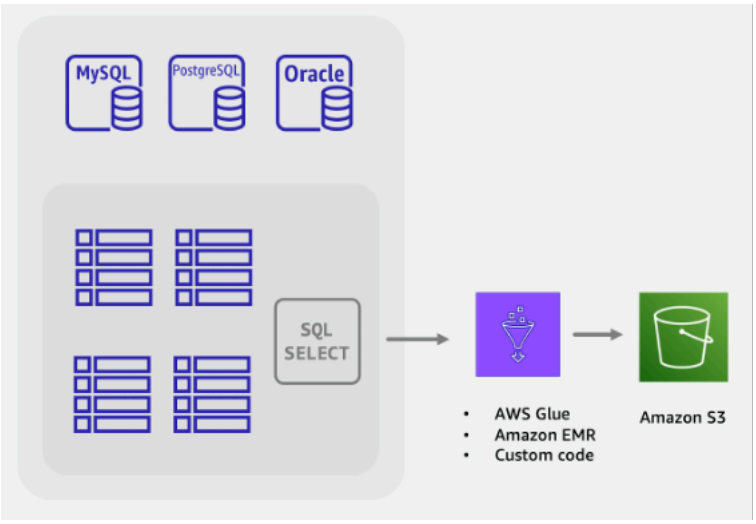
Offline-Migration mit Amazon S3 durchführen

Tools

- Ein ETL-Auftrag zum Extrahieren und Transformieren von SQL-Daten und zum Speichern in einem S3-Bucket, z. B.:
  - AWS Database Migration Service, ein Dienst, mit dem eine Massenladung historischer Daten und auch die Verarbeitung von CDC-Datensätzen möglich ist, um Quell- und Zieltabellen zu synchronisieren.
  - AWS Glue
  - Amazon EMR
  - Eigener benutzerdefinierter Code
- Feature zum Importieren von DynamoDB-Daten aus S3

Schritte für die Offline-Migration:

1. Erstellen Sie einen ETL-Auftrag, der die SQL-Datenbank abfragen, Tabellendaten in das DynamoDB-JSON- oder CSV-Format umwandeln und in einem S3-Bucket speichern kann.



2. Das Feature zum Importieren von DynamoDB-Daten aus S3 wird aufgerufen; es wird eine neue Tabelle erstellt und es werden automatisch Daten aus dem S3-Bucket geladen.

Die vollständige Offline-Migration ist zwar einfach, aber bei Anwendungsbesitzern und -benutzern möglicherweise weniger beliebt. Für die Benutzer wäre es von Vorteil, wenn die Anwendung während der Migration zumindest einen geringeren Service-Level aufrecht erhalten könnte.

Es besteht die Möglichkeit, Funktionen hinzuzufügen, die Schreibvorgänge während der Offline-Migration unterbinden, Lesevorgänge aber aufrecht erhalten. Die Benutzer der Anwendung könnten dann während der Migration der relationalen Daten weiterhin sicher nach vorhandenen Daten suchen und Abfragen dafür durchführen. Wenn Sie genau das suchen, lesen Sie weiter, um mehr über Hybridmigrationen zu erfahren.

Durchführen einer Hybridmigration zu DynamoDB

Lese- und Schreibvorgänge werden zwar in alle Datenbankanwendungen durchgeführt, bei der Planung einer Hybrid- oder Online-Migration ist allerdings die Art des jeweiligen Vorgangs zu berücksichtigen. Datenbankschreibvorgänge lassen sich in drei Bereiche unterteilen: Einfügungen, Aktualisierungen und Löschungen. Es gibt Anwendungen, in den Datenlöschungen nicht sofort verarbeitet werden müssen. Hier können die Löschungen beispielsweise in einem Massenbereinigungsprozess am Monatsende erfolgen. Solche Anwendungen lassen sich einfacher und unter teilweiser Aufrechterhaltung der Betriebszeit migrieren.

Plan

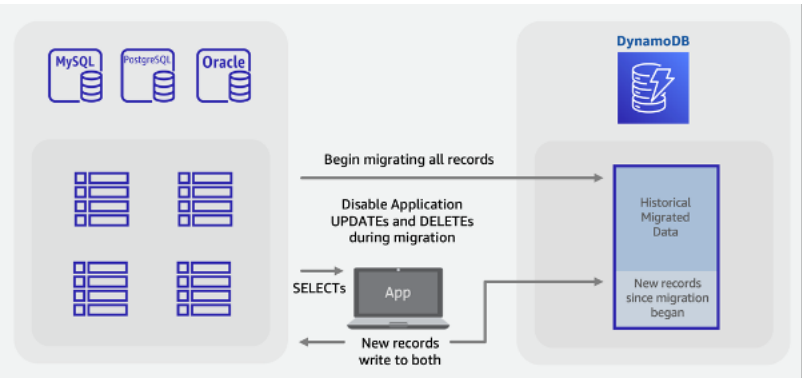
Führen Sie eine online/offline Hybridmigration mit dualen Schreibvorgängen für Anwendungen durch

Tools

- Ein ETL-Auftrag zum Extrahieren und Transformieren von SQL-Daten und zum Speichern in einem S3-Bucket, z. B.:
  - AWS DMS
  - AWS Glue
  - Amazon EMR
  - Eigener benutzerdefinierter Code

Schritte für die Hybridmigration:

1. Erstellen Sie die DynamoDB-Zieltabelle. In diese Tabelle gehen sowohl historische Massendaten als auch neue Live-Daten ein.
2. Erstellen Sie eine Version der Legacy-Anwendung, bei der Löschungen und Aktualisierungen deaktiviert sind, alle Einfügungen in die SQL-Datenbank und in DynamoDB jedoch als duale Schreibvorgänge ausgeführt werden.
3. Beginnen Sie mit dem ETL-Job oder der AWS DMS ETL-Aufgabe, um bestehende Daten aufzufüllen und gleichzeitig die neue Anwendungsversion bereitzustellen
4. Nach Abschluss des Backfill-Auftrags enthält DynamoDB alle vorhandenen und neuen Datensätze und der Cutover der Anwendung kann beginnen.



> **Note**
>
> Beim Backfill-Auftrag werden die Daten direkt von SQL nach DynamoDB geschrieben. Das Feature für den Import aus S3 kann nicht wie im Beispiel für die Offline-Migration verwendet

werden, da hierdurch eine neue Tabelle erstellt wird, die erst verfügbar ist, nachdem die Daten in DynamoDB geladen wurden.

Durchführen einer Online-Migration zu DynamoDB mit 1:1-Migration aller Tabellen

Viele relationale Datenbanken verfügen über das Feature „Erfassung von Datenänderungen“ (Change Data Capture, CDC), mit dem Benutzer eine Liste der Änderungen an einer Datenbanktabelle anfordern können, die vor oder nach einem bestimmten Zeitpunkt vorgenommen wurden. CDC aktiviert dieses Feature mit internen Protokollen, sodass die Tabelle keine Spalte mit Zeitstempel enthalten muss, damit dies funktioniert.

Wenn Sie ein Schema von SQL-Tabellen in eine NoSQL-Datenbank migrieren, können Sie die Daten kombinieren und so umstrukturieren, dass weniger Tabellen erforderlich sind. Auf diese Weise werden Daten an einem einzigen Ort erfasst, wobei verwandte Daten nicht manuell in mehrstufigen Lesevorgängen zusammengefügt werden müssen. Die Strukturierung von Daten in einer Tabelle ist jedoch nicht immer erforderlich; manchmal werden die Tabellen auch 1:1 zu DynamoDB migriert. Solche 1:1-Migrationen sind weniger kompliziert, da Sie das CDC-Feature der Quelldatenbank nutzen und gängige ETL-Tools verwenden können, die diese Art der Migration unterstützen. Die Daten jeder Zeile können dann weiterhin transformiert werden, aber der Umfang der einzelnen Tabellen bleibt gleich.

Erwägen Sie die 1:1-Migration von SQL-Tabellen zu DynamoDB unter dem Vorbehalt, dass DynamoDB keine serverseitigen Joins unterstützt. Sie müssen Ihrer Anwendung Logik hinzufügen, um Daten aus mehreren Tabellen zu kombinieren.

Plan

Führen Sie eine Online-Migration jeder Tabelle zu DynamoDB durch, indem Sie AWS DMS

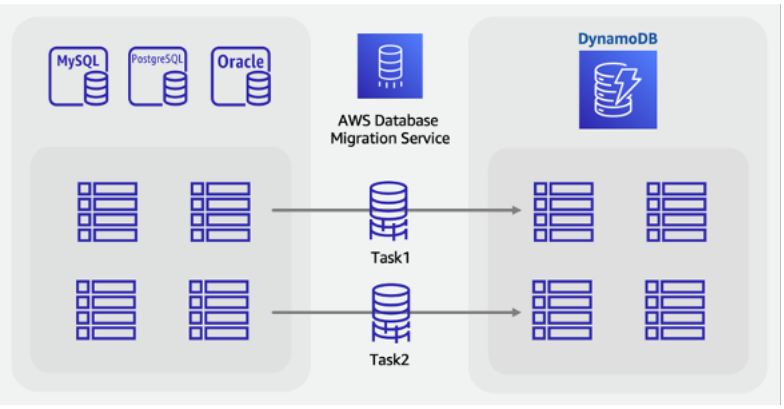
Tools

- AWS DMS (DMS)

Schritte für die Online-Migration:

1. Identifizieren Sie in Ihrem Quellschema die zu migrierenden Tabellen.

2. Erstellen Sie in DynamoDB dieselbe Anzahl von Tabellen mit identischer Schlüsselstruktur wie in der Quelle.
3. Erstellen Sie einen Replikationsserver auf AWS DMS und konfigurieren Sie die Quell- und Zielendpunkte
4. Definieren Sie alle erforderlichen Transformationen pro Zeile (z. B. Verkettung von Spalten oder Umrechnung von Datumsangaben in das Zeichenfolgenformat ISO-8601).
5. Erstellen Sie für jede Tabelle eine Migrationsaufgabe für Volllast und CDC.
6. Überwachen Sie diese Aufgaben, bis die laufende Replikationsphase begonnen hat
7. Zu diesem Zeitpunkt können Sie alle Validierungsaudits durchführen und dann Benutzer zu der Anwendung umleiten, die in DynamoDB liest und schreibt.



Durchführen einer Online-Migration zu DynamoDB mithilfe einer benutzerdefinierten Staging-Tabelle

Wie im obigen Offline-Migrationsszenario können Sie Tabellen kombinieren, um eindeutige NoSQL-Zugriffsmuster zu nutzen (z. B. Transformation von vier Legacy-Tabellen in eine einzige DynamoDB-Tabelle). Eine SQL `VIEW` könnte die Arbeit innerhalb der Quelldatenbank übernehmen und einen einzelnen Datensatz vorbereiten, der alle vier Tabellen in einem Satz darstellt.

Bei Online-Migrationen mit sich ändernden Live-Daten können Sie allerdings keine CDC-Features nutzen, da sie von `VIEW`s nicht unterstützt werden. Wenn die Tabellen eine Spalte mit den Zeitstempeln der letzten Aktualisierung enthalten und diese in die `VIEW` integriert werden, können Sie einen benutzerdefinierten ETL-Auftrag erstellen, der anhand dieser Zeitstempel einen Massenladevorgang mit Synchronisation durchführt.

Ein neuer Ansatz zur Bewältigung dieser Herausforderung besteht darin, mithilfe von SQL-Standardfunktionen wie Ansichten, gespeicherten Prozeduren und Triggern eine neue SQL-Tabelle zu erstellen, die im endgültig gewünschten DynamoDB-NoSQL-Format vorliegt.

Wenn der Datenbankserver freie Kapazitäten aufweist, kann diese einzelne Staging-Tabelle vor Beginn der Migration erstellt werden. Schreiben Sie dazu eine gespeicherte Prozedur, die Daten aus vorhandenen Tabellen liest, Daten nach Bedarf transformiert und in die neue Staging-Tabelle schreibt. Sie können eine Reihe von Triggern hinzufügen, damit Änderungen an Tabellen in Echtzeit in der Staging-Tabelle repliziert werden. Wenn die Unternehmensrichtlinie keine Trigger zulässt, lässt sich durch Änderungen an gespeicherten Prozeduren dasselbe Ergebnis erzielen. Sie fügen dann jeder Prozedur, die Daten schreibt, einige Codezeilen hinzu, sodass dieselben Änderungen zusätzlich in die Staging-Tabelle geschrieben werden.

Diese Staging-Tabelle, die vollständig mit den Legacy-Anwendungstabellen synchronisiert wird, stellt einen guten Ausgangspunkt für eine Live-Migration dar. Tools, die Datenbank-CDC verwenden, um Live-Migrationen durchzuführen, wie z. B. AWS DMS, können jetzt für diese Tabelle verwendet werden. Ein Vorteil dieses Ansatzes besteht darin, dass er auf vertraute SQL-Kenntnisse und -Features zurückgreift, die in der relationalen Datenbank-Engine verfügbar sind.

Plan

Führen Sie eine Online-Migration mit einer SQL-Staging-Tabelle durch AWS DMS

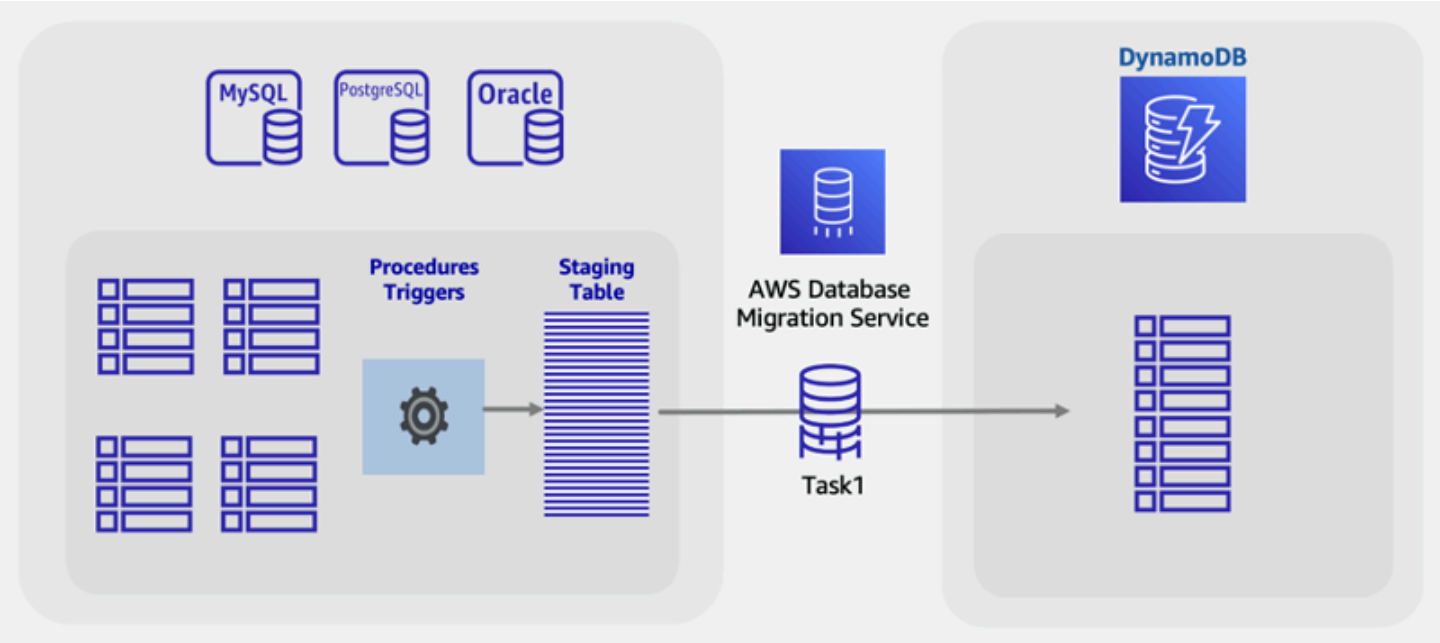
Tools

- Benutzerdefinierte gespeicherte SQL-Prozeduren oder -Trigger
- AWS DMS

Schritte für die Online-Migration:

1. Vergewissern Sie sich, dass die relationale Quelldatenbank-Engine ausreichend Festplattenspeicher und Verarbeitungskapazität aufweist.
2. Erstellen Sie eine neue Staging-Tabelle in der SQL-Datenbank mit aktivierten Zeitstempeln oder CDC-Features.
3. Schreiben Sie eine gespeicherte Prozedur und führen Sie sie aus, um vorhandene relationale Tabellendaten in die Staging-Tabelle zu kopieren.

4. Stellen Sie Trigger bereit oder ändern Sie bestehende Prozeduren, um in der neuen Staging-Tabelle duales Schreiben und gleichzeitig in vorhandenen Tabellen normale Schreibvorgänge zuzulassen.
5. Ausführen AWS DMS , um diese Quelltabelle zu migrieren und mit einer DynamoDB-Zieltabelle zu synchronisieren



In diesem Leitfaden wurden verschiedene Überlegungen und Ansätze für die Migration relationaler Datenbankdaten zu DynamoDB vorgestellt, wobei der Schwerpunkt auf der Minimierung von Ausfallzeiten und der Verwendung gängiger Datenbanktools und -techniken lag. Weitere Informationen finden Sie hier:

- AWS DMS Benutzerhandbuch
- AWS Glue Benutzerhandbuch
- Bewährte Methoden für die Migration von RDBMS zu DynamoDB

NoSQL-Workbench für DynamoDB

NoSQL Workbench for Amazon DynamoDB ist eine plattformübergreifende clientseitige GUI-Anwendung für moderne Datenbankentwicklung und -operationen. Sie ist für Windows, macOS und Linux verfügbar. Mit NoSQL Workbench können Sie DynamoDB-Datenmodelle entwerfen, Zugriffsmuster als echte DynamoDB-Operationen definieren und sie anhand von Beispieldaten validieren. Darüber hinaus können Sie Ihre Datenmodelle in Projekten organisieren. NoSQL Workbench beinhaltet DynamoDB Local, sodass Sie Ihre Tabellen und Indizes testen können, bevor Sie Ihr Datenmodell in die Cloud übertragen. Weitere Informationen über die lokale Version von DynamoDB und ihre Anforderungen finden Sie unter Lokale Einrichtung von DynamoDB (herunterladbare Version).

Data Modeler

Mit NoSQL Workbench für DynamoDB können Sie ein neues Projekt von Grund auf neu starten oder ein Beispielprojekt verwenden, das Ihrem Anwendungsfall entspricht. Anschließend entwerfen Sie Tabellen und globale Sekundärindizes, definieren Attribute und konfigurieren Beispieldaten. Sie können Ihre Zugriffsmuster auch als echte DynamoDB-Operationen (PutItem, UpdateItem, Query und andere) visualisieren und diese Operationen anhand der konfigurierten Beispieldaten ausführen, um zu überprüfen, ob das Zugriffsmuster wie beabsichtigt funktioniert, und Anpassungen am Datenmodell auf der Grundlage der Überprüfungsergebnisse vornehmen. Schließlich übertragen Sie das Modell nach der Validierung entweder lokal auf DynamoDB oder auf Ihr AWS Konto, um es weiter zu testen und in der Produktion zu verwenden. Für die Zusammenarbeit können Sie entworfene Datenmodelle importieren und exportieren. Weitere Informationen finden Sie unter Erstellen von Datenmodellen mit NoSQL Workbench.

Operation Builder

NoSQL Workbench stellt eine umfassende Grafikbenutzeroberfläche für die Entwicklung und den Test von Abfragen bereit. Sie können den Operation Builder verwenden, um Live-Datensätze anzuzeigen, zu erkunden und abzufragen. Der strukturierte Operation Builder unterstützt Projektionsausdrücke und Bedingungsausdrücke und generiert Beispielcode in mehreren Sprachen. Sie können Tabellen von einem Amazon-DynamoDB-Konto direkt in ein anderes in verschiedenen Regionen klonen. Sie können Tabellen auch direkt zwischen einer lokalen Version von DynamoDB und einem Amazon-DynamoDB-Konto klonen, um das Schlüsselschema Ihrer Tabelle (und optional das Schema und Elemente des GSI) schneller zwischen Ihren Entwicklungsumgebungen zu kopieren. Weitere Informationen finden Sie unter Erkunden von Datasets und Erstellen von Vorgängen mit NoSQL Workbench.

Im folgenden Video werden die Konzepte der Datenmodellierung mit NoSQL Workbench vorgestellt.

Themen

- Herunterladen von NoSQL Workbench for DynamoDB
- Erstellen von Datenmodellen mit NoSQL Workbench
- Erkunden von Datasets und Erstellen von Vorgängen mit NoSQL Workbench
- Beispieldatenmodelle für NoSQL Workbench
- Versionsverlauf für NoSQL-Workbench

Herunterladen von NoSQL Workbench for DynamoDB

Folgen Sie diesen Anweisungen, um NoSQL Workbench und DynamoDB local für Amazon DynamoDB herunterzuladen.

So laden Sie NoSQL Workbench und die lokale Version von DynamoDB herunter

- Laden Sie die entsprechende Version von NoSQL-Workbench für Ihr Betriebssystem herunter.

Betriebssystem	Download-Link
macOS (Intel)	Für macOS (Intel) herunterladen
macOS (Apple silicon)	Für macOS (Apple silicon) herunterladen
Windows	Für Windows herunterladen
Linux	Download für Linux

> **Note**
>
> NoSQL Workbench beinhaltet DynamoDB Local als Teil des Installationsvorgangs. Für die lokale Ausführung von DynamoDB ist Java Runtime Environment (JRE) Version 17.x oder neuer erforderlich.

> **Note**
>
> NoSQL Workbench unterstützt Ubuntu 12.04, Fedora 21 und Debian 8 oder neuere Versionen dieser Linux-Distributionen.
> Für Ubuntu-Installationen sind zwei grundlegende Softwarekomponenten erforderlich: und. `libfuse2` `curl`
> Ab Ubuntu 22.04 ist libfuse2 nicht mehr standardmäßig installiert. Um dieses Problem zu lösen, führen Sie `sudo add-apt-repository universe && sudo apt install libfuse2` den Befehl to install für die neueste Ubuntu-Version aus.
> Führen `sudo apt update && sudo apt upgrade && sudo apt install curl` Sie für cURL aus.

Erstellen von Datenmodellen mit NoSQL Workbench

Sie können das Datenmodellierungstool in NoSQL Workbench für Amazon DynamoDB verwenden, um neue Datenmodelle zu erstellen oder Modelle auf der Grundlage vorhandener Datenmodelle zu entwerfen, die Ihren Anwendungsdatenzugriffsmustern entsprechen. Der Data Modeler enthält ein paar Beispiel-Datenmodelle für den Einstieg.

Themen

- Erstellen eines neuen Datenmodells
- Importieren eines vorhandenen Datenmodells
- Bearbeiten eines vorhandenen Datenmodells
- Hinzufügen von Beispieldaten zu einem Datenmodell
- Zugriffsmuster hinzufügen und validieren
- Importieren von Beispieldaten aus einer CSV-Datei
- Facets
- Anzeigen aller Tabellen in einem Datenmodell mithilfe der aggregierten Ansicht
- Exportieren eines Datenmodells
- Übergeben eines Datenmodells an DynamoDB

Erstellen eines neuen Datenmodells

Befolgen Sie diese Schritte, um ein neues Datenmodell in Amazon DynamoDB mit NoSQL Workbench zu erstellen.

Erstellen eines neuen Datenmodells

1. Öffnen Sie NoSQL Workbench und wählen Sie auf dem Hauptbildschirm die Option Modell manuell erstellen aus.

   Eine neue Seite mit einer leeren Konfiguration für Ihre erste Tabelle wird geöffnet. NoSQL Workbench erstellt alle neuen Datenmodelle mit einem Standardnamen (d. h. untitled-2) und fügt sie dem Projektordner Drafts hinzu.
2. Geben Sie auf dem Bildschirm für die Tabellenkonfiguration Folgendes an:

   - Tabellenname – Einen eindeutigen Namen für die Tabelle eingeben.
   - Partitionsschlüssel — Geben Sie einen Namen für den Partitionsschlüssel ein und geben Sie dessen Typ an. Optional können Sie außerdem ein detaillierteres Datentypformat für die Generierung von Beispieldaten auswählen.
   - Wenn Sie einen Sortierschlüssel hinzufügen möchten, geben Sie den Namen und den Typ des Sortierschlüssels an. Optional können Sie ein detaillierteres Datentypformat für die Generierung von Beispieldaten auswählen.

   > **Note**
   >
   > Weitere Informationen zum Entwerfen von Primärschlüsseln, zum effektiven Entwerfen und Verwenden von Partitionsschlüsseln und zur Verwendung von Sortierschlüsseln finden Sie in folgenden Abschnitten:
   >
   > - Primärschlüssel
   > - Bewährte Methoden für das Entwerfen und effektive Verwenden von Partitionsschlüsseln in DynamoDB
   > - Bewährte Methoden für die Verwendung von Sortierschlüsseln zur Organisation von Daten in DynamoDB
3. Sie können weitere Attribute hinzufügen, um Ihr Modell und Ihre Zugriffsmuster genauer zu validieren. Um weitere Attribute hinzuzufügen:

- Wählen Sie Attribut hinzufügen aus.
- Geben Sie den Namen und den Typ des Attributs an.
- Optional können Sie ein detaillierteres Datentypformat für die Generierung von Beispieldaten auswählen.

4. Falls Sie einen globalen sekundären Index hinzufügen möchten, wählen Sie Add global secondary index (Globalen sekundären Index hinzufügen). Geben Sie Global secondary index name (Name des globalen sekundären Indexes), das Attribut Partition key (Partitionsschlüssel) und Projection type (Projektionstyp) an.

   Weitere Informationen über das Arbeiten mit globalen sekundären Indizes in DynamoDB finden Sie unter Globale sekundäre Indizes.

5. Fügen Sie optional eine Facette hinzu. Eine Facette ist ein virtuelles Konstrukt in NoSQL Workbench. Es ist kein funktionales Konstrukt in DynamoDB. Facetten in NoSQL Workbench helfen Ihnen, die unterschiedlichen Datenzugriffsmuster einer Anwendung für DynamoDB mit nur einer Teilmenge der Daten in einer Tabelle zu visualisieren.

   > **Note**
   >
   > Wir empfehlen Ihnen, diese Option anstelle von Zugriffsmuster hinzufügen und validieren Facets zu verwenden, um zu visualisieren, wie Ihre Anwendung auf Daten in DynamoDB zugreift. Zugriffsmuster spiegeln Ihre tatsächlichen Datenbankinteraktionen wider und helfen Ihnen, das richtige Datenmodell für Ihren Anwendungsfall zu erstellen, während Facetten nicht funktionale Visualisierungen sind.

   Klicken Sie auf Add Facet (Facette hinzufügen). Machen Sie folgende Angaben:

   - den Facet name (Name der Facette),
   - Ein Alias für den Partitionsschlüssel, der die Unterscheidung dieser Facettenansicht erleichtert.
   - Ein Alias für den Sortierschlüssel, wenn Sie einen Sortierschlüssel für die Tabelle angegeben haben.
   - Wählen Sie die Attribute aus, die Teil dieser Facette sind.

   Wenn Sie weitere Facetten hinzufügen möchten, wiederholen Sie diesen Schritt.

6. Klicken Sie abschließend auf die Schaltfläche Speichern, um die Tabelle zu erstellen.

7. Wenn Sie weitere Tabellen oder globale Sekundärindizes benötigen, klicken Sie auf das +-Symbol über der Tabelle, die Sie gerade erstellt haben.

Importieren eines vorhandenen Datenmodells

Sie können NoSQL Workbench für Amazon DynamoDB verwenden, um ein Datenmodell zu erstellen, indem Sie ein vorhandenes Modell importieren und bearbeiten. Sie können Datenmodelle entweder im NoSQL-Workbench-Modellformat oder in AWS CloudFormation JSON-Vorlagenformat importieren.

Importieren eines Datenmodells

1. Öffnen Sie NoSQL Workbench und wählen Sie auf dem Hauptbildschirm Modell importieren aus. Sie können entweder ein NoSQL Workbench-Modellformat oder eine CloudFormation JSON-Vorlage importieren.
2. Sie können das importierte Modell zu einem vorhandenen Projekt hinzufügen oder ein neues Projekt erstellen. Wenn Sie kein anderes Projekt auswählen, wird Ihr Modell dem Projektordner für Entwürfe hinzugefügt.
3. Klicken Sie auf Desktop durchsuchen und wählen Sie ein Modell aus, das Sie von Ihrem Computer importieren möchten.
4. Once your model is imported, you can add access patterns, edit tables/indexes, and modify the data locally.

Bearbeiten eines vorhandenen Datenmodells

So bearbeiten Sie ein vorhandenes Modell

1. Um Änderungen an einem vorhandenen Modell vorzunehmen, öffnen Sie das Modell auf der Modeler-Seite.
2. Im Bereich zur Ressourcenauswahl sehen Sie die Liste der Tabellen und globalen sekundären Indizes.
3. Um eine Tabelle oder einen globalen sekundären Index zu bearbeiten, klicken Sie zunächst im Ressourcenauswahlfenster auf den entsprechenden Namen und verwenden Sie dann die Aktionssymbole oben. Verfügbare Aktionen sind Löschen, Duplizieren und Bearbeiten.
4. Wenn Sie Modelldetails bearbeiten möchten, klicken Sie auf das Dreipunktsymbol neben dem Modellnamen.

5. Von dort aus können Sie auf Modelldetails bearbeiten klicken und die Informationen entsprechend bearbeiten.
6. Sie können das Modell auch über dasselbe Menü duplizieren, verschieben, löschen und exportieren.
7. Um zu einem anderen Modell zu wechseln, können Sie entweder erneut den Hauptbildschirm aufrufen oder das Drop-down-Menü für die Modellauswahl verwenden.

Hinzufügen von Beispieldaten zu einem Datenmodell

So generieren Sie automatisch Beispieldaten

1. Öffnen Sie NoSQL Workbench und klicken Sie auf dem Hauptbildschirm auf den Namen des Modells, dem Sie Beispieldaten hinzufügen möchten.
2. Klicken Sie in der Hauptinhaltssymbolleiste auf die zusätzlichen Aktionen (Dreipunktsymbol) und wählen Sie Beispieldaten hinzufügen aus.
3. Geben Sie die Anzahl der Beispieldaten ein, die Sie generieren möchten, und wählen Sie dann Bestätigen aus.
4. Mit der automatischen Generierung von Beispieldaten können Sie zwischen 1 und 5000 Datenzeilen zur sofortigen Verwendung generieren. Sie können einen detaillierten Beispieldatentyp angeben, um realistische Daten zu erstellen, die Ihren Entwurfs- und Testanforderungen entsprechen. Um die Fähigkeit zur Generierung realistischer Daten nutzen zu können, müssen Sie das Format des Beispieldatentyps für Ihre Attribute im Datenmodellierer angeben. Informationen Erstellen eines neuen Datenmodells zur Angabe von Beispielformaten für Datentypen finden Sie unter.

So fügen Sie ein Element nach dem anderen Beispieldaten hinzu

1. Öffnen Sie das Modell, das Sie bearbeiten möchten, und wählen Sie dann die Tabelle aus, zu der Sie Beispieldaten hinzufügen möchten. Klicken Sie in der Hauptinhaltssymbolleiste auf die zusätzlichen Aktionen (Dreipunktsymbol) und wählen Sie Daten bearbeiten aus.
2. Jetzt können Sie Zeilen hinzufügen, löschen und bearbeiten. Nachdem Sie die erforderlichen Änderungen vorgenommen haben, klicken Sie auf die Schaltfläche Speichern.

Zugriffsmuster hinzufügen und validieren

Sie können NoSQL Workbench for Amazon DynamoDB verwenden, um Zugriffsmuster zu erstellen, zu speichern und zu validieren.

> **Note**
>
> Weitere Informationen zur Identifizierung der richtigen Zugriffsmuster finden Sie unter Identifizieren Sie Ihre Datenzugriffsmuster.

Um ein Zugriffsmuster zu erstellen

1. Öffnen Sie NoSQL Workbench und klicken Sie auf dem Hauptbildschirm auf den Namen des Modells, dem Sie Zugriffsmuster hinzufügen möchten.
2. Wählen Sie auf der linken Seite die Registerkarte Zugriffsmuster und klicken Sie auf das Pluszeichen.
3. Geben Sie auf dem nächsten Bildschirm einen Namen, eine optionale Beschreibung, den Typ des Zugriffsmusters und die Tabelle oder den globalen sekundären Index ein, anhand dessen das Zugriffsmuster getestet werden soll.

   > **Note**
   >
   > NoSQL Workbench unterstützt derzeit die folgenden Operationen für Zugriffsmuster:`Scan`,,,`Query`, `GetItem``PutItem`,`UpdateItem`. `DeleteItem` Amazon DynamoDB unterstützt eine breitere Liste von Vorgängen.

4. Nachdem Sie ein Zugriffsmuster erstellt haben, können Sie zur Registerkarte Validieren wechseln, um zu überprüfen, ob Ihr Datenmodell darauf ausgelegt ist, die erwarteten Ergebnisse für das Zugriffsmuster zurückzugeben. Einzelheiten Hinzufügen von Beispieldaten zu einem Datenmodell zur automatischen Generierung von Beispieldaten für Ihre Tabellen finden Sie unter. Verschiedene Arten von Zugriffsmustern unterstützen unterschiedliche Eingabeparameter.

   > **Note**
   >
   > Um Zugriffsmuster zu überprüfen, startet NoSQL Workbench eine separate lokale DynamoDB-Datenbank am Port `8001` (standardmäßig) mit Tabellen und Indizes, die im Speicher gespeichert sind.

- NoSQL Workbench fügt automatisch die Beispieldaten aus Ihrem Modell zu den temporären Tabellen hinzu.
- Wenn Sie die Beispieldaten oder das Datenmodell selbst bearbeiten, aktualisiert NoSQL Workbench die temporären Tabellen.
- Diese temporäre Datenbank wird gelöscht, wenn Sie die Anwendung schließen.

Um Ihre Zugriffsmuster zu bearbeiten

1. Öffnen Sie NoSQL Workbench und klicken Sie auf dem Hauptbildschirm auf den Namen des Modells, für das Sie Zugriffsmuster bearbeiten möchten.
2. Wählen Sie auf der linken Seite die Registerkarte Zugriffsmuster aus.
3. Um ein Zugriffsmuster zu bearbeiten, wählen Sie es aus der Liste auf der linken Seite aus.
4. Klicken Sie in der oberen Leiste auf die Aktionsschaltfläche Bearbeiten.

Importieren von Beispieldaten aus einer CSV-Datei

Wenn Sie über bereits vorhandene Beispieldaten in einer CSV-Datei verfügen, können Sie diese in NoSQL-Workbench importieren. Auf diese Weise können Sie Ihr Modell schnell mit Beispieldaten füllen, ohne es zeilenweise eingeben zu müssen.

Die Spaltennamen in der CSV-Datei müssen mit den Attributnamen in Ihrem Datenmodell übereinstimmen, müssen jedoch nicht in der gleichen Reihenfolge sein. Wenn Ihr Datenmodell beispielsweise Attribute hat, die als `LoginAlias`, `FirstName`, und `LastName`benannt sind, könnten Ihre CSV-Spalten `LastName`, `FirstName`, und `LoginAlias` sein.

Sie können bis zu 150 Zeilen gleichzeitig aus einer CSV-Datei importieren.

So importieren Sie Daten aus einer CSV-Datei in NoSQL-Workbench

1. Um CSV-Daten in eine Tabelle zu importieren, klicken Sie zuerst im Ressourcenbereich auf den Tabellennamen und dann in der Hauptinhaltssymbolleiste auf die zusätzlichen Aktionen (Dreipunktsymbol).
2. Wählen Sie Beispieldaten importieren aus.
3. Wählen Sie Ihre CSV-Datei aus und klicken Sie auf Open (Öffnen). Die CSV-Daten werden an Ihre Tabelle angehängt.

Facets

In NoSQL Workbench können Sie mit Facets eine Teilmenge der Daten in einer Tabelle anzeigen, ohne Datensätze sehen zu müssen, die die Einschränkungen der Facette nicht erfüllen. Facets gelten als visuelles Datenmodellierungswerkzeug und existieren nicht als brauchbares Konstrukt in DynamoDB, da sie eine reine Hilfe zur Modellierung von Zugriffsmustern darstellen.

> **Note**
>
> Wir empfehlen Ihnen, diese Option anstelle von Zugriffsmuster hinzufügen und validieren Facets zu verwenden, um zu visualisieren, wie Ihre Anwendung auf Daten in DynamoDB zugreift. Zugriffsmuster spiegeln Ihre tatsächlichen Datenbankinteraktionen wider und helfen Ihnen, das richtige Datenmodell für Ihren Anwendungsfall zu erstellen, während Facetten nicht funktionale Visualisierungen sind.

Um eine Facette zu erstellen

1. Wählen Sie im Ressourcenauswahlfenster eine Tabelle aus, die Sie bearbeiten möchten
2. Klicken Sie in der oberen Leiste auf das Aktionssymbol Bearbeiten.
3. Scrollen Sie nach unten zum Abschnitt Facettenfilter.
4. Klicken Sie auf Add Facet (Facette hinzufügen). Machen Sie folgende Angaben:

   - den Facet name (Name der Facette),
   - Ein Alias für den Partitionsschlüssel, der die Unterscheidung dieser Facettenansicht erleichtert.
   - Ein Alias für den Sortierschlüssel, wenn Sie einen Sortierschlüssel für die Tabelle angegeben haben.
   - Wählen Sie die Attribute aus, die Teil dieser Facette sind.

   Wenn Sie weitere Facetten hinzufügen möchten, wiederholen Sie diesen Schritt.

Anzeigen aller Tabellen in einem Datenmodell mithilfe der aggregierten Ansicht

Die Aggregatansicht in NoSQL Workbench für Amazon DynamoDB ermöglicht es Ihnen, alle Tabellen und Indizes in einem Datenmodell zu visualisieren. Für jede Tabelle werden die folgenden Informationen angezeigt:

- Tabellenspaltennamen
- Beispieldaten
- Alle globalen sekundären Indizes, die der Tabelle zugeordnet sind. NoSQL Workbench zeigt die folgenden Informationen für jeden Index an:
  - Indexspaltennamen
  - Beispieldaten

Anzeigen aller Tabelleninformationen

1. Öffnen Sie NoSQL Workbench und klicken Sie auf dem Hauptbildschirm auf den Namen des Modells, das Sie öffnen möchten.
2. Klicken Sie in der oberen Leiste auf Aggregatansicht. Sie sehen die Daten aller Tabellen und Indizes auf demselben Bildschirm.

Um die Gesamtansicht als Bilder zu exportieren

1. Wenn die Gesamtansicht ausgewählt ist, klicken Sie auf das Dreipunktsymbol und wählen Sie Gesamtansicht exportieren
2. Ein Archiv mit PNG-Bildern aller Tabellen und Indizes wird zum Herunterladen angeboten.

Exportieren eines Datenmodells

Nach dem Erstellen eines Datenmodells mit NoSQL Workbench für Amazon DynamoDB können Sie das Modell entweder im NoSQL-Workbench-Modellformat oder AWS CloudFormation JSON-Vorlagenformat speichern und exportieren.

Exportieren eines Datenmodells

1. Öffnen Sie NoSQL Workbench und klicken Sie auf dem Hauptbildschirm auf den Namen des Modells, das Sie bearbeiten möchten.
2. Klicken Sie auf das Dreipunktsymbol neben dem Namen des Datenmodells und wählen Sie Modell exportieren aus.
3. Wählen Sie aus, ob Sie Ihr Datenmodell im NoSQL Workbench-Modellformat oder im CloudFormation JSON-Vorlagenformat exportieren möchten.

   - Wählen Sie das NoSQL Workbench-Modellformat, wenn Sie Ihr Modell mit anderen Teammitgliedern teilen möchten, die NoSQL Workbench verwenden, oder es später in NoSQL Workbench importieren möchten.
   - Wählen Sie das CloudFormation JSON-Vorlagenformat, wenn Sie Ihr Modell direkt in Ihren Workflow implementieren AWS oder in Ihren Workflow integrieren möchten. infrastructure-as-code

Übergeben eines Datenmodells an DynamoDB

Wenn Sie mit Ihrem Datenmodell zufrieden sind, können Sie das Modell an Amazon DynamoDB übergeben.

> **Note**
>
> - Diese Aktion erstellt serverseitige Ressourcen AWS für die Tabellen und globalen Sekundärindizes, die im Datenmodell dargestellt werden.
> - NoSQL Workbench erstellt standardmäßig Tabellen und Indizes mit On-Demand-Kapazität.

Übergeben des Datenmodells an DynamoDB

1. Öffnen Sie NoSQL Workbench und klicken Sie auf dem Hauptbildschirm auf den Namen des Modells, das Sie übernehmen möchten.
2. Klicken Sie in der oberen Leiste auf Commit.
3. Wählen Sie eine bestehende Verbindung aus oder erstellen Sie eine neue Verbindung, indem Sie auf die Schaltfläche Neue Verbindung hinzufügen klicken.

   - Zum Hinzufgen einer neuen Verbindung geben Sie die folgenden Informationen an.

- Account Alias (Konto-Alias)
- AWS Region
- Zugriffsschlüssel-ID
- Geheimer Zugriffsschlüssel

Weitere Informationen darüber, wie Sie die Zugriffsschlüssel erhalten, finden Sie unter Einen AWS Zugriffsschlüssel abrufen.

- Optional können Sie Folgendes angeben:
  - Session token (Sitzungstoken)
  - IAM role ARN (IAM-Rollen-ARN)

4. Wenn Sie DynamoDB lieber lokal verwenden möchten:
   1. Wählen Sie die Registerkarte Lokale Verbindung.
   2. Klicken Sie auf die Schaltfläche Neue Verbindung hinzufügen.
   3. Geben Sie den Connection name (Verbindungsnamen) und den Port an.

> **Note**
>
> Um DynamoDB lokal zu verwenden, müssen Sie es mithilfe des lokalen DynamoDB-Schalters unten links auf dem NoSQL Workbench-Bildschirm aktivieren.

5. Klicken Sie auf Commit.

Erkunden von Datasets und Erstellen von Vorgängen mit NoSQL Workbench

NoSQL Workbench für Amazon DynamoDB stellt eine umfassende Grafikbenutzeroberfläche für die Entwicklung und den Test von Abfragen bereit. Sie können den Operation Builder in NoSQL Workbench verwenden, um Live-Datensätze anzuzeigen, zu erkunden und abzufragen. Der strukturierte Operation Builder unterstützt Projektionsausdrücke und Bedingungsausdrücke und generiert Beispielcode in mehreren Sprachen. Sie können Tabellen von einem Amazon-DynamoDB-Konto direkt in ein anderes in verschiedenen Regionen klonen. Sie können Tabellen auch direkt zwischen einer lokalen Version von DynamoDB und einem Amazon-DynamoDB-Konto klonen, um das Schlüsselschema Ihrer Tabelle (und optional das Schema und Elemente des GSI) schneller

zwischen Ihren Entwicklungsumgebungen zu kopieren. Im Operation Builder lassen sich bis zu 50 DynamoDB-Datenoperationen speichern.

Themen

- Herstellen einer Verbindung zu Live-Datensätzen
- Erstellen komplexer Operationen
- Klonen von Tabellen mit NoSQL Workbench
- Exportieren von Daten in eine CSV-Datei

Herstellen einer Verbindung zu Live-Datensätzen

Um mit NoSQL Workbench eine Verbindung zu Ihren Amazon DynamoDB-Tabellen herzustellen, müssen Sie zunächst eine Verbindung zu Ihrem Konto herstellen. AWS

Hinzufügen einer Verbindung zu Ihrer Datenbank

1. Wählen Sie in NoSQL Workbench im Navigationsbereich auf der linken Seite das Symbol Operation Builder.
2. Wählen Sie Add connection (Verbindung hinzufügen).
3. Geben Sie folgende Informationen an:

   - Verbindungsname
   - AWS Region
   - Zugriffsschlüssel-ID
   - Geheimer Zugriffsschlüssel

   Weitere Informationen darüber, wie Sie die Zugriffsschlüssel erhalten, finden Sie unter Einen Zugriffsschlüssel abrufen. AWS

   Optional können Sie Folgendes angeben:

   - Session token (Sitzungstoken)
   - IAM role ARN (IAM-Rollen-ARN)
4. Wählen Sie Connect (Verbinden) aus.

Wenn Sie sich nicht für ein Konto mit kostenlosem Kontingent registrieren möchten und lieber die lokale Version von DynamoDB (herunterladbare Version) verwenden möchten:

a. Wählen Sie die Registerkarte Local (Lokal) auf dem Verbindungsbildschirm aus.
b. Geben Sie folgende Informationen an:
   - Verbindungsname
   - Port
c. Wählen Sie die Schaltfläche connect (Verbinden).

> **Note**
>
> Um eine Verbindung zur lokalen Version von DynamoDB herzustellen, starten Sie diese entweder manuell über Ihr Terminal (siehe Bereitstellen von DynamoDB Locally lokal auf Ihrem Computer) oder direkt mit dem Schalter für die lokale Version von DDB im NoSQL-Workbench-Navigationsmenü. Stellen Sie sicher, dass der Verbindungsport mit Ihrem lokalen DynamoDB-Port identisch ist.

5. Wählen Sie für die erstellte Verbindung die Option Open (Öffnen).

Nach dem Herstellen der Verbindung mit Ihrer DynamoDB-Datenbank erscheint die Liste der verfügbaren Tabellen im linken Bereich. Wählen Sie eine der Tabellen aus, um ein Beispiel der in der Tabelle gespeicherten Daten zurückzugeben.

Sie können jetzt Abfragen an der ausgewählten Tabelle ausführen.

Informationen zum Ausführen von Abfragen für eine Tabelle finden Sie im nächsten Abschnitt über das Erstellen von Operationen unter Erstellen komplexer Operationen.

Erstellen komplexer Operationen

Der Operation Builder in NoSQL Workbench für Amazon DynamoDB stellt eine visuelle Schnittstelle bereit, auf der Sie komplexe Datenebenen-Operationen durchführen können. Er beinhaltet Unterstützung für Projektionsausdrücke und Bedingungsausdrücke. Nachdem Sie einen Vorgang erstellt haben, können Sie ihn für die spätere Verwendung speichern (bis zu 50 Vorgänge können gespeichert werden). Anschließend können Sie im Menü Gespeicherte Vorgänge eine Liste der häufig verwendeten Datenebenenvorgänge durchsuchen und diese verwenden, um automatisch

einen neuen Vorgang aufzufüllen und zu erstellen. Sie können auch Beispielcode für diese Vorgänge in mehreren Sprachen erstellen.

NoSQL Workbench unterstützt das Erstellen von PartiQL für DynamoDB-Anweisungen, sodass Sie mithilfe einer SQL-kompatiblen Abfragesprache mit DynamoDB interagieren können. NoSQL Workbench unterstützt auch das Erstellen von DynamoDB-CRUD-API-Operationen.

Um NoSQL Workbench zum Erstellen von Operationen zu verwenden, wählen Sie im Navigationsbereich auf der linken Seite das Symbol Operation Builder aus.

Themen

- Erstellen von PartiQL-Anweisungen
- Erstellen von API-Operationen

Erstellen von PartiQL-Anweisungen

Wenn Sie mit NoSQL Workbench PartiQL-für-DynamoDB-Anweisungen erstellen möchten, wählen Sie oben in NoSQL Workbench PartiQL-Editor aus.

Sie können die folgenden PartiQL-Anweisungstypen in Operation Builder erstellen:

Themen

- Singleton-Anweisungen
- Transaktionen
- Batch

Singleton-Anweisungen

Zum Ausführen oder Generieren von Code für eine PartiQL-Anweisung gehen Sie folgendermaßen vor.

1. Wählen Sie oben im Fenster den PartiQL-Editor aus.
2. Geben Sie eine gültige PartiQL-Anweisung ein.
3. Wenn Ihre Anweisung Parameter verwendet:
    a. Klicken Sie auf Optional request parameter (Optionale Anfrageparameter).
    b. Klicken Sie auf Add new parameters (Fügen Sie neue Parameter hinzu).

c. Geben Sie den Attributtyp und -wert ein.

d. Wenn Sie zusätzliche Parameter hinzufügen möchten, wiederholen Sie die Schritte b und c.

4. Falls Sie Code generieren möchten, wählen Sie Generate code (Code generieren).

   Wählen Sie in den angezeigten Tabs Ihre gewünschte Sprache aus. Sie können diesen Code jetzt kopieren und in Ihrer Anwendung verwenden.

5. Falls die Operation sofort ausgeführt werden soll, wählen Sie Run (Ausführen).

6. Wenn Sie diese Operation für die spätere Verwendung speichern möchten, wählen Sie Save operation (Operation speichern) aus. Geben Sie dann einen Namen für Ihre Operation ein und wählen Sie Save (Speichern) aus.

Transaktionen

Zum Ausführen oder Generieren von Code für eine PartiQL-Transaktion gehen Sie folgendermaßen vor.

1. Wählen Sie Parti QLTransaction aus der Dropdownliste Weitere Operationen aus.

2. Wählen Sie Add new statement (Neue Anweisung hinzufügen) aus.

3. Geben Sie eine gültige PartiQL-Anweisung ein.

   > **Note**
   >
   > Lese- und Schreibvorgänge werden in derselben PartiQL-Transaktionsanforderung nicht unterstützt. Eine SELECT-Anweisung darf sich nicht in derselben Anforderung mit INSERT-, UPDATE- und DELETE-Anweisungen befinden. Weitere Details finden Sie in unter Ausführen von Transaktionen mit PartiQL für DynamoDB

4. Wenn Ihre Anweisung Parameter verwendet

   a. Wählen Sie Optional request parameters (Optionale Anforderungsparameter) aus.

   b. Klicken Sie auf Add new parameters (Fügen Sie neue Parameter hinzu).

   c. Geben Sie den Attributtyp und -wert ein.

   d. Wenn Sie zusätzliche Parameter hinzufügen möchten, wiederholen Sie die Schritte b und c.

5. Wenn Sie weitere Anweisungen hinzufügen möchten, wiederholen Sie die Schritte 2 bis 4.

6. Falls Sie Code generieren möchten, wählen Sie Generate code (Code generieren) aus.

Wählen Sie in den angezeigten Tabs Ihre gewünschte Sprache aus. Sie können diesen Code jetzt kopieren und in Ihrer Anwendung verwenden.

7. Falls die Operation sofort ausgeführt werden soll, wählen Sie Run (Ausführen).
8. Wenn Sie diese Operation für die spätere Verwendung speichern möchten, wählen Sie Save operation (Operation speichern) aus. Geben Sie dann einen Namen für Ihre Operation ein und wählen Sie Save (Speichern) aus.

Batch

Zum Ausführen oder Generieren von Code für einen PartiQL-Batch gehen Sie folgendermaßen vor.

1. Wählen Sie Parti QLBatch aus dem Drop-down-Menü Weitere Operationen aus.
2. Wählen Sie Add new statement (Neue Anweisung hinzufügen) aus.
3. Geben Sie eine gültige PartiQL-Anweisung ein.

> **Note**
>
> Lese- und Schreibvorgänge werden in derselben PartiQL-Batchanforderung nicht unterstützt, was bedeutet, dass sich eine SELECT-Anweisung nicht in derselben Anforderung mit INSERT-, UPDATE- und DELETE-Anweisungen befinden kann. Schreibvorgänge auf dasselbe Element sind nicht zulässig. Wie bei der BatchGetItem Operation werden nur Singleton-Lesevorgänge unterstützt. Scan- und Abfragevorgänge werden nicht unterstützt. Weitere Details finden Sie in unter Ausführen von Batchvorgängen mit PartiQL für DynamoDB.

4. Wenn Ihre Anweisung Parameter verwendet:
   a. Klicken Sie auf Optional request parameter (Optionale Anfrageparameter).
   b. Klicken Sie auf Add new parameters (Fügen Sie neue Parameter hinzu).
   c. Geben Sie den Attributtyp und -wert ein.
   d. Wenn Sie zusätzliche Parameter hinzufügen möchten, wiederholen Sie die Schritte b und c.
5. Wenn Sie weitere Anweisungen hinzufügen möchten, wiederholen Sie die Schritte 2 bis 4.
6. Falls Sie Code generieren möchten, wählen Sie Generate code (Code generieren) aus.

   Wählen Sie in den angezeigten Tabs Ihre gewünschte Sprache aus. Sie können diesen Code jetzt kopieren und in Ihrer Anwendung verwenden.

7. Falls die Operation sofort ausgeführt werden soll, wählen Sie Run (Ausführen).
8. Wenn Sie diese Operation für die spätere Verwendung speichern möchten, wählen Sie Save operation (Operation speichern) aus. Geben Sie dann einen Namen für Ihre Operation ein und wählen Sie Save (Speichern) aus.

Erstellen von API-Operationen

Um NoSQL Workbench zum Erstellen von DynamoDB CRUD zu verwenden APIs, wählen Sie Operation Builder auf der linken Seite der NoSQL Workbench-Benutzeroberfläche aus.

Wählen Sie dann Öffnen und anschließend eine Verbindung aus.

Sie können die folgenden Aktionen in Operation Builder durchführen:

- Tabelle löschen
- Tabelle erstellen
- Tabelle aktualisieren

- PutItem
- Element aktualisieren
- Element löschen
- Query
- Scan
- TransactGetItems
- TransactWriteItems

Tabelle löschen

Gehen Sie folgendermaßen vor, um die Operation `Delete Table` auszuführen.

1. Suchen Sie im Abschnitt Tabellen nach der zu löschenden Tabelle.
2. Wählen Sie im Dreipunktmenü die Option Tabelle löschen aus.
3. Bestätigen Sie, dass Sie die Tabelle löschen möchten, indem Sie den Tabellennamen eingeben.
4. Wählen Sie Löschen aus.

Weitere Informationen zu dieser Operation finden Sie unter Tabelle löschen in der Amazon-DynamoDB-API-Referenz.

Löschen eines GSI

Gehen Sie folgendermaßen vor, um die Operation `Delete GSI` auszuführen.

1. Suchen Sie im Abschnitt Tabellen nach dem zu löschenden GSI einer Tabelle.
2. Wählen Sie im Dreipunktmenü die Option GSI löschen aus.
3. Bestätigen Sie, dass Sie den GSI löschen möchten, indem Sie den Namen des GSI eingeben.
4. Wählen Sie Löschen aus.

Weitere Informationen zu dieser Operation finden Sie unter Tabelle löschen in der Amazon-DynamoDB-API-Referenz.

Tabelle erstellen

Gehen Sie folgendermaßen vor, um die Operation `Create Table` auszuführen.

1. Klicken Sie neben dem Abschnitt Tabellen auf das Pluszeichen.
2. Geben Sie den gewünschten Tabellennamen ein.
3. Erstellen Sie einen Partitionsschlüssel.
4. Optional: Erstellen Sie einen Sortierschlüssel.
5. Wenn Sie die Kapazitätseinstellungen anpassen möchten, deaktivieren Sie das Kontrollkästchen neben Standardkapazitätseinstellungen verwenden.
   - Sie können jetzt entweder Bereitgestellt oder On-Demand-Kapazität auswählen.

     Wenn Bereitgestellt ausgewählt ist, können Sie minimale und maximale Lese- und Schreibkapazitätseinheiten festlegen. Sie können außerdem Auto Scaling aktivieren oder deaktivieren.
   - Wenn die Tabelle derzeit auf „On-Demand“ eingestellt ist, können Sie keinen bereitgestellten Durchsatz angeben.
   - Wenn Sie von On-Demand-Durchsatz zu Bereitgestelltem Durchsatz wechseln, wird Autoscaling automatisch auf alle GSIs mit folgenden Werten angewendet: min: 1, max.: 10; Ziel: 70%
6. Wählen Sie Überspringen GSIs und erstellen aus, um diese Tabelle ohne GSI zu erstellen. Optional können Sie Weiter auswählen, um einen GSI mit dieser neuen Tabelle zu erstellen.

Weitere Informationen zu dieser Operation finden Sie in der Amazon-DynamoDB-API-Referenz unter Tabelle erstellen.

Erstellen eines GSI

Gehen Sie folgendermaßen vor, um die Operation `Create GSI` auszuführen.

1. Suchen Sie eine Tabelle, der Sie einen GSI hinzufügen möchten.
2. Wählen Sie im Dreipunktmenü die Option GSI erstellen aus.
3. Benennen Sie den GSI unter Index-Name.
4. Erstellen Sie einen Partitionsschlüssel.
5. Optional: Erstellen Sie einen Sortierschlüssel.
6. Wählen Sie aus der Dropdown-Liste einen Projektionstyp aus.
7. Wählen Sie GSI erstellen.

Weitere Informationen zu dieser Operation finden Sie unter Tabelle erstellen in der Amazon-DynamoDB-API-Referenz.

Tabelle aktualisieren

Gehen Sie folgendermaßen vor, um die Kapazitätseinstellungen für eine Tabelle mit der Operation `Update Table` zu aktualisieren.

1. Suchen Sie die Tabelle, deren Kapazitätseinstellungen aktualisiert werden sollen.
2. Wählen Sie im Dreipunktmenü die Option Kapazitätseinstellungen aktualisieren aus.
3. Sie können jetzt entweder Bereitgestellt oder On-Demand-Kapazität auswählen.

   Wenn Bereitgestellt ausgewählt ist, können Sie minimale und maximale Lese- und Schreibkapazitätseinheiten festlegen. Sie können außerdem Auto Scaling aktivieren oder deaktivieren.
4. Wählen Sie Aktualisieren.

Weitere Informationen zu dieser Operation finden Sie in der Amazon-DynamoDB-API-Referenz unter Tabelle aktualisieren.

Aktualisieren eines GSI

Gehen Sie folgendermaßen vor, um die Kapazitätseinstellungen für einen GSI mit der Operation `Update Table` zu aktualisieren.

> **Note**
>
> Standardmäßig übernehmen globale sekundäre Indizes die Kapazitätseinstellungen der Basistabelle. Globale sekundäre Indizes können nur dann einen anderen Kapazitätsmodus haben, wenn sich die Basistabelle im Modus „Bereitgestellte Kapazität“ befindet. Wenn Sie einen globalen sekundären Index für eine Tabelle mit dem Modus bereitgestellter Kapazität erstellen, müssen Sie Lese- und Schreibkapazitätseinheiten für den erwarteten Workload dieses Indexes angeben. Weitere Informationen finden Sie unter Überlegungen im Hinblick auf die bereitgestellte Durchsatzkapazität für globale sekundäre Indizes.

1. Suchen Sie den GSI, dessen Kapazitätseinstellungen aktualisiert werden sollen.
2. Wählen Sie im Dreipunktmenü die Option Kapazitätseinstellungen aktualisieren aus.
3. Sie können jetzt entweder Bereitgestellt oder On-Demand-Kapazität auswählen.

   Wenn Bereitgestellt ausgewählt ist, können Sie minimale und maximale Lese- und Schreibkapazitätseinheiten festlegen. Sie können außerdem Auto Scaling aktivieren oder deaktivieren.
4. Wählen Sie Aktualisieren.

Weitere Informationen zu dieser Operation finden Sie in der Amazon-DynamoDB-API-Referenz unter Tabelle aktualisieren.

Element einfügen

Sie erstellen ein Element mithilfe der Operation `Put Item`. Zum Ausführen oder Generieren von Code für eine `Put Item`-Operation gehen Sie folgendermaßen vor.

1. Suchen Sie die Tabelle, in der Sie ein Element erstellen möchten.
2. Wählen Sie im Dropdown-Menü Aktionen die Option Element erstellen aus.
3. Geben Sie den Partitionsschlüsselwert ein.
4. Geben Sie den Sortierschlüsselwert ein, falls vorhanden.

5. Gehen Sie folgendermaßen vor, falls Sie Nicht-Schlüssel-Attribute hinzufügen möchten:

   a. Wählen Sie + Weitere Attribute hinzufügen aus.

   b. Geben Sie Attribute name (Attributname), Type (Typ) und Value (Wert) an.

6. Falls ein Bedingungsausdruck erfüllt sein muss, damit die `Put Item`-Operation erfolgreich ist, gehen Sie folgendermaßen vor:

   a. Wählen Sie Condition (Bedingung).

   b. Geben Sie den Attributnamen, Vergleichsoperator, Attributtyp und Attributwert an.

   c. Falls weitere Bedingungen erforderlich sind, wählen Sie erneut Condition (Bedingung).

   Weitere Informationen finden Sie unter CLI-Beispiel für DynamoDB-Bedingungsausdrücke.

7. Falls Sie Code generieren möchten, wählen Sie Generate code (Code generieren) aus.

   Wählen Sie in den angezeigten Tabs Ihre gewünschte Sprache aus. Sie können diesen Code jetzt kopieren und in Ihrer Anwendung verwenden.

8. Falls die Operation sofort ausgeführt werden soll, wählen Sie Run (Ausführen).

9. Wenn Sie diese Operation für die spätere Verwendung speichern möchten, wählen Sie Save operation (Operation speichern) aus, geben Sie einen Namen für Ihre Operation ein und wählen Sie Save (Speichern) aus.

Weitere Informationen zu diesem Vorgang finden Sie PutItemin der Amazon DynamoDB DynamoDB-API-Referenz.

Element aktualisieren

Zum Ausführen oder Generieren von Code für eine `Update Item`-Operation gehen Sie folgendermaßen vor.

1. Suchen Sie die Tabelle, in der Sie ein Element aktualisieren möchten.
2. Wählen Sie das Element aus.
3. Geben Sie den Namen und den Wert des Attributs für den ausgewählten Ausdruck ein.
4. Falls Sie weitere Ausdrücke hinzufügen möchten, wählen Sie einen anderen Ausdruck in der Dropdown-Liste Ausdruck aktualisieren aus und klicken Sie anschließend auf das Pluszeichen.
5. Falls ein Bedingungsausdruck erfüllt sein muss, damit die `Update Item`-Operation erfolgreich ist, gehen Sie folgendermaßen vor:

a. Wählen Sie Condition (Bedingung).
b. Geben Sie den Attributnamen, Vergleichsoperator, Attributtyp und Attributwert an.
c. Falls weitere Bedingungen erforderlich sind, wählen Sie erneut Condition (Bedingung).

Weitere Informationen finden Sie unter CLI-Beispiel für DynamoDB-Bedingungsausdrücke.

6. Falls Sie Code generieren möchten, wählen Sie Generate code (Code generieren).

   Wählen Sie den Tab für die gewünschte Sprache. Sie können diesen Code jetzt kopieren und in Ihrer Anwendung verwenden.
7. Falls die Operation sofort ausgeführt werden soll, wählen Sie Run (Ausführen).
8. Wenn Sie diese Operation für die spätere Verwendung speichern möchten, wählen Sie Save operation (Operation speichern) aus, geben Sie einen Namen für Ihre Operation ein und wählen Sie Save (Speichern) aus.

Weitere Informationen zu diesem Vorgang finden Sie UpdateItemin der Amazon DynamoDB DynamoDB-API-Referenz.

Element löschen

Gehen Sie folgendermaßen vor, um die Operation `Delete Item` auszuführen.

1. Suchen Sie die Tabelle, in der Sie ein Element löschen möchten.
2. Wählen Sie das Element aus.
3. Wählen Sie im Dropdown-Menü Aktionen die Option Element löschen aus.
4. Bestätigen Sie, dass Sie das Element löschen möchten, indem Sie Löschen auswählen.

Weitere Informationen zu diesem Vorgang finden Sie DeleteItemin der Amazon DynamoDB DynamoDB-API-Referenz.

Duplizieren eines Elements

Sie können ein Element duplizieren, indem Sie ein neues Element mit den gleichen Attributen erstellen. Gehen Sie folgendermaßen vor, um ein Element zu duplizieren.

1. Suchen Sie die Tabelle, in der Sie ein Element duplizieren möchten.
2. Wählen Sie das Element aus.

3. Wählen Sie im Dropdown-Menü Aktionen die Option Element duplizieren aus.
4. Geben Sie einen neuen Partitionsschlüssel an.
5. Geben Sie einen neuen Sortierschlüssel an (falls erforderlich).
6. Wählen Sie Ausführen aus.

Weitere Informationen zu diesem Vorgang finden Sie DeleteItemin der Amazon DynamoDB DynamoDB-API-Referenz.

Query

Zum Ausführen oder Generieren von Code für eine `Query`-Operation gehen Sie folgendermaßen vor.

1. Wählen Sie oben in der NoSQL Workbench-Benutzeroberfläche die Option Abfrage aus.
2. Geben Sie den Partitionsschlüsselwert an.
3. Falls ein Sortierschlüssel für die `Query`-Operation erforderlich ist:
   a. Wählen Sie Sortierschlüssel aus:
   b. Geben Sie den Vergleichsoperator und Attributwert an.
4. Wählen Sie Abfrage aus, um den Abfragevorgang auszuführen. Wenn weitere Optionen benötigt werden, aktivieren Sie das Kontrollkästchen Weitere Optionen und fahren Sie mit den folgenden Schritten fort.
5. Falls nicht alle Attribute mit dem Operationsergebnis zurückgegeben werden sollen, wählen Sie Projection expression (Projektionsausdruck)aus.
6. Klicken Sie auf das Pluszeichen (+).
7. Geben Sie das Attribut ein, das mit dem Abfrageergebnis zurückgegeben werden soll.
8. Falls weitere Attribute erforderlich sind, wählen Sie + (Pluszeichen).
9. Falls ein Bedingungsausdruck erfüllt sein muss, damit die `Query`-Operation erfolgreich ist, gehen Sie folgendermaßen vor:
   a. Wählen Sie Condition (Bedingung).
   b. Geben Sie den Attributnamen, Vergleichsoperator, Attributtyp und Attributwert an.
   c. Falls weitere Bedingungen erforderlich sind, wählen Sie erneut Condition (Bedingung).

   Weitere Informationen finden Sie unter CLI-Beispiel für DynamoDB-Bedingungsausdrücke.

10. Falls Sie Code generieren möchten, wählen Sie Generate code (Code generieren).

    Wählen Sie den Tab für die gewünschte Sprache. Sie können diesen Code jetzt kopieren und in Ihrer Anwendung verwenden.
11. Falls die Operation sofort ausgeführt werden soll, wählen Sie Run (Ausführen).
12. Wenn Sie diese Operation für die spätere Verwendung speichern möchten, wählen Sie Save operation (Operation speichern) aus, geben Sie einen Namen für Ihre Operation ein und wählen Sie Save (Speichern) aus.

Weitere Informationen zu dieser Operation finden Sie unter Abfragen in den Amazon-DynamoDB-API-Referenzen.

Scan

Zum Ausführen oder Generieren von Code für eine `Scan`-Operation gehen Sie folgendermaßen vor.

1. Wählen Sie oben in der NoSQL Workbench-Benutzeroberfläche die Option Scan aus.
2. Klicken Sie auf die Schaltfläche Scan, um diesen einfachen Scanvorgang durchzuführen. Wenn weitere Optionen benötigt werden, aktivieren Sie das Kontrollkästchen Weitere Optionen und fahren Sie mit den folgenden Schritten fort.
3. Geben Sie einen Attributnamen an, um die Scanergebnisse zu filtern.
4. Falls nicht alle Attribute mit dem Operationsergebnis zurückgegeben werden sollen, wählen Sie Projection expression (Projektionsausdruck)aus.
5. Falls ein Bedingungsausdruck erfüllt sein muss, damit die Scan-Operation erfolgreich ist, gehen Sie folgendermaßen vor:

    a. Wählen Sie Condition (Bedingung).

    b. Geben Sie den Attributnamen, Vergleichsoperator, Attributtyp und Attributwert an.

    c. Falls weitere Bedingungen erforderlich sind, wählen Sie erneut Condition (Bedingung).

    Weitere Informationen finden Sie unter CLI-Beispiel für DynamoDB-Bedingungsausdrücke.
6. Falls Sie Code generieren möchten, wählen Sie Generate code (Code generieren).

    Wählen Sie den Tab für die gewünschte Sprache. Sie können diesen Code jetzt kopieren und in Ihrer Anwendung verwenden.
7. Falls die Operation sofort ausgeführt werden soll, wählen Sie Run (Ausführen).

8. Wenn Sie diese Operation für die spätere Verwendung speichern möchten, wählen Sie Save operation (Operation speichern) aus, geben Sie einen Namen für Ihre Operation ein und wählen Sie Save (Speichern) aus.

TransactGetItems

Zum Ausführen oder Generieren von Code für eine `TransactGetItems`-Operation gehen Sie folgendermaßen vor.

1. Wählen Sie in der Dropdownliste Weitere Operationen oben in der NoSQL Workbench-Benutzeroberfläche die Option aus. TransactGetItems
2. Wählen Sie das Pluszeichen (+) in der Nähe. TransactGetItem
3. Geben Sie einen Partitionsschlüssel an.
4. Geben Sie einen Sortierschlüssel an (falls erforderlich).
5. Wählen Sie Ausführen, um den Vorgang auszuführen, Vorgang speichern, um ihn zu speichern, oder Code generieren, um Code dafür zu generieren.

Weitere Informationen über Transaktionen finden Sie unter Amazon DynamoDB Transactions.

TransactWriteItems

Zum Ausführen oder Generieren von Code für eine `TransactWriteItems`-Operation gehen Sie folgendermaßen vor.

1. Wählen Sie in der Dropdownliste Weitere Operationen oben in der NoSQL Workbench-Benutzeroberfläche die Option aus. TransactWriteItems
2. Wählen Sie aus dem Dropdown-Menü Aktionen einen Vorgang aus.
3. Wählen Sie das Pluszeichen (+) in der Nähe. TransactWriteItem
4. Wählen Sie im Dropdown-Menü Aktionen die gewünschte Operation.
   - Befolgen Sie für `DeleteItem` die Anweisungen für die Element löschen-Operation.
   - Befolgen Sie für `PutItem` die Anweisungen für die Element einfügen-Operation.
   - Befolgen Sie für `UpdateItem` die Anweisungen für die Element aktualisieren-Operation.

Zum Ändern der Reihenfolge der Aktionen wählen Sie eine Aktion in der Liste auf der linken Seite und dann den Nach-oben- oder Nach-unten-Pfeil, um sie in der Liste nach oben oder unten zu verschieben.

Zum Löschen einer Aktion wählen Sie diese in der Liste und dann das Symbol Delete (Löschen) (Papierkorb).

5. Wählen Sie Ausführen, um den Vorgang auszuführen, Vorgang speichern, um ihn zu speichern, oder Code generieren, um Code dafür zu generieren.

Weitere Informationen über Transaktionen finden Sie unter Amazon DynamoDB Transactions.

Klonen von Tabellen mit NoSQL Workbench

Durch das Klonen von Tabellen werden die wichtigsten Schemata einer Tabelle (und optional das GSI-Schema und Elemente) zwischen Ihren Entwicklungsumgebungen kopiert. Sie können eine Tabelle von der lokalen Version von DynamoDB zu einem Amazon-DynamoDB-Konto klonen und sogar von einem Konto in ein anderes in verschiedenen Regionen, um Tests zu beschleunigen.

So klonen Sie eine Tabelle

1. Wählen Sie im Operation Builder Ihre Verbindung und Region aus (die Regionsauswahl ist für die lokale Version von DynamoDB nicht verfügbar).
2. Sobald Sie mit DynamoDB verbunden sind, durchsuchen Sie Ihre Tabellen und wählen Sie die Tabelle aus, die Sie klonen möchten.
3. Wählen Sie im Dreipunktmenü die Option Klonen aus.
4. Geben Sie die Details zum Klonziel ein:

    a. Wählen Sie eine Verbindung aus.
    b. Wählen Sie eine Region aus („Region“ ist für die lokale Version von DynamoDB nicht verfügbar).
    c. Geben Sie einen neuen Tabellennamen ein.
    d. Wählen Sie eine Klonoption:

        i. Das Schlüsselschema ist standardmäßig ausgewählt und kann nicht deaktiviert werden. Standardmäßig werden beim Klonen einer Tabelle der Primärschlüssel und der Sortierschlüssel kopiert, sofern sie verfügbar sind.

ii. Das GSI-Schema ist standardmäßig ausgewählt, wenn die zu klonende Tabelle über einen GSI verfügt. Standardmäßig werden beim Klonen einer Tabelle der Primärschlüssel und der Sortierschlüssel des GSI kopiert, sofern sie verfügbar sind. Sie können das GSI-Schema deaktivieren, um das Klonen des GSI-Schemas zu überspringen. Beim Klonen einer Tabelle werden die Kapazitätseinstellungen der Basistabelle für den GSI kopiert. Sie können den Vorgang `UpdateTable` in Operation Builder verwenden, um die GSI-Kapazitätseinstellung der Tabelle zu aktualisieren, nachdem das Klonen abgeschlossen ist.

5. Geben Sie die Anzahl der zu klonenden Elemente ein. Um nur das Schlüsselschema und optional das GSI-Schema zu klonen, können Sie den Wert Zu klonende Elemente bei 0 belassen. Maximal können 5 000 Elemente geklont werden.
6. Wählen Sie einen Kapazitätsmodus:

   a. Der Modus On-Demand ist standardmäßig ausgewählt. DynamoDB On-Demand bietet pay-per-request Preise für Lese- und Schreibanforderungen, sodass Sie nur für das bezahlen, was Sie tatsächlich nutzen. Weitere Informationen finden Sie unter DynamoDB On-demand mode.

   b. Im bereitgestellten Modus können Sie die Anzahl der Lese- und Schreibvorgänge pro Sekunde festlegen, die Sie für Ihre Anwendung benötigen. Die bereitgestellte Kapazität Ihrer Tabelle kann mithilfe von Auto Scaling als Reaktion auf Datenverkehrsänderungen automatisch angepasst werden. Weitere Informationen finden Sie unter DynamoDB Provisioned mode.
7. Wählen Sie Klonen, um mit dem Klonen zu beginnen.
8. Der Klonvorgang wird im Hintergrund ausgeführt. Auf der Registerkarte Operation Builder wird eine Benachrichtigung angezeigt, wenn sich der Status der Klontabelle ändert. Sie können diesen Status abrufen, indem Sie die Registerkarte Operation Builder und dann die Pfeilschaltfläche auswählen. Die Pfeilschaltfläche befindet sich im Status-Widget zum Klonen von Tabellen am unteren Rand der Menüseitenleiste.

Exportieren von Daten in eine CSV-Datei

Sie können die Ergebnisse einer Abfrage aus dem Operation Builder in eine CSV-Datei exportieren. Auf diese Weise können Sie die Daten in eine Tabelle laden oder in Ihrer bevorzugten Programmiersprache verarbeiten.

Exportieren in CSV

1. Führen Sie im Operation Builder einen Vorgang Ihrer Wahl aus, z. B. einen Scan oder eine Abfrage.

   > **Note**
   >
   > - Sie können nur Ergebnisse von API-Lesevorgängen und PartiQL-Anweisungen in eine CSV-Datei exportieren. Ergebnisse von Transaktionsleseanweisungen können nicht exportiert werden.
   > - Derzeit können Sie Ergebnisse nur seitenweise in eine CSV-Datei exportieren. Wenn mehrere Ergebnisseiten vorliegen, muss jede Seite einzeln exportiert werden.

2. Wählen Sie aus den Ergebnissen die Elemente aus, die exportiert werden sollen.
3. Wählen Sie im Dropdown-Menü Aktionen die Option Als CSV exportieren aus.
4. Wählen Sie einen Dateinamen und einen Speicherort für die CSV-Datei und anschließend Speichern aus.

Beispieldatenmodelle für NoSQL Workbench

Auf der Homepage für den Modeler werden eine Reihe von Beispielmodellen angezeigt, die im Lieferumfang der NoSQL Workbench enthalten sind. In diesem Abschnitt werden diese Modelle und ihre möglichen Verwendungsmöglichkeiten beschrieben.

Themen

- Mitarbeiterdatenmodell
- Datenmodell des Diskussionsforums
- Datenmodell der Musikbibliothek
- Datenmodell für Skigebiet
- Datenmodell für Kreditkartenangebote
- Datenmodell für Lesezeichen

Mitarbeiterdatenmodell

Dieses Datenmodell ist ein Einführungsmodell. Es enthält grundlegende Details eines Mitarbeiters, wie eindeutiger Alias, Vorname, Nachname, Bezeichnung, Manager und Fähigkeiten.

Dieses Datenmodell zeigt einige Techniken, wie etwa den Umgang mit komplexen Attributen, z. B. mit mehr als einer Fertigkeit. Dieses Modell ist auch ein Beispiel für eine one-to-many Beziehung zwischen dem Manager und seinen berichtspflichtigen Mitarbeitern, die durch den sekundären Index erreicht wurde. DirectReports

Folgende Zugriffsmuster werden durch dieses Datenmodell erleichtert:

- Abrufen eines Mitarbeiterdatensatzes unter Verwendung des Login-Alias des Mitarbeiters, erleichtert durch eine Tabelle namens `Employee`.
- Suchen nach Mitarbeitern nach Namen, erleichtert durch den globalen sekundären Index der Mitarbeitertabelle mit der Bezeichnung `Name`.
- Abrufen aller direkt unterstellten Mitarbeiter eines Vorgesetzten unter Verwendung des Login-Alias des Vorgesetzten, erleichtert durch den globalen sekundären Index der Mitarbeitertabelle mit der Bezeichnung `DirectReports`.

Datenmodell des Diskussionsforums

Dieses Datenmodell steht für ein Diskussionsforum. Mit diesem Modell können Kunden mit der Entwickler-Community interagieren, Fragen stellen und auf Beiträge anderer Kunden antworten. Jeder AWS -Service hat ein dediziertes Forum. Jeder kann einen neuen Diskussionsthread starten, indem er eine Nachricht in einem Forum postet. Jeder Thread erhält eine beliebige Anzahl von Antworten.

Folgende Zugriffsmuster werden durch dieses Datenmodell erleichtert:

- Abrufen eines Forumsdatensatzes anhand des Namens des Forums, erleichtert durch eine Tabelle mit der Bezeichnung `Forum`.
- Abrufen eines bestimmten Threads oder aller Threads für ein Forum, erleichtert durch eine Tabelle mit der Bezeichnung `Thread`.
- Suchen nach Antworten anhand der E-Mail-Adresse des Benutzers, erleichtert durch den globalen sekundären Index der Antworttabelle mit der Bezeichnung `PostedBy-Message-Index`.

Datenmodell der Musikbibliothek

Dieses Datenmodell stellt eine Musikbibliothek mit einer großen Sammlung von Songs dar und präsentiert die am meisten heruntergeladenen Songs nahezu in Echtzeit.

Folgende Zugriffsmuster werden durch dieses Datenmodell erleichtert:

- Abrufen eines Songs, erleichtert durch eine Tabelle mit der Bezeichnung `Songs`.
- Abrufen eines bestimmten Downloads oder aller Downloads für einen Song, erleichtert durch eine Tabelle mit der Bezeichnung `Songs`.
- Abrufen einer bestimmten Anzahl monatlicher Downloads oder aller monatlichen Downloads für einen Song, erleichtert durch eine Tabelle mit der Bezeichnung `Song`.
- Abrufen aller Datensätze (einschließlich Song, Downloads und monatlicher Downloads) für einen Song, erleichtert durch eine Tabelle mit der Bezeichnung `Songs`.
- Suchen nach den am meisten heruntergeladenen Songs, erleichtert durch den globalen sekundären Index der Songtabelle mit der Bezeichnung `DownloadsByMonth`.

Datenmodell für Skigebiet

Dieses Datenmodell steht für ein Skigebiet, das täglich eine große Datenerfassung für jeden Skilift durchführt.

Folgende Zugriffsmuster werden durch dieses Datenmodell erleichtert:

- Abrufen aller dynamischen und statischen Daten für einen bestimmten Skilift oder das gesamte Skigebiet, erleichtert durch eine Tabelle mit der Bezeichnung `SkiLifts`.
- Abrufen aller dynamischen Daten (einschließlich einzelner Liftfahrer, Schneedecke, Lawinengefahr und Liftstatus) für einen Skilift oder das gesamte Skigebiet an einem bestimmten Datum, erleichtert durch eine Tabelle mit der Bezeichnung `SkiLifts`.
- Abrufen aller statischen Daten (einschließlich Empfehlung nur für erfahrene Skifahrer, Anstieg in Höhenmetern und Fahrzeit) für einen bestimmten Skilift, erleichtert durch eine Tabelle mit der Bezeichnung `SkiLifts`.
- Das Abrufen des Datums von Daten, die für einen bestimmten Skilift oder das gesamte Skigebiet aufgezeichnet wurden, sortiert nach der Gesamtzahl der einzelnen Fahrer, wird durch den in der SkiLifts Tabelle genannten `SkiLiftsByRiders` globalen Sekundärindex erleichtert.

Datenmodell für Kreditkartenangebote

Dieses Datenmodell wird von einer Anwendung für Kreditkartenangebote verwendet.

Ein Kreditkartenanbieter erstellt im Laufe der Zeit Angebote. Zu diesen Angeboten gehören Überweisungen ohne Gebühren, erhöhte Kreditlimits, niedrigere Zinssätze, Rückzahlungen und Flugmeilen. Nachdem ein Kunde diese Angebote annimmt oder ablehnt, wird der jeweilige Angebotsstatus entsprechend aktualisiert.

Folgende Zugriffsmuster werden durch dieses Datenmodell erleichtert:

- Abrufen von Kontodatensätzen mit `AccountId`, erleichtert durch die Haupttabelle.
- Abrufen aller Konten mit wenigen projizierten Posten, erleichtert durch den sekundären Index `AccountIndex`.
- Abrufen von Konten und allen Angebotsdatensätzen, die mit diesen Konten in Verbindung stehen, mithilfe von `AccountId`, erleichtert durch die Haupttabelle.
- Abrufen von Konten und spezifischen Angebotsdatensätzen, die mit diesen Konten in Verbindung stehen, mithilfe von `AccountId` und `OfferId`, erleichtert durch die Haupttabelle.
- Abrufen aller `ACCEPTED/DECLINED`-Angebotsdatensätze mit spezifischem `OfferType`, die mit Konten in Verbindung stehen, mithilfe von `AccountId`, `OfferType` und `Status`, erleichtert durch den sekundären Index `GSI1`.
- Abrufen von Angeboten und damit in Verbindung stehenden Angebotspositionsdatensätzen mit `OfferId`, erleichtert durch die Haupttabelle.

Datenmodell für Lesezeichen

Dieses Datenmodell wird zum Speichern von Lesezeichen für Kunden verwendet.

Ein Kunde kann viele Lesezeichen haben und ein Lesezeichen kann zu vielen Kunden gehören. Dieses Datenmodell stellt eine many-to-many Beziehung dar.

Folgende Zugriffsmuster werden durch dieses Datenmodell erleichtert:

- Eine einzelne Abfrage anhand von `customerId` kann nun Kundendaten sowie Lesezeichen zurückgeben.
- Ein `ByEmail`-Abfrageindex gibt Kundendaten nach E-Mail-Adresse zurück. Beachten Sie, dass Lesezeichen nicht durch diesen Index abgerufen werden.

- Ein `ByUrl`-Abfrageindex ruft Lesezeichendaten nach URL ab. Beachten Sie, dass wir CustomerID als Sortierschlüssel für den Index haben, da dieselbe URL von mehreren Kunden mit einem Lesezeichen versehen werden kann.
- Ein `ByCustomerFolder`-Abfrageindex ruft Lesezeichen nach Ordner für jeden Kunden ab.

Versionsverlauf für NoSQL-Workbench

In der folgenden Tabelle werden die wichtigen Änderungen in den einzelnen Versionen des Client-Tools NoSQL-Workbench beschrieben.

Version	Änderungen	Beschreibung	Date
3,20,0	Neuer Data Modeler für DynamoDB	Data Modeler for DynamoDB bietet eine aktualisierte Benutzererfahrung. Data Modeler für DynamoDB unterstützt jetzt Zugriffsmuster.	16. Februar 2026
3.13.5	Kapazitätsmodus für standardmäßige Tabelleneinstellungen ist jetzt „On-Demand“	Wenn Sie eine Tabelle mit Standardeinstellungen erstellen, erstellt DynamoDB eine Tabelle, die anstelle des Modus bereitgestellter Kapazität den Modus „On-Demand-Kapazität“ verwendet.	24. Februar 2025
3.13,0	Verbesserungen von NoSQL Workbench Operation Builder	NoSQL Workbench bietet jetzt native Unterstützung für den dunklen Modus. Die Tabellen- und Elementoperationen	24. April 2024

Version	Änderungen	Beschreibung	Date
		in Operation Builder wurden verbesser t. Die Ergebnisse zu Elementen und Informationen zu Operation-Builder-Anfragen sind im JSON-Format verfügbar.	
3.12.0	Klonen von Tabellen mit NoSQL Workbench und Zurückgeb en verbrauchter Kapazität	Sie können jetzt Tabellen zwischen lokalen DynamoDB-Versionen und einem DynamoDB-Webdienstkonto oder zwischen DynamoDB-Webdienstkonten klonen, um die Entwicklungsiterat ionen zu beschleun igen. Zeigen Sie verbrauchte Lese- oder Schreibka pazitätseinheiten an, nachdem ein Vorgang mit dem Operation Builder ausgeführ t wurde. Wir haben das Problem beim Überschreiben von Daten beim Import aus einer CSV-Datei behoben.	26. Februar 2024

Version	Änderungen	Beschreibung	Date
3.11.0	Verbesserungen an der lokalen Version von DynamoDB	Sie können jetzt den Port angeben, wenn Sie die integrierte lokale DynamoDB-Instance starten. NoSQL Workbench kann jetzt ohne Administratorrechte unter Windows installiert werden. Wir haben die Datenmodellvorlagen aktualisiert.	17. Januar 2024
3.10.0	Native Unterstützung für Apple silicon	NoSQL Workbench bietet jetzt native Unterstützung für Mac mit Apple silicon. Sie können jetzt das Format für die Generierung von Beispieldaten für Attribute des Typs `Number` konfigurieren.	05. Dezember 2023
3.9.0	Verbesserungen des Datenmodellierers	Visualizer unterstützt jetzt die Übergabe von Datenmodellen an die lokale Version von DynamoDB mit der Option, bestehende Tabellen zu überschreiben.	3. November 2023

Version	Änderungen	Beschreibung	Date
3.8.0	Beispieldatengenerierung	NoSQL Workbench unterstützt jetzt die Generierung von Beispieldaten für Ihre DynamoDB-Datenmodelle.	25. September 2023
3.6.0	Verbesserungen in Operation Builder	Verbesserungen der Verbindungsverwaltung in Operation Builder. Attributnamen in Data Modeler können jetzt geändert werden, ohne Daten zu löschen. Weitere Fehlerbehebungen	11. April 2023
3.5.0	Support für neue AWS Regionen	NoSQL Workbench unterstützt jetzt die Regionen ap-south-2, ap-southeast-3, ap-southeast-4, eu-central-2, eu-south-2, me-central-1 und me-west-1.	23. Februar 2023
3.4.0	Support für die lokale Version von DynamoDB	NoSQL Workbench unterstützt jetzt die Installation der lokalen Version von DynamoDB als Bestandteil des Installationsprozesses.	6. Dezember 2022

Version	Änderungen	Beschreibung	Date
3.3.0	Support für Operation en auf Steuerebene	Operation Builder unterstützt jetzt Operationen auf Steuerebene.	1. Juni 2022
3.2.0	CSV-Import und Export	Sie können jetzt Beispieldaten aus einer CSV-Datei im Visualizer-Tool importieren und auch die Ergebnisse einer Operation Builder-A bfrage in eine CSV-Datei exportieren.	11. Oktober 2021
3.1.0	Speichern von Vorgängen	Der Operation-Builder in NoSQL-Workbench unterstützt jetzt das Speichern von Vorgängen für die spätere Verwendung.	12. Juli 2021
3.0.0	Kapazitätseinstell ungen und CloudForm ation Import/Export	Das NoSQL-Wor kbench für Amazon DynamoDB unterstüt zt jetzt die Angabe eines Lese-/Sch reibkapazitätsmodus für Tabellen und kann nun Datenmodelle im Format CloudForm ation importieren und exportieren.	21. April 2021

Version	Änderungen	Beschreibung	Date
2.2.0	PartiQL-Support	Das NoSQL-Workbench für Amazon DynamoDB bietet Unterstützung für das Erstellen von PartiQL-Anweisungen für DynamoDB.	4. Dezember 2020
1.1.0	Unterstützung für Linux.	Das NoSQL-Workbench für Amazon DynamoDB wird auf Linux-Ubuntu, Fedora und Debian unterstützt.	4. Mai 2020
1.0.0	NoSQL-Workbench für Amazon DynamoDB – GA.	NoSQL-Workbench für Amazon DynamoDB ist allgemein verfügbar.	2. März 2020
0.4.1	Unterstützung für IAM-Rollen und temporäre Anmeldeinformationen.	Das NoSQL-Workbench für Amazon DynamoDB fügt Unterstützung für AWS Identity and Access Management -(IAM-)Rollen und temporäre Sicherheitsanmeldeinformationen hinzu.	19. Dezember 2019

Version	Änderungen	Beschreibung	Date
0.3.1	Support für die lokale Version von DynamoDB (herunterladbare Version).	NoSQL-Workbench unterstützt jetzt Verbindungen zu Lokale Version von DynamoDB (herunterladbare Version) für den Entwurf, die Erstellung, die Abfrage und die Verwaltung von DynamoDB-Tabellen.	8. November 2019
0.2.1	NoSQL-Workbench (Vorversion) veröffentlicht.	Dies ist die erste Version von NoSQL-Workbench für DynamoDB. Verwenden Sie NoSQL Workbench für zum Gestalten, Erstellen, Abfragen und Verwalten von DynamoDB-Tabellen.	16. September 2019

Backup und Wiederherstellung für DynamoDB

DynamoDB bietet On-Demand-Backups und Backups für die zeitpunktbezogene Wiederherstellung (PITR), um Ihre DynamoDB-Daten bei Notfallereignissen zu schützen. Außerdem gibt es die Datenarchivierung für die langfristige Aufbewahrung. Sie können Tabellen von wenigen Megabyte bis zu Hunderten von Terabyte an Daten sichern, ohne dass dies Auswirkungen auf die Leistung und Verfügbarkeit Ihrer Produktionsanwendungen hat. Alle Backups werden automatisch verschlüsselt, katalogisiert und sind leicht auffindbar.

Mit On-Demand-Backups können Sie ein Snapshot-Backup Ihrer Tabelle erstellen, die DynamoDB speichert und verwaltet. Die Gebühren richten sich nach der Größe und Dauer Ihrer Backups. Mit On-Demand-Backups können Sie Ihre gesamte DynamoDB-Tabelle in den Zustand zurücksetzen, in dem sie sich befand, als das Backup erstellt wurde.

Zum Erstellen und Verwalten von DynamoDB-On-Demand-Backups gibt es zwei Möglichkeiten:

- DynamoDB
- AWS Backup

Sie können die DynamoDB-On-Demand-Backupfunktion nutzen, um vollständige Backups Ihrer Tabellen für die langfristige Aufbewahrung und Archivierung für behördliche Compliance-Anforderungen zu erstellen. Sie können Ihre Tabellendaten jederzeit in der AWS-Managementkonsole oder mit einem einzigen API-Aufruf sichern und wiederherstellen.

Backups für die zeitpunktbezogene Wiederherstellung (PITR) werden vollständig von DynamoDB verwaltet und stellen Wiederherstellungspunkte für bis zu 35 Tage mit einer Granularität pro Sekunde bereit. Um die zeitpunktbezogene Wiederherstellung, also fortlaufende Backups, zu verwenden, aktivieren Sie die zeitpunktbezogene Wiederherstellung (PITR) in Ihrer DynamoDB-Tabelle. Für die Dauer, die PITR für die Tabelle aktiviert ist, wird Ihnen eine Gebühr auf der Grundlage der Größe Ihrer DynamoDB-Tabelle berechnet. Wenn Sie die zeitpunktbezogene Wiederherstellung (PITR) für Ihre DynamoDB-Tabelle aktivieren, werden Ihre Tabellendaten kontinuierlich gesichert. Auf diese Weise können Sie Ihre Tabelle auf einen bestimmten Zeitpunkt innerhalb des PITR-Wiederherstellungszeitraums wiederherstellen, indem Sie eine neue DynamoDB-Tabelle mit dem exakten Status der Originaltabelle zum gewünschten Zeitpunkt erstellen.

Mit der zeitpunktbezogenen Wiederherstellung schützen Sie Ihre DynamoDB-Tabellen vor versehentlichen Schreib- und Löschoperationen. Mit der zeitpunktbezogenen Wiederherstellung

müssen Sie sich keine Gedanken über das Erstellen, Warten oder Planen von On-Demand-Backups machen. Angenommen, Sie schreiben einen Skript-Test versehentlich in eine aktive DynamoDB-Tabelle.

Mit der zeitpunktbezogenen Wiederherstellung können Sie diese Tabelle im Zustand eines beliebigen Zeitpunkts innerhalb der vergangenen 35 Tage wiederherstellen. Der Wiederherstellungszeitraum kann auf einen Wert zwischen 1 und 35 Tagen festgelegt werden. Nachdem Sie die zeitpunktbezogene Wiederherstellung aktiviert haben, können Sie die Wiederherstellung im Zustand eines beliebigen Zeitpunkts innerhalb von fünf Minuten vor der aktuellen Uhrzeit bis zum Ende des konfigurierten Wiederherstellungszeitraums durchführen. DynamoDB verwaltet inkrementelle Backups Ihrer Tabelle.

Darüber hinaus wirken sich zeitpunktbezogene Operationen nicht auf die Leistung oder API-Latenzen aus.

Sie können eine DynamoDB-Tabelle mithilfe der AWS-Managementkonsole, der AWS Command Line Interface (AWS CLI) oder der DynamoDB-API zu einem Zeitpunkt wiederherstellen. Bei der zeitpunktbezogenen Wiederherstellung wird immer eine neue Tabelle erstellt.

Weitere Informationen zur Verfügbarkeit und den Preisen für AWS-Regionen finden Sie unter Amazon-DynamoDB-Preise.

> **Note**
>
> - Tagging und attributbasierte Zugriffskontrolle (ABAC) werden für DynamoDB-Backups nicht unterstützt. Um ABAC mit Backups zu verwenden, empfehlen wir die Verwendung von AWS Backup.
> - Tags werden in wiederhergestellten Tabellen nicht beibehalten. Sie müssen den wiederhergestellten Tabellen Tags hinzufügen, bevor Sie Tag-basierte Bedingungen in Ihren Richtlinien verwenden können.

Das folgende Video gibt Ihnen einen einführenden Einblick in das Sicherungs- und Wiederherstellungskonzept und bietet nähere Informationen zur zeitpunktbezogenen Wiederherstellung.

Backup und Wiederherstellung

Themen

- Zeitpunktbezogene Backups für DynamoDB
- Verwendung von On-Demand-Backup und -Wiederherstellung in DynamoDB
- Grundlegendes zur Amazon-DynamoDB-Fakturierung für Backups
- Wiederherstellen einer Tabelle in DynamoDB
- Verwendung von AWS Backup mit DynamoDB

Zeitpunktbezogene Backups für DynamoDB

Die zeitpunktbezogene Wiederherstellung (Point-in-time Recovery, PITR) in Amazon DynamoDB stellt fortlaufend automatische Backups Ihrer DynamoDB-Tabellendaten bereit. Backups für die zeitpunktbezogene Wiederherstellung (PITR) werden vollständig von DynamoDB verwaltet und stellen Wiederherstellungspunkte für bis zu 35 Tage mit einer Granularität pro Sekunde bereit. Mit der zeitpunktbezogenen Wiederherstellung müssen Sie sich keine Gedanken über das Erstellen, Warten oder Planen von On-Demand-Backups machen. Dieser Abschnitt erhält eine Übersicht über die Funktionsweise dieses Prozesses in DynamoDB.

Bevor Sie beginnen

Bevor Sie die zeitpunktbezogene Wiederherstellung (PITR) für eine Amazon-DynamoDB-Tabelle aktivieren, sollten Sie Folgendes berücksichtigen:

- Durch die Einstellung von `RecoveryPeriodinDays` können Sie den Zeitraum verkürzen, für den fortlaufende Backups erstellt werden. Standardmäßig ist Ihr `RecoveryPeriodinDays` 35. Sie können es jedoch auf einen Wert zwischen 1 und 35 festlegen. Eine Verkürzung des `RecoveryPeriodinDays` hat keine Auswirkungen auf die PITR-Preisgestaltung, da der Preis auf der Größe der Tabelle und den lokalen Sekundärindizes basiert.
- Wenn Sie die zeitpunktbezogene Wiederherstellung deaktivieren und später für eine Tabelle wieder aktivieren, wird hierdurch der Startzeitpunkt für die Wiederherstellung dieser Tabelle zurückgesetzt. Daher können Sie die Tabelle nur mit `LatestRestorableDateTime` sofort wiederherstellen.
- Sie können die zeitpunktbezogene Wiederherstellung für jede lokale Replika einer globalen Tabelle aktivieren. Wenn Sie die Tabelle wiederherstellen, wird eine unabhängige Tabelle erstellt, die nicht Bestandteil der globalen Tabelle ist. Wenn Sie die Globale Tabellenversion 2019.11.21 (aktuell) von globalen Tabellen verwenden, können Sie eine neue globale Tabelle aus der wiederhergestellten Tabelle erstellen. Weitere Informationen finden Sie unter So funktionieren globale DynamoDB-Tabellen.

- Sie können Ihre DynamoDB-Tabellendaten auch in AWS-Regionen so wiederherstellen, dass die wiederhergestellte Tabelle in einer anderen Region als der erstellt wird, in der sich die Quelltabelle befindet. Sie können regionsübergreifende Wiederherstellungen zwischen AWS kommerziellen Regionen, AWS China-Regionen und AWS GovCloud (USA)-Regionen durchführen. Sie zahlen nur für die Daten, die Sie aus der Quellregion übertragen, und für die Wiederherstellung in einer neuen Tabelle in der Zielregion.
- AWS CloudTrail protokolliert alle Konsolen- und API-Aktionen für die zeitpunktbezogene Wiederherstellung, um die Protokollierung, Überwachung und Prüfung zu ermöglichen. Weitere Informationen finden Sie unter Protokollieren von DynamoDB-Vorgängen mithilfe von AWS CloudTrail.

Themen

- point-in-timeWiederherstellung in DynamoDB aktivieren

point-in-timeWiederherstellung in DynamoDB aktivieren

Amazon DynamoDB point-in-time Recovery (PITR) bietet automatische Backups Ihrer DynamoDB-Tabellendaten. Dieser Abschnitt erhält eine Übersicht über die Funktionsweise dieses Prozesses in DynamoDB.

> **Note**
>
> DynamoDB berechnet PITR auf Basis der Größe jeder DynamoDB-Tabelle, einschließlich Tabellendaten und lokaler Sekundärindizes. Eine Änderung des Wiederherstellungsfensters (z. B. von 35 Tagen auf 1 Tag) reduziert den Preis nicht. Die Kosten bleiben unabhängig vom gewählten Wiederherstellungszeitraum gleich. Der konfigurierte maximale Wiederherstellungszeitraum hat keinen Einfluss auf den Preis, der Ihnen für die Aktivierung von PITR berechnet wird. DynamoDB überwacht kontinuierlich die Größe der Tabellen, für die PITR aktiviert ist, um Ihre Backup-Gebühren zu ermitteln. Ihnen wird die PITR-Nutzung in Rechnung gestellt, bis Sie PITR für die einzelnen Tabelle ausschalten.

Themen

- point-in-timeWiederherstellung aktivieren
- Aktivieren von PITR (Konsole)

- Aktivieren von PITR (AWS CLI)
- Aktivieren von PITR (CloudFormation)
- Aktivieren von PITR (API)
- Wiederherstellungszeitraum
- Bearbeiten der PITR
- Löschen einer Tabelle mit aktiviertem PITR

point-in-timeWiederherstellung aktivieren

Sie können die point-in-time Wiederherstellung mit der AWS-Managementkonsole, AWS Command Line Interface (AWS CLI) oder der DynamoDB-API aktivieren. Wenn diese Option aktiviert ist, bietet die point-in-time Wiederherstellung fortlaufende Backups, bis Sie sie explizit deaktivieren.

Nachdem Sie die point-in-time Wiederherstellung aktiviert haben, können Sie die Wiederherstellung zu einem beliebigen Zeitpunkt innerhalb von `EarliestRestorableDateTime` und durchführen`LatestRestorableDateTime`. `LatestRestorableDateTime`liegt in der Regel fünf Minuten vor der aktuellen Uhrzeit. Weitere Informationen finden Sie unter Wiederherstellen einer DynamoDB-Tabelle auf einen bestimmten Zeitpunkt.

> **Note**
>
> Der point-in-time Wiederherstellungsprozess führt immer eine Wiederherstellung in einer neuen Tabelle durch.

Aktivieren von PITR (Konsole)

Aktivieren von PITR über die DynamoDB-Konsole

1. Navigieren Sie zur DynamoDB-Konsole.
2. Klicken Sie in der linken Navigationsleiste auf Tabellen und wählen Sie Ihre DynamoDB-Tabelle aus.
3. Wählen Sie auf der Registerkarte Backups für die Option Zeitpunktbezogene Wiederherstellung Bearbeiten aus.
4. Wählen Sie point-in-timeWiederherstellung einschalten aus.

5. Wählen Sie einen Wert zwischen 1 und 35 für Ihren Wiederherstellungszeitraum. Dies gibt den maximalen Zeitraum an, für den das kontinuierliche Backup wiederherstellbar ist.

Aktivieren von PITR (AWS CLI)

> **Note**
>
> Wenn Sie beim Ausführen von AWS CLI Befehlen Fehler erhalten, finden Sie weitere Informationen unter AWS CLI Fehler beheben. Vergewissern Sie sich, dass Sie die neueste AWS CLI -Version verwenden.

Führen Sie den update-continuous-backupsBefehl mit aktivierter Einstellung point-in-time-recovery-specification aus:

```
aws dynamodb update-continuous-backups \
--table-name Music \
--point-in-time-recovery-specification
 PointInTimeRecoveryEnabled=true,RecoveryPeriodInDays=35
```

Aktivieren von PITR (CloudFormation)

Verwenden Sie die AWS::DynamoDB::TableRessource mit `PointInTimeRecoverySpecification` aktivierter Eigenschaft:

```
Resources:
  iotCatalog:
    Type: AWS::DynamoDB::Table
      Properties:
      ...
      PointInTimeRecoverySpecification:
        PointInTimeRecoveryEnabled: true
        RecoveryPeriodInDays: 35
```

Beispiel einer Anforderungssyntax:

```
{
   "PointInTimeRecoverySpecification": {
      "PointInTimeRecoveryEnabled": boolean,
      "RecoveryPeriodInDays: number
```

```
   },
   "TableName": "string"
}
```

Aktivieren von PITR (API)

Führen Sie den UpdateContinuousBackupsAPI-Vorgang mit eingeschaltetem `PointInTimeRecoverySpecification` Parameter aus.

Beispiel einer Anforderungssyntax:

```
{
   "PointInTimeRecoverySpecification": {
      "PointInTimeRecoveryEnabled": boolean,
      "RecoveryPeriodInDays" : number
   },
   "TableName": "string"
}
```

Beispiel einer Antwortsyntax:

```
{
   "ContinuousBackupsDescription": {
      "ContinuousBackupsStatus": "string",
      "PointInTimeRecoveryDescription": {
         "PointInTimeRecoveryStatus": "string",
         "EarliestRestorableDateTime": number,
         "RecoveryPeriodInDays": number,
         "LatestRestorableDateTime": number
      }
   }
}
```

Python

```
import boto3

dynamodb = boto3.client('dynamodb')

response = dynamodb.update_continuous_backups(
    TableName=<table_name>,
    PointInTimeRecoverySpecification={
```

```
        'PointInTimeRecoveryEnabled': True,
        'RecoveryPeriodInDays': 35
    }
)
```

Wiederherstellungszeitraum

Der Wiederherstellungszeitraum für kontinuierliche Backups kann auf einen Wert zwischen 1 und 35 Tagen festgelegt werden. Dieser `RecoveryPeriodInDays` bestimmt den Zeitraum, für den Ihre kontinuierlichen Backups aufbewahrt werden. Wenn Sie diesen Wert beispielsweise auf 30 Tage festlegen, können Sie Ihre Tabelle auf einen beliebigen Zeitpunkt innerhalb der letzten 30 Tage wiederherstellen.

> **Note**
>
> DynamoDB berechnet PITR auf Basis der Größe jeder DynamoDB-Tabelle, einschließlich Tabellendaten und lokaler Sekundärindizes. Der konfigurierte maximale Wiederherstellungszeitraum hat keinen Einfluss auf den Preis, der Ihnen für die Aktivierung von PITR berechnet wird. Siehe Amazon-DynamoDB-Preise für weitere Informationen zur Preisgestaltung.

Bearbeiten der PITR

Sie können die PITR-Einstellung in Ihrer Tabelle bearbeiten und den Wiederherstellungszeitraum ändern. Wenn Sie den Wiederherstellungszeitraum ändern und ihn auf einen höheren Wert als den zuvor festgelegten Wert legen, ändert sich Ihr `EarliestRestorePoint`-Wert nicht sofort. Da der Wiederherstellungszeitraum ein gleitendes Zeitfenster ist, erstellt DynamoDB weiterhin automatische Backups, bis der neu verlängerte Zeitraum erreicht ist. Wenn Sie den Wiederherstellungszeitraum ändern und ihn auf einen niedrigeren Wert als den zuvor festgelegten Wert verringern, `EarliestRestorePoint` wird er sofort an Ihren Wiederherstellungszeitraum angepasst, und alle kontinuierlichen Backups, die außerhalb des neuen festgelegten Werts liegen, können nicht wiederhergestellt werden.

Löschen einer Tabelle mit aktiviertem PITR

Wenn Sie eine Tabelle löschen, für die point-in-time Wiederherstellung aktiviert ist, erstellt DynamoDB automatisch einen Backup-Snapshot, der als Systemsicherung bezeichnet wird, und bewahrt ihn 35 Tage lang auf (ohne zusätzliche Kosten). Sie können das System-Backup verwenden,

um die gelöschte Tabelle in dem Zustand wiederherzustellen, in dem sie sich kurz vor dem Löschen befand. Alle Systemsicherungen folgen der Standardbenennungskonvention von. *table-name* `$DeletedTableBackup`

> **Note**
>
> Sobald eine Tabelle mit aktivierter point-in-time Wiederherstellung gelöscht wurde, können Sie die Systemsicherung verwenden, um diese Tabelle auf einen einzigen Zeitpunkt zurückzusetzen. Das System-Backup wird beim Löschen der Tabelle erstellt und stellt eine Momentaufnahme der Tabelle unmittelbar vor ihrer Löschung dar.

Verwendung von On-Demand-Backup und -Wiederherstellung in DynamoDB

Amazon DynamoDB unterstützt eigenständige On-Demand-Backup- und Wiederherstellungsfunktionen. Diese Funktionen stehen Ihnen unabhängig davon zur Verfügung, ob Sie AWS Backup verwenden.

Sie können die DynamoDB-On-Demand-Backupfunktion nutzen, um vollständige Backups Ihrer Tabellen für die langfristige Aufbewahrung und Archivierung für behördliche Compliance-Anforderungen zu erstellen. Sie können Ihre Tabellendaten jederzeit mit einem einzigen Klick auf die AWS Management Console oder mit einem einzigen API-Aufruf sichern und wiederherstellen. Backup- und Wiederherstellungsaktionen werden ohne Auswirkungen auf die Tabellenleistung oder -verfügbarkeit ausgeführt.

Sie können Tabellensicherungen mit der Konsole, der AWS Befehlszeilenschnittstelle (AWS CLI) oder der DynamoDB-API erstellen. Weitere Informationen finden Sie unter Backup einer DynamoDB-Tabelle.

Informationen zum Wiederherstellen einer Tabelle aus einem Backup finden Sie unter Wiederherstellen einer DynamoDB-Tabelle aus einem Backup.

Backup und Wiederherstellen von DynamoDB-Tabellen mit DynamoDB: So funktioniert es

Sie können mit der DynamoDB-On-Demand-Backupfunktion vollständige Backups Ihrer Amazon-DynamoDB-Tabellen erstellen. Diese Funktion ist unabhängig vom AWS Backup verfügbar.

Dieser Abschnitt bietet eine Übersicht über die Aktionen während des DynamoDB-Backup- und Wiederherstellungsvorgangs.

Sicherungen

Wenn Sie ein On-Demand-Backup mit DynamoDB erstellen, wird ein Zeitmarker der Anforderung katalogisiert. Das Backup wird durch Anwenden aller Änderungen bis zur Uhrzeit der Anforderung für den letzten vollständigen Tabellen-Snapshot asynchron erstellt. DynamoDB-Backupanforderungen werden sofort verarbeitet und stehen innerhalb weniger Minuten für die Wiederherstellung zur Verfügung.

> **Note**
>
> Bei jedem On-Demand-Backup werden die gesamten Tabellendaten gesichert. Es gibt keine Beschränkungen in Bezug auf die Anzahl der On-Demand-Backups, die erstellt werden können.

Alle Backups in DynamoDB funktionieren, ohne den bereitgestellten Durchsatz für die Tabelle zu beanspruchen.

DynamoDB-Backups garantieren keine ursächliche Konsistenz der Elemente. Doch der Versatz zwischen Updates in einem Backup beträgt in der Regel viel weniger als eine Sekunde.

Während ein Backup ausgeführt wird, können Sie die folgenden Vorgänge nicht ausführen:

- Den Backupvorgang anhalten oder abbrechen.
- Die Quelltabelle des Backups löschen.
- Backups für eine Tabelle deaktivieren, wenn ein Backup für diese Tabelle gerade ausgeführt wird.

Wenn Sie keine Planungsskripts und Bereinigungsaufträge erstellen möchten, können AWS Backup Sie Backup-Pläne mit Zeitplänen und Aufbewahrungsrichtlinien für Ihre DynamoDB-Tabellen erstellen. AWS Backup führt die Backups aus und löscht sie, wenn sie ablaufen. Weitere Informationen finden Sie im AWS Backup -Entwicklerhandbuch.

Darüber hinaus können Sie mithilfe von AWS Lambda Funktionen regelmäßige oder future Backups planen. AWS Backup Weitere Informationen finden Sie im Blog-Eintrag A Serverless solution to schedule your Amazon DynamoDB On-Demand Backup.

Wenn Sie die Konsole verwenden, AWS Backup werden alle mit dieser Methode erstellten Backups auf der Registerkarte Backups aufgeführt, wobei der Backup-Typ auf eingestellt ist`AWS`.

> **Note**
>
> Sie können Backups, die mit dem Backuptyp gekennzeichnet sind, nicht AWS mithilfe der DynamoDB-Konsole löschen. Verwenden Sie die AWS Backup Konsole, um diese Backups zu verwalten.

Weitere Informationen zum Ausführen eines Backups finden Sie unter Backup einer DynamoDB-Tabelle.

Wiederherstellen

Sie stellen eine Tabelle wieder her, ohne den bereitgestellten Durchsatz für die Tabelle zu beanspruchen. Sie können eine vollständige Tabellenwiederherstellung aus Ihrem DynamoDB-Backup durchführen oder die Zieltabelleneinstellungen konfigurieren. Wenn Sie eine Wiederherstellung durchführen, können Sie die folgenden Tabelleneinstellungen ändern:

- Globale sekundäre Indizes () GSIs
- Lokale Sekundärindizes () LSIs
- Fakturierungsmodus
- Bereitgestellte Lese- und Schreibkapazität
- Verschlüsselungseinstellungen

> **Important**
>
> Bei einer vollständigen Wiederherstellung der Tabelle werden für die Zieltabelle die gleichen bereitgestellten Lese- und Schreibkapazitätseinheiten wie für die Quelltabelle festgelegt, die zum Zeitpunkt der Backupanforderung erfasst wurden. Der Wiederherstellungsvorgang stellt auch die lokalen und die globalen sekundären Indizes wieder her.

Sie können Ihre DynamoDB-Tabellendaten auch regionsübergreifend wiederherstellen, AWS sodass die wiederhergestellte Tabelle in einer anderen Region als der, in der sich das Backup befindet,

erstellt wird. Sie können regionsübergreifende Wiederherstellungen zwischen AWS Handelsregionen, AWS China Regionen und AWS GovCloud (US-) Regionen durchführen. Sie zahlen nur für die Daten, die Sie aus der Quellregion übertragen, und für die Wiederherstellung in einer neuen Tabelle in der Zielregion.

Wiederherstellungen können schneller und kosteneffizienter sei, wenn Sie die Erstellung einiger oder aller sekundärer Indizes auf der neu wiederhergestellten Tabelle ausschließen.

Sie müssen für die wiederhergestellte Tabelle Folgendes einrichten:

- Auto Scaling-Richtlinien
- AWS Identity and Access Management (IAM) -Richtlinien
- CloudWatch Amazon-Metriken und Alarme
- Tags (Markierungen)
- Stream-Einstellungen
- Einstellungen für Gültigkeitsdauer (TTL)
- Einstellungen für den Löschschutz
- Einstellung für die zeitpunktbezogene Wiederherstellung (PITR)

Sie können die gesamten Tabellendaten nur in einer neuen Tabelle aus einem Backup wiederherstellen. Sie können erst in die wiederhergestellte Tabelle schreiben, wenn sie aktiv ist.

> **Note**
>
> Es ist nicht möglich, eine vorhandene Tabelle während einer Wiederherstellung zu überschreiben.

Die Wiederherstellungszeiten stehen in direktem Zusammenhang mit der Konfiguration Ihrer Tabellen (z. B. der Größe Ihrer Tabellen und der Anzahl der zugrunde liegenden Partitionen) und anderen verwandten Variablen. Eine bewährte Methode bei der Planung der Notfallwiederherstellung besteht darin, die durchschnittlichen Wiederherstellungszeiten regelmäßig zu dokumentieren und festzulegen, wie sich diese Zeiten auf Ihr gesamtes Recovery-Zeitziel auswirken.

Weitere Informationen zum Ausführen einer Wiederherstellung finden Sie unter Wiederherstellen einer DynamoDB-Tabelle aus einem Backup.

Sie können IAM-Richtlinien für die Zugriffskontrolle einsetzen. Weitere Informationen finden Sie unter Verwenden von IAM mit DynamoDB-Backup und -Wiederherstellung.

Sämtliche Konsolen- und API-Aktionen werden zum Backup und Wiederherstellung in AWS CloudTrail für die Protokollierung, Überwachung und Prüfung erfasst und aufgezeichnet.

Backup einer DynamoDB-Tabelle

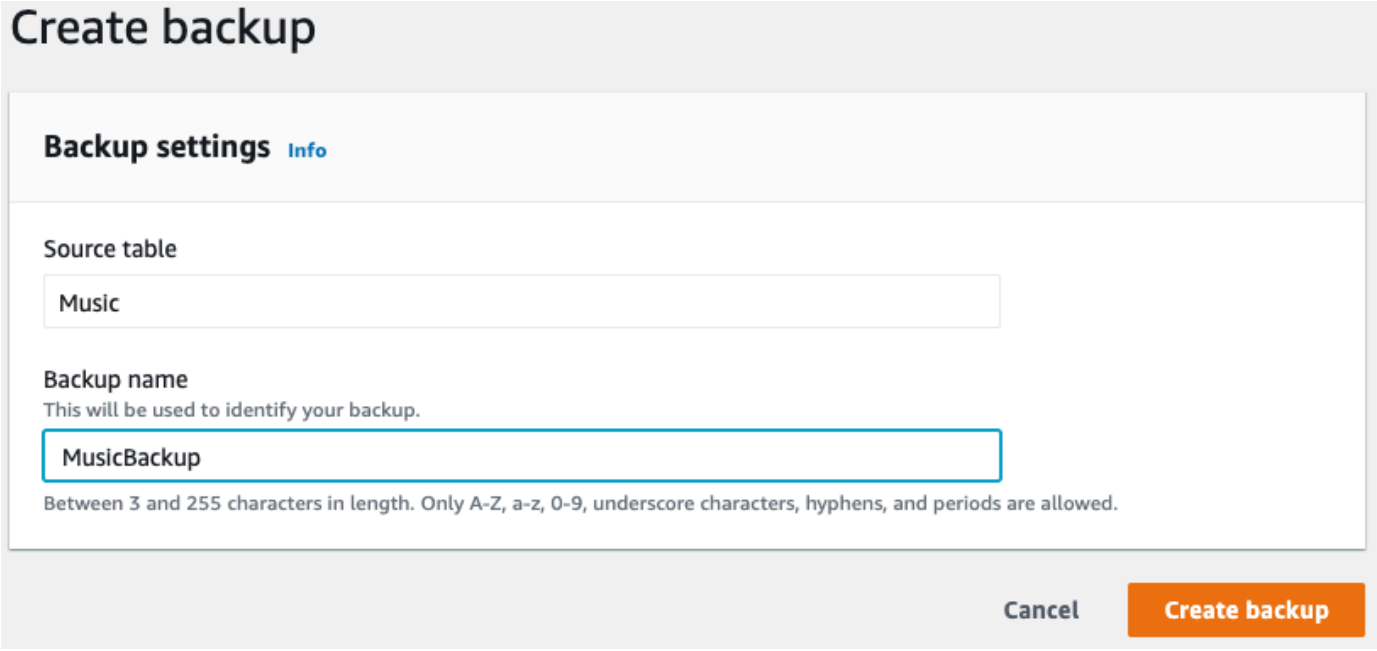
In diesem Abschnitt wird beschrieben, wie Sie die Amazon DynamoDB DynamoDB-Konsole verwenden oder eine Tabelle sichern.AWS Command Line Interface

Erstellen eines Tabellen-Backups (Konsole)

Gehen Sie wie folgt vor, um ein Backup mit dem Namen `MusicBackup` für eine vorhandene `Music`-Tabelle mithilfe der AWS-Managementkonsole zu erstellen.

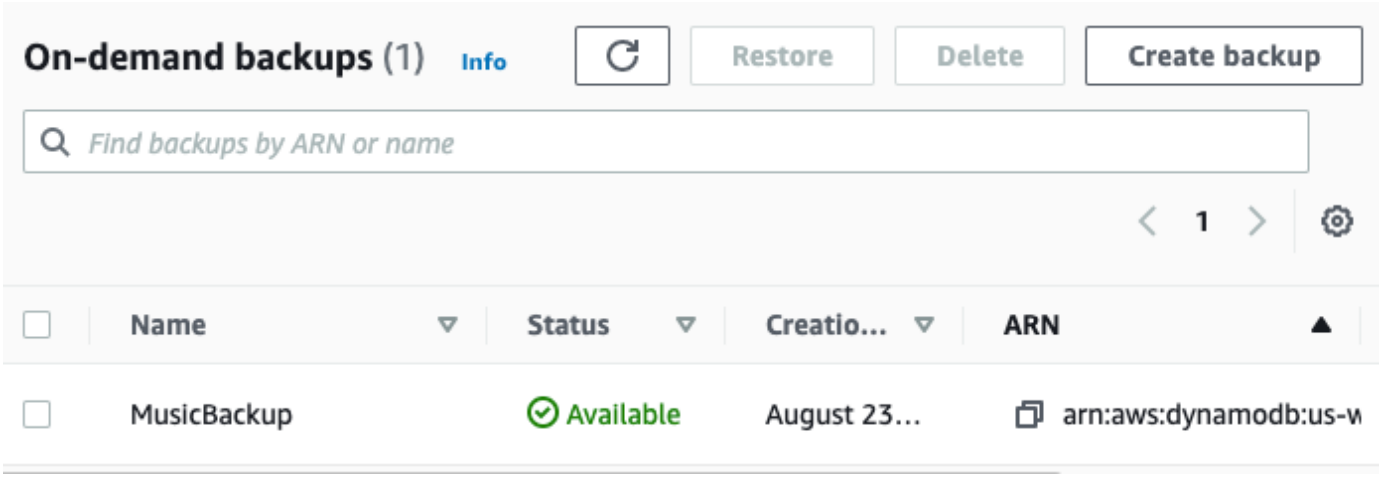
So erstellen Sie ein Tabellen-Backup

1. Melden Sie sich bei der an AWS-Managementkonsole und öffnen Sie die DynamoDB-Konsole unter. https://console.aws.amazon.com/dynamodb/
2. Sie können ein Backup mit einem der folgenden Verfahren erstellen:
   - Klicken Sie auf der Registerkarte Backups (Backups) der Tabelle `Music` auf Create backup (Backup erstellen).
   - Klicken Sie im Navigationsbereich auf der linken Seite der Konsole auf Backups (Backups). Wählen Sie Create backup (Backup erstellen) aus.
3. Stellen Sie sicher, dass `Music` der Tabellenname ist, und geben Sie als Backupnamen **`MusicBackup`** ein. Wählen Sie Sicherung erstellen aus, um das Backup zu erstellen.



> **Note**
>
> Wenn Sie Backups mit dem Abschnitt Backups im Navigationsbereich erstellen, wird die Tabelle nicht automatisch vorab ausgewählt. Sie müssen den Namen für die Quelltabelle des Backup manuell auswählen.

Während das Backup erstellt wird, lautet der Backupstatus Creating. Nachdem das Backup abgeschlossen wurde, ändert sich der Backupstatus in Available (Verfügbar).



Erstellen eines Tabellen-Backups (AWS CLI)

Gehen Sie wie folgt vor, um eine Backup für eine vorhandene `Music`-Tabelle mithilfe der AWS CLI zu erstellen.

So erstellen Sie ein Tabellen-Backup

- Erstellen Sie ein Backup mit dem Namen `MusicBackup` für die Tabelle`Music`.

```
aws dynamodb create-backup --table-name Music \
 --backup-name MusicBackup
```

Während das Backup erstellt wird, lautet der Backupstatus `CREATING`.

```
{
    "BackupDetails": {
        "BackupName": "MusicBackup",
        "BackupArn": "arn:aws:dynamodb:us-east-1:123456789012:table/Music/
backup/01489602797149-73d8d5bc",
        "BackupStatus": "CREATING",
        "BackupCreationDateTime": 1489602797.149
    }
}
```

Nachdem das Backup abgeschlossen wurde, sollte sich ihr `BackupStatus` in `AVAILABLE` ändern. Verwenden Sie den Befehl `describe-backup`, um den Status zu überprüfen. Sie können den Eingabewert von `backup-arn` aus der Ausgabe des vorherigen Schritts oder mit dem Befehl `list-backups` abrufen.

```
aws dynamodb describe-backup \
--backup-arn arn:aws:dynamodb:us-east-1:123456789012:table/Music/backup/01489173575360-
b308cd7d
```

Verwenden Sie den Befehl `list-backups`, um eine Übersicht über Ihre Backups zu erhalten. Es werden alle Backups mit dem Status `CREATING` oder `AVAILABLE` aufgeführt.

```
aws dynamodb list-backups
```

Die Befehle `list-backups` und `describe-backup` sind nützlich, um die Informationen über die Quelltabelle des Backups zu prüfen.

Wiederherstellen einer DynamoDB-Tabelle aus einem Backup

In diesem Abschnitt wird beschrieben, wie Sie eine Tabelle aus einer Sicherung mithilfe der Amazon DynamoDB DynamoDB-Konsole oder der AWS Command Line Interface()AWS CLI wiederherstellen.

> **Note**
>
> Wenn Sie das verwenden möchten AWS CLI, müssen Sie es zuerst konfigurieren. Weitere Informationen finden Sie unter Zugreifen auf DynamoDB.

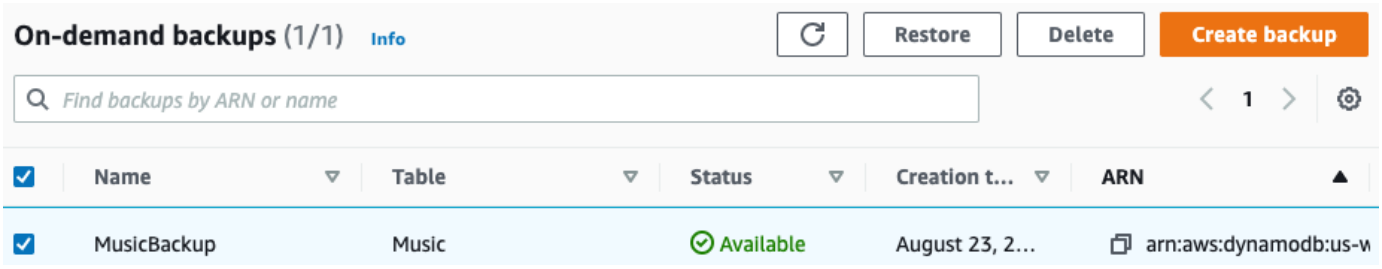
Wiederherstellen einer Tabelle aus einem Backup (Konsole)

Der folgende Vorgang zeigt, wie die Tabelle `Music` mit dem Backup `MusicBackup`, die im Tutorial Backup einer DynamoDB-Tabelle erstellt wurde, wiederhergestellt wird.

> **Note**
>
> Dieses Verfahren setzt voraus, dass die Tabelle `Music` nicht mehr vorhanden ist, bevor sie anhand der Datei `MusicBackup` wiederhergestellt wird.

So stellen Sie eine Tabelle aus einem Backup wieder her

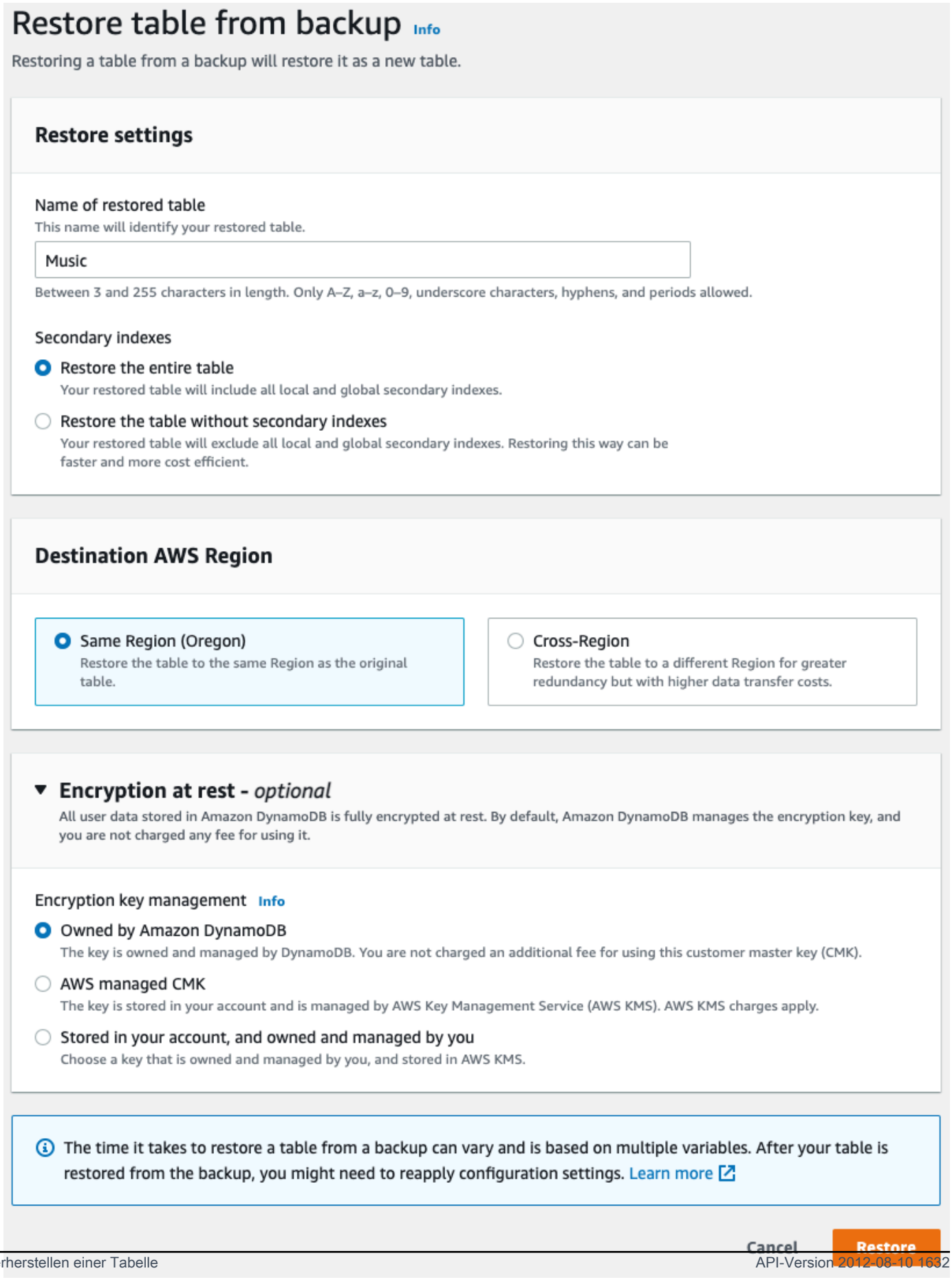
1. Melden Sie sich bei der an AWS-Managementkonsole und öffnen Sie die DynamoDB-Konsole unter. https://console.aws.amazon.com/dynamodb/
2. Klicken Sie im Navigationsbereich auf der linken Seite der Konsole auf Backups (Backups).
3. Wählen Sie aus der Backupliste die Datei `MusicBackup` aus.



4. Wählen Sie Restore (Wiederherstellen).

5. Geben Sie als neuen Tabellennamen **`Music`** ein. Überprüfen Sie den Namen und weitere Details des Backups. Wählen Sie dann Restore table (Tabelle wiederherstellen) aus, um den Wiederherstellungsvorgang zu starten.

> **Note**
>
> Sie können die Tabelle in derselben AWS Region oder in einer anderen Region wiederherstellen, in der sich das Backup befindet. Sie können auch sekundäre Indizes von der Erstellung für die neu wiederhergestellte Tabelle ausschließen. Darüber hinaus können Sie einen anderen Verschlüsselungsmodus angeben.
> Aus Backups wiederhergestellte Tabellen werden immer mit der Tabellenklasse DynamoDB Standard erstellt.


Restore table from backup Info
Restoring a table from a backup will restore it as a new table.
Restore settings
Name of restored table
This name will identify your restored table.
Music
Between 3 and 255 characters in length. Only A–Z, a–z, 0–9, underscore characters, hyphens, and periods allowed.
Secondary indexes
Restore the entire table
Your restored table will include all local and global secondary indexes.
Restore the table without secondary indexes
Your restored table will exclude all local and global secondary indexes. Restoring this way can be faster and more cost efficient.
Destination AWS Region
Same Region (Oregon)
Restore the table to the same Region as the original table.
Cross-Region
Restore the table to a different Region for greater redundancy but with higher data transfer costs.
Encryption at rest - optional
All user data stored in Amazon DynamoDB is fully encrypted at rest. By default, Amazon DynamoDB manages the encryption key, and you are not charged any fee for using it.
Encryption key management Info
Owned by Amazon DynamoDB
The key is owned and managed by DynamoDB. You are not charged an additional fee for using this customer master key (CMK).
AWS managed CMK
The key is stored in your account and is managed by AWS Key Management Service (AWS KMS). AWS KMS charges apply.
Stored in your account, and owned and managed by you
Choose a key that is owned and managed by you, and stored in AWS KMS.
The time it takes to restore a table from a backup can vary and is based on multiple variables. After your table is restored from the backup, you might need to reapply configuration settings. Learn more
Cancel
Restore

Die Tabelle, die wiederhergestellt wird, erhält den Status Creating. Nach Abschluss des Wiederherstellungsvorgangs ändert sich der Status der Tabelle `Music` in Active (Aktiv).

Wiederherstellen einer Tabelle aus einem Backup (AWS CLI)

Gehen Sie wie folgt vor AWS CLI, um die `Music` Tabelle mithilfe der wiederherzustellen`MusicBackup`, die im Backup einer DynamoDB-Tabelle Tutorial erstellt wurde.

So stellen Sie eine Tabelle aus einem Backup wieder her

1. Bestätigen Sie das Backup, das Sie wiederherstellen möchten, indem Sie den Befehl `list-backups` verwenden. Dieses Beispiel verwendet `MusicBackup`.

   ```
   aws dynamodb list-backups
   ```

   Verwenden Sie den Befehl `describe-backup`, um weitere Details für das Backup abzurufen. Sie können die Eingabe `backup-arn` mit dem vorherigen Schritt ermitteln.

   ```
   aws dynamodb describe-backup \
   --backup-arn arn:aws:dynamodb:us-east-1:123456789012:table/Music/
   backup/01489173575360-b308cd7d
   ```

2. Stellen Sie die Tabelle aus dem Backup wieder her. In diesem Fall `MusicBackup` stellt der die `Music` Tabelle in derselben AWS Region wieder her.

   ```
   aws dynamodb restore-table-from-backup \
   --target-table-name Music \
   --backup-arn arn:aws:dynamodb:us-east-1:123456789012:table/Music/
   backup/01489173575360-b308cd7d
   ```

3. Stellen Sie die Tabelle mit benutzerdefinierten Tabelleneinstellungen aus dem Backup wieder her. In diesem Fall wird aus dem Backup `MusicBackup` die Tabelle `Music` wiederhergestellt und ein Verschlüsselungsmodus für die wiederhergestellte Tabelle angegeben.

   > **Note**
   >
   > Der Parameter `sse-specification-override` verwendet dieselben Werte wie der im Befehl `CreateTable` verwendete Parameter `sse-specification-override`.

Weitere Informationen hierzu finden Sie unter Verwalten von verschlüsselten Tabellen in DynamoDB.

```
aws dynamodb restore-table-from-backup \
--target-table-name Music \
--backup-arn arn:aws:dynamodb:us-east-1:123456789012:table/Music/
backup/01581080476474-e177ebe2 \
--sse-specification-override Enabled=true,SSEType=KMS,KMSMasterKeyId=abcd1234-
abcd-1234-a123-ab1234a1b234
```

Sie können die Tabelle in einer anderen AWS Region wiederherstellen, als in der sich das Backup befindet.

Note

- Der Parameter `sse-specification-override` ist für bereichsübergreifende Wiederherstellungen obligatorisch, für Wiederherstellungen in derselben Region wie die Quelltabelle aber optional.
- Wenn Sie eine regionsübergreifende Wiederherstellung von der Befehlszeile aus durchführen, müssen Sie die AWS Standardregion auf die gewünschte Zielregion festlegen. Weitere Informationen finden Sie unter Befehlszeilenoptionen im AWS Command Line Interface-Benutzerhandbuch.

```
aws dynamodb restore-table-from-backup \
--target-table-name Music \
--backup-arn arn:aws:dynamodb:us-east-1:123456789012:table/Music/
backup/01581080476474-e177ebe2 \
--sse-specification-override Enabled=true,SSEType=KMS
```

Sie können den Abrechnungsmodus und den bereitgestellten Durchsatz für die wiederhergestellte Tabelle überschreiben.

```
aws dynamodb restore-table-from-backup \
--target-table-name Music \
--backup-arn arn:aws:dynamodb:us-east-1:123456789012:table/Music/
backup/01489173575360-b308cd7d \
```

```
--billing-mode-override PAY_PER_REQUEST
```

Sie können einige oder alle sekundären Indizes von der Erstellung für die neu wiederhergestellte Tabelle ausschließen.

> **Note**
>
> Wiederherstellungen können schneller und kosteneffizienter sei, wenn Sie die Erstellung einiger oder aller sekundärer Indizes für die wiederhergestellte Tabelle ausschließen.

```
aws dynamodb restore-table-from-backup \
--target-table-name Music \
--backup-arn arn:aws:dynamodb:us-east-1:123456789012:table/Music/
backup/01581081403719-db9c1f91 \
--global-secondary-index-override '[]' \
--sse-specification-override Enabled=true,SSEType=KMS
```

> **Note**
>
> Die angegebenen sekundären Indizes sollten vorhandenen Indizes entsprechen. Sie können keine neuen Indizes zum Zeitpunkt der Wiederherstellung erstellen.

Sie können eine Kombination verschiedener Überschreibungen verwenden. Sie können beispielsweise einen einzelnen globalen sekundären Index verwenden und gleichzeitig den bereitgestellten Durchsatz wie folgt ändern.

```
aws dynamodb restore-table-from-backup \
--target-table-name Music \
--backup-arn arn:aws:dynamodb:eu-west-1:123456789012:table/Music/
backup/01581082594992-303b6239 \
--billing-mode-override PROVISIONED \
--provisioned-throughput-override ReadCapacityUnits=100,WriteCapacityUnits=100 \
--global-secondary-index-override IndexName=singers-
index,KeySchema=["{AttributeName=SingerName,KeyType=HASH}"],Projection="{ProjectionType=KEYS_
 \
--sse-specification-override Enabled=true,SSEType=KMS
```

Verwenden Sie zum Überprüfen der Wiederherstellung den Befehl `describe-table`, um die Tabelle `Music` zu beschreiben.

```
aws dynamodb describe-table --table-name Music
```

Die Tabelle, die aus dem Backup wiederhergestellt wird, erhält den Status Creating. Nach Abschluss des Wiederherstellungsvorgangs ändert sich der Status der Tabelle `Music` in Active (Aktiv).

Important

Ändern oder löschen Sie die IAM-Rollenrichtlinie nicht, während eine Wiederherstellung ausgeführt wird. Andernfalls kann es zu unerwartetem Verhalten kommen. Angenommen, Sie haben die Schreibberechtigungen für eine Tabelle entfernt, während diese Tabelle wiederhergestellt wurde. In diesem Fall kann die zugrunde liegende `RestoreTableFromBackup`-Operation keine wiederhergestellten Daten in die Tabelle schreiben.

Nachdem die Wiederherstellung abgeschlossen ist, können Sie Ihre IAM-Rollenrichtlinie ändern oder löschen.

IAM-Richtlinien, die Quell-IP-Einschränkungen für den Zugriff auf die Zielwiederherstellungstabelle beinhalten, sollten den `aws:ViaAWSService` Schlüssel zu `false` festgelegt haben, um sicherzustellen, dass die Einschränkungen nur für Anforderungen gelten, die direkt von einem Prinzipal gestellt werden. Andernfalls wird die Wiederherstellung abgebrochen.

Wenn Ihr Backup mit einem Von AWS verwalteter Schlüssel oder einem vom Kunden verwalteten Schlüssel verschlüsselt ist, deaktivieren oder löschen Sie den Schlüssel nicht, während eine Wiederherstellung läuft, da sonst die Wiederherstellung fehlschlägt.

Nach Abschluss der Wiederherstellungsoperation können Sie den Verschlüsselungsschlüssel der wiederhergestellten Tabelle ändern und den alten Schlüssel deaktivieren oder löschen.

Löschen eines DynamoDB-Tabellen-Backups

In diesem Abschnitt wird beschrieben, wie Sie das AWS-Managementkonsole oder das AWS Command Line Interface(AWS CLI) verwenden, um eine Amazon DynamoDB-Tabellensicherung zu löschen.

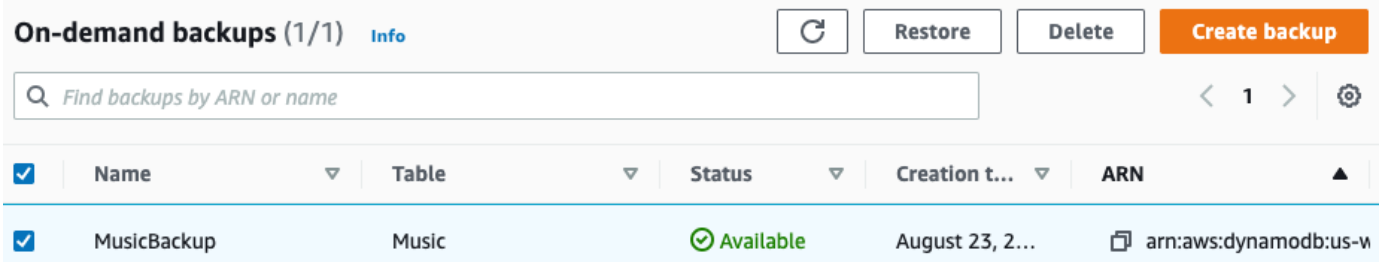
> **Note**
>
> Wenn Sie das verwenden möchten AWS CLI, müssen Sie es zuerst konfigurieren. Weitere Informationen finden Sie unter Mit dem AWS CLI.

Löschen eines Tabellen-Backups (Konsole)

Das folgende Verfahren zeigt, wie Sie mithilfe der Konsole das Backup `MusicBackup` löschen, das im Backup einer DynamoDB-Tabelle-Tutorial erstellt wurde.

So löschen Sie ein Backup

1. Melden Sie sich bei der an AWS-Managementkonsole und öffnen Sie die DynamoDB-Konsole unter. https://console.aws.amazon.com/dynamodb/
2. Klicken Sie im Navigationsbereich auf der linken Seite der Konsole auf Backups (Backups).
3. Wählen Sie aus der Backupliste die Datei `MusicBackup` aus.



4. Wählen Sie Löschen aus. Bestätigen Sie, dass Sie das Backup löschen möchten, indem Sie **`delete`** eingeben und auf Delete (Löschen) klicken.

Löschen eines Tabellen-Backups (AWS CLI)

Das folgende Beispiel löscht ein Backup der vorhandenen Tabelle `Music` mithilfe der AWS CLI.

```
aws dynamodb delete-backup \
--backup-arn arn:aws:dynamodb:us-east-1:123456789012:table/Music/
backup/01489602797149-73d8d5bc
```

Verwenden von IAM mit DynamoDB-Backup und -Wiederherstellung

Sie können AWS Identity and Access Management (IAM) verwenden, um die Sicherungs- und Wiederherstellungsaktionen von Amazon DynamoDB für einige Ressourcen einzuschränken. Die `CreateBackup` und werden für `RestoreTableFromBackup` APIs jede Tabelle einzeln ausgeführt.

Weitere Informationen zur Verwendung der IAM-Richtlinien in DynamoDB finden Sie unter Identitätsbasierte Richtlinien für DynamoDB.

Es folgen Beispiele für die IAM-Richtlinien, die Sie zum Konfigurieren bestimmter Backup- und Wiederherstellungsfunktionen in DynamoDB verwenden können.

Beispiel 1: Die Aktionen CreateBackup und RestoreTableFromBackup zulassen

Mit der folgenden IAM-Richtlinie werden die Berechtigungen zum Zulassen der DynamoDB-Aktionen `CreateBackup` und `RestoreTableFromBackup` für alle Tabellen erteilt:

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "dynamodb:CreateBackup",
                "dynamodb:RestoreTableFromBackup",
                "dynamodb:PutItem",
                "dynamodb:UpdateItem",
                "dynamodb:DeleteItem",
                "dynamodb:GetItem",
                "dynamodb:Query",
                "dynamodb:Scan",
                "dynamodb:BatchWriteItem"
            ],
            "Resource": "*"
        }
    ]
}
```

> **Important**
>
> RestoreTableFromBackup DynamoDB-Berechtigungen sind für das Quell-Backup erforderlich, und DynamoDB-Lese- und Schreibberechtigungen für die Zieltabelle sind für die Wiederherstellungsfunktion erforderlich.
> RestoreTableToPointInTime DynamoDB-Berechtigungen sind für die Quelltabelle erforderlich, und DynamoDB-Lese- und Schreibberechtigungen für die Zieltabelle sind für die Wiederherstellungsfunktion erforderlich.

Beispiel 2: Zulassen und Verweigern CreateBackup RestoreTableFromBackup

Mit der folgenden IAM-Richtlinie werden die Berechtigungen für die Aktion `CreateBackup` erteilt und für die Aktion `RestoreTableFromBackup` verweigert:

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": ["dynamodb:CreateBackup"],
            "Resource": "*"
        },
        {
            "Effect": "Deny",
            "Action": ["dynamodb:RestoreTableFromBackup"],
            "Resource": "*"
        }

    ]
}
```

Beispiel 3: Zulassen ListBackups und verweigern CreateBackup und RestoreTableFromBackup

Mit der folgenden IAM-Richtlinie werden die Berechtigungen für die Aktion `ListBackups` erteilt und für die Aktionen `CreateBackup` und `RestoreTableFromBackup` verweigert:

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": ["dynamodb:ListBackups"],
            "Resource": "*"
        },
        {
            "Effect": "Deny",
            "Action": [
                "dynamodb:CreateBackup",
                "dynamodb:RestoreTableFromBackup"
            ],
            "Resource": "*"
        }

    ]
}
```

Beispiel 4: Zulassen ListBackups und Verweigern DeleteBackup

Mit der folgenden IAM-Richtlinie werden die Berechtigungen für die Aktion `ListBackups` erteilt und für die Aktion `DeleteBackup` verweigert:

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": ["dynamodb:ListBackups"],
            "Resource": "*"
        },
        {
            "Effect": "Deny",
            "Action": ["dynamodb:DeleteBackup"],
```

```
            "Resource": "*"
        }

    ]
}
```

Beispiel 5: Zulassen RestoreTableFromBackup und DescribeBackup für alle Ressourcen und Verweigern DeleteBackup für ein bestimmtes Backup

Mit der folgenden IAM-Richtlinie werden die Berechtigungen für die Aktionen `RestoreTableFromBackup` und `DescribeBackup` erteilt und die Aktion `DeleteBackup` für eine bestimmte Backupressource abgelehnt:

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "dynamodb:DescribeBackup",
                "dynamodb:RestoreTableFromBackup"
            ],
            "Resource": "arn:aws:dynamodb:us-east-1:123456789012:table/Music/
backup/01489173575360-b308cd7d"
        },
        {
            "Effect": "Allow",
            "Action": [
                "dynamodb:PutItem",
                "dynamodb:UpdateItem",
                "dynamodb:DeleteItem",
                "dynamodb:GetItem",
                "dynamodb:Query",
                "dynamodb:Scan",
                "dynamodb:BatchWriteItem"
            ],
            "Resource": "*"
        },
        {
```

```
            "Effect": "Deny",
            "Action": [
                "dynamodb:DeleteBackup"
            ],
            "Resource": "arn:aws:dynamodb:us-east-1:123456789012:table/Music/
backup/01489173575360-b308cd7d"
        }
    ]
}
```

> **Important**
>
> RestoreTableFromBackup DynamoDB-Berechtigungen sind für das Quell-Backup erforderlich, und DynamoDB-Lese- und Schreibberechtigungen für die Zieltabelle sind für die Wiederherstellungsfunktion erforderlich.
> RestoreTableToPointInTime DynamoDB-Berechtigungen sind für die Quelltabelle erforderlich, und DynamoDB-Lese- und Schreibberechtigungen für die Zieltabelle sind für die Wiederherstellungsfunktion erforderlich.

Beispiel 6: Erlauben Sie eine bestimmte Tabelle CreateBackup

Mit der folgenden IAM-Richtlinie werden die Berechtigungen für die Aktion `CreateBackup` nur für die Tabelle `Movies` erteilt:

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": ["dynamodb:CreateBackup"],
            "Resource": [
                "arn:aws:dynamodb:us-east-1:123456789012:table/Movies"
            ]
        }
    ]
}
```

Beispiel 7: Zulassen ListBackups

Die folgende IAM-Richtlinie erteilt Berechtigungen für die `ListBackups` Aktion:

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": ["dynamodb:ListBackups"],
            "Resource": "*"
        }
    ]
}
```

> **Important**
>
> Sie können keine Berechtigungen für die Aktion `ListBackups` für eine bestimmte Tabelle erteilen.

Beispiel 8: Zugriff auf AWS Backup Funktionen zulassen

Sie benötigen API-Berechtigungen für die `StartAwsBackupJob`-Aktion für ein erfolgreiches Backup mit erweiterten Funktionen und die `dynamodb:RestoreTableFromAwsBackup`-Aktion, um dieses Backup erfolgreich wiederherzustellen.

Die folgende IAM-Richtlinie gewährt AWS Backup die Berechtigungen zum Auslösen von Backups mit erweiterten Funktionen und Wiederherstellungen. Beachten Sie auch, dass die Richtlinie Zugriff auf den AWS -KMS-Schlüssel benötigt, wenn die Tabellen verschlüsselt sind.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
```

```
            "Sid": "DescribeQueryScanBooksTable",
            "Effect": "Allow",
            "Action": [
                "dynamodb:StartAwsBackupJob",
                "dynamodb:DescribeTable",
                "dynamodb:Query",
                "dynamodb:Scan"
            ],
            "Resource": "arn:aws:dynamodb:us-west-2:111122223333:table/Books"
        },
        {
            "Sid": "AllowRestoreFromAwsBackup",
            "Effect": "Allow",
            "Action": [
                "dynamodb:RestoreTableFromAwsBackup"
            ],
            "Resource": "*"
        }
    ]
}
```

Beispiel 9: RestoreTableToPointInTime Für eine bestimmte Quelltabelle verweigern

Die folgende IAM-Richtlinie verweigert Berechtigungen für die Aktion `RestoreTableToPointInTime` für eine bestimmte Quelltabelle:

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Deny",
            "Action": [
                "dynamodb:RestoreTableToPointInTime"
            ],
            "Resource": "arn:aws:dynamodb:us-east-1:123456789012:table/Music"
        }
    ]
}
```

Beispiel 10: Alle Backups für eine bestimmte Quelltabelle ablehnen RestoreTableFromBackup

Die folgende IAM-Richtlinie verweigert Berechtigungen für die Aktion `RestoreTableToPointInTime` für alle Backups für eine bestimmte Quelltabelle:

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Deny",
            "Action": [
                "dynamodb:RestoreTableFromBackup"
            ],
            "Resource": "arn:aws:dynamodb:us-east-1:123456789012:table/Music/
backup/*"
        }
    ]
}
```

Grundlegendes zur Amazon-DynamoDB-Fakturierung für Backups

Dieses Handbuch enthält Einzelheiten zur Funktionsweise der DynamoDB-Fakturierung für Backups. Wir werden die verschiedenen Komponenten, die zu den Gesamtkosten beitragen, aufschlüsseln und klare Erklärungen und praktische Beispiele geben.

DynamoDB bietet On-Demand-Backups und Backups für die zeitpunktbezogene Wiederherstellung (PITR), um Ihre DynamoDB-Daten bei Notfallereignissen zu schützen. Außerdem gibt es die Datenarchivierung für die langfristige Aufbewahrung.

Funktionsweise

DynamoDB-On-Demand-Backups werden monatlich berechnet. Wenn Sie an einem bestimmten Tag des Monats ein Backup erstellen, wird eine einzelne Gebühr für dieses Backup für die verbleibenden Tage des Monats berechnet. Beispiel: Wenn Sie ein Backup am 27. erstellen, werden Ihnen nur

die wenigen verbleibenden Tage in diesem Monat in Rechnung gestellt. Diese werden am 27. als Einzelgebühr berechnet.

Wenn Sie Ihre zuvor erstellten Backups für die folgenden Monate beibehalten, wird Ihnen immer am 1. des Monats eine volle Monatsgebühr für dieses Backup berechnet. Wenn das Backup vor Monatsende entfernt wird, werden die Gebühren auf der Grundlage der tatsächlichen Nutzung angepasst.

Wenn Sie beispielsweise am 27. Juli ein Backup erstellt haben und dieses über den gesamten August beibehalten, fallen für dieses Backup die folgenden Gebühren an:

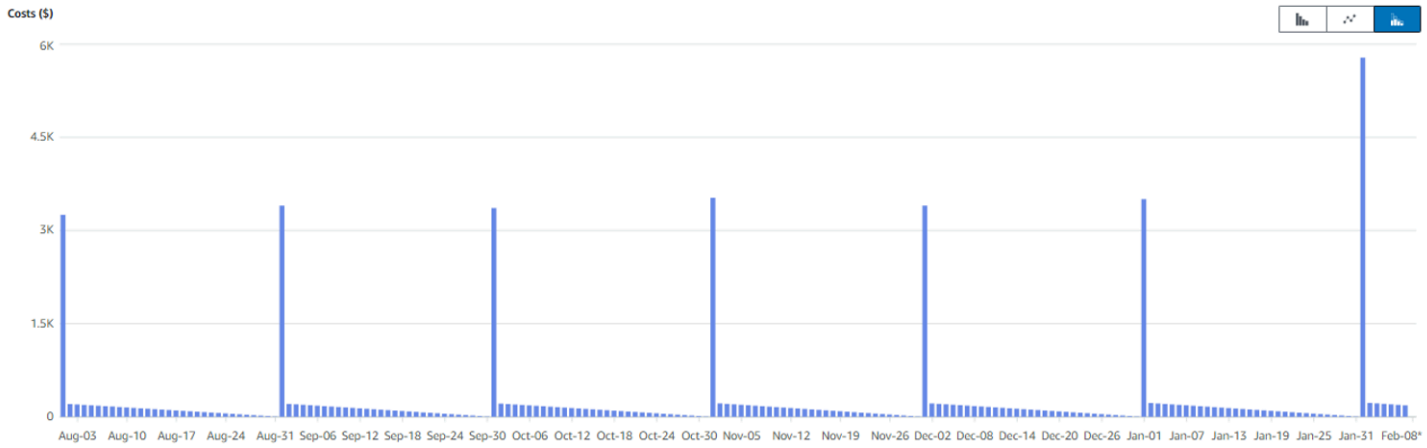
- Eine Gebühr am 27. Juli für die verbleibenden Tage im Juli
- Eine Gebühr am 1. August für den gesamten Monat August
- Eine Gebühr am 1. jedes Folgemonats, in dem dieses Backup beibehalten wird.

Wenn Backups für DynamoDB-Tabellen verwaltet werden, stellen Sie möglicherweise fest, dass die Kosten für die `DynamoDB (Region)-TimedBackupStorage-ByteHrs`-Nutzungsmetrik am Monatsersten ungewöhnlich hoch erscheinen. Wenn Sie diese Metrik zu Beginn eines neuen Monats überprüfen und sie mit früheren Abrechnungszeiträumen vergleichen, stellen Sie außerdem möglicherweise fest, dass die Nutzung stark anzusteigen scheint. Dies ist beabsichtigt. Am 1. jeden Monats fallen für alle vorhandenen DynamoDB-Backups Nutzungsgebühren für den gesamten Monat an. Für alle DynamoDB-Backups, die im Laufe des Monats entfernt werden, werden die Nutzungskosten anteilig entsprechend der tatsächlichen Nutzung berechnet. Infolgedessen kann es sein, dass die Gebühr (die am ersten Tag des Monats berechnet wird) im Laufe des Monats sinkt. Dies liegt daran, dass Aufbewahrungsrichtlinien Ablaufdaten anwenden oder bei aus dem Vormonat übernommenen Backups manuelle Löschungen vorgenommen werden. Dies wird im Folgenden anhand eines Szenarios erläutert.

In ähnlicher Weise werden Sie im Laufe des Monats kleinere Spitzen feststellen, wenn neue Backups erstellt werden, deren Gebühren am Tag der Erstellung für den Rest des Monats anfallen.

Beispiel für die Fakturierung von DynamoDB-Backups

Hier finden Sie ein Beispiel für Gebühren, die Sie am Anfang des Monats in Cost Explorer sehen könnten:



Sie sehen, dass der 1. Februar im Vergleich zu den Vormonaten offenbar einen viel größeren Anstieg aufweist. Lassen Sie uns aufschlüsseln, warum das passiert.

Von der DynamoDB-Preisseite:

„Der gesamte Speicherbedarf für die Sicherung, der Ihnen monatlich in Rechnung gestellt wird, besteht aus der Summe aller Sicherungen von DynamoDB-Tabellen. DynamoDB überwacht laufend den Monat über die Größe der On-Demand-Sicherungen, woraus die Gebühren für die Sicherung ermittelt werden.“

Dies erklärt, warum in der Rechnung immer am 1. eines Monats ein starker Anstieg der Nutzung verzeichnet wird. Für alle bestehenden Backups, die einen weiteren Monat beibehalten werden, fallen am Monatsersten die vollen Monatsgebühren an. Anders ausgedrückt: Wenn Sie den Monat mit 300 DynamoDB-Backups beginnen, werden für alle 300 Backups die Nutzungsgebühren für den ganzen Monat am 1. des Monats berechnet.

Für neue Backups, die im Laufe des Monats erstellt werden, fallen vom Tag der Erstellung bis zum Monatsende Nutzungsgebühren an.

Was passiert, wenn das Backup in der Mitte des Monats gelöscht wird?

Hier sind einige Szenarien, die Sie berücksichtigen sollten:

1. Wenn ein Backup aus dem Vormonat am 15. des aktuellen Monats gelöscht wird, werden die Nutzungsgebühren für dieses Backup, die am 1. des Monat erhoben wurden, an die tatsächliche Nutzung angepasst und es wird nicht der angenommene volle Nutzungsmonat, der zuvor berechnet wurde, in Rechnung gestellt. Das folgende Beispiel erklärt dies ausführlicher.
2. Wenn Sie im Laufe des Monats ein Backup erstellen, werden die Nutzungsgebühren für den Rest des Monats an dem Tag berechnet, an dem es erstellt wurde. Wenn Sie dieses Backup jedoch

vor Monatsende löschen, werden Ihre Nutzungsgebühren so angepasst, dass sie nur die Tage umfassen, an denen das Backup aktiv war, und weiter am Erstellungsdatum erhoben.

Warum scheint die Nutzung für den aktuellen Monat am Monatsersten so viel höher zu sein als in den Vormonaten und was passiert, wenn ich die Backups entferne?

Um diese wichtige zweiteilige Frage zu beantworten, lassen Sie uns anhand der folgenden Informationen ein Beispielszenario entwerfen:

- Länge des Monats: 30 Tage
- Häufigkeit der DynamoDB-Backups: 10 pro Tag, 300 pro Monat
- Aufbewahrungsrichtlinie für DynamoDB-Backups: 30 Tage
- Kosten pro DynamoDB-Backup: 2 $/Tag, 60 $/Monat
- Gesamtbetrag am letzten Monatsersten (`TimedBackupStorage-ByteHrs`, am 1. des aktuellen Monats geprüft): 9 300 $
- Summe des Vormonats (`TimedBackupStorage-ByteHrs`): 18 600 $
- Gesamtbetrag am aktuellen Monatsersten (`TimedBackupStorage-ByteHrs`, am 1. des Monats geprüft): 18 000 $
- Änderungen der DynamoDB-Nutzung von Monat zu Monat: Keine

Anhand der obigen Informationen können wir erkennen, dass im Vormonat 300 Backups erstellt wurden, die laut Richtlinie 30 Tage lang aufbewahrt werden. Am 1. eines neuen Monats bleiben all diese Backups weiterhin bestehen, da sie das Ende ihrer Wiederherstellungszeit noch nicht erreicht haben. Mit jedem Tag, der vergeht, werden jedoch die ältesten Backups gelöscht, wie hier gezeigt:

Dropoff-Tabelle für DynamoDB-Backups

Neuer Monat	Tag 1	Tag 2	Tag 3	Tag 4	Tag 5
Gesamtzahl der aus dem Vormonat übernommenen Backups	300	290	280	270	260

- Am Monatsersten können wir 300 Backups mit einem Preis von 60 $ pro Monat pro Backup sehen, was insgesamt 18 000 $ für `TimedBackupStorage-ByteHrs` entspricht. Dies steht im Gegensatz zum Vormonat, wo der Gesamtbetrag für den ganzen Monat 18 600 $ war.
- Am 2. werden 10 dieser Backups abgelaufen sein und gelöscht werden. Wenn dies geschieht, werden die Gebühren für diese Backups auf die tatsächliche Nutzung und nicht die angenommene Nutzung aktualisiert. Dies führt dazu, dass diese 10 Backups, für die zuvor am Monatsersten eine Gebühr von 600 $ (10 Backups x 30 Tage) berechnet wurde, auf 20 $ (10 Backups x 1 Tag) gesenkt werden.
- Am darauffolgenden Tag läuft der nächste Satz von 10 Backups ab und wird gelöscht, wodurch die Nutzung von 30 Tagen auf 2 Tage und die Gebühr auf 40 $ (10 Backups x 2 Tage) sinkt.

Mit jedem Tag, der vergeht, werden wir feststellen, dass der Anstieg, der größer als im Vormonat war, allmählich abnimmt. Wenn wir dies auf den gesamten Monat ausweiten, können wir Folgendes beobachten:

Entwicklung der DynamoDB-Backup-Gebühren (1. Tag des Monats)

300 Backups in 10er-Blöcken	1.	10.	20.	30.
Block 1	600 $	20 $	20 $	20 $
Block 2	600 $	40 $	40 $	40 $
Block 3	600 $	60 $	60 $	60 $
Block 4	600 $	80 $	80 $	80 $
Block 5	600 $	100 $	100 $	100 $
Block 6	600 $	120 $	120 $	120 $
Block 7	600 $	140 $	140 $	140 $
Block 8	600 $	160 $	160 $	160 $
Block 9	600 $	180 $	180 $	180 $
Block 10	600 $	600 $	200 $	200 $
Block 11	600 $	600 $	220 $	220 $

300 Backups in 10er-Blöcken	1.	10.	20.	30.
Block 12	600 $	600 $	240 $	240 $
Block 13	600 $	600 $	260 $	260 $
Block 14	600 $	600 $	280 $	280 $
Block 15	600 $	600 $	300 $	300 $
Block 16	600 $	600 $	320 $	320 $
Block 17	600 $	600 $	340 $	340 $
Block 18	600 $	600 $	360 $	360 $
Block 19	600 $	600 $	380 $	380 $
Block 20	600 $	600 $	600 $	400 $
Block 21	600 $	600 $	600 $	420 $
Block 22	600 $	600 $	600 $	440 $
Block 23	600 $	600 $	600 $	460 $
Block 24	600 $	600 $	600 $	480 $
Block 25	600 $	600 $	600 $	500 $
Block 26	600 $	600 $	600 $	520 $
Block 27	600 $	600 $	600 $	540 $
Block 28	600 $	600 $	600 $	560 $
Block 29	600 $	600 $	600 $	580 $
Block 30	600 $	600 $	600 $	600 $
Gesamtbetrag am 1. des Monats ($)	18 000 $	13 500 $	10 400 $	9 300 $

Da jeden Tag ein neuer Block gelöscht wird, wird seine Nutzung an die Anzahl der Tage angepasst, an denen er vorhanden war, und nicht der Gesamtbetrag für den Monat angegeben. Das hat zur Folge, dass am Monatsende die Gebühren, die am 1. des Monats angegeben wurden, von den anfänglichen 18 000 $ auf die erwarteten 9 300 $ gesunken sein werden. Diese Zahl, kombiniert mit den im Laufe des Monats neu erstellten Backups (für die eine Abrechnungstabelle ähnlich der oben genannten, aber mit umgekehrtem Verlauf vorhanden sein wird), wird zu monatlichen Ausgaben führen, die den 18 600 $ des Vormonats entsprechen.

Wiederherstellen einer Tabelle in DynamoDB

Sie können eine DynamoDB-Tabelle aus Ihrem PITR-Backup oder Ihren On-Demand-Backups mithilfe der AWS-Managementkonsole AWS Befehlszeilenschnittstelle (AWS CLI) oder der DynamoDB-API wiederherstellen. Der Wiederherstellungsprozess stellt die Daten in einer neuen DynamoDB-Tabelle wieder her.

Eine Tabelle mithilfe der Wiederherstellung wiederherstellen point-in-time

Sie können Ihre Tabelle zu jedem beliebigen Zeitpunkt bis zum `EarliestRestoreableDateTime` wiederherstellen.

> **Important**
>
> Wenn Sie die point-in-time Wiederherstellung deaktivieren und später für eine Tabelle aktivieren, setzen Sie die Startzeit zurück, für die Sie diese Tabelle wiederherstellen können. Daher können Sie die Tabelle nur mit `LatestRestorableDateTime` sofort wiederherstellen.

Wenn Sie mithilfe point-in-time der Wiederherstellung wiederherstellen, stellt DynamoDB Ihre Tabellendaten auf den Status zurück, der auf dem ausgewählten Datum und der ausgewählten Uhrzeit (Tag:Stunde:Minute:Sekunde) in einer neuen Tabelle basiert. Sie stellen eine Tabelle wieder her, ohne den bereitgestellten Durchsatz für die Tabelle zu beanspruchen. Sie können mithilfe der Wiederherstellung eine vollständige point-in-time Tabellenwiederherstellung durchführen oder die Einstellungen der Zieltabelle konfigurieren. Sie können die folgenden Tabelleneinstellungen für die wiederhergestellte Tabelle ändern:

- Globale sekundäre Indizes () GSIs

- Lokale Sekundärindizes () LSIs
- Fakturierungsmodus
- Bereitgestellte Lese- und Schreibkapazität
- Verschlüsselungseinstellungen

> **Important**
>
> Bei einer vollständigen Wiederherstellung der Tabelle werden für die Zieltabelle die gleichen bereitgestellten Lese- und Schreibkapazitätseinheiten festgelegt, die für Quelltabelle festgelegt waren, als die Backupanforderung erfasst wurden. Beispiel: Angenommen, der bereitgestellte Durchsatz einer Tabelle wurde jüngst auf 50 Lese- und 50 Schreibkapazitätseinheiten verringert. Anschließend stellen Sie den Zustand dieser Tabelle vor drei Wochen wieder her. Dabei lag der bereitgestellte Durchsatz zu diesem Zeitpunkt bei 100 Lesekapazitätseinheiten und bei 100 Schreibkapazitätseinheiten. In diesem Fall setzt DynamoDB Ihre Tabellendaten auf den Zustand dieses bestimmten Zeitpunkts mit dem bereitgestellten Durchsatz dieses Zeitpunkts zurück (100 Lesekapazitätseinheiten und 100 Schreibkapazitätseinheiten).

Sie können Ihre DynamoDB-Tabellendaten auch AWS-Regionen so wiederherstellen, dass die wiederhergestellte Tabelle in einer anderen Region als der Region erstellt wird, in der sich die Quelltabelle befindet. Sie können regionsübergreifende Wiederherstellungen zwischen AWS Handelsregionen, AWS China Regionen und AWS GovCloud (US) durchführen. Sie zahlen nur für die Daten, die Sie aus der Quellregion übertragen, und für die Wiederherstellung in einer neuen Tabelle in der Zielregion.

> **Note**
>
> Eine regionsübergreifende Wiederherstellung wird nicht unterstützt, wenn die Quell- oder Zielregion Asien-Pazifik (Hongkong) oder Naher Osten (Bahrain) ist.

Wiederherstellungen können schneller und kosteneffizienter sei, wenn Sie die Erstellung einiger oder aller Indizes für die wiederhergestellte Tabelle ausschließen. Sie müssen für die wiederhergestellte Tabelle manuell Folgendes einrichten:

- Auto Scaling-Richtlinien

- AWS Identity and Access Management Richtlinien
- Kennzahlen und Alarme von Amazon CloudWatch Events
- Tags (Markierungen)
- Stream-Einstellungen
- Einstellungen für Gültigkeitsdauer (TTL)
- Point-in-time Einstellungen für die Wiederherstellung

Die Zeit, die Sie zum Wiederherstellen einer Tabelle benötigen, hängt von mehreren Faktoren ab und korreliert nicht immer mit der Größe der Tabelle.

Wiederherstellen einer DynamoDB-Tabelle auf einen bestimmten Zeitpunkt

Amazon DynamoDB point-in-time Recovery (PITR) bietet kontinuierliche Backups Ihrer DynamoDB-Tabellendaten. Sie können eine Tabelle mit der DynamoDB-Konsole oder AWS Command Line Interface (AWS CLI) im Zustand eines bestimmten Zeitpunkts wiederherstellen. Der point-in-time Wiederherstellungsprozess wird in einer neuen Tabelle wiederhergestellt.

Wenn Sie den verwenden möchten AWS CLI, müssen Sie ihn zuerst konfigurieren. Weitere Informationen finden Sie unter Zugreifen auf DynamoDB.

Themen

- Wiederherstellen einer DynamoDB-Tabelle auf einen bestimmten Zeitpunkt (Konsole)
- Wiederherstellen einer Tabelle auf einen bestimmten Zeitpunkt (AWS CLI)

Wiederherstellen einer DynamoDB-Tabelle auf einen bestimmten Zeitpunkt (Konsole)

Das folgende Beispiel zeigt, wie Sie mit der DynamoDB-Konsole eine vorhandene Tabelle namens `Music` im Zustand eines bestimmten Zeitpunkts wiederherstellen.

> **Note**
>
> Bei diesem Verfahren wird davon ausgegangen, dass Sie die point-in-time Wiederherstellung aktiviert haben. Um es für die `Music` Tabelle zu aktivieren, wählen Sie auf der Registerkarte Backups im Abschnitt Point-in-time Wiederherstellung (PITR) die Option Bearbeiten aus und aktivieren Sie dann das Kontrollkästchen neben Aktivieren point-in-time-recovery.

So stellen Sie Tabelle im Zustand eines bestimmten Zeitpunkts wieder her

1. Melden Sie sich bei der an AWS-Managementkonsole und öffnen Sie die DynamoDB-Konsole unter. https://console.aws.amazon.com/dynamodb/
2. Klicken Sie im Navigationsbereich auf der linken Seite der Konsole auf Tables (Tabellen).
3. Wählen Sie in der Tabellenliste die Tabelle `Music` aus.
4. Wählen Sie auf der Registerkarte Backups der **`Music`** Tabelle im Abschnitt Point-in-time Recovery (PITR) die Option Restore aus.
5. Geben Sie als neuen Tabellennamen **`MusicMinutesAgo`** ein.

   > **Note**
   >
   > Sie können die Tabelle in derselben AWS Region oder in einer anderen Region wiederherstellen, in der sich die Quelltabelle befindet. Sie können sekundäre Indizes von der Erstellung für die wiederhergestellte Tabelle ausschließen. Darüber hinaus können Sie einen anderen Verschlüsselungsmodus angeben.

6. Um die Wiederherstellungszeit zu bestätigen, setzen Sie Datum und Uhrzeit für die Wiederherstellung auf Frühestmöglich. Wählen Sie dann Restore (Wiederherstellen) aus, um den Wiederherstellungsvorgang zu starten.

   Die Tabelle, die wiederhergestellt wird, erhält den Status Restoring. Nach Abschluss des Wiederherstellungsvorgangs ändert sich der Status der Tabelle `MusicMinutesAgo` in Active (Aktiv).

Wiederherstellen einer Tabelle auf einen bestimmten Zeitpunkt (AWS CLI)

Das folgende Verfahren zeigt, wie Sie mit dem AWS CLI eine vorhandene Tabelle wiederherstellen können, die `Music` nach einem bestimmten Zeitpunkt benannt ist.

> **Note**
>
> Bei diesem Verfahren wird davon ausgegangen, dass Sie die point-in-time Wiederherstellung aktiviert haben. Führen Sie den folgenden Befehl aus, um diese Funktion für die Tabelle `Music` zu aktivieren.

```
aws dynamodb update-continuous-backups \
    --table-name Music \
```

```
    --point-in-time-recovery-specification PointInTimeRecoveryEnabled=True
```

So stellen Sie Tabelle im Zustand eines bestimmten Zeitpunkts wieder her

1. Vergewissern Sie sich mithilfe des `describe-continuous-backups` Befehls, dass die point-in-time Wiederherstellung für die `Music` Tabelle aktiviert ist.

   ```
   aws dynamodb describe-continuous-backups \
       --table-name Music
   ```

   Kontinuierliche Backups (automatisch bei der Tabellenerstellung aktiviert) und point-in-time Recovery sind aktiviert.

   ```
   {
       "ContinuousBackupsDescription": {
           "PointInTimeRecoveryDescription": {
               "PointInTimeRecoveryStatus": "ENABLED",
               "EarliestRestorableDateTime": 1519257118.0,
               "LatestRestorableDateTime": 1520018653.01
           },
           "ContinuousBackupsStatus": "ENABLED"
       }
   }
   ```

2. Stellen Sie die Tabelle auf einen bestimmten Zeitpunkt wieder her. In diesem Fall wird die Tabelle `Music` in der `LatestRestorableDateTime` (vor ~5 Minuten) in derselben AWS -Region wiederhergestellt.

   ```
   aws dynamodb restore-table-to-point-in-time \
       --source-table-name Music \
       --target-table-name MusicMinutesAgo \
       --use-latest-restorable-time
   ```

   > **Note**
   >
   > Sie können auch auf einen bestimmten Zeitpunkt wiederherstellen. Führen Sie dazu den Befehl mit dem Argument `--restore-date-time` aus und geben Sie einen Zeitstempel an. Sie können einen beliebigen Zeitpunkt innerhalb des konfigurierten

Wiederherstellungszeitraums angeben, der auf einen beliebigen Wert zwischen 1 und 35 Tagen festgelegt werden kann. Beispielsweise wird mit dem folgendem Befehl die Tabelle auf den Zeitpunkt `EarliestRestorableDateTime` widerhergestellt.

```
aws dynamodb restore-table-to-point-in-time \
    --source-table-name Music \
    --target-table-name MusicEarliestRestorableDateTime \
    --no-use-latest-restorable-time \
    --restore-date-time 1519257118.0
```

Für die Wiederherstellung auf einen bestimmten Zeitpunkt ist die Angabe des Arguments `--no-use-latest-restorable-time` optional.

3. Stellen Sie die Tabelle mit benutzerdefinierten Tabelleneinstellungen zu einem bestimmten Zeitpunkt wieder her. In diesem Fall wird die Tabelle `Music` auf den Zeitpunkt `LatestRestorableDateTime` (etwa vor 5 Minuten) wiederhergestellt.

   Sie können wie folgt einen anderen Verschlüsselungsmodus für die wiederhergestellte Tabelle angeben.

   > **Note**
   >
   > Der Parameter `sse-specification-override` verwendet dieselben Werte wie der im Befehl `CreateTable` verwendete Parameter `sse-specification-override`. Weitere Informationen hierzu finden Sie unter Verwalten von verschlüsselten Tabellen in DynamoDB.

   ```
   aws dynamodb restore-table-to-point-in-time \
       --source-table-name Music \
       --target-table-name MusicMinutesAgo \
       --use-latest-restorable-time \
       --sse-specification-override Enabled=true,SSEType=KMS,KMSMasterKeyId=abcd1234-
   abcd-1234-a123-ab1234a1b234
   ```

   Sie können die Tabelle in einer anderen AWS Region wiederherstellen, als in der sich die Quelltabelle befindet.

> **Note**
>
> - Der Parameter `sse-specification-override` ist für bereichsübergreifende Wiederherstellungen obligatorisch, für Wiederherstellungen in derselben Region wie die Quelltabelle aber optional.
> - Der Parameter `source-table-arn` muss für regionsübergreifende Wiederherstellungen bereitgestellt werden.
> - Wenn Sie eine regionsübergreifende Wiederherstellung von der Befehlszeile aus durchführen, müssen Sie die AWS Standardregion auf die gewünschte Zielregion festlegen. Weitere Informationen finden Sie unter Befehlszeilenoptionen im AWS Command Line Interface -Benutzerhandbuch.

```
aws dynamodb restore-table-to-point-in-time \
    --source-table-arn arn:aws:dynamodb:us-east-1:123456789012:table/Music \
    --target-table-name MusicMinutesAgo \
    --use-latest-restorable-time \
    --sse-specification-override Enabled=true,SSEType=KMS,KMSMasterKeyId=abcd1234-
abcd-1234-a123-ab1234a1b234
```

Sie können den Abrechnungsmodus und den bereitgestellten Durchsatz für die wiederhergestellte Tabelle überschreiben.

```
aws dynamodb restore-table-to-point-in-time \
    --source-table-name Music \
    --target-table-name MusicMinutesAgo \
    --use-latest-restorable-time \
    --billing-mode-override PAY_PER_REQUEST
```

Sie können einige oder alle sekundären Indizes von der Erstellung für die neu wiederhergestellte Tabelle ausschließen.

> **Note**
>
> Wiederherstellungen können schneller und kosteneffizienter sei, wenn Sie die Erstellung einiger oder aller sekundärer Indizes für die neu wiederhergestellte Tabelle ausschließen.

```
aws dynamodb restore-table-to-point-in-time \
    --source-table-name Music \
    --target-table-name MusicMinutesAgo \
    --use-latest-restorable-time \
    --global-secondary-index-override '[]'
```

Sie können eine Kombination verschiedener Überschreibungen verwenden. Sie können beispielsweise einen einzelnen globalen sekundären Index verwenden und gleichzeitig den bereitgestellten Durchsatz wie folgt ändern.

```
aws dynamodb restore-table-to-point-in-time \
    --source-table-name Music \
    --target-table-name MusicMinutesAgo \
    --billing-mode-override PROVISIONED \
    --provisioned-throughput-override ReadCapacityUnits=100,WriteCapacityUnits=100
 \
    --global-secondary-index-override IndexName=singers-
index,KeySchema=["{AttributeName=SingerName,KeyType=HASH}"],Projection="{ProjectionType=KEY
 \
    --sse-specification-override Enabled=true,SSEType=KMS \
    --use-latest-restorable-time
```

Verwenden Sie zum Überprüfen der Wiederherstellung den Befehl `describe-table`, um die Tabelle `MusicEarliestRestorableDateTime` zu beschreiben.

```
aws dynamodb describe-table --table-name MusicEarliestRestorableDateTime
```

Die Tabelle, die wiederhergestellt wird, erhält den Status Creating und für die laufende Wiederherstellung den Wert true. Nach Abschluss des Wiederherstellungsvorgangs ändert sich der Status der Tabelle `MusicEarliestRestorableDateTime` in Active (Aktiv).

> **Important**
>
> Ändern oder löschen Sie während einer Wiederherstellung nicht die AWS Identity and Access Management (IAM-) Richtlinien, die der IAM-Entität (z. B. Benutzer, Gruppe oder Rolle) die Berechtigung zur Durchführung der Wiederherstellung gewähren. Dies könnte andernfalls zu unerwartetem Verhalten führen. Angenommen, Sie haben die Schreibberechtigungen für eine Tabelle entfernt, während diese Tabelle wiederhergestellt wurde. In diesem Fall kann die zugrunde liegende `RestoreTableToPointInTime`-Operation keine wiederhergestellten Daten in die Tabelle schreiben. Beachten Sie, dass IAM-Richtlinien, die Quell-IP-Beschränkungen für den Zugriff auf die Zielwiederherstellungstabelle vorsehen, ebenfalls Probleme verursachen können.
> Sie können Berechtigungen erst modifizieren oder löschen, wenn die Wiederherstellung abgeschlossen ist.

Verwendung von AWS Backup mit DynamoDB

Amazon DynamoDB kann Ihnen dabei helfen, mit den erweiterten Backup-Funktionen in AWS Backup die gesetzlichen Vorschriften einzuhalten und die Anforderungen in Bezug auf Business Continuity zu erfüllen. AWS Backup ist ein vollständig verwalteter Datenschutzservice, der das Zentralisieren und Automatisieren von Backups in AWS-Services, in der Cloud und On-Premises leichter macht. Mit diesem Service können Sie Backup-Richtlinien konfigurieren und Aktivitäten für Ihre AWS-Ressourcen an einem Ort überwachen. Um AWS Backup zu verwenden, müssen Sie sich zustimmend opt-in (Anmelden). Opt-In-Optionen gelten für das spezifische Konto und die AWS-Region, daher müssen Sie sich möglicherweise bei mehreren Regionen mit demselben Konto anmelden. Weitere Informationen finden Sie im AWS-Backup-Entwicklerleitfaden.

Amazon DynamoDB ist nativ integriert in AWS Backup. Sie können AWS Backup verwenden, um Ihre DynamoDB-On-Demand-Backups automatisch zu planen, zu kopieren, zu markieren und die Lebensdauer festzulegen. Sie können diese Backups weiterhin von der DynamoDB-Konsole aus anzeigen und wiederherstellen. Sie können die DynamoDB-Konsole, die API und AWS-Befehlszeilenschnittstelle (AWS-CLI), um automatische Backups für Ihre DynamoDB-Tabellen zu aktivieren.

> **Note**
>
> Alle Backups, die über DynamoDB erstellt wurden, bleiben unverändert. Sie können weiterhin Backups über den aktuellen DynamoDB-Workflow erstellen.

Verbesserte Backup-Funktionen verfügbar über AWS Backup umfassen:

Scheduled backups (Geplante Backups) – Sie können regelmäßig geplante Backups Ihrer DynamoDB-Tabellen mithilfe von Backup-Plänen einrichten.

Account- und regionenübergreifendes Kopieren - Sie können Ihre Backups automatisch in einen anderen Backup-Tresor in einer anderen AWS-Region oder Konto kopieren, mit dem Sie Ihre Datenschutzanforderungen unterstützen können.

Tiering für Cold-Storage - Sie können Ihre Backups so konfigurieren, dass Lebenszyklusregeln implementiert werden, um Backups zu löschen oder in Cold-Storage umzuwandeln. Dies kann Ihnen helfen, Ihre Backup-Kosten zu optimieren.

Tags – Sie können Ihre Backups automatisch für Abrechnungs- und Kostenzuteilungszwecke markieren.

Encryption (Verschlüsselung) – DynamoDB On-Demand-Backups verwaltet durch AWS Backup werden jetzt im AWS Backup-Tresor gespeichert. Auf diese Weise können Sie Ihre Backups verschlüsseln und sichern, indem Sie einen AWS KMS key, der unabhängig von Ihrem DynamoDB-Tabellen-Verschlüsselungsschlüssel ist, verwenden.

Audit-Backups – Sie können AWS Backup-Audit Manager zur Prüfung der Compliance Ihrer AWS Backup-Richtlinien verwenden und um Backup-Aktivitäten und -Ressourcen zu finden, die noch nicht mit den von Ihnen definierten Steuerelementen übereinstimmen. Sie können damit auch automatisch einen Audit-Trail mit täglichen und On-Demand-Berichten für Ihre Backup-Governance-Zwecke generieren.

Sichere Backups mithilfe des WORM-Modells – Sie können AWS Backup-Vault Lock verwenden, um eine Einstellung für die einmalige Lesung (WORM) für Ihre Sicherungen zu aktivieren. Mit der AWS Backup-Tresorsperre können Sie eine zusätzliche Verteidigungsebene hinzufügen, die Backups vor unbeabsichtigten oder böswilligen Löschvorgängen, Änderungen am Aufbewahrungszeitraum für Backups und Aktualisierungen der Lebenszykluseinstellungen schützt. Weitere Informationen hierzu finden Sie unter AWS Backup-Vault Lock.

Diese erweiterten Backup-Funktionen sind in allen AWS-Regionen verfügbar. Weitere Informationen zu diesen Funktionen finden Sie im AWS Backup-Entwicklerleitfaden.

Themen

- DynamoDB-Tabellen sichern und wiederherstellen mit AWS Backup: So funktioniert's
- Erstellen von Backups von DynamoDB-Tabellen mit AWS Backup
- Kopieren eines Backups einer DynamoDB-Tabelle mit AWS Backup
- Wiederherstellen eines Backups einer DynamoDB-Tabelle aus AWS Backup
- Löschen eines Backups einer DynamoDB-Tabelle mit AWS Backup
- Nutzungshinweis: Unterschiede zwischen On-Demand-Backups, die von AWS Backup und DynamoDB verwaltet werden

DynamoDB-Tabellen sichern und wiederherstellen mit AWS Backup: So funktioniert's

Sie können mit der On-Demand-Backupfunktion vollständige Backups Ihrer Amazon-DynamoDB-Tabellen erstellen. Dieser Abschnitt bietet eine Übersicht über die Aktionen während des Backup- und Wiederherstellungsvorgangs.

Sicherungen

Wenn Sie ein On-Demand-Backup mit erstellen AWS Backup, wird eine Zeitmarkierung der Anfrage katalogisiert. Das Backup wird durch Anwenden aller Änderungen bis zur Uhrzeit der Anforderung für den letzten vollständigen Tabellen-Snapshot asynchron erstellt.

Bei jedem On-Demand-Backup werden die gesamten Tabellendaten gesichert. Es gibt keine Beschränkungen in Bezug auf die Anzahl der On-Demand-Backups, die erstellt werden können.

> **Note**
>
> Im Gegensatz zu DynamoDB-Backups erfolgen Backups, die mit erstellt wurden, nicht AWS Backup sofort.

Während ein Backup ausgeführt wird, können Sie die folgenden Vorgänge nicht ausführen:

- Den Backupvorgang anhalten oder abbrechen.
- Die Quelltabelle des Backups löschen.
- Backups für eine Tabelle deaktivieren, wenn ein Backup für diese Tabelle gerade ausgeführt wird.

AWS Backup bietet automatisierte Backup-Zeitpläne, Aufbewahrungsmanagement und Lebenszyklusmanagement. Dadurch werden benutzerdefinierte Skripts und manuelle Prozesse überflüssig. AWS Backup führt die Backups aus und löscht sie, wenn sie ablaufen. Weitere Informationen finden Sie im AWS Backup -Entwicklerhandbuch.

Wenn Sie die Konsole verwenden, AWS Backup werden alle mit dieser Methode erstellten Backups auf der Registerkarte Backups aufgeführt, wobei der Backup-Typ auf eingestellt ist`AWS_BACKUP`.

> **Note**
>
> Sie können Backups, die mit dem Backuptyp gekennzeichnet sind, nicht `AWS_BACKUP` mithilfe der DynamoDB-Konsole löschen. Verwenden Sie die AWS Backup Konsole, um diese Backups zu verwalten.

Weitere Informationen zum Ausführen eines Backups finden Sie unter Backup einer DynamoDB-Tabelle.

Wiederherstellen

Sie stellen eine Tabelle wieder her, ohne den bereitgestellten Durchsatz für die Tabelle zu beanspruchen. Sie können eine vollständige Tabellenwiederherstellung aus Ihrem DynamoDB-Backup durchführen oder die Zieltabelleneinstellungen konfigurieren. Wenn Sie eine Wiederherstellung durchführen, können Sie die folgenden Tabelleneinstellungen ändern:

- Verschlüsselungseinstellungen

> **Important**
>
> Bei einer vollständigen Wiederherstellung der Tabelle werden für die Zieltabelle die gleichen bereitgestellten Lese- und Schreibkapazitätseinheiten festgelegt, die für Quelltabelle festgelegt waren, als die Backupanforderung erfasst wurden. Der Wiederherstellungsvorgang stellt auch die lokalen und die globalen sekundären Indizes wieder her.

Sie können eine Sicherungskopie Ihrer DynamoDB-Tabellendaten in eine andere AWS Region kopieren und sie dann in dieser neuen Region wiederherstellen. Sie können Backups zwischen AWS kommerziellen Regionen, AWS China Regionen und AWS GovCloud (US-) Regionen kopieren und dann wiederherstellen. Sie zahlen nur für die Daten, die Sie aus der Quellregion kopieren, und Daten, die Sie in eine neue Tabelle in der Zielregion wieder herstellen.

AWS Backup stellt die Tabellen mit allen ursprünglichen Indizes wieder her.

Sie müssen für die wiederhergestellte Tabelle Folgendes einrichten:

- Auto Scaling-Richtlinien
- AWS Identity and Access Management (IAM) -Richtlinien
- CloudWatch Amazon-Metriken und Alarme
- Tags (Markierungen)
- Stream-Einstellungen
- Einstellungen für Gültigkeitsdauer (TTL)
- Einstellungen für den Löschschutz
- Einstellung für die zeitpunktbezogene Wiederherstellung (PITR)

Sie können die gesamten Tabellendaten nur in einer neuen Tabelle aus einem Backup wiederherstellen. Sie können erst in die wiederhergestellte Tabelle schreiben, wenn sie aktiv ist.

> **Note**
>
> AWS Backup Wiederherstellungen sind zerstörungsfrei. Es ist nicht möglich, eine vorhandene Tabelle während einer Wiederherstellung zu überschreiben.

Die Wiederherstellungszeiten stehen in direktem Zusammenhang mit der Konfiguration Ihrer Tabellen (z. B. der Größe Ihrer Tabellen und der Anzahl der zugrunde liegenden Partitionen) und anderen verwandten Variablen. Eine bewährte Methode bei der Planung der Notfallwiederherstellung besteht darin, die durchschnittlichen Wiederherstellungszeiten regelmäßig zu dokumentieren und festzulegen, wie sich diese Zeiten auf Ihr gesamtes Recovery-Zeitziel auswirken.

Weitere Informationen zum Ausführen einer Wiederherstellung finden Sie unter Wiederherstellen einer DynamoDB-Tabelle aus einem Backup.

Sie können IAM-Richtlinien für die Zugriffskontrolle einsetzen. Weitere Informationen finden Sie unter Verwenden von IAM mit DynamoDB-Backup und -Wiederherstellung.

Sämtliche Konsolen- und API-Aktionen werden zum Backup und Wiederherstellung in AWS CloudTrail für die Protokollierung, Überwachung und Prüfung erfasst und aufgezeichnet.

Erstellen von Backups von DynamoDB-Tabellen mit AWS Backup

In diesem Abschnitt wird beschrieben, wie Sie AWS Backup einschalten können, um On-Demand-Backups und geplante Backups aus Ihren DynamoDB-Tabellen zu erstellen.

Themen

- Einschalten von AWS Backup-Funktionen
- On-Demand-Backups
- Geplante Backups

Einschalten von AWS Backup-Funktionen

Sie müssen AWS Backup einschalten, um es mit DynamoDB zu verwenden.

Führen Sie die folgenden Schritte aus, um AWS Backup einzuschalten:

1. Melden Sie sich bei der AWS-Managementkonsole an und öffnen Sie die DynamoDB-Konsole unter https://console.aws.amazon.com/dynamodb/.
2. Klicken Sie im Navigationsbereich auf der linken Seite der Konsole auf Backups (Backups).
3. Wählen Sie im Fenster „Backup-Einstellungen“ die Option Einschalten aus.
4. Es wird ein Bestätigungsbildschirm angezeigt. Wählen Sie Funktionen einschalten aus.

AWS Backup-Funktionen sind jetzt für Ihre DynamoDB-Tabellen verfügbar.

Wenn Sie eingeschaltete AWS Backup-Funktionen wieder ausschalten möchten, gehen Sie folgendermaßen vor:

1. Melden Sie sich bei der AWS-Managementkonsole an und öffnen Sie die DynamoDB-Konsole unter https://console.aws.amazon.com/dynamodb/.
2. Klicken Sie im Navigationsbereich auf der linken Seite der Konsole auf Backups (Backups).

3. Wählen Sie im Fenster „Backup-Einstellungen“ die Option Ausschalten aus.
4. Es wird ein Bestätigungsbildschirm angezeigt. Wählen Sie Funktionen ausschalten aus.

Wenn Sie die AWS Backup-Funktionen nicht ein- oder ausschalten können, muss möglicherweise Ihr AWS-Administrator diese Aktionen ausführen.

On-Demand-Backups

Gehen Sie wie folgt vor, um eine On-Demand-Backup einer DynamoDB-Tabelle zu erstellen:

1. Melden Sie sich bei der AWS-Managementkonsole an und öffnen Sie die DynamoDB-Konsole unter https://console.aws.amazon.com/dynamodb/.
2. Klicken Sie im Navigationsbereich auf der linken Seite der Konsole auf Backups (Backups).
3. Wählen Sie Create backup (Backup erstellen).
4. Wählen Sie im angezeigten Dropdown-Menü Create an on-demand backup (Erstellen eines On-Demand-Backups) aus.
5. Um ein von AWS Backup verwaltetes Backup mit Warm-Storage und anderen grundlegenden Funktionen zu erstellen, wählen Sie Default Settings (Standardeinstellungen). Um ein Backup zu erstellen, das auf Cold Storage umgestellt werden kann, oder um ein Backup mit DynamoDB-Funktionen anstelle von AWS Backup zu erstellen, wählen Sie Customize settings (Anpassen der Einstellungen) aus.

   Wenn Sie stattdessen dieses Backup mit früheren DynamoDB-Funktionen erstellen möchten, wählen Sie Customize settings (Anpassen der Einstellungen) und dann wählen Sie Backup with DynamoDB (Backup mit DynamoDB) aus.
6. Wenn Sie die Einstellungen vorgenommen haben, wählen Sie Create backup (Backup erstellen).

Geplante Backups

Gehen Sie folgendermaßen vor, um ein Backup zu planen.

1. Melden Sie sich bei der AWS-Managementkonsole an und öffnen Sie die DynamoDB-Konsole unter https://console.aws.amazon.com/dynamodb/.
2. Klicken Sie im Navigationsbereich auf der linken Seite der Konsole auf Backups (Backups).
3. Wählen Sie im angezeigten Dropdown-Menü Schedule backups with AWS Backup(Backups planen mit) aus.

4. Sie werden auf AWS Backup umgeleitet, um einen Backupplan zu erstellen.

Kopieren eines Backups einer DynamoDB-Tabelle mit AWS Backup

Sie können eine Kopie eines aktuellen Backups erstellen. Sie können Backups auf mehrere AWS-Konten oder AWS-Regionen nach Bedarf oder automatisch als Teil eines geplanten Backupplans kopieren. Sie können auch eine Folge von konto- und regionsübergreifenden Kopien für Amazon DynamoDB Encryption Client automatisieren.

Regionsübergreifende Replikation ist insbesondere wertvoll, wenn Sie über Betriebskontinuität oder Compliance-Anforderungen verfügen, um Backups in einem Mindestabstand von Ihren Produktionsdaten zu speichern.

Kontoübergreifende Backups sind nützlich, um Ihre Backups sicher in eine oder mehrere AWS-Konten in Ihrer Organisation aus betrieblichen oder Sicherheitsgründen zu kopieren. Wenn Ihr ursprüngliches Backup versehentlich gelöscht wird, können Sie das Backup von seinem Zielkonto in das Quellkonto kopieren und dann die Wiederherstellung starten. Bevor Sie dies tun können, benötigen Sie zwei Konten, die zur selben Organisation im Organisationsdienst gehören.

Kopien erben die Konfiguration des Quellbackups, sofern Sie nichts anderes angeben, mit einer Ausnahme: Wenn Sie angeben, dass Ihre neue Kopie „Nie“ abläuft. Mit dieser Einstellung erbt die neue Kopie weiterhin ihr Ablaufdatum von der Quelle. Wenn Sie möchten, dass Ihr neues Backup dauerhaft ist, legen Sie entweder fest, dass Ihre Quellbackups niemals ablaufen, oder geben Sie an, dass Ihre neue Kopie 100 Jahre nach ihrer Erstellung abläuft.

> **Note**
>
> Wenn Sie auf ein anderes Konto kopieren, müssen Sie zuerst über die Berechtigung für dieses Konto verfügen.

Gehen Sie wie folgt vor, um ein Backup zu kopieren:

1. Melden Sie sich bei der AWS-Managementkonsole an und öffnen Sie die DynamoDB-Konsole unter https://console.aws.amazon.com/dynamodb/.
2. Klicken Sie im Navigationsbereich auf der linken Seite der Konsole auf Backups (Backups).
3. Markieren Sie das Kontrollkästchen neben den Backups, die Sie kopieren möchten.

- Wenn das Backup, das Sie kopieren möchten, ausgegraut dargestellt ist, müssen Sie erweiterte Funktionen mit AWS Backup aktivieren. Erstellen Sie dann ein neues Backup. Sie können dieses neue Backup jetzt in andere Regionen und Konten kopieren und haben die Möglichkeit, weitere neue Backups zu kopieren.

4. Wählen Sie die Option Copy Kopieren aus.
5. Wenn Sie das Backup in ein anderes Konto oder eine andere Region kopieren möchten, aktivieren Sie das Kontrollkästchen neben Kopieren Sie den Wiederherstellungspunkt an ein anderes Ziel aus. Wählen Sie dann aus, ob Sie in eine andere Region in Ihrem Konto oder in ein anderes Konto in einer anderen Region kopieren möchten.

> **Note**
>
> Um ein Backup in einer anderen Region oder einem anderen Konto wiederherzustellen, müssen Sie das Backup zuerst in diese Region oder dieses Konto kopieren.

6. Wählen Sie den gewünschten Tresor aus, in den die Datei kopiert werden soll. Sie können bei Bedarf auch einen neuen Backuptresor erstellen.
7. Klicken Sie auf Copy backup (Backup kopieren).

Wiederherstellen eines Backups einer DynamoDB-Tabelle aus AWS Backup

In diesem Abschnitt wird beschrieben, wie Sie ein Backup einer DynamoDB-Tabelle aus AWS Backup wiederherstellen.

Themen

- Wiederherstellen einer DynamoDB-Tabelle aus AWS Backup
- Wiederherstellen einer DynamoDB-Tabelle in einer anderen Region oder einem anderen Konto

Wiederherstellen einer DynamoDB-Tabelle aus AWS Backup

Um Ihre DynamoDB-Tabellen aus AWS Backup wiederherzustellen, führen Sie die folgenden Schritte aus:

1. Melden Sie sich bei der AWS-Managementkonsole an und öffnen Sie die DynamoDB-Konsole unter https://console.aws.amazon.com/dynamodb/.

2. Klicken Sie im Navigationsbereich auf der linken Seite der Konsole auf Tables (Tabellen).
3. Wählen Sie die Registerkarte Backups (Backups).
4. Aktivieren Sie das Kontrollkästchen neben dem Backup, von dem Sie wiederherstellen möchten.
5. Wählen Sie Restore (Wiederherstellen). Sie werden zum Bildschirm Wiederherstellen der Tabelle aus Backup weitergeleitet.
6. Geben Sie den Namen für die neu wiederhergestellte Tabelle, die Verschlüsselung, die diese neue Tabelle haben wird, den Schlüssel, mit dem die Wiederherstellung verschlüsselt werden soll, und andere Optionen ein.
7. Wenn Sie fertig sind, wählen Sie Restore (Wiederherstellen) aus.

Wiederherstellen einer DynamoDB-Tabelle in einer anderen Region oder einem anderen Konto

Um eine DynamoDB-Tabelle in einer anderen Region oder einem anderen Konto wiederherzustellen, müssen Sie das Backup zuerst in die neue Region oder das Konto kopieren. Um auf ein anderes Konto zu kopieren, muss dieses Konto Ihnen zuerst die Erlaubnis erteilen. Nachdem Sie Ihr DynamoDB-Backup in die neue Region oder das neue Konto kopiert haben, kann es mit dem Prozess im vorherigen Abschnitt wiederhergestellt werden.

Löschen eines Backups einer DynamoDB-Tabelle mit AWS Backup

In diesem Abschnitt wird beschrieben, wie Sie ein Backup einer DynamoDB-Tabelle mit AWS Backup löschen.

Ein DynamoDB-Backup, das anhand der AWS-Backup-Funktionen erstellt wurde, wird in einem AWS-Backup-Tresor gespeichert.

Gehen Sie wie folgt vor, um diese Art eines Backups zu löschen:

1. Melden Sie sich bei AWS-Managementkonsole an und öffnen Sie die DynamoDB-Konsole unter https://console.aws.amazon.com/dynamodb/.
2. Klicken Sie im Navigationsbereich auf der linken Seite der Konsole auf Backups (Backups).
3. Wählen Sie auf dem folgenden Bildschirm Continue to AWS Backup (Weiter zu Backup) aus.

   Sie werden zur AWS-Backup-Konsole umgeleitet. Weitere Informationen zum Löschen von Backups in der AWS-Backup-Konsole finden Sie unter Löschen von Backups.

Weitere Informationen zu AWS Backup finden Sie unter Backup and recovery using AWS Backup in der AWS Prescriptive Guidance.

Nutzungshinweis: Unterschiede zwischen On-Demand-Backups, die von AWS Backup und DynamoDB verwaltet werden

In diesem Abschnitt werden die technischen Unterschiede zwischen On-Demand-Backups durch AWS Backup und DynamoDB beschrieben.

AWS Backup hat einige andere Workflows und Verhaltensweisen als DynamoDB. Dazu zählen:

Verschlüsselung — Mit dem AWS Backup Plan erstellte Backups werden in einem verschlüsselten Tresor mit einem Schlüssel gespeichert, der AWS Backup vom Dienst verwaltet wird. Der Tresor verfügt über Zugriffssteuerungsrichtlinien für zusätzliche Sicherheit.

Backup-ARN — Die von AWS Backup erstellten Sicherungsdateien haben jetzt einen AWS Backup ARN, was sich auf das Benutzerberechtigungsmodell auswirken könnte. Die Namen der Backup-Ressourcen (ARNs) werden von `arn:aws:dynamodb` zu geändert`arn:aws:backup`.

Backups löschen — Backups, die mit erstellt wurden, AWS Backup können nur aus dem AWS Backup Tresor gelöscht werden. Sie können keine AWS Backup Dateien aus der DynamoDB-Konsole löschen.

Backup-Vorgang - Im Gegensatz zu DynamoDB-Backups sind Backups mit AWS Backup nicht unmittelbar.

Abrechnung — Backups von DynamoDB-Tabellen mit AWS Backup Funktionen werden von abgerechnet. AWS Backup

IAM-Rollen - Wenn Sie den Zugriff über IAM-Rollen verwalten, müssen Sie auch eine neue IAM-Rolle mit diesen neuen Berechtigungen konfigurieren:

```
"dynamodb:StartAwsBackupJob",
"dynamodb:RestoreTableFromAwsBackup"
```

`dynamodb:StartAwsBackupJob`wird für eine erfolgreiche Sicherung mit AWS Backup Funktionen und `dynamodb:RestoreTableFromAwsBackup` für die Wiederherstellung aus einer mit Funktionen erstellten Sicherung benötigt. AWS Backup

Informationen zum Anzeigen dieser Berechtigungen in einer vollständigen IAM-Richtlinie finden Sie unter Beispiel 8 in Verwenden von IAM.

Codebeispiele für DynamoDB mit AWS SDKs

Die folgenden Codebeispiele zeigen, wie DynamoDB mit einem AWS Software Development Kit (SDK) verwendet wird.

Bei Grundlagen handelt es sich um Codebeispiele, die Ihnen zeigen, wie Sie die wesentlichen Vorgänge innerhalb eines Services ausführen.

Aktionen sind Codeauszüge aus größeren Programmen und müssen im Kontext ausgeführt werden. Während Aktionen Ihnen zeigen, wie Sie einzelne Service-Funktionen aufrufen, können Sie Aktionen im Kontext der zugehörigen Szenarien anzeigen.

Szenarien sind Codebeispiele, die Ihnen zeigen, wie Sie bestimmte Aufgaben ausführen, indem Sie mehrere Funktionen innerhalb eines Service aufrufen oder mit anderen AWS-Services kombinieren.

AWS Community-Beiträge sind Beispiele, die von mehreren Teams erstellt wurden und verwaltet werden. AWS Um Feedback zu geben, verwenden Sie den Mechanismus, der in den verlinkten Repositorys bereitgestellt wird.

Eine vollständige Liste der AWS SDK-Entwicklerhandbücher und Codebeispiele finden Sie unterDynamoDB mit einem SDK verwenden AWS. Dieses Thema enthält auch Informationen zu den ersten Schritten und Details zu früheren SDK-Versionen.

Codebeispiele

- Grundlegende Beispiele für die Verwendung von DynamoDB AWS SDKs
  - Hello DynamoDB
  - Erlernen Sie die Grundlagen von DynamoDB mit einem SDK AWS
  - Aktionen für DynamoDB mit AWS SDKs
    - BatchExecuteStatementMit einem AWS SDK verwenden
    - Verwendung BatchGetItem mit einem AWS SDK oder CLI
    - Verwendung BatchWriteItem mit einem AWS SDK oder CLI
    - Verwendung CreateTable mit einem AWS SDK oder CLI
    - Verwendung DeleteItem mit einem AWS SDK oder CLI
    - Verwendung DeleteTable mit einem AWS SDK oder CLI
    - Verwendung DescribeTable mit einem AWS SDK oder CLI

    - Verwendung DescribeTimeToLive mit einem AWS SDK oder CLI
    - ExecuteStatementMit einem AWS SDK verwenden
    - Verwendung GetItem mit einem AWS SDK oder CLI
    - Verwendung ListTables mit einem AWS SDK oder CLI
    - Verwendung PutItem mit einem AWS SDK oder CLI
    - Verwendung Query mit einem AWS SDK oder CLI
    - Verwendung Scan mit einem AWS SDK oder CLI
    - Verwendung UpdateItem mit einem AWS SDK oder CLI
    - Verwendung UpdateTable mit einem AWS SDK oder CLI
    - Verwendung UpdateTimeToLive mit einem AWS SDK oder CLI
- Szenarien für die Verwendung von DynamoDB AWS SDKs
    - Beschleunigen Sie DynamoDB-Lesevorgänge mit DAX mithilfe eines SDK AWS
    - Arbeiten Sie mit erweiterten DynamoDB Global Secondary Index-Szenarien unter Verwendung von v2 AWS Command Line Interface
    - Erstellen Sie eine Anwendung zum Senden von Daten an eine DynamoDB-Tabelle
    - Vergleichen Sie mehrere Werte mit einem einzigen Attribut in DynamoDB mit einem SDK AWS
    - Bedingtes Aktualisieren eines DynamoDB-Elements mit einer TTL mithilfe eines SDK AWS
    - Stellen Sie mithilfe eines SDK eine Connect zu einer lokalen DynamoDB-Instanz her AWS
    - Zählen Sie Ausdrucksoperatoren in DynamoDB mit einem SDK AWS
    - Erstellen einer API-Gateway-REST-API zur Verfolgung von COVID-19-Daten
    - Erstellen einer Messenger-Anwendung mit Step Functions
    - Eine Anwendung für Foto-Asset-Management erstellen, mit der Benutzer Fotos mithilfe von Labels verwalten können
    - Erstellen Sie mithilfe des SDK eine DynamoDB-Tabelle mit einem globalen sekundären Index AWS
    - Erstellen Sie mithilfe eines SDK eine DynamoDB-Tabelle mit Warmdurchsatzeinstellung AWS
    - Erstellen einer Webanwendung zur Verfolgung von DynamoDB-Daten
    - Erstellen einer Websocket-Chat-Anwendung mit API Gateway
    - Erstellen Sie ein DynamoDB-Element mit einer TTL mithilfe eines SDK AWS
    - Erstellen und verwalten Sie globale DynamoDB-Tabellen mit Multi-Region Strong Consistency mithilfe eines SDK AWS

- Erstellen und verwalten Sie globale DynamoDB-Tabellen zur Demonstration von MREC mithilfe eines SDK AWS
- Löschen Sie DynamoDB-Daten mithilfe von PartiQL DELETE-Anweisungen mit einem SDK AWS
- Ermitteln Sie persönliche Schutzausrüstung in Bildern mit Amazon Rekognition mithilfe eines SDK AWS
- Fügen Sie DynamoDB-Daten mithilfe von PartiQL INSERT-Anweisungen mit einem SDK ein AWS
- Aufrufen einer Lambda-Funktion von einem Browser aus
- Globale sekundäre DynamoDB-Indizes mit Version 2 verwalten AWS Command Line Interface
- Verwaltung ressourcenbasierter DynamoDB-Richtlinien mit Version 2 AWS Command Line Interface
- Überwachen Sie die Leistung von Amazon DynamoDB mithilfe eines SDK AWS
- Führen Sie erweiterte DynamoDB-Abfrageoperationen mit einem SDK durch AWS
- Führen Sie Listenoperationen in DynamoDB mit einem SDK durch AWS
- Führen Sie Kartenoperationen in DynamoDB mit einem SDK durch AWS
- Führen Sie Set-Operationen in DynamoDB mit einem SDK durch AWS
- Abfragen einer DynamoDB-Tabelle mithilfe von Batches von PartiQL-Anweisungen und einem SDK AWS
- Fragen Sie eine DynamoDB-Tabelle mit PartiQL und einem SDK ab AWS
- Abfragen einer DynamoDB-Tabelle mithilfe eines globalen sekundären Indexes mit einem SDK AWS
- Fragen Sie eine DynamoDB-Tabelle mithilfe einer Begins_with-Bedingung mit einem SDK ab AWS
- Fragen Sie eine DynamoDB-Tabelle mithilfe eines Datumsbereichs im Sortierschlüssel mit einem SDK ab AWS
- Fragen Sie eine DynamoDB-Tabelle mit einem komplexen Filterausdruck mit einem SDK ab AWS
- Fragen Sie eine DynamoDB-Tabelle mit einem dynamischen Filterausdruck mit einem SDK ab AWS
- Fragen Sie eine DynamoDB-Tabelle mit einem Filterausdruck und einem Limit mit einem SDK ab AWS
- Abfragen einer DynamoDB-Tabelle mit verschachtelten Attributen mithilfe eines SDK AWS

- Abfragen einer DynamoDB-Tabelle mit Paginierung mithilfe eines SDK AWS
- Abfragen einer DynamoDB-Tabelle mit stark konsistenten Lesevorgängen mithilfe eines SDK AWS
- Abfragen von DynamoDB-Daten mithilfe von PartiQL SELECT-Anweisungen mit einem SDK AWS
- Abfragen einer DynamoDB-Tabelle nach TTL-Elementen mithilfe eines SDK AWS
- Abfragen von DynamoDB-Tabellen mithilfe von Datums- und Uhrzeitmustern mit einem SDK AWS
- Speichern Sie EXIF und andere Bildinformationen mit einem SDK AWS
- Richten Sie die attributebasierte Zugriffskontrolle für DynamoDB mit v2 ein AWS Command Line Interface
- Die Reihenfolge der Aktualisierungsausdrücke in DynamoDB mit einem SDK verstehen AWS
- Aktualisieren Sie eine DynamoDB-Tabelleneinstellung mit warmem Durchsatz mithilfe eines SDK AWS
- Aktualisieren Sie ein DynamoDB-Element mit einer TTL mithilfe eines SDK AWS
- Aktualisieren Sie DynamoDB-Daten mithilfe von PartiQL UPDATE-Anweisungen mit einem SDK AWS
- Verwenden von API Gateway zum Aufrufen einer Lambda-Funktion
- Verwenden von Step Functions, um Lambda-Funktionen aufzurufen
- Verwenden Sie ein Dokumentmodell für DynamoDB mithilfe eines SDK AWS
- Verwenden Sie ein Objektpersistenzmodell auf hoher Ebene für DynamoDB mithilfe eines SDK AWS
- Verwenden Sie atomare Zähleroperationen in DynamoDB mit einem SDK AWS
- Verwenden Sie bedingte Operationen in DynamoDB mit einem SDK AWS
- Verwenden Sie Ausdrucksattributnamen in DynamoDB mit einem SDK AWS
- Verwendung geplanter Ereignisse zum Aufrufen einer Lambda-Funktion
- Arbeiten Sie mit lokalen sekundären DynamoDB-Indizes unter Verwendung von v2 AWS Command Line Interface
- Arbeiten Sie mit DynamoDB Streams und Time-to-Live verwenden Sie v2 AWS Command Line Interface
- Arbeiten Sie mit globalen DynamoDB-Tabellen und Multi-Region-Replikation mit Eventual Consistency (MREC) unter Verwendung von v2 AWS Command Line Interface

- Arbeiten Sie mit DynamoDB-Ressourcen-Tagging mithilfe von v2 AWS Command Line Interface
- Arbeiten Sie mit DynamoDB-Tabellenverschlüsselung mit Version 2 AWS Command Line Interface

- Serverless-Beispiele für DynamoDB
  - Aufrufen einer Lambda-Funktion über einen DynamoDB-Auslöser
  - Melden von Batch-Elementfehlern für Lambda-Funktionen mit einem DynamoDB-Auslöser
- AWS Community-Beiträge für DynamoDB
  - Erstellen und Testen einer Serverless-Anwendung

Grundlegende Beispiele für die Verwendung von DynamoDB AWS SDKs

Die folgenden Codebeispiele zeigen, wie Sie die Grundlagen von Amazon DynamoDB mit verwenden können. AWS SDKs

Beispiele

- Hello DynamoDB
- Erlernen Sie die Grundlagen von DynamoDB mit einem SDK AWS
- Aktionen für DynamoDB mit AWS SDKs
  - BatchExecuteStatementMit einem AWS SDK verwenden
  - Verwendung BatchGetItem mit einem AWS SDK oder CLI
  - Verwendung BatchWriteItem mit einem AWS SDK oder CLI
  - Verwendung CreateTable mit einem AWS SDK oder CLI
  - Verwendung DeleteItem mit einem AWS SDK oder CLI
  - Verwendung DeleteTable mit einem AWS SDK oder CLI
  - Verwendung DescribeTable mit einem AWS SDK oder CLI
  - Verwendung DescribeTimeToLive mit einem AWS SDK oder CLI
  - ExecuteStatementMit einem AWS SDK verwenden
  - Verwendung GetItem mit einem AWS SDK oder CLI
  - Verwendung ListTables mit einem AWS SDK oder CLI
  - Verwendung PutItem mit einem AWS SDK oder CLI

- Verwendung Query mit einem AWS SDK oder CLI
- Verwendung Scan mit einem AWS SDK oder CLI
- Verwendung UpdateItem mit einem AWS SDK oder CLI
- Verwendung UpdateTable mit einem AWS SDK oder CLI
- Verwendung UpdateTimeToLive mit einem AWS SDK oder CLI

Hello DynamoDB

Die folgenden Codebeispiele zeigen, wie Sie mit der Verwendung von DynamoDB beginnen.

.NET

SDK für .NET (v4)

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
using Amazon.DynamoDBv2;
using Amazon.DynamoDBv2.Model;
using Microsoft.Extensions.DependencyInjection;

namespace DynamoDBActions;

/// <summary>
/// A simple example that demonstrates basic DynamoDB operations.
/// </summary>
public class HelloDynamoDB
{
    /// <summary>
    /// HelloDynamoDB lists the existing DynamoDB tables for the default user.
    /// </summary>
    /// <param name="args">Command line arguments</param>
    /// <returns>Async task.</returns>
    static async Task Main(string[] args)
    {
```

```
        // Set up dependency injection for Amazon DynamoDB.
        using var host =
 Microsoft.Extensions.Hosting.Host.CreateDefaultBuilder(args)
            .ConfigureServices((_, services) =>
                services.AddAWSService<IAmazonDynamoDB>()
            )
            .Build();

        // Now the client is available for injection.
        var dynamoDbClient = host.Services.GetRequiredService<IAmazonDynamoDB>();

        try
        {
            var request = new ListTablesRequest();
            var tableNames = new List<string>();

            var paginatorForTables =
 dynamoDbClient.Paginators.ListTables(request);

            await foreach (var tableName in paginatorForTables.TableNames)
            {
                tableNames.Add(tableName);
            }

            Console.WriteLine("Welcome to the DynamoDB Hello Service example. " +
                              "\nLet's list your DynamoDB tables:");
            tableNames.ForEach(table =>
            {
                Console.WriteLine($"Table: {table}");
            });
        }
        catch (AmazonDynamoDBException ex)
        {
            Console.WriteLine($"An Amazon DynamoDB service error occurred while
 listing tables. {ex.Message}");
        }
        catch (Exception ex)
        {
            Console.WriteLine($"An error occurred while listing tables.
 {ex.Message}");
        }
    }
}
```

- Einzelheiten zur API finden Sie ListTablesin der AWS SDK für .NET API-Referenz.

C++

SDK für C++

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

Code für die CMake Datei CMake Lists.txt.

```
# Set the minimum required version of CMake for this project.
cmake_minimum_required(VERSION 3.13)

# Set the AWS service components used by this project.
set(SERVICE_COMPONENTS dynamodb)

# Set this project's name.
project("hello_dynamodb")

# Set the C++ standard to use to build this target.
# At least C++ 11 is required for the AWS SDK for C++.
set(CMAKE_CXX_STANDARD 11)

# Use the MSVC variable to determine if this is a Windows build.
set(WINDOWS_BUILD ${MSVC})

if (WINDOWS_BUILD) # Set the location where CMake can find the installed
 libraries for the AWS SDK.
    string(REPLACE ";" "/aws-cpp-sdk-all;" SYSTEM_MODULE_PATH
 "${CMAKE_SYSTEM_PREFIX_PATH}/aws-cpp-sdk-all")
    list(APPEND CMAKE_PREFIX_PATH ${SYSTEM_MODULE_PATH})
endif ()

# Find the AWS SDK for C++ package.
find_package(AWSSDK REQUIRED COMPONENTS ${SERVICE_COMPONENTS})
```

```
if (WINDOWS_BUILD AND AWSSDK_INSTALL_AS_SHARED_LIBS)
     # Copy relevant AWS SDK for C++ libraries into the current binary directory
 for running and debugging.

     # set(BIN_SUB_DIR "/Debug") # if you are building from the command line you
 may need to uncomment this
                                     # and set the proper subdirectory to the
 executables' location.

     AWSSDK_CPY_DYN_LIBS(SERVICE_COMPONENTS ""
 ${CMAKE_CURRENT_BINARY_DIR}${BIN_SUB_DIR})
endif ()

add_executable(${PROJECT_NAME}
        hello_dynamodb.cpp)

target_link_libraries(${PROJECT_NAME}
        ${AWSSDK_LINK_LIBRARIES})
```

Code für die Quelldatei „hello_dynamodb.cpp“.

```
#include <aws/core/Aws.h>
#include <aws/dynamodb/DynamoDBClient.h>
#include <aws/dynamodb/model/ListTablesRequest.h>
#include <iostream>

/*
 *  A "Hello DynamoDB" starter application which initializes an Amazon DynamoDB
 (DynamoDB) client and lists the
 *  DynamoDB tables.
 *
 *  main function
 *
 *  Usage: 'hello_dynamodb'
 *
 */

int main(int argc, char **argv) {
    Aws::SDKOptions options;
    // Optionally change the log level for debugging.
//   options.loggingOptions.logLevel = Utils::Logging::LogLevel::Debug;
    Aws::InitAPI(options); // Should only be called once.
```

```
    int result = 0;
    {
        Aws::Client::ClientConfiguration clientConfig;
        // Optional: Set to the AWS Region (overrides config file).
        // clientConfig.region = "us-east-1";

        Aws::DynamoDB::DynamoDBClient dynamodbClient(clientConfig);
        Aws::DynamoDB::Model::ListTablesRequest listTablesRequest;
        listTablesRequest.SetLimit(50);
        do {
            const Aws::DynamoDB::Model::ListTablesOutcome &outcome =
 dynamodbClient.ListTables(
                    listTablesRequest);
            if (!outcome.IsSuccess()) {
                std::cout << "Error: " << outcome.GetError().GetMessage() <<
 std::endl;
                result = 1;
                break;
            }

            for (const auto &tableName: outcome.GetResult().GetTableNames()) {
                std::cout << tableName << std::endl;
            }

            listTablesRequest.SetExclusiveStartTableName(
                    outcome.GetResult().GetLastEvaluatedTableName());

        } while (!listTablesRequest.GetExclusiveStartTableName().empty());
    }


    Aws::ShutdownAPI(options); // Should only be called once.
    return result;
}
```

- Einzelheiten zur API finden Sie ListTablesunter AWS SDK für C++ API-Referenz.

Java

SDK für Java 2.x

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.model.ListTablesRequest;
import software.amazon.awssdk.services.dynamodb.model.ListTablesResponse;
import java.util.List;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 */
public class ListTables {
    public static void main(String[] args) {
        System.out.println("Listing your Amazon DynamoDB tables:\n");
        Region region = Region.US_EAST_1;
        DynamoDbClient ddb = DynamoDbClient.builder()
                .region(region)
                .build();
        listAllTables(ddb);
        ddb.close();
    }

    public static void listAllTables(DynamoDbClient ddb) {
        boolean moreTables = true;
        String lastName = null;

        while (moreTables) {
```

```
            try {
                ListTablesResponse response = null;
                if (lastName == null) {
                    ListTablesRequest request =
 ListTablesRequest.builder().build();
                    response = ddb.listTables(request);
                } else {
                    ListTablesRequest request = ListTablesRequest.builder()
                            .exclusiveStartTableName(lastName).build();
                    response = ddb.listTables(request);
                }

                List<String> tableNames = response.tableNames();
                if (tableNames.size() > 0) {
                    for (String curName : tableNames) {
                        System.out.format("* %s\n", curName);
                    }
                } else {
                    System.out.println("No tables found!");
                    System.exit(0);
                }

                lastName = response.lastEvaluatedTableName();
                if (lastName == null) {
                    moreTables = false;
                }

            } catch (DynamoDbException e) {
                System.err.println(e.getMessage());
                System.exit(1);
            }
        }
        System.out.println("\nDone!");
    }
}
```

- Einzelheiten zur API finden Sie ListTablesin der AWS SDK for Java 2.x API-Referenz.

JavaScript

SDK für JavaScript (v3)

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

Weitere Informationen zur Arbeit mit DynamoDB finden Sie unter DynamoDB programmieren mit. AWS SDK für JavaScript JavaScript

```
import { ListTablesCommand, DynamoDBClient } from "@aws-sdk/client-dynamodb";

const client = new DynamoDBClient({});

export const main = async () => {
  const command = new ListTablesCommand({});

  const response = await client.send(command);
  console.log(response.TableNames.join("\n"));
  return response;
};
```

- Einzelheiten zur API finden Sie unter ListTablesAPI-Referenz.AWS SDK für JavaScript

Python

SDK für Python (Boto3)

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
import boto3

# Create a DynamoDB client using the default credentials and region
dynamodb = boto3.client("dynamodb")

# Initialize a paginator for the list_tables operation
paginator = dynamodb.get_paginator("list_tables")

# Create a PageIterator from the paginator
page_iterator = paginator.paginate(Limit=10)

# List the tables in the current AWS account
print("Here are the DynamoDB tables in your account:")

# Use pagination to list all tables
table_names = []

for page in page_iterator:
    for table_name in page.get("TableNames", []):
        print(f"- {table_name}")
        table_names.append(table_name)

if not table_names:
    print("You don't have any DynamoDB tables in your account.")
else:
    print(f"\nFound {len(table_names)} tables.")
```

- Einzelheiten zur API finden Sie ListTablesin AWS SDK for Python (Boto3) API Reference.

Ruby

SDK für Ruby

> **Note**
>
> Es gibt noch mehr dazu. GitHub Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
require 'aws-sdk-dynamodb'
require 'logger'

# DynamoDBManager is a class responsible for managing DynamoDB operations
# such as listing all tables in the current AWS account.
class DynamoDBManager
  def initialize(client)
    @client = client
    @logger = Logger.new($stdout)
  end

  # Lists and prints all DynamoDB tables in the current AWS account.
  def list_tables
    @logger.info('Here are the DynamoDB tables in your account:')

    paginator = @client.list_tables(limit: 10)
    table_names = []

    paginator.each_page do |page|
      page.table_names.each do |table_name|
        @logger.info("- #{table_name}")
        table_names << table_name
      end
    end

    if table_names.empty?
      @logger.info("You don't have any DynamoDB tables in your account.")
    else
      @logger.info("\nFound #{table_names.length} tables.")
    end
  end
end

if $PROGRAM_NAME == __FILE__
  dynamodb_client = Aws::DynamoDB::Client.new
  manager = DynamoDBManager.new(dynamodb_client)
  manager.list_tables
end
```

- Einzelheiten zur API finden Sie ListTablesin der AWS SDK für Ruby API-Referenz.

Eine vollständige Liste der AWS SDK-Entwicklerhandbücher und Codebeispiele finden Sie unterDynamoDB mit einem SDK verwenden AWS. Dieses Thema enthält auch Informationen zu den ersten Schritten und Details zu früheren SDK-Versionen.

Erlernen Sie die Grundlagen von DynamoDB mit einem SDK AWS

Die folgenden Code-Beispiele veranschaulichen Folgendes:

- Erstellen einer Tabelle, die Filmdaten enthalten kann.
- Einfügen, Abrufen und Aktualisieren eines einzelnen Films in der Tabelle.
- Schreiben von Filmdaten in die Tabelle anhand einer JSON-Beispieldatei.
- Abfragen nach Filmen, die in einem bestimmten Jahr veröffentlicht wurden.
- Scan nach Filmen, die in mehreren Jahren veröffentlicht wurden.
- Löschen eines Films aus der Tabelle und anschließendes Löschen der Tabelle.

.NET

SDK für .NET (v4)

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
/// <summary>
/// This example application performs the following basic Amazon DynamoDB
/// functions:
///     CreateTableAsync
///     PutItemAsync
///     UpdateItemAsync
///     BatchWriteItemAsync
///     GetItemAsync
///     DeleteItemAsync
///     Query
///     Scan
///     DeleteItemAsync.
/// </summary>
```

```
public class DynamoDbBasics
{
    public static bool IsInteractive = true;

    // Separator for the console display.
    private static readonly string SepBar = new string('-', 80);

    /// <summary>
    /// The main entry point for the DynamoDB Basics example application.
    /// </summary>
    /// <param name="args">Command line arguments.</param>
    /// <returns>A task representing the asynchronous operation.</returns>
    public static async Task Main(string[] args)
    {
        // Set up dependency injection for Amazon DynamoDB.
        using var host =
 Microsoft.Extensions.Hosting.Host.CreateDefaultBuilder(args)
            .ConfigureServices((_, services) =>
                services.AddAWSService<IAmazonDynamoDB>()
                    .AddTransient<DynamoDbWrapper>())
            .Build();

        // Now the wrapper is available for injection.
        var dynamoDbWrapper =
 host.Services.GetRequiredService<DynamoDbWrapper>();

        var tableName = "movie_table";

        var movieFileName = @"movies.json";

        DisplayInstructions();

        // Create a new table and wait for it to be active.
        Console.WriteLine($"Creating the new table: {tableName}");

        var success = await dynamoDbWrapper.CreateMovieTableAsync(tableName);

        Console.WriteLine(success
            ? $"\nTable: {tableName} successfully created."
            : $"\nCould not create {tableName}.");

        WaitForEnter();

        // Add a single new movie to the table.
```

```
        var newMovie = new Movie
        {
            Year = 2021,
            Title = "Spider-Man: No Way Home",
        };

        success = await dynamoDbWrapper.PutItemAsync(newMovie, tableName);
        if (success)
        {
            Console.WriteLine($"Added {newMovie.Title} to the table.");
        }
        else
        {
            Console.WriteLine("Could not add movie to table.");
        }

        WaitForEnter();

        // Update the new movie by adding a plot and rank.
        var newInfo = new MovieInfo
        {
            Plot = "With Spider-Man's identity now revealed, Peter asks" +
                   "Doctor Strange for help. When a spell goes wrong, dangerous"
+
                   "foes from other worlds start to appear, forcing Peter to" +
                   "discover what it truly means to be Spider-Man.",
            Rank = 9,
        };

        success = await dynamoDbWrapper.UpdateItemAsync(newMovie, newInfo,
tableName);
        if (success)
        {
            Console.WriteLine($"Successfully updated the movie:
{newMovie.Title}");
        }
        else
        {
            Console.WriteLine("Could not update the movie.");
        }

        WaitForEnter();

        // Add a batch of movies to the DynamoDB table from a list of
```

```
        // movies in a JSON file.
        var itemCount = await dynamoDbWrapper.BatchWriteItemsAsync(movieFileName,
tableName);
        Console.WriteLine($"Added {itemCount} movies to the table.");

        WaitForEnter();

        // Get a movie by key. (partition + sort)
        var lookupMovie = new Movie
        {
            Title = "Jurassic Park",
            Year = 1993,
        };

        Console.WriteLine("Looking for the movie \"Jurassic Park\".");
        var item = await dynamoDbWrapper.GetItemAsync(lookupMovie, tableName);
        if (item?.Count > 0)
        {
            dynamoDbWrapper.DisplayItem(item);
        }
        else
        {
            Console.WriteLine($"Couldn't find {lookupMovie.Title}");
        }

        WaitForEnter();

        // Delete a movie.
        var movieToDelete = new Movie
        {
            Title = "The Town",
            Year = 2010,
        };

        success = await dynamoDbWrapper.DeleteItemAsync(tableName,
movieToDelete);

        if (success)
        {
            Console.WriteLine($"Successfully deleted {movieToDelete.Title}.");
        }
        else
        {
            Console.WriteLine($"Could not delete {movieToDelete.Title}.");
```

```
        }

        WaitForEnter();

        // Use Query to find all the movies released in 2010.
        int findYear = 2010;
        Console.WriteLine($"Movies released in {findYear}");
        var queryCount = await dynamoDbWrapper.QueryMoviesAsync(tableName,
findYear);
        Console.WriteLine($"Found {queryCount} movies released in {findYear}");

        WaitForEnter();

        // Use Scan to get a list of movies from 2001 to 2011.
        int startYear = 2001;
        int endYear = 2011;
        var scanCount = await dynamoDbWrapper.ScanTableAsync(tableName,
startYear, endYear);
        Console.WriteLine($"Found {scanCount} movies released between {startYear}
and {endYear}");

        WaitForEnter();

        // Delete the table.
        success = await dynamoDbWrapper.DeleteTableAsync(tableName);

        if (success)
        {
            Console.WriteLine($"Successfully deleted {tableName}");
        }
        else
        {
            Console.WriteLine($"Could not delete {tableName}");
        }

        Console.WriteLine("The DynamoDB Basics example application is
complete.");

        WaitForEnter();
    }

    /// <summary>
    /// Displays the description of the application on the console.
    /// </summary>
```

```
   private static void DisplayInstructions()
   {
       if (!IsInteractive)
       {
           return;
       }

       Console.Clear();
       Console.WriteLine();
       Console.Write(new string(' ', 28));
       Console.WriteLine("DynamoDB Basics Example");
       Console.WriteLine(SepBar);
       Console.WriteLine("This demo application shows the basics of using
DynamoDB with the AWS SDK.");
       Console.WriteLine(SepBar);
       Console.WriteLine("The application does the following:");
       Console.WriteLine("\t1. Creates a table with partition: year and
sort:title.");
       Console.WriteLine("\t2. Adds a single movie to the table.");
       Console.WriteLine("\t3. Adds movies to the table from moviedata.json.");
       Console.WriteLine("\t4. Updates the rating and plot of the movie that was
just added.");
       Console.WriteLine("\t5. Gets a movie using its key (partition + sort).");
       Console.WriteLine("\t6. Deletes a movie.");
       Console.WriteLine("\t7. Uses QueryAsync to return all movies released in
a given year.");
       Console.WriteLine("\t8. Uses ScanAsync to return all movies released
within a range of years.");
       Console.WriteLine("\t9. Finally, it deletes the table that was just
created.");
       WaitForEnter();
   }

   /// <summary>
   /// Simple method to wait for the Enter key to be pressed.
   /// </summary>
   private static void WaitForEnter()
   {
       if (IsInteractive)
       {
           Console.WriteLine("\nPress <Enter> to continue.");
           Console.WriteLine(SepBar);
           _ = Console.ReadLine();
       }
```

```
    }
}
```

Verwenden Sie den injizierten Client für Tabellenoperationen.

```
using System.Text.Json;
using Amazon.DynamoDBv2;
using Amazon.DynamoDBv2.DataModel;
using Amazon.DynamoDBv2.DocumentModel;
using Amazon.DynamoDBv2.Model;

namespace DynamoDBActions;

/// <summary>
/// Methods of this class perform Amazon DynamoDB operations.
/// </summary>
public class DynamoDbWrapper
{
    private readonly IAmazonDynamoDB _amazonDynamoDB;

    /// <summary>
    /// Constructor for the DynamoDbWrapper class.
    /// </summary>
    /// <param name="amazonDynamoDB">The injected DynamoDB client.</param>
    public DynamoDbWrapper(IAmazonDynamoDB amazonDynamoDB)
    {
        _amazonDynamoDB = amazonDynamoDB;
    }
```

Erstellt eine Tabelle, die Filmdaten enthält.

```
    /// <summary>
    /// Creates a new Amazon DynamoDB table and then waits for the new
    /// table to become active.
    /// </summary>
    /// <param name="tableName">The name of the table to create.</param>
    /// <returns>A Boolean value indicating the success of the operation.</
returns>
    public async Task<bool> CreateMovieTableAsync(string tableName)
```

```
    {
        try
        {
            var response = await _amazonDynamoDB.CreateTableAsync(new
CreateTableRequest
            {
                TableName = tableName,
                AttributeDefinitions = new List<AttributeDefinition>()
                {
                    new AttributeDefinition
                    {
                        AttributeName = "title",
                        AttributeType = ScalarAttributeType.S,
                    },
                    new AttributeDefinition
                    {
                        AttributeName = "year",
                        AttributeType = ScalarAttributeType.N,
                    },
                },
                KeySchema = new List<KeySchemaElement>()
                {
                    new KeySchemaElement
                    {
                        AttributeName = "year",
                        KeyType = KeyType.HASH,
                    },
                    new KeySchemaElement
                    {
                        AttributeName = "title",
                        KeyType = KeyType.RANGE,
                    },
                },
                BillingMode = BillingMode.PAY_PER_REQUEST,
            });

            // Wait until the table is ACTIVE and then report success.
            Console.Write("Waiting for table to become active...");

            var request = new DescribeTableRequest
            {
                TableName = response.TableDescription.TableName,
            };
```

```
            TableStatus status;

            int sleepDuration = 2000;

            do
            {
                Thread.Sleep(sleepDuration);

                var describeTableResponse = await
_amazonDynamoDB.DescribeTableAsync(request);
                status = describeTableResponse.Table.TableStatus;

                Console.Write(".");
            }
            while (status != "ACTIVE");

            return status == TableStatus.ACTIVE;
        }
        catch (ResourceInUseException ex)
        {
            Console.WriteLine($"Table {tableName} already exists. {ex.Message}");
            throw;
        }
        catch (AmazonDynamoDBException ex)
        {
            Console.WriteLine($"An Amazon DynamoDB error occurred while creating
table {tableName}. {ex.Message}");
            throw;
        }
        catch (Exception ex)
        {
            Console.WriteLine($"An error occurred while creating table
{tableName}. {ex.Message}");
            throw;
        }
    }
```

Fügt der Tabelle einen einzelnen Film hinzu.

```
    /// <summary>
```

```
    /// Adds a new item to the table.
    /// </summary>
    /// <param name="newMovie">A Movie object containing informtation for
    /// the movie to add to the table.</param>
    /// <param name="tableName">The name of the table where the item will be
 added.</param>
    /// <returns>A Boolean value that indicates the results of adding the item.</
returns>
    public async Task<bool> PutItemAsync(Movie newMovie, string tableName)
    {
        try
        {
            var item = new Dictionary<string, AttributeValue>
            {
                ["title"] = new AttributeValue { S = newMovie.Title },
                ["year"] = new AttributeValue { N = newMovie.Year.ToString() },
            };

            var request = new PutItemRequest
            {
                TableName = tableName,
                Item = item,
            };

            await _amazonDynamoDB.PutItemAsync(request);
            return true;
        }
        catch (ResourceNotFoundException ex)
        {
            Console.WriteLine($"Table {tableName} was not found. {ex.Message}");
            return false;
        }
        catch (AmazonDynamoDBException ex)
        {
            Console.WriteLine($"An Amazon DynamoDB error occurred while putting
 item. {ex.Message}");
            throw;
        }
        catch (Exception ex)
        {
            Console.WriteLine($"An error occurred while putting item.
 {ex.Message}");
            throw;
        }
```

```
    }
```

Aktualisiert ein einzelnes Element in einer Tabelle.

```
    /// <summary>
    /// Updates an existing item in the movies table.
    /// </summary>
    /// <param name="newMovie">A Movie object containing information for
    /// the movie to update.</param>
    /// <param name="newInfo">A MovieInfo object that contains the
    /// information that will be changed.</param>
    /// <param name="tableName">The name of the table that contains the movie.</
param>
    /// <returns>A Boolean value that indicates the success of the operation.</
returns>
    public async Task<bool> UpdateItemAsync(
        Movie newMovie,
        MovieInfo newInfo,
        string tableName)
    {
        try
        {
            var key = new Dictionary<string, AttributeValue>
            {
                ["title"] = new AttributeValue { S = newMovie.Title },
                ["year"] = new AttributeValue { N = newMovie.Year.ToString() },
            };
            var updates = new Dictionary<string, AttributeValueUpdate>
            {
                ["info.plot"] = new AttributeValueUpdate
                {
                    Action = AttributeAction.PUT,
                    Value = new AttributeValue { S = newInfo.Plot },
                },

                ["info.rating"] = new AttributeValueUpdate
                {
                    Action = AttributeAction.PUT,
                    Value = new AttributeValue { N = newInfo.Rank.ToString() },
                },
```

```
            };

            var request = new UpdateItemRequest
            {
                AttributeUpdates = updates,
                Key = key,
                TableName = tableName,
            };

            await _amazonDynamoDB.UpdateItemAsync(request);
            return true;
        }
        catch (ResourceNotFoundException ex)
        {
            Console.WriteLine($"Table {tableName} or item was not found.
{ex.Message}");
            return false;
        }
        catch (AmazonDynamoDBException ex)
        {
            Console.WriteLine($"An Amazon DynamoDB error occurred while updating
item. {ex.Message}");
            throw;
        }
        catch (Exception ex)
        {
            Console.WriteLine($"An error occurred while updating item.
{ex.Message}");
            throw;
        }
    }
```

Ruft ein einzelnes Element aus der Filmtabelle ab.

```
    /// <summary>
    /// Gets information about an existing movie from the table.
    /// </summary>
    /// <param name="newMovie">A Movie object containing information about
    /// the movie to retrieve.</param>
```

```
    /// <param name="tableName">The name of the table containing the movie.</
param>
    /// <returns>A Dictionary object containing information about the item
    /// retrieved.</returns>
    public async Task<Dictionary<string, AttributeValue>> GetItemAsync(Movie
 newMovie, string tableName)
    {
        try
        {
            var key = new Dictionary<string, AttributeValue>
            {
                ["title"] = new AttributeValue { S = newMovie.Title },
                ["year"] = new AttributeValue { N = newMovie.Year.ToString() },
            };

            var request = new GetItemRequest
            {
                Key = key,
                TableName = tableName,
            };

            var response = await _amazonDynamoDB.GetItemAsync(request);
            return response.Item;
        }
        catch (ResourceNotFoundException ex)
        {
            Console.WriteLine($"Table {tableName} was not found. {ex.Message}");
            return new Dictionary<string, AttributeValue>();
        }
        catch (AmazonDynamoDBException ex)
        {
            Console.WriteLine($"An Amazon DynamoDB error occurred while getting
 item. {ex.Message}");
            throw;
        }
        catch (Exception ex)
        {
            Console.WriteLine($"An error occurred while getting item.
 {ex.Message}");
            throw;
        }
    }
```

Schreibt einen Stapel von Elementen in die Filmtabelle.

```
/// <summary>
/// Loads the contents of a JSON file into a list of movies to be
/// added to the DynamoDB table.
/// </summary>
/// <param name="movieFileName">The name of the JSON file.</param>
/// <returns>A generic list of movie objects.</returns>
public List<Movie> ImportMovies(string movieFileName)
{
    var moviesList = new List<Movie>();
    if (!File.Exists(movieFileName))
    {
        return moviesList;
    }

    using var sr = new StreamReader(movieFileName);
    string json = sr.ReadToEnd();
    var allMovies = JsonSerializer.Deserialize<List<Movie>>(
        json,
        new JsonSerializerOptions
        {
            PropertyNameCaseInsensitive = true
        });

    // Now return the first 250 entries.
    if (allMovies != null && allMovies.Any())
    {
        moviesList = allMovies.GetRange(0, 250);
    }
    return moviesList;
}

/// <summary>
/// Writes 250 items to the movie table.
/// </summary>
/// <param name="movieFileName">A string containing the full path to
/// the JSON file containing movie data.</param>
/// <param name="tableName">The name of the table to write items to.</param>
/// <returns>A long integer value representing the number of movies
```

```
    /// imported from the JSON file.</returns>
    public async Task<long> BatchWriteItemsAsync(
        string movieFileName, string tableName)
    {
        try
        {
            var movies = ImportMovies(movieFileName);
            if (!movies.Any())
            {
                Console.WriteLine("Couldn't find the JSON file with movie
data.");
                return 0;
            }

            var context = new DynamoDBContextBuilder()
                // Optional call to provide a specific instance of
IAmazonDynamoDB
                .WithDynamoDBClient(() => _amazonDynamoDB)
                .Build();

            var movieBatch = context.CreateBatchWrite<Movie>(
                new BatchWriteConfig()
                {
                    OverrideTableName = tableName
                });
            movieBatch.AddPutItems(movies);

            Console.WriteLine("Adding imported movies to the table.");
            await movieBatch.ExecuteAsync();

            return movies.Count;
        }
        catch (ResourceNotFoundException ex)
        {
            Console.WriteLine($"Table was not found during batch write operation.
{ex.Message}");
            throw;
        }
        catch (AmazonDynamoDBException ex)
        {
            Console.WriteLine($"An Amazon DynamoDB error occurred during batch
write operation. {ex.Message}");
            throw;
        }
```

```
        catch (Exception ex)
        {
            Console.WriteLine($"An error occurred during batch write operation.
{ex.Message}");
            throw;
        }
    }
```

Löscht ein einzelnes Element aus der Tabelle.

```
    /// <summary>
    /// Deletes a single item from a DynamoDB table.
    /// </summary>
    /// <param name="tableName">The name of the table from which the item
    /// will be deleted.</param>
    /// <param name="movieToDelete">A movie object containing the title and
    /// year of the movie to delete.</param>
    /// <returns>A Boolean value indicating the success or failure of the
    /// delete operation.</returns>
    public async Task<bool> DeleteItemAsync(
        string tableName,
        Movie movieToDelete)
    {
        try
        {
            var key = new Dictionary<string, AttributeValue>
            {
                ["title"] = new AttributeValue { S = movieToDelete.Title },
                ["year"] = new AttributeValue { N =
movieToDelete.Year.ToString() },
            };

            var request = new DeleteItemRequest { TableName = tableName, Key =
key, };

            await _amazonDynamoDB.DeleteItemAsync(request);
            return true;
        }
        catch (ResourceNotFoundException ex)
        {
```

```
            Console.WriteLine($"Table {tableName} was not found. {ex.Message}");
            return false;
        }
        catch (AmazonDynamoDBException ex)
        {
            Console.WriteLine($"An Amazon DynamoDB error occurred while deleting
item. {ex.Message}");
            throw;
        }
        catch (Exception ex)
        {
            Console.WriteLine($"An error occurred while deleting item.
{ex.Message}");
            throw;
        }
    }
```

Fragt die Tabelle nach Filmen ab, die in einem bestimmten Jahr veröffentlicht wurden.

```
    /// <summary>
    /// Queries the table for movies released in a particular year and
    /// then displays the information for the movies returned.
    /// </summary>
    /// <param name="tableName">The name of the table to query.</param>
    /// <param name="year">The release year for which we want to
    /// view movies.</param>
    /// <returns>The number of movies that match the query.</returns>
    public async Task<int> QueryMoviesAsync(string tableName, int year)
    {
        try
        {
            var movieTable = new TableBuilder(_amazonDynamoDB, tableName)
                .AddHashKey("year", DynamoDBEntryType.Numeric)
                .AddRangeKey("title", DynamoDBEntryType.String)
                .Build();

            var filter = new QueryFilter("year", QueryOperator.Equal, year);

            Console.WriteLine("\nFind movies released in: {year}:");
```

```
            var config = new QueryOperationConfig()
            {
                Limit = 10, // 10 items per page.
                Select = SelectValues.SpecificAttributes,
                AttributesToGet = new List<string>
                {
                    "title",
                    "year",
                },
                ConsistentRead = true,
                Filter = filter,
            };

            // Value used to track how many movies match the
            // supplied criteria.
            var moviesFound = 0;

            var search = movieTable.Query(config);
            do
            {
                var movieList = await search.GetNextSetAsync();
                moviesFound += movieList.Count;

                foreach (var movie in movieList)
                {
                    DisplayDocument(movie);
                }
            }
            while (!search.IsDone);

            return moviesFound;
        }
        catch (ResourceNotFoundException ex)
        {
            Console.WriteLine($"Table {tableName} was not found. {ex.Message}");
            return 0;
        }
        catch (AmazonDynamoDBException ex)
        {
            Console.WriteLine($"An Amazon DynamoDB error occurred while querying
movies. {ex.Message}");
            throw;
        }
        catch (Exception ex)
```

```
        {
            Console.WriteLine($"An error occurred while querying movies.
 {ex.Message}");
            throw;
        }
    }
```

Scannt die Tabelle nach Filmen, die in einem bestimmten Zeitraum veröffentlicht wurden.

```
    /// <summary>
    /// Scans the table for movies released between the specified years.
    /// </summary>
    /// <param name="tableName">The name of the table to scan.</param>
    /// <param name="startYear">The starting year for the range.</param>
    /// <param name="endYear">The ending year for the range.</param>
    /// <returns>The number of movies found in the specified year range.</
returns>
    public async Task<int> ScanTableAsync(
        string tableName,
        int startYear,
        int endYear)
    {
        try
        {
            var request = new ScanRequest
            {
                TableName = tableName,
                ExpressionAttributeNames = new Dictionary<string, string>
                {
                    { "#yr", "year" },
                },
                ExpressionAttributeValues = new Dictionary<string,
 AttributeValue>
                {
                    { ":y_a", new AttributeValue { N = startYear.ToString() } },
                    { ":y_z", new AttributeValue { N = endYear.ToString() } },
                },
                FilterExpression = "#yr between :y_a and :y_z",
                ProjectionExpression = "#yr, title, info.actors[0],
 info.directors, info.running_time_secs",
```

```
                Limit = 10 // Set a limit to demonstrate using the
LastEvaluatedKey.
            };

            // Keep track of how many movies were found.
            int foundCount = 0;

            var response = new ScanResponse();
            do
            {
                response = await _amazonDynamoDB.ScanAsync(request);
                foundCount += response.Items.Count;
                response.Items.ForEach(i => DisplayItem(i));
                request.ExclusiveStartKey = response.LastEvaluatedKey;
            }
            while (response?.LastEvaluatedKey?.Count > 0);
            return foundCount;
        }
        catch (ResourceNotFoundException ex)
        {
            Console.WriteLine($"Table {tableName} was not found. {ex.Message}");
            return 0;
        }
        catch (AmazonDynamoDBException ex)
        {
            Console.WriteLine($"An Amazon DynamoDB error occurred while scanning
table. {ex.Message}");
            throw;
        }
        catch (Exception ex)
        {
            Console.WriteLine($"An error occurred while scanning table.
{ex.Message}");
            throw;
        }
    }
```

Löscht die Filmtabelle.

```
    /// <summary>
    /// Deletes a DynamoDB table.
```

```
    /// </summary>
    /// <param name="tableName">The name of the table to delete.</param>
    /// <returns>A Boolean value indicating the success of the operation.</
returns>
    public async Task<bool> DeleteTableAsync(string tableName)
    {
        try
        {
            var request = new DeleteTableRequest
            {
                TableName = tableName,
            };

            var response = await _amazonDynamoDB.DeleteTableAsync(request);

            Console.WriteLine($"Table {response.TableDescription.TableName}
 successfully deleted.");
            return true;

        }
        catch (ResourceNotFoundException ex)
        {
            Console.WriteLine($"Table {tableName} was not found and cannot be
 deleted. {ex.Message}");
            return false;
        }
        catch (AmazonDynamoDBException ex)
        {
            Console.WriteLine($"An Amazon DynamoDB error occurred while deleting
 table {tableName}. {ex.Message}");
            return false;
        }
        catch (Exception ex)
        {
            Console.WriteLine($"An error occurred while deleting table
 {tableName}. {ex.Message}");
            return false;
        }
    }
```

- Weitere API-Informationen finden Sie in den folgenden Themen der AWS SDK für .NET - API-Referenz.
  - BatchWriteItem
  - CreateTable
  - DeleteItem
  - DeleteTable
  - DescribeTable
  - GetItem
  - PutItem
  - Query
  - Scan
  - UpdateItem

Bash

AWS CLI mit Bash-Skript

> **Note**
>
> Es gibt noch mehr dazu. GitHub Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

Das DynamoDB-Szenario „Erste Schritte“.

```
###############################################################################
# function dynamodb_getting_started_movies
#
# Scenario to create an Amazon DynamoDB table and perform a series of operations
 on the table.
#
# Returns:
#       0 - If successful.
#       1 - If an error occurred.
###############################################################################
function dynamodb_getting_started_movies() {
```

```
  source ./dynamodb_operations.sh

  key_schema_json_file="dynamodb_key_schema.json"
  attribute_definitions_json_file="dynamodb_attr_def.json"
  item_json_file="movie_item.json"
  key_json_file="movie_key.json"
  batch_json_file="batch.json"
  attribute_names_json_file="attribute_names.json"
  attributes_values_json_file="attribute_values.json"

  echo_repeat "*" 88
  echo
  echo "Welcome to the Amazon DynamoDB getting started demo."
  echo
  echo_repeat "*" 88
  echo

  local table_name
  echo -n "Enter a name for a new DynamoDB table: "
  get_input
  table_name=$get_input_result

  echo '[
  {"AttributeName": "year", "KeyType": "HASH"},
   {"AttributeName": "title", "KeyType": "RANGE"}
  ]' >"$key_schema_json_file"

  echo '[
  {"AttributeName": "year", "AttributeType": "N"},
   {"AttributeName": "title", "AttributeType": "S"}
  ]' >"$attribute_definitions_json_file"

  if dynamodb_create_table -n "$table_name" -a "$attribute_definitions_json_file"
\
    -k "$key_schema_json_file" 1>/dev/null; then
    echo "Created a DynamoDB table named $table_name"
  else
    errecho "The table failed to create. This demo will exit."
    clean_up
    return 1
  fi

  echo "Waiting for the table to become active...."
```

```
  if dynamodb_wait_table_active -n "$table_name"; then
    echo "The table is now active."
  else
    errecho "The table failed to become active. This demo will exit."
    cleanup "$table_name"
    return 1
  fi

  echo
  echo_repeat "*" 88
  echo

  echo -n "Enter the title of a movie you want to add to the table: "
  get_input
  local added_title
  added_title=$get_input_result

  local added_year
  get_int_input "What year was it released? "
  added_year=$get_input_result

  local rating
  get_float_input "On a scale of 1 - 10, how do you rate it? " "1" "10"
  rating=$get_input_result

  local plot
  echo -n "Summarize the plot for me: "
  get_input
  plot=$get_input_result

  echo '{
    "year": {"N" :"'"$added_year"'"},
    "title": {"S" :  "'"$added_title"'"},
    "info": {"M" : {"plot": {"S" : "'"$plot"'"}, "rating":
{"N" :"'"$rating"'"} } }
  }' >"$item_json_file"

  if dynamodb_put_item -n "$table_name" -i "$item_json_file"; then
    echo "The movie '$added_title' was successfully added to the table
'$table_name'."
  else
    errecho "Put item failed. This demo will exit."
    clean_up "$table_name"
    return 1
```

```
  fi

  echo
  echo_repeat "*" 88
  echo

  echo "Let's update your movie '$added_title'."
  get_float_input "You rated it $rating, what new rating would you give it? " "1"
"10"
  rating=$get_input_result

  echo -n "You summarized the plot as '$plot'."
  echo "What would you say now? "
  get_input
  plot=$get_input_result

  echo '{
    "year": {"N" :"'"$added_year"'"},
    "title": {"S" :  "'"$added_title"'"}
    }' >"$key_json_file"

  echo '{
    ":r": {"N" :"'"$rating"'"},
    ":p": {"S" : "'"$plot"'"}
   }' >"$item_json_file"

  local update_expression="SET info.rating = :r, info.plot = :p"

  if dynamodb_update_item -n "$table_name" -k "$key_json_file" -e
"$update_expression" -v "$item_json_file"; then
    echo "Updated '$added_title' with new attributes."
  else
    errecho "Update item failed. This demo will exit."
    clean_up "$table_name"
    return 1
  fi

  echo
  echo_repeat "*" 88
  echo

  echo "We will now use batch write to upload 150 movie entries into the table."

  local batch_json
```

```
  for batch_json in movie_files/movies_*.json; do
    echo "{ \"$table_name\" : $(<"$batch_json") }" >"$batch_json_file"
    if dynamodb_batch_write_item -i "$batch_json_file" 1>/dev/null; then
      echo "Entries in $batch_json added to table."
    else
      errecho "Batch write failed. This demo will exit."
      clean_up "$table_name"
      return 1
    fi
  done

  local title="The Lord of the Rings: The Fellowship of the Ring"
  local year="2001"

  if get_yes_no_input "Let's move on...do you want to get info about '$title'?
(y/n) "; then
    echo '{
  "year": {"N" :"'"$year"'"},
  "title": {"S" :  "'"$title"'"}
  }' >"$key_json_file"
    local info
    info=$(dynamodb_get_item -n "$table_name" -k "$key_json_file")

    # shellcheck disable=SC2181
    if [[ ${?} -ne 0 ]]; then
      errecho "Get item failed. This demo will exit."
      clean_up "$table_name"
      return 1
    fi

    echo "Here is what I found:"
    echo "$info"
  fi

  local ask_for_year=true
  while [[ "$ask_for_year" == true ]]; do
    echo "Let's get a list of movies released in a given year."
    get_int_input "Enter a year between 1972 and 2018: " "1972" "2018"
    year=$get_input_result
    echo '{
    "#n": "year"
    }' >"$attribute_names_json_file"

    echo '{
```

```
    ":v": {"N" :"'"$year"'"}
    }' >"$attributes_values_json_file"

    response=$(dynamodb_query -n "$table_name" -k "#n=:v" -a
"$attribute_names_json_file" -v "$attributes_values_json_file")

    # shellcheck disable=SC2181
    if [[ ${?} -ne 0 ]]; then
      errecho "Query table failed. This demo will exit."
      clean_up "$table_name"
      return 1
    fi

    echo "Here is what I found:"
    echo "$response"

    if ! get_yes_no_input "Try another year? (y/n) "; then
      ask_for_year=false
    fi
  done

  echo "Now let's scan for movies released in a range of years. Enter a year: "
  get_int_input "Enter a year between 1972 and 2018: " "1972" "2018"
  local start=$get_input_result

  get_int_input "Enter another year: " "1972" "2018"
  local end=$get_input_result

  echo '{
    "#n": "year"
    }' >"$attribute_names_json_file"

  echo '{
    ":v1": {"N" : "'"$start"'"},
    ":v2": {"N" : "'"$end"'"}
    }' >"$attributes_values_json_file"

  response=$(dynamodb_scan -n "$table_name" -f "#n BETWEEN :v1 AND :v2" -a
"$attribute_names_json_file" -v "$attributes_values_json_file")

  # shellcheck disable=SC2181
  if [[ ${?} -ne 0 ]]; then
    errecho "Scan table failed. This demo will exit."
    clean_up "$table_name"
```

```
    return 1
  fi

  echo "Here is what I found:"
  echo "$response"

  echo
  echo_repeat "*" 88
  echo

  echo "Let's remove your movie '$added_title' from the table."

  if get_yes_no_input "Do you want to remove '$added_title'? (y/n) "; then
    echo '{
  "year": {"N" :"'"$added_year"'"},
  "title": {"S" :  "'"$added_title"'"}
  }' >"$key_json_file"

    if ! dynamodb_delete_item -n "$table_name" -k "$key_json_file"; then
      errecho "Delete item failed. This demo will exit."
      clean_up "$table_name"
      return 1
    fi
  fi

  if get_yes_no_input "Do you want to delete the table '$table_name'? (y/n) ";
 then
    if ! clean_up "$table_name"; then
      return 1
    fi
  else
    if ! clean_up; then
      return 1
    fi
  fi

  return 0
}
```

Die in diesem Szenario verwendeten „DynamoDB“-Funktionen.

```
##############################################################################
```

```
# function dynamodb_create_table
#
# This function creates an Amazon DynamoDB table.
#
# Parameters:
#       -n table_name  -- The name of the table to create.
#       -a attribute_definitions -- JSON file path of a list of attributes and
 their types.
#       -k key_schema -- JSON file path of a list of attributes and their key
 types.
#
#  Returns:
#       0 - If successful.
#       1 - If it fails.
###############################################################################
function dynamodb_create_table() {
  local table_name attribute_definitions key_schema response
  local option OPTARG # Required to use getopts command in a function.

  ######################################
  # Function usage explanation
  ######################################
  function usage() {
    echo "function dynamodb_create_table"
    echo "Creates an Amazon DynamoDB table with on-demand billing."
    echo " -n table_name  -- The name of the table to create."
    echo " -a attribute_definitions -- JSON file path of a list of attributes and
 their types."
    echo " -k key_schema -- JSON file path of a list of attributes and their key
 types."
    echo ""
  }

  # Retrieve the calling parameters.
  while getopts "n:a:k:h" option; do
    case "${option}" in
      n) table_name="${OPTARG}" ;;
      a) attribute_definitions="${OPTARG}" ;;
      k) key_schema="${OPTARG}" ;;
      h)
        usage
        return 0
        ;;
      \?)
```

```
        echo "Invalid parameter"
        usage
        return 1
        ;;
    esac
  done
  export OPTIND=1

  if [[ -z "$table_name" ]]; then
    errecho "ERROR: You must provide a table name with the -n parameter."
    usage
    return 1
  fi

  if [[ -z "$attribute_definitions" ]]; then
    errecho "ERROR: You must provide an attribute definitions json file path the
-a parameter."
    usage
    return 1
  fi

  if [[ -z "$key_schema" ]]; then
    errecho "ERROR: You must provide a key schema json file path the -k
parameter."
    usage
    return 1
  fi

  iecho "Parameters:\n"
  iecho "    table_name:   $table_name"
  iecho "    attribute_definitions:   $attribute_definitions"
  iecho "    key_schema:   $key_schema"
  iecho ""

  response=$(aws dynamodb create-table \
    --table-name "$table_name" \
    --attribute-definitions file://"$attribute_definitions" \
    --billing-mode PAY_PER_REQUEST \
    --key-schema file://"$key_schema" )

  local error_code=${?}

  if [[ $error_code -ne 0 ]]; then
    aws_cli_error_log $error_code
```

```
    errecho "ERROR: AWS reports create-table operation failed.$response"
    return 1
  fi

  return 0
}

#############################################################################
# function dynamodb_describe_table
#
# This function returns the status of a DynamoDB table.
#
# Parameters:
#       -n table_name  -- The name of the table.
#
#  Response:
#       - TableStatus:
#     And:
#       0 - Table is active.
#       1 - If it fails.
#############################################################################
function dynamodb_describe_table {
  local table_name
  local option OPTARG # Required to use getopts command in a function.

  ######################################
  # Function usage explanation
  ######################################
  function usage() {
    echo "function dynamodb_describe_table"
    echo "Describe the status of a DynamoDB table."
    echo "  -n table_name  -- The name of the table."
    echo ""
  }

  # Retrieve the calling parameters.
  while getopts "n:h" option; do
    case "${option}" in
      n) table_name="${OPTARG}" ;;
      h)
        usage
        return 0
        ;;
      \?)
```

```
        echo "Invalid parameter"
        usage
        return 1
        ;;
    esac
  done
  export OPTIND=1

  if [[ -z "$table_name" ]]; then
    errecho "ERROR: You must provide a table name with the -n parameter."
    usage
    return 1
  fi

  local table_status
    table_status=$(
      aws dynamodb describe-table \
        --table-name "$table_name" \
        --output text \
        --query 'Table.TableStatus'
    )

   local error_code=${?}

    if [[ $error_code -ne 0 ]]; then
      aws_cli_error_log "$error_code"
      errecho "ERROR: AWS reports describe-table operation failed.$table_status"
      return 1
    fi

  echo "$table_status"

  return 0
}

###############################################################################
# function dynamodb_put_item
#
# This function puts an item into a DynamoDB table.
#
# Parameters:
#       -n table_name  -- The name of the table.
#       -i item  -- Path to json file containing the item values.
#
```

```
#  Returns:
#       0 - If successful.
#       1 - If it fails.
###############################################################################
function dynamodb_put_item() {
  local table_name item response
  local option OPTARG # Required to use getopts command in a function.

  ######################################
  # Function usage explanation
  ######################################
  function usage() {
    echo "function dynamodb_put_item"
    echo "Put an item into a DynamoDB table."
    echo " -n table_name  -- The name of the table."
    echo " -i item  -- Path to json file containing the item values."
    echo ""
  }

  while getopts "n:i:h" option; do
    case "${option}" in
      n) table_name="${OPTARG}" ;;
      i) item="${OPTARG}" ;;
      h)
        usage
        return 0
        ;;
      \?)
        echo "Invalid parameter"
        usage
        return 1
        ;;
    esac
  done
  export OPTIND=1

  if [[ -z "$table_name" ]]; then
    errecho "ERROR: You must provide a table name with the -n parameter."
    usage
    return 1
  fi

  if [[ -z "$item" ]]; then
    errecho "ERROR: You must provide an item with the -i parameter."
```

```
    usage
    return 1
  fi

  iecho "Parameters:\n"
  iecho "    table_name:   $table_name"
  iecho "    item:   $item"
  iecho ""
  iecho ""

  response=$(aws dynamodb put-item \
    --table-name "$table_name" \
    --item file://"$item")

  local error_code=${?}

  if [[ $error_code -ne 0 ]]; then
    aws_cli_error_log $error_code
    errecho "ERROR: AWS reports put-item operation failed.$response"
    return 1
  fi

  return 0

}

###############################################################################
# function dynamodb_update_item
#
# This function updates an item in a DynamoDB table.
#
#
# Parameters:
#       -n table_name  -- The name of the table.
#       -k keys  -- Path to json file containing the keys that identify the item
 to update.
#       -e update expression  -- An expression that defines one or more
 attributes to be updated.
#       -v values  -- Path to json file containing the update values.
#
#  Returns:
#       0 - If successful.
#       1 - If it fails.
###############################################################################
```

```
function dynamodb_update_item() {
  local table_name keys update_expression values response
  local option OPTARG # Required to use getopts command in a function.

  ######################################
  # Function usage explanation
  ######################################
  function usage() {
    echo "function dynamodb_update_item"
    echo "Update an item in a DynamoDB table."
    echo " -n table_name  -- The name of the table."
    echo " -k keys  -- Path to json file containing the keys that identify the
 item to update."
    echo " -e update expression  -- An expression that defines one or more
 attributes to be updated."
    echo " -v values  -- Path to json file containing the update values."
    echo ""
  }

  while getopts "n:k:e:v:h" option; do
    case "${option}" in
      n) table_name="${OPTARG}" ;;
      k) keys="${OPTARG}" ;;
      e) update_expression="${OPTARG}" ;;
      v) values="${OPTARG}" ;;
      h)
        usage
        return 0
        ;;
      \?)
        echo "Invalid parameter"
        usage
        return 1
        ;;
    esac
  done
  export OPTIND=1

  if [[ -z "$table_name" ]]; then
    errecho "ERROR: You must provide a table name with the -n parameter."
    usage
    return 1
  fi
```

```
  if [[ -z "$keys" ]]; then
    errecho "ERROR: You must provide a keys json file path the -k parameter."
    usage
    return 1
  fi
  if [[ -z "$update_expression" ]]; then
    errecho "ERROR: You must provide an update expression with the -e parameter."
    usage
    return 1
  fi

  if [[ -z "$values" ]]; then
    errecho "ERROR: You must provide a values json file path the -v parameter."
    usage
    return 1
  fi

  iecho "Parameters:\n"
  iecho "    table_name:   $table_name"
  iecho "    keys:   $keys"
  iecho "    update_expression:   $update_expression"
  iecho "    values:   $values"

  response=$(aws dynamodb update-item \
    --table-name "$table_name" \
    --key file://"$keys" \
    --update-expression "$update_expression" \
    --expression-attribute-values file://"$values")

  local error_code=${?}

  if [[ $error_code -ne 0 ]]; then
    aws_cli_error_log $error_code
    errecho "ERROR: AWS reports update-item operation failed.$response"
    return 1
  fi

  return 0

}

###############################################################################
# function dynamodb_batch_write_item
#
```

```
# This function writes a batch of items into a DynamoDB table.
#
# Parameters:
#       -i item  -- Path to json file containing the items to write.
#
#  Returns:
#       0 - If successful.
#       1 - If it fails.
###############################################################################
function dynamodb_batch_write_item() {
  local item response
  local option OPTARG # Required to use getopts command in a function.

  ######################################
  # Function usage explanation
  ######################################
  function usage() {
    echo "function dynamodb_batch_write_item"
    echo "Write a batch of items into a DynamoDB table."
    echo " -i item  -- Path to json file containing the items to write."
    echo ""
  }
  while getopts "i:h" option; do
    case "${option}" in
      i) item="${OPTARG}" ;;
      h)
        usage
        return 0
        ;;
      \?)
        echo "Invalid parameter"
        usage
        return 1
        ;;
    esac
  done
  export OPTIND=1

  if [[ -z "$item" ]]; then
    errecho "ERROR: You must provide an item with the -i parameter."
    usage
    return 1
  fi
```

```
  iecho "Parameters:\n"
  iecho "    table_name:   $table_name"
  iecho "    item:   $item"
  iecho ""

  response=$(aws dynamodb batch-write-item \
    --request-items file://"$item")

  local error_code=${?}

  if [[ $error_code -ne 0 ]]; then
    aws_cli_error_log $error_code
    errecho "ERROR: AWS reports batch-write-item operation failed.$response"
    return 1
  fi

  return 0
}

###############################################################################
# function dynamodb_get_item
#
# This function gets an item from a DynamoDB table.
#
# Parameters:
#       -n table_name  -- The name of the table.
#       -k keys  -- Path to json file containing the keys that identify the item
 to get.
#       [-q query]  -- Optional JMESPath query expression.
#
#  Returns:
#       The item as text output.
#  And:
#       0 - If successful.
#       1 - If it fails.
###############################################################################
function dynamodb_get_item() {
  local table_name keys query response
  local option OPTARG # Required to use getopts command in a function.

  # ######################################
  # Function usage explanation
  ######################################
  function usage() {
```

```
    echo "function dynamodb_get_item"
    echo "Get an item from a DynamoDB table."
    echo " -n table_name  -- The name of the table."
    echo " -k keys  -- Path to json file containing the keys that identify the
item to get."
    echo " [-q query]  -- Optional JMESPath query expression."
    echo ""
  }
  query=""
  while getopts "n:k:q:h" option; do
    case "${option}" in
      n) table_name="${OPTARG}" ;;
      k) keys="${OPTARG}" ;;
      q) query="${OPTARG}" ;;
      h)
        usage
        return 0
        ;;
      \?)
        echo "Invalid parameter"
        usage
        return 1
        ;;
    esac
  done
  export OPTIND=1

  if [[ -z "$table_name" ]]; then
    errecho "ERROR: You must provide a table name with the -n parameter."
    usage
    return 1
  fi

  if [[ -z "$keys" ]]; then
    errecho "ERROR: You must provide a keys json file path the -k parameter."
    usage
    return 1
  fi

  if [[ -n "$query" ]]; then
    response=$(aws dynamodb get-item \
      --table-name "$table_name" \
      --key file://"$keys" \
      --output text \
```

```
      --query "$query")
  else
    response=$(
      aws dynamodb get-item \
        --table-name "$table_name" \
        --key file://"$keys" \
        --output text
    )
  fi

  local error_code=${?}

  if [[ $error_code -ne 0 ]]; then
    aws_cli_error_log $error_code
    errecho "ERROR: AWS reports get-item operation failed.$response"
    return 1
  fi

  if [[ -n "$query" ]]; then
    echo "$response" | sed "/^\t/s/\t//1" # Remove initial tab that the JMSEPath
 query inserts on some strings.
  else
    echo "$response"
  fi

  return 0
}

###############################################################################
# function dynamodb_query
#
# This function queries a DynamoDB table.
#
# Parameters:
#       -n table_name  -- The name of the table.
#       -k key_condition_expression -- The key condition expression.
#       -a attribute_names -- Path to JSON file containing the attribute names.
#       -v attribute_values -- Path to JSON file containing the attribute values.
#       [-p projection_expression]  -- Optional projection expression.
#
#  Returns:
#       The items as json output.
#  And:
#       0 - If successful.
```

```
#       1 - If it fails.
###############################################################################
function dynamodb_query() {
  local table_name key_condition_expression attribute_names attribute_values
 projection_expression response
  local option OPTARG # Required to use getopts command in a function.

  # ######################################
  # Function usage explanation
  ######################################
  function usage() {
    echo "function dynamodb_query"
    echo "Query a DynamoDB table."
    echo " -n table_name  -- The name of the table."
    echo " -k key_condition_expression -- The key condition expression."
    echo " -a attribute_names -- Path to JSON file containing the attribute
 names."
    echo " -v attribute_values -- Path to JSON file containing the attribute
 values."
    echo " [-p projection_expression]  -- Optional projection expression."
    echo ""
  }

  while getopts "n:k:a:v:p:h" option; do
    case "${option}" in
      n) table_name="${OPTARG}" ;;
      k) key_condition_expression="${OPTARG}" ;;
      a) attribute_names="${OPTARG}" ;;
      v) attribute_values="${OPTARG}" ;;
      p) projection_expression="${OPTARG}" ;;
      h)
        usage
        return 0
        ;;
      \?)
        echo "Invalid parameter"
        usage
        return 1
        ;;
    esac
  done
  export OPTIND=1

  if [[ -z "$table_name" ]]; then
```

```
    errecho "ERROR: You must provide a table name with the -n parameter."
    usage
    return 1
  fi

  if [[ -z "$key_condition_expression" ]]; then
    errecho "ERROR: You must provide a key condition expression with the -k
parameter."
    usage
    return 1
  fi

  if [[ -z "$attribute_names" ]]; then
    errecho "ERROR: You must provide a attribute names with the -a parameter."
    usage
    return 1
  fi

  if [[ -z "$attribute_values" ]]; then
    errecho "ERROR: You must provide a attribute values with the -v parameter."
    usage
    return 1
  fi

  if [[ -z "$projection_expression" ]]; then
    response=$(aws dynamodb query \
      --table-name "$table_name" \
      --key-condition-expression "$key_condition_expression" \
      --expression-attribute-names file://"$attribute_names" \
      --expression-attribute-values file://"$attribute_values")
  else
    response=$(aws dynamodb query \
      --table-name "$table_name" \
      --key-condition-expression "$key_condition_expression" \
      --expression-attribute-names file://"$attribute_names" \
      --expression-attribute-values file://"$attribute_values" \
      --projection-expression "$projection_expression")
  fi

  local error_code=${?}

  if [[ $error_code -ne 0 ]]; then
    aws_cli_error_log $error_code
    errecho "ERROR: AWS reports query operation failed.$response"
```

```
    return 1
  fi

  echo "$response"

  return 0
}

###############################################################################
# function dynamodb_scan
#
# This function scans a DynamoDB table.
#
# Parameters:
#       -n table_name  -- The name of the table.
#       -f filter_expression  -- The filter expression.
#       -a expression_attribute_names -- Path to JSON file containing the
 expression attribute names.
#       -v expression_attribute_values -- Path to JSON file containing the
 expression attribute values.
#       [-p projection_expression]  -- Optional projection expression.
#
#  Returns:
#       The items as json output.
#  And:
#       0 - If successful.
#       1 - If it fails.
###############################################################################
function dynamodb_scan() {
  local table_name filter_expression expression_attribute_names
 expression_attribute_values projection_expression response
  local option OPTARG # Required to use getopts command in a function.

  # ######################################
  # Function usage explanation
  ######################################
  function usage() {
    echo "function dynamodb_scan"
    echo "Scan a DynamoDB table."
    echo " -n table_name  -- The name of the table."
    echo " -f filter_expression  -- The filter expression."
    echo " -a expression_attribute_names -- Path to JSON file containing the
 expression attribute names."
```

```
    echo " -v expression_attribute_values -- Path to JSON file containing the
expression attribute values."
    echo " [-p projection_expression]  -- Optional projection expression."
    echo ""
 }

 while getopts "n:f:a:v:p:h" option; do
    case "${option}" in
      n) table_name="${OPTARG}" ;;
      f) filter_expression="${OPTARG}" ;;
      a) expression_attribute_names="${OPTARG}" ;;
      v) expression_attribute_values="${OPTARG}" ;;
      p) projection_expression="${OPTARG}" ;;
      h)
        usage
        return 0
        ;;
      \?)
        echo "Invalid parameter"
        usage
        return 1
        ;;
    esac
 done
 export OPTIND=1

 if [[ -z "$table_name" ]]; then
    errecho "ERROR: You must provide a table name with the -n parameter."
    usage
    return 1
 fi

 if [[ -z "$filter_expression" ]]; then
    errecho "ERROR: You must provide a filter expression with the -f parameter."
    usage
    return 1
 fi

 if [[ -z "$expression_attribute_names" ]]; then
    errecho "ERROR: You must provide expression attribute names with the -a
parameter."
    usage
    return 1
 fi
```

```
  if [[ -z "$expression_attribute_values" ]]; then
    errecho "ERROR: You must provide expression attribute values with the -v
 parameter."
    usage
    return 1
  fi

  if [[ -z "$projection_expression" ]]; then
    response=$(aws dynamodb scan \
      --table-name "$table_name" \
      --filter-expression "$filter_expression" \
      --expression-attribute-names file://"$expression_attribute_names" \
      --expression-attribute-values file://"$expression_attribute_values")
  else
    response=$(aws dynamodb scan \
      --table-name "$table_name" \
      --filter-expression "$filter_expression" \
      --expression-attribute-names file://"$expression_attribute_names" \
      --expression-attribute-values file://"$expression_attribute_values" \
      --projection-expression "$projection_expression")
  fi

  local error_code=${?}

  if [[ $error_code -ne 0 ]]; then
    aws_cli_error_log $error_code
    errecho "ERROR: AWS reports scan operation failed.$response"
    return 1
  fi

  echo "$response"

  return 0
}

########################################################################
# function dynamodb_delete_item
#
# This function deletes an item from a DynamoDB table.
#
# Parameters:
#       -n table_name  -- The name of the table.
```

```
#       -k keys  -- Path to json file containing the keys that identify the item
 to delete.
#
#  Returns:
#       0 - If successful.
#       1 - If it fails.
###############################################################################
function dynamodb_delete_item() {
  local table_name keys response
  local option OPTARG # Required to use getopts command in a function.

  # ######################################
  # Function usage explanation
  ######################################
  function usage() {
    echo "function dynamodb_delete_item"
    echo "Delete an item from a DynamoDB table."
    echo " -n table_name  -- The name of the table."
    echo " -k keys  -- Path to json file containing the keys that identify the
 item to delete."
    echo ""
  }
  while getopts "n:k:h" option; do
    case "${option}" in
      n) table_name="${OPTARG}" ;;
      k) keys="${OPTARG}" ;;
      h)
        usage
        return 0
        ;;
      \?)
        echo "Invalid parameter"
        usage
        return 1
        ;;
    esac
  done
  export OPTIND=1

  if [[ -z "$table_name" ]]; then
    errecho "ERROR: You must provide a table name with the -n parameter."
    usage
    return 1
  fi
```

```
  if [[ -z "$keys" ]]; then
    errecho "ERROR: You must provide a keys json file path the -k parameter."
    usage
    return 1
  fi

  iecho "Parameters:\n"
  iecho "    table_name:   $table_name"
  iecho "    keys:   $keys"
  iecho ""

  response=$(aws dynamodb delete-item \
    --table-name "$table_name" \
    --key file://"$keys")

  local error_code=${?}

  if [[ $error_code -ne 0 ]]; then
    aws_cli_error_log $error_code
    errecho "ERROR: AWS reports delete-item operation failed.$response"
    return 1
  fi

  return 0

}

###############################################################################
# function dynamodb_delete_table
#
# This function deletes a DynamoDB table.
#
# Parameters:
#       -n table_name  -- The name of the table to delete.
#
#  Returns:
#       0 - If successful.
#       1 - If it fails.
###############################################################################
function dynamodb_delete_table() {
  local table_name response
  local option OPTARG # Required to use getopts command in a function.
```

```
# bashsupport disable=BP5008
function usage() {
  echo "function dynamodb_delete_table"
  echo "Deletes an Amazon DynamoDB table."
  echo " -n table_name  -- The name of the table to delete."
  echo ""
}

# Retrieve the calling parameters.
while getopts "n:h" option; do
  case "${option}" in
    n) table_name="${OPTARG}" ;;
    h)
      usage
      return 0
      ;;
    \?)
      echo "Invalid parameter"
      usage
      return 1
      ;;
  esac
done
export OPTIND=1

if [[ -z "$table_name" ]]; then
  errecho "ERROR: You must provide a table name with the -n parameter."
  usage
  return 1
fi

iecho "Parameters:\n"
iecho "    table_name:   $table_name"
iecho ""

response=$(aws dynamodb delete-table \
  --table-name "$table_name")

local error_code=${?}

if [[ $error_code -ne 0 ]]; then
  aws_cli_error_log $error_code
  errecho "ERROR: AWS reports delete-table operation failed.$response"
  return 1
```

```
  fi

  return 0
}
```

Die in diesem Szenario verwendeten Hilfsfunktionen.

```
###############################################################################
# function iecho
#
# This function enables the script to display the specified text only if
# the global variable $VERBOSE is set to true.
###############################################################################
function iecho() {
  if [[ $VERBOSE == true ]]; then
    echo "$@"
  fi
}

###############################################################################
# function errecho
#
# This function outputs everything sent to it to STDERR (standard error output).
###############################################################################
function errecho() {
  printf "%s\n" "$*" 1>&2
}

###############################################################################
# function aws_cli_error_log()
#
# This function is used to log the error messages from the AWS CLI.
#
# See https://docs.aws.amazon.com/cli/latest/topic/return-codes.html#cli-aws-
help-return-codes.
#
# The function expects the following argument:
#         $1 - The error code returned by the AWS CLI.
#
#  Returns:
#          0: - Success.
#
```

```
################################################################################
function aws_cli_error_log() {
  local err_code=$1
  errecho "Error code : $err_code"
  if [ "$err_code" == 1 ]; then
    errecho "  One or more S3 transfers failed."
  elif [ "$err_code" == 2 ]; then
    errecho "  Command line failed to parse."
  elif [ "$err_code" == 130 ]; then
    errecho "  Process received SIGINT."
  elif [ "$err_code" == 252 ]; then
    errecho "  Command syntax invalid."
  elif [ "$err_code" == 253 ]; then
    errecho "  The system environment or configuration was invalid."
  elif [ "$err_code" == 254 ]; then
    errecho "  The service returned an error."
  elif [ "$err_code" == 255 ]; then
    errecho "  255 is a catch-all error."
  fi

  return 0
}
```

- Weitere API-Informationen finden Sie in den folgenden Themen der AWS CLI - Befehlsreferenz.
  - BatchWriteItem
  - CreateTable
  - DeleteItem
  - DeleteTable
  - DescribeTable
  - GetItem
  - PutItem
  - Query
  - Scan
  - UpdateItem

C++

SDK für C++

> **Note**
>
> Es gibt noch mehr GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
    {
        Aws::Client::ClientConfiguration clientConfig;
        //  1. Create a table with partition: year (N) and sort: title (S).
 (CreateTable)
        if (AwsDoc::DynamoDB::createMoviesDynamoDBTable(clientConfig)) {

            AwsDoc::DynamoDB::dynamodbGettingStartedScenario(clientConfig);

            // 9. Delete the table. (DeleteTable)
            AwsDoc::DynamoDB::deleteMoviesDynamoDBTable(clientConfig);
        }
    }

//! Scenario to modify and query a DynamoDB table.
/*!
  \sa dynamodbGettingStartedScenario()
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
 */
bool AwsDoc::DynamoDB::dynamodbGettingStartedScenario(
        const Aws::Client::ClientConfiguration &clientConfiguration) {
    std::cout << std::setfill('*') << std::setw(ASTERISK_FILL_WIDTH) << " "
              << std::endl;
    std::cout << "Welcome to the Amazon DynamoDB getting started demo." <<
 std::endl;
    std::cout << std::setfill('*') << std::setw(ASTERISK_FILL_WIDTH) << " "
              << std::endl;

    Aws::DynamoDB::DynamoDBClient dynamoClient(clientConfiguration);

    // 2. Add a new movie.
    Aws::String title;
```

```
   float rating;
   int year;
   Aws::String plot;
   {
       title = askQuestion(
               "Enter the title of a movie you want to add to the table: ");
       year = askQuestionForInt("What year was it released? ");
       rating = askQuestionForFloatRange("On a scale of 1 - 10, how do you rate
it? ",
                                         1, 10);
       plot = askQuestion("Summarize the plot for me: ");

       Aws::DynamoDB::Model::PutItemRequest putItemRequest;
       putItemRequest.SetTableName(MOVIE_TABLE_NAME);

       putItemRequest.AddItem(YEAR_KEY,

Aws::DynamoDB::Model::AttributeValue().SetN(year));
       putItemRequest.AddItem(TITLE_KEY,

Aws::DynamoDB::Model::AttributeValue().SetS(title));

       // Create attribute for the info map.
       Aws::DynamoDB::Model::AttributeValue infoMapAttribute;

       std::shared_ptr<Aws::DynamoDB::Model::AttributeValue> ratingAttribute =
Aws::MakeShared<Aws::DynamoDB::Model::AttributeValue>(
               ALLOCATION_TAG.c_str());
       ratingAttribute->SetN(rating);
       infoMapAttribute.AddMEntry(RATING_KEY, ratingAttribute);

       std::shared_ptr<Aws::DynamoDB::Model::AttributeValue> plotAttribute =
Aws::MakeShared<Aws::DynamoDB::Model::AttributeValue>(
               ALLOCATION_TAG.c_str());
       plotAttribute->SetS(plot);
       infoMapAttribute.AddMEntry(PLOT_KEY, plotAttribute);

       putItemRequest.AddItem(INFO_KEY, infoMapAttribute);

       Aws::DynamoDB::Model::PutItemOutcome outcome = dynamoClient.PutItem(
               putItemRequest);
       if (!outcome.IsSuccess()) {
           std::cerr << "Failed to add an item: " <<
outcome.GetError().GetMessage()
```

```
                    << std::endl;
            return false;
        }
    }

    std::cout << "\nAdded '" << title << "' to '" << MOVIE_TABLE_NAME << "'."
              << std::endl;

    // 3. Update the rating and plot of the movie by using an update expression.
    {
        rating = askQuestionForFloatRange(
                Aws::String("\nLet's update your movie.\nYou rated it  ") +
                std::to_string(rating)
                + ", what new rating would you give it? ", 1, 10);
        plot = askQuestion(Aws::String("You summarized the plot as '") + plot +
                           "'.\nWhat would you say now? ");

        Aws::DynamoDB::Model::UpdateItemRequest request;
        request.SetTableName(MOVIE_TABLE_NAME);
        request.AddKey(TITLE_KEY,
Aws::DynamoDB::Model::AttributeValue().SetS(title));
        request.AddKey(YEAR_KEY,
Aws::DynamoDB::Model::AttributeValue().SetN(year));
        std::stringstream expressionStream;
        expressionStream << "set " << INFO_KEY << "." << RATING_KEY << " =:r, "
                         << INFO_KEY << "." << PLOT_KEY << " =:p";
        request.SetUpdateExpression(expressionStream.str());
        request.SetExpressionAttributeValues({
                                                     {":r",
Aws::DynamoDB::Model::AttributeValue().SetN(
                                                             rating)},
                                                     {":p",
Aws::DynamoDB::Model::AttributeValue().SetS(
                                                             plot)}
                                             });

        request.SetReturnValues(Aws::DynamoDB::Model::ReturnValue::UPDATED_NEW);

        const Aws::DynamoDB::Model::UpdateItemOutcome &result =
dynamoClient.UpdateItem(
                request);
        if (!result.IsSuccess()) {
            std::cerr << "Error updating movie " + result.GetError().GetMessage()
                      << std::endl;
```

```
            return false;
        }
    }

    std::cout << "\nUpdated '" << title << "' with new attributes:" << std::endl;

    // 4. Put 250 movies in the table from moviedata.json.
    {
        std::cout << "Adding movies from a json file to the database." <<
std::endl;
        const size_t MAX_SIZE_FOR_BATCH_WRITE = 25;
        const size_t MOVIES_TO_WRITE = 10 * MAX_SIZE_FOR_BATCH_WRITE;
        Aws::String jsonString = getMovieJSON();
        if (!jsonString.empty()) {
            Aws::Utils::Json::JsonValue json(jsonString);
            Aws::Utils::Array<Aws::Utils::Json::JsonView> movieJsons =
json.View().AsArray();
            Aws::Vector<Aws::DynamoDB::Model::WriteRequest> writeRequests;

            // To add movies with a cross-section of years, use an appropriate
increment
            // value for iterating through the database.
            size_t increment = movieJsons.GetLength() / MOVIES_TO_WRITE;
            for (size_t i = 0; i < movieJsons.GetLength(); i += increment) {
                writeRequests.push_back(Aws::DynamoDB::Model::WriteRequest());
                Aws::Map<Aws::String, Aws::DynamoDB::Model::AttributeValue>
putItems = movieJsonViewToAttributeMap(
                        movieJsons[i]);
                Aws::DynamoDB::Model::PutRequest putRequest;
                putRequest.SetItem(putItems);
                writeRequests.back().SetPutRequest(putRequest);
                if (writeRequests.size() == MAX_SIZE_FOR_BATCH_WRITE) {
                    Aws::DynamoDB::Model::BatchWriteItemRequest request;
                    request.AddRequestItems(MOVIE_TABLE_NAME, writeRequests);
                    const Aws::DynamoDB::Model::BatchWriteItemOutcome &outcome =
dynamoClient.BatchWriteItem(
                            request);
                    if (!outcome.IsSuccess()) {
                        std::cerr << "Unable to batch write movie data: "
                                  << outcome.GetError().GetMessage()
                                  << std::endl;
                        writeRequests.clear();
                        break;
                    }
```

```
                    else {
                        std::cout << "Added batch of " << writeRequests.size()
                                  << " movies to the database."
                                  << std::endl;
                    }
                    writeRequests.clear();
                }
            }
        }

    std::cout << std::setfill('*') << std::setw(ASTERISK_FILL_WIDTH) << " "
              << std::endl;

    // 5. Get a movie by Key (partition + sort).
    {
        Aws::String titleToGet("King Kong");
        Aws::String answer = askQuestion(Aws::String(
                "Let's move on...Would you like to get info about '" + titleToGet
+
                "'? (y/n) "));
        if (answer == "y") {
            Aws::DynamoDB::Model::GetItemRequest request;
            request.SetTableName(MOVIE_TABLE_NAME);
            request.AddKey(TITLE_KEY,

Aws::DynamoDB::Model::AttributeValue().SetS(titleToGet));
            request.AddKey(YEAR_KEY,
Aws::DynamoDB::Model::AttributeValue().SetN(1933));

            const Aws::DynamoDB::Model::GetItemOutcome &result =
dynamoClient.GetItem(
                    request);
            if (!result.IsSuccess()) {
                std::cerr << "Error " << result.GetError().GetMessage();
            }
            else {
                const Aws::Map<Aws::String, Aws::DynamoDB::Model::AttributeValue>
&item = result.GetResult().GetItem();
                if (!item.empty()) {
                    std::cout << "\nHere's what I found:" << std::endl;
                    printMovieInfo(item);
                }
                else {
```

```
                    std::cout << "\nThe movie was not found in the database."
                              << std::endl;
                }
            }
        }
    }

    // 6. Use Query with a key condition expression to return all movies
    //    released in a given year.
    Aws::String doAgain = "n";
    do {
        Aws::DynamoDB::Model::QueryRequest req;

        req.SetTableName(MOVIE_TABLE_NAME);

        // "year" is a DynamoDB reserved keyword and must be replaced with an
        // expression attribute name.
        req.SetKeyConditionExpression("#dynobase_year = :valueToMatch");
        req.SetExpressionAttributeNames({{"#dynobase_year", YEAR_KEY}});

        int yearToMatch = askQuestionForIntRange(
                "\nLet's get a list of movies released in"
                " a given year. Enter a year between 1972 and 2018 ",
                1972, 2018);
        Aws::Map<Aws::String, Aws::DynamoDB::Model::AttributeValue>
attributeValues;
        attributeValues.emplace(":valueToMatch",
                                Aws::DynamoDB::Model::AttributeValue().SetN(
                                        yearToMatch));
        req.SetExpressionAttributeValues(attributeValues);

        const Aws::DynamoDB::Model::QueryOutcome &result =
dynamoClient.Query(req);
        if (result.IsSuccess()) {
            const Aws::Vector<Aws::Map<Aws::String,
Aws::DynamoDB::Model::AttributeValue>> &items = result.GetResult().GetItems();
            if (!items.empty()) {
                std::cout << "\nThere were " << items.size()
                          << " movies in the database from "
                          << yearToMatch << "." << std::endl;
                for (const auto &item: items) {
                    printMovieInfo(item);
                }
                doAgain = "n";
```

```
            }
            else {
                std::cout << "\nNo movies from " << yearToMatch
                          << " were found in the database"
                          << std::endl;
                doAgain = askQuestion(Aws::String("Try another year? (y/n) "));
            }
        }
        else {
            std::cerr << "Failed to Query items: " <<
result.GetError().GetMessage()
                      << std::endl;
        }

    } while (doAgain == "y");

    //  7. Use Scan to return movies released within a range of years.
    //     Show how to paginate data using ExclusiveStartKey. (Scan +
FilterExpression)
    {
        int startYear = askQuestionForIntRange("\nNow let's scan a range of years
"
                                               "for movies in the database. Enter
a start year: ",
                                               1972, 2018);
        int endYear = askQuestionForIntRange("\nEnter an end year: ",
                                             startYear, 2018);
        Aws::Map<Aws::String, Aws::DynamoDB::Model::AttributeValue>
exclusiveStartKey;
        do {
            Aws::DynamoDB::Model::ScanRequest scanRequest;
            scanRequest.SetTableName(MOVIE_TABLE_NAME);
            scanRequest.SetFilterExpression(
                    "#dynobase_year >= :startYear AND #dynobase_year
<= :endYear");
            scanRequest.SetExpressionAttributeNames({{"#dynobase_year",
YEAR_KEY}});

            Aws::Map<Aws::String, Aws::DynamoDB::Model::AttributeValue>
attributeValues;
            attributeValues.emplace(":startYear",
                                    Aws::DynamoDB::Model::AttributeValue().SetN(
                                            startYear));
            attributeValues.emplace(":endYear",
```

```
                                    Aws::DynamoDB::Model::AttributeValue().SetN(
                                            endYear));
            scanRequest.SetExpressionAttributeValues(attributeValues);

            if (!exclusiveStartKey.empty()) {
                scanRequest.SetExclusiveStartKey(exclusiveStartKey);
            }

            const Aws::DynamoDB::Model::ScanOutcome &result = dynamoClient.Scan(
                    scanRequest);
            if (result.IsSuccess()) {
                const Aws::Vector<Aws::Map<Aws::String,
Aws::DynamoDB::Model::AttributeValue>> &items = result.GetResult().GetItems();
                if (!items.empty()) {
                    std::stringstream stringStream;
                    stringStream << "\nFound " << items.size() << " movies in one
scan."
                                 << " How many would you like to see? ";
                    size_t count = askQuestionForInt(stringStream.str());
                    for (size_t i = 0; i < count && i < items.size(); ++i) {
                        printMovieInfo(items[i]);
                    }
                }
                else {
                    std::cout << "\nNo movies in the database between " <<
startYear <<
                              " and " << endYear << "." << std::endl;
                }

                exclusiveStartKey = result.GetResult().GetLastEvaluatedKey();
                if (!exclusiveStartKey.empty()) {
                    std::cout << "Not all movies were retrieved. Scanning for
more."
                              << std::endl;
                }
                else {
                    std::cout << "All movies were retrieved with this scan."
                              << std::endl;
                }
            }
            else {
                std::cerr << "Failed to Scan movies: "
                          << result.GetError().GetMessage() << std::endl;
            }
```

```
        } while (!exclusiveStartKey.empty());
    }

    // 8. Delete a movie. (DeleteItem)
    {
        std::stringstream stringStream;
        stringStream << "\nWould you like to delete the movie " << title
                     << " from the database? (y/n) ";
        Aws::String answer = askQuestion(stringStream.str());
        if (answer == "y") {
            Aws::DynamoDB::Model::DeleteItemRequest request;
            request.AddKey(YEAR_KEY,
 Aws::DynamoDB::Model::AttributeValue().SetN(year));
            request.AddKey(TITLE_KEY,
                           Aws::DynamoDB::Model::AttributeValue().SetS(title));
            request.SetTableName(MOVIE_TABLE_NAME);

            const Aws::DynamoDB::Model::DeleteItemOutcome &result =
 dynamoClient.DeleteItem(
                    request);
            if (result.IsSuccess()) {
                std::cout << "\nRemoved \"" << title << "\" from the database."
                          << std::endl;
            }
            else {
                std::cerr << "Failed to delete the movie: "
                          << result.GetError().GetMessage()
                          << std::endl;
            }
        }
    }

    return true;
}

//! Routine to convert a JsonView object to an attribute map.
/*!
  \sa movieJsonViewToAttributeMap()
  \param jsonView: Json view object.
  \return map: Map that can be used in a DynamoDB request.
 */
Aws::Map<Aws::String, Aws::DynamoDB::Model::AttributeValue>
AwsDoc::DynamoDB::movieJsonViewToAttributeMap(
        const Aws::Utils::Json::JsonView &jsonView) {
```

```
    Aws::Map<Aws::String, Aws::DynamoDB::Model::AttributeValue> result;

    if (jsonView.KeyExists(YEAR_KEY)) {
        result[YEAR_KEY].SetN(jsonView.GetInteger(YEAR_KEY));
    }
    if (jsonView.KeyExists(TITLE_KEY)) {
        result[TITLE_KEY].SetS(jsonView.GetString(TITLE_KEY));
    }
    if (jsonView.KeyExists(INFO_KEY)) {
        Aws::Map<Aws::String, const
 std::shared_ptr<Aws::DynamoDB::Model::AttributeValue>> infoMap;
        Aws::Utils::Json::JsonView infoView = jsonView.GetObject(INFO_KEY);
        if (infoView.KeyExists(RATING_KEY)) {
            std::shared_ptr<Aws::DynamoDB::Model::AttributeValue> attributeValue
 = std::make_shared<Aws::DynamoDB::Model::AttributeValue>();
            attributeValue->SetN(infoView.GetDouble(RATING_KEY));
            infoMap.emplace(std::make_pair(RATING_KEY, attributeValue));
        }
        if (infoView.KeyExists(PLOT_KEY)) {
            std::shared_ptr<Aws::DynamoDB::Model::AttributeValue> attributeValue
 = std::make_shared<Aws::DynamoDB::Model::AttributeValue>();
            attributeValue->SetS(infoView.GetString(PLOT_KEY));
            infoMap.emplace(std::make_pair(PLOT_KEY, attributeValue));
        }

        result[INFO_KEY].SetM(infoMap);
    }

    return result;
}

//! Create a DynamoDB table to be used in sample code scenarios.
/*!
  \sa createMoviesDynamoDBTable()
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
*/
bool AwsDoc::DynamoDB::createMoviesDynamoDBTable(
        const Aws::Client::ClientConfiguration &clientConfiguration) {
    Aws::DynamoDB::DynamoDBClient dynamoClient(clientConfiguration);

    bool movieTableAlreadyExisted = false;

    {
```

```
        Aws::DynamoDB::Model::CreateTableRequest request;

        Aws::DynamoDB::Model::AttributeDefinition yearAttributeDefinition;
        yearAttributeDefinition.SetAttributeName(YEAR_KEY);
        yearAttributeDefinition.SetAttributeType(
                Aws::DynamoDB::Model::ScalarAttributeType::N);
        request.AddAttributeDefinitions(yearAttributeDefinition);

        Aws::DynamoDB::Model::AttributeDefinition titleAttributeDefinition;
        yearAttributeDefinition.SetAttributeName(TITLE_KEY);
        yearAttributeDefinition.SetAttributeType(
                Aws::DynamoDB::Model::ScalarAttributeType::S);
        request.AddAttributeDefinitions(yearAttributeDefinition);

        Aws::DynamoDB::Model::KeySchemaElement yearKeySchema;
        yearKeySchema.WithAttributeName(YEAR_KEY).WithKeyType(
                Aws::DynamoDB::Model::KeyType::HASH);
        request.AddKeySchema(yearKeySchema);

        Aws::DynamoDB::Model::KeySchemaElement titleKeySchema;
        yearKeySchema.WithAttributeName(TITLE_KEY).WithKeyType(
                Aws::DynamoDB::Model::KeyType::RANGE);
        request.AddKeySchema(yearKeySchema);

        Aws::DynamoDB::Model::ProvisionedThroughput throughput;
        throughput.WithReadCapacityUnits(
                PROVISIONED_THROUGHPUT_UNITS).WithWriteCapacityUnits(
                PROVISIONED_THROUGHPUT_UNITS);
        request.SetProvisionedThroughput(throughput);
        request.SetTableName(MOVIE_TABLE_NAME);

        std::cout << "Creating table '" << MOVIE_TABLE_NAME << "'..." <<
std::endl;
        const Aws::DynamoDB::Model::CreateTableOutcome &result =
dynamoClient.CreateTable(
                request);
        if (!result.IsSuccess()) {
            if (result.GetError().GetErrorType() ==
                Aws::DynamoDB::DynamoDBErrors::RESOURCE_IN_USE) {
                std::cout << "Table already exists." << std::endl;
                movieTableAlreadyExisted = true;
            }
            else {
                std::cerr << "Failed to create table: "
```

```
                          << result.GetError().GetMessage();
                return false;
            }
        }
    }

    // Wait for table to become active.
    if (!movieTableAlreadyExisted) {
        std::cout << "Waiting for table '" << MOVIE_TABLE_NAME
                  << "' to become active...." << std::endl;
        if (!AwsDoc::DynamoDB::waitTableActive(MOVIE_TABLE_NAME,
 clientConfiguration)) {
            return false;
        }
        std::cout << "Table '" << MOVIE_TABLE_NAME << "' created and active."
                  << std::endl;
    }

    return true;
}

//! Delete the DynamoDB table used for sample code scenarios.
/*!
  \sa deleteMoviesDynamoDBTable()
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
*/
bool AwsDoc::DynamoDB::deleteMoviesDynamoDBTable(
        const Aws::Client::ClientConfiguration &clientConfiguration) {
    Aws::DynamoDB::DynamoDBClient dynamoClient(clientConfiguration);

    Aws::DynamoDB::Model::DeleteTableRequest request;
    request.SetTableName(MOVIE_TABLE_NAME);

    const Aws::DynamoDB::Model::DeleteTableOutcome &result =
 dynamoClient.DeleteTable(
            request);
    if (result.IsSuccess()) {
        std::cout << "Your table \""
                  << result.GetResult().GetTableDescription().GetTableName()
                  << " was deleted.\n";
    }
    else {
        std::cerr << "Failed to delete table: " << result.GetError().GetMessage()
```

```
                  << std::endl;
    }

    return result.IsSuccess();
}

//! Query a newly created DynamoDB table until it is active.
/*!
  \sa waitTableActive()
  \param waitTableActive: The DynamoDB table's name.
  \param dynamoClient: A DynamoDB client.
  \return bool: Function succeeded.
*/
bool AwsDoc::DynamoDB::waitTableActive(const Aws::String &tableName,
                                       const Aws::DynamoDB::DynamoDBClient
 &dynamoClient) {

    // Repeatedly call DescribeTable until table is ACTIVE.
    const int MAX_QUERIES = 20;
    Aws::DynamoDB::Model::DescribeTableRequest request;
    request.SetTableName(tableName);

    int count = 0;
    while (count < MAX_QUERIES) {
        const Aws::DynamoDB::Model::DescribeTableOutcome &result =
 dynamoClient.DescribeTable(
                request);
        if (result.IsSuccess()) {
            Aws::DynamoDB::Model::TableStatus status =
 result.GetResult().GetTable().GetTableStatus();

            if (Aws::DynamoDB::Model::TableStatus::ACTIVE != status) {
                std::this_thread::sleep_for(std::chrono::seconds(1));
            }
            else {
                return true;
            }
        }
        else {
            std::cerr << "Error DynamoDB::waitTableActive "
                      << result.GetError().GetMessage() << std::endl;
            return false;
        }
        count++;
```

```
    }
    return false;
}
```

- Weitere API-Informationen finden Sie in den folgenden Themen der AWS SDK für C++ -API-Referenz.
  - BatchWriteItem
  - CreateTable
  - DeleteItem
  - DeleteTable
  - DescribeTable
  - GetItem
  - PutItem
  - Query
  - Scan
  - UpdateItem

Go

SDK für Go V2

> **Note**
>
> Es gibt noch mehr GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

Führen Sie ein interaktives Szenario aus, um die Tabelle zu erstellen und Aktionen darauf auszuführen.

```
import (
 "context"
 "fmt"
 "log"
 "strings"
```

```
 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb"
 "github.com/awsdocs/aws-doc-sdk-examples/gov2/demotools"
 "github.com/awsdocs/aws-doc-sdk-examples/gov2/dynamodb/actions"
)

// RunMovieScenario is an interactive example that shows you how to use the AWS
 SDK for Go
// to create and use an Amazon DynamoDB table that stores data about movies.
//
//  1. Create a table that can hold movie data.
//  2. Put, get, and update a single movie in the table.
//  3. Write movie data to the table from a sample JSON file.
//  4. Query for movies that were released in a given year.
//  5. Scan for movies that were released in a range of years.
//  6. Delete a movie from the table.
//  7. Delete the table.
//
// This example creates a DynamoDB service client from the specified sdkConfig so
 that
// you can replace it with a mocked or stubbed config for unit testing.
//
// It uses a questioner from the `demotools` package to get input during the
 example.
// This package can be found in the ..\..\demotools folder of this repo.
//
// The specified movie sampler is used to get sample data from a URL that is
 loaded
// into the named table.
func RunMovieScenario(
 ctx context.Context, sdkConfig aws.Config, questioner demotools.IQuestioner,
 tableName string,
 movieSampler actions.IMovieSampler) {
 defer func() {
  if r := recover(); r != nil {
   fmt.Printf("Something went wrong with the demo.")
  }
 }()

 log.Println(strings.Repeat("-", 88))
 log.Println("Welcome to the Amazon DynamoDB getting started demo.")
 log.Println(strings.Repeat("-", 88))
```

```
tableBasics := actions.TableBasics{TableName: tableName,
 DynamoDbClient: dynamodb.NewFromConfig(sdkConfig)}

exists, err := tableBasics.TableExists(ctx)
if err != nil {
 panic(err)
}
if !exists {
 log.Printf("Creating table %v...\n", tableName)
 _, err = tableBasics.CreateMovieTable(ctx)
 if err != nil {
  panic(err)
 } else {
  log.Printf("Created table %v.\n", tableName)
 }
} else {
 log.Printf("Table %v already exists.\n", tableName)
}

var customMovie actions.Movie
customMovie.Title = questioner.Ask("Enter a movie title to add to the table:",
 demotools.NotEmpty{})
customMovie.Year = questioner.AskInt("What year was it released?",
 demotools.NotEmpty{}, demotools.InIntRange{Lower: 1900, Upper: 2030})
customMovie.Info = map[string]interface{}{}
customMovie.Info["rating"] = questioner.AskFloat64(
 "Enter a rating between 1 and 10:",
 demotools.NotEmpty{}, demotools.InFloatRange{Lower: 1, Upper: 10})
customMovie.Info["plot"] = questioner.Ask("What's the plot? ",
 demotools.NotEmpty{})
err = tableBasics.AddMovie(ctx, customMovie)
if err == nil {
 log.Printf("Added %v to the movie table.\n", customMovie.Title)
}
log.Println(strings.Repeat("-", 88))

log.Printf("Let's update your movie. You previously rated it %v.\n",
customMovie.Info["rating"])
customMovie.Info["rating"] = questioner.AskFloat64(
 "What new rating would you give it?",
 demotools.NotEmpty{}, demotools.InFloatRange{Lower: 1, Upper: 10})
log.Printf("You summarized the plot as '%v'.\n", customMovie.Info["plot"])
customMovie.Info["plot"] = questioner.Ask("What would you say now?",
 demotools.NotEmpty{})
```

```
attributes, err := tableBasics.UpdateMovie(ctx, customMovie)
if err == nil {
 log.Printf("Updated %v with new values.\n", customMovie.Title)
 for _, attVal := range attributes {
  for valKey, val := range attVal {
   log.Printf("\t%v: %v\n", valKey, val)
  }
 }
}
log.Println(strings.Repeat("-", 88))

log.Printf("Getting movie data from %v and adding 250 movies to the table...\n",
 movieSampler.GetURL())
movies := movieSampler.GetSampleMovies()
written, err := tableBasics.AddMovieBatch(ctx, movies, 250)
if err != nil {
 panic(err)
} else {
 log.Printf("Added %v movies to the table.\n", written)
}

show := 10
if show > written {
 show = written
}
log.Printf("The first %v movies in the table are:", show)
for index, movie := range movies[:show] {
 log.Printf("\t%v. %v\n", index+1, movie.Title)
}
movieIndex := questioner.AskInt(
 "Enter the number of a movie to get info about it: ",
 demotools.InIntRange{Lower: 1, Upper: show},
)
movie, err := tableBasics.GetMovie(ctx, movies[movieIndex-1].Title,
movies[movieIndex-1].Year)
if err == nil {
 log.Println(movie)
}
log.Println(strings.Repeat("-", 88))

log.Println("Let's get a list of movies released in a given year.")
releaseYear := questioner.AskInt("Enter a year between 1972 and 2018: ",
 demotools.InIntRange{Lower: 1972, Upper: 2018},
)
```

```
releases, err := tableBasics.Query(ctx, releaseYear)
if err == nil {
 if len(releases) == 0 {
  log.Printf("I couldn't find any movies released in %v!\n", releaseYear)
 } else {
  for _, movie = range releases {
   log.Println(movie)
  }
 }
}
log.Println(strings.Repeat("-", 88))

log.Println("Now let's scan for movies released in a range of years.")
startYear := questioner.AskInt("Enter a year: ",
 demotools.InIntRange{Lower: 1972, Upper: 2018})
endYear := questioner.AskInt("Enter another year: ",
 demotools.InIntRange{Lower: 1972, Upper: 2018})
releases, err = tableBasics.Scan(ctx, startYear, endYear)
if err == nil {
 if len(releases) == 0 {
  log.Printf("I couldn't find any movies released between %v and %v!\n",
startYear, endYear)
 } else {
  log.Printf("Found %v movies. In this list, the plot is <nil> because "+
   "we used a projection expression when scanning for items to return only "+
   "the title, year, and rating.\n", len(releases))
  for _, movie = range releases {
   log.Println(movie)
  }
 }
}
log.Println(strings.Repeat("-", 88))

var tables []string
if questioner.AskBool("Do you want to list all of your tables? (y/n) ", "y") {
 tables, err = tableBasics.ListTables(ctx)
 if err == nil {
  log.Printf("Found %v tables:", len(tables))
  for _, table := range tables {
   log.Printf("\t%v", table)
  }
 }
}
log.Println(strings.Repeat("-", 88))
```

```
 log.Printf("Let's remove your movie '%v'.\n", customMovie.Title)
 if questioner.AskBool("Do you want to delete it from the table? (y/n) ", "y") {
  err = tableBasics.DeleteMovie(ctx, customMovie)
 }
 if err == nil {
  log.Printf("Deleted %v.\n", customMovie.Title)
 }

 if questioner.AskBool("Delete the table, too? (y/n)", "y") {
  err = tableBasics.DeleteTable(ctx)
 } else {
  log.Println("Don't forget to delete the table when you're done or you might " +
   "incur charges on your account.")
 }
 if err == nil {
  log.Printf("Deleted table %v.\n", tableBasics.TableName)
 }

 log.Println(strings.Repeat("-", 88))
 log.Println("Thanks for watching!")
 log.Println(strings.Repeat("-", 88))
}
```

Definieren Sie eine Movie-Struktur, die in diesem Beispiel verwendet wird.

```
import (
 "archive/zip"
 "bytes"
 "encoding/json"
 "fmt"
 "io"
 "log"
 "net/http"

 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/attributevalue"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb/types"
)
```

```
// Movie encapsulates data about a movie. Title and Year are the composite
 primary key
// of the movie in Amazon DynamoDB. Title is the sort key, Year is the partition
 key,
// and Info is additional data.
type Movie struct {
 Title string                `dynamodbav:"title"`
 Year  int                   `dynamodbav:"year"`
 Info  map[string]interface{} `dynamodbav:"info"`
}

// GetKey returns the composite primary key of the movie in a format that can be
// sent to DynamoDB.
func (movie Movie) GetKey() map[string]types.AttributeValue {
 title, err := attributevalue.Marshal(movie.Title)
 if err != nil {
  panic(err)
 }
 year, err := attributevalue.Marshal(movie.Year)
 if err != nil {
  panic(err)
 }
 return map[string]types.AttributeValue{"title": title, "year": year}
}

// String returns the title, year, rating, and plot of a movie, formatted for the
 example.
func (movie Movie) String() string {
 return fmt.Sprintf("%v\n\tReleased: %v\n\tRating: %v\n\tPlot: %v\n",
  movie.Title, movie.Year, movie.Info["rating"], movie.Info["plot"])
}
```

Erstellen Sie eine Struktur und Methoden, die DynamoDB-Aktionen aufrufen.

```
import (
 "context"
 "errors"
 "log"
 "time"
```

```
 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/attributevalue"
 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/expression"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb/types"
)

// TableBasics encapsulates the Amazon DynamoDB service actions used in the
 examples.
// It contains a DynamoDB service client that is used to act on the specified
 table.
type TableBasics struct {
 DynamoDbClient *dynamodb.Client
 TableName      string
}



// TableExists determines whether a DynamoDB table exists.
func (basics TableBasics) TableExists(ctx context.Context) (bool, error) {
 exists := true
 _, err := basics.DynamoDbClient.DescribeTable(
  ctx, &dynamodb.DescribeTableInput{TableName: aws.String(basics.TableName)},
 )
 if err != nil {
  var notFoundEx *types.ResourceNotFoundException
  if errors.As(err, &notFoundEx) {
   log.Printf("Table %v does not exist.\n", basics.TableName)
   err = nil
  } else {
   log.Printf("Couldn't determine existence of table %v. Here's why: %v\n",
 basics.TableName, err)
  }
  exists = false
 }
 return exists, err
}



// CreateMovieTable creates a DynamoDB table with a composite primary key defined
 as
// a string sort key named `title`, and a numeric partition key named `year`.
// This function uses NewTableExistsWaiter to wait for the table to be created by
```

```
// DynamoDB before it returns.
func (basics TableBasics) CreateMovieTable(ctx context.Context)
 (*types.TableDescription, error) {
 var tableDesc *types.TableDescription
 table, err := basics.DynamoDbClient.CreateTable(ctx, &dynamodb.CreateTableInput{
  AttributeDefinitions: []types.AttributeDefinition{{
   AttributeName: aws.String("year"),
   AttributeType: types.ScalarAttributeTypeN,
  }, {
   AttributeName: aws.String("title"),
   AttributeType: types.ScalarAttributeTypeS,
  }},
  KeySchema: []types.KeySchemaElement{{
   AttributeName: aws.String("year"),
   KeyType:       types.KeyTypeHash,
  }, {
   AttributeName: aws.String("title"),
   KeyType:       types.KeyTypeRange,
  }},
  TableName:   aws.String(basics.TableName),
  BillingMode: types.BillingModePayPerRequest,
 })
 if err != nil {
  log.Printf("Couldn't create table %v. Here's why: %v\n", basics.TableName, err)
 } else {
  waiter := dynamodb.NewTableExistsWaiter(basics.DynamoDbClient)
  err = waiter.Wait(ctx, &dynamodb.DescribeTableInput{
   TableName: aws.String(basics.TableName)}, 5*time.Minute)
  if err != nil {
   log.Printf("Wait for table exists failed. Here's why: %v\n", err)
  }
  tableDesc = table.TableDescription
  log.Printf("Ccreating table test")
 }
 return tableDesc, err
}



// ListTables lists the DynamoDB table names for the current account.
func (basics TableBasics) ListTables(ctx context.Context) ([]string, error) {
 var tableNames []string
 var output *dynamodb.ListTablesOutput
 var err error
```

```
 tablePaginator := dynamodb.NewListTablesPaginator(basics.DynamoDbClient,
 &dynamodb.ListTablesInput{})
 for tablePaginator.HasMorePages() {
  output, err = tablePaginator.NextPage(ctx)
  if err != nil {
   log.Printf("Couldn't list tables. Here's why: %v\n", err)
   break
  } else {
   tableNames = append(tableNames, output.TableNames...)
  }
 }
 return tableNames, err
}



// AddMovie adds a movie the DynamoDB table.
func (basics TableBasics) AddMovie(ctx context.Context, movie Movie) error {
 item, err := attributevalue.MarshalMap(movie)
 if err != nil {
  panic(err)
 }
 _, err = basics.DynamoDbClient.PutItem(ctx, &dynamodb.PutItemInput{
  TableName: aws.String(basics.TableName), Item: item,
 })
 if err != nil {
  log.Printf("Couldn't add item to table. Here's why: %v\n", err)
 }
 return err
}



// UpdateMovie updates the rating and plot of a movie that already exists in the
// DynamoDB table. This function uses the `expression` package to build the
 update
// expression.
func (basics TableBasics) UpdateMovie(ctx context.Context, movie Movie)
 (map[string]map[string]interface{}, error) {
 var err error
 var response *dynamodb.UpdateItemOutput
 var attributeMap map[string]map[string]interface{}
 update := expression.Set(expression.Name("info.rating"),
 expression.Value(movie.Info["rating"]))
```

```
 update.Set(expression.Name("info.plot"), expression.Value(movie.Info["plot"]))
 expr, err := expression.NewBuilder().WithUpdate(update).Build()
 if err != nil {
  log.Printf("Couldn't build expression for update. Here's why: %v\n", err)
 } else {
  response, err = basics.DynamoDbClient.UpdateItem(ctx,
 &dynamodb.UpdateItemInput{
   TableName:                 aws.String(basics.TableName),
   Key:                       movie.GetKey(),
   ExpressionAttributeNames:  expr.Names(),
   ExpressionAttributeValues: expr.Values(),
   UpdateExpression:          expr.Update(),
   ReturnValues:              types.ReturnValueUpdatedNew,
  })
  if err != nil {
   log.Printf("Couldn't update movie %v. Here's why: %v\n", movie.Title, err)
  } else {
   err = attributevalue.UnmarshalMap(response.Attributes, &attributeMap)
   if err != nil {
    log.Printf("Couldn't unmarshall update response. Here's why: %v\n", err)
   }
  }
 }
 return attributeMap, err
}



// AddMovieBatch adds a slice of movies to the DynamoDB table. The function sends
// batches of 25 movies to DynamoDB until all movies are added or it reaches the
// specified maximum.
func (basics TableBasics) AddMovieBatch(ctx context.Context, movies []Movie,
 maxMovies int) (int, error) {
 var err error
 var item map[string]types.AttributeValue
 written := 0
 batchSize := 25 // DynamoDB allows a maximum batch size of 25 items.
 start := 0
 end := start + batchSize
 for start < maxMovies && start < len(movies) {
  var writeReqs []types.WriteRequest
  if end > len(movies) {
   end = len(movies)
  }
```

```
  for _, movie := range movies[start:end] {
   item, err = attributevalue.MarshalMap(movie)
   if err != nil {
    log.Printf("Couldn't marshal movie %v for batch writing. Here's why: %v\n",
 movie.Title, err)
   } else {
    writeReqs = append(
     writeReqs,
     types.WriteRequest{PutRequest: &types.PutRequest{Item: item}},
    )
   }
  }
  _, err = basics.DynamoDbClient.BatchWriteItem(ctx,
 &dynamodb.BatchWriteItemInput{
   RequestItems: map[string][]types.WriteRequest{basics.TableName: writeReqs}})
  if err != nil {
   log.Printf("Couldn't add a batch of movies to %v. Here's why: %v\n",
 basics.TableName, err)
  } else {
   written += len(writeReqs)
  }
  start = end
  end += batchSize
 }

 return written, err
}



// GetMovie gets movie data from the DynamoDB table by using the primary
 composite key
// made of title and year.
func (basics TableBasics) GetMovie(ctx context.Context, title string, year int)
 (Movie, error) {
 movie := Movie{Title: title, Year: year}
 response, err := basics.DynamoDbClient.GetItem(ctx, &dynamodb.GetItemInput{
  Key: movie.GetKey(), TableName: aws.String(basics.TableName),
 })
 if err != nil {
  log.Printf("Couldn't get info about %v. Here's why: %v\n", title, err)
 } else {
  err = attributevalue.UnmarshalMap(response.Item, &movie)
  if err != nil {
```

```
   log.Printf("Couldn't unmarshal response. Here's why: %v\n", err)
  }
 }
 return movie, err
}



// Query gets all movies in the DynamoDB table that were released in the
 specified year.
// The function uses the `expression` package to build the key condition
 expression
// that is used in the query.
func (basics TableBasics) Query(ctx context.Context, releaseYear int) ([]Movie,
 error) {
 var err error
 var response *dynamodb.QueryOutput
 var movies []Movie
 keyEx := expression.Key("year").Equal(expression.Value(releaseYear))
 expr, err := expression.NewBuilder().WithKeyCondition(keyEx).Build()
 if err != nil {
  log.Printf("Couldn't build expression for query. Here's why: %v\n", err)
 } else {
  queryPaginator := dynamodb.NewQueryPaginator(basics.DynamoDbClient,
 &dynamodb.QueryInput{
   TableName:                 aws.String(basics.TableName),
   ExpressionAttributeNames:  expr.Names(),
   ExpressionAttributeValues: expr.Values(),
   KeyConditionExpression:    expr.KeyCondition(),
  })
  for queryPaginator.HasMorePages() {
   response, err = queryPaginator.NextPage(ctx)
   if err != nil {
    log.Printf("Couldn't query for movies released in %v. Here's why: %v\n",
 releaseYear, err)
    break
   } else {
    var moviePage []Movie
    err = attributevalue.UnmarshalListOfMaps(response.Items, &moviePage)
    if err != nil {
     log.Printf("Couldn't unmarshal query response. Here's why: %v\n", err)
     break
    } else {
     movies = append(movies, moviePage...)
```

```
    }
   }
  }
 }
 return movies, err
}



// Scan gets all movies in the DynamoDB table that were released in a range of
 years
// and projects them to return a reduced set of fields.
// The function uses the `expression` package to build the filter and projection
// expressions.
func (basics TableBasics) Scan(ctx context.Context, startYear int, endYear int)
 ([]Movie, error) {
 var movies []Movie
 var err error
 var response *dynamodb.ScanOutput
 filtEx := expression.Name("year").Between(expression.Value(startYear),
 expression.Value(endYear))
 projEx := expression.NamesList(
  expression.Name("year"), expression.Name("title"),
 expression.Name("info.rating"))
 expr, err :=
 expression.NewBuilder().WithFilter(filtEx).WithProjection(projEx).Build()
 if err != nil {
  log.Printf("Couldn't build expressions for scan. Here's why: %v\n", err)
 } else {
  scanPaginator := dynamodb.NewScanPaginator(basics.DynamoDbClient,
 &dynamodb.ScanInput{
   TableName:                 aws.String(basics.TableName),
   ExpressionAttributeNames:  expr.Names(),
   ExpressionAttributeValues: expr.Values(),
   FilterExpression:          expr.Filter(),
   ProjectionExpression:      expr.Projection(),
  })
  for scanPaginator.HasMorePages() {
   response, err = scanPaginator.NextPage(ctx)
   if err != nil {
    log.Printf("Couldn't scan for movies released between %v and %v. Here's why:
 %v\n",
     startYear, endYear, err)
    break
```

```
   } else {
    var moviePage []Movie
    err = attributevalue.UnmarshalListOfMaps(response.Items, &moviePage)
    if err != nil {
     log.Printf("Couldn't unmarshal query response. Here's why: %v\n", err)
     break
    } else {
     movies = append(movies, moviePage...)
    }
   }
  }
 }
 return movies, err
}



// DeleteMovie removes a movie from the DynamoDB table.
func (basics TableBasics) DeleteMovie(ctx context.Context, movie Movie) error {
 _, err := basics.DynamoDbClient.DeleteItem(ctx, &dynamodb.DeleteItemInput{
  TableName: aws.String(basics.TableName), Key: movie.GetKey(),
 })
 if err != nil {
  log.Printf("Couldn't delete %v from the table. Here's why: %v\n", movie.Title,
 err)
 }
 return err
}



// DeleteTable deletes the DynamoDB table and all of its data.
func (basics TableBasics) DeleteTable(ctx context.Context) error {
 _, err := basics.DynamoDbClient.DeleteTable(ctx, &dynamodb.DeleteTableInput{
  TableName: aws.String(basics.TableName)})
 if err != nil {
  log.Printf("Couldn't delete table %v. Here's why: %v\n", basics.TableName, err)
 }
 return err
}
```

- Weitere API-Informationen finden Sie in den folgenden Themen der AWS SDK für Go -API-Referenz.
  - BatchWriteItem
  - CreateTable
  - DeleteItem
  - DeleteTable
  - DescribeTable
  - GetItem
  - PutItem
  - Query
  - Scan
  - UpdateItem

Java

SDK für Java 2.x

> **Note**
>
> Es gibt noch mehr GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

Erstellen Sie eine DynamoDB-Tabelle.

```
// Create a table with a Sort key.
public static void createTable(DynamoDbClient ddb, String tableName) {
    DynamoDbWaiter dbWaiter = ddb.waiter();
    ArrayList<AttributeDefinition> attributeDefinitions = new ArrayList<>();

    // Define attributes.
    attributeDefinitions.add(AttributeDefinition.builder()
        .attributeName("year")
        .attributeType("N")
        .build());

    attributeDefinitions.add(AttributeDefinition.builder()
```

```
            .attributeName("title")
            .attributeType("S")
            .build());

        ArrayList<KeySchemaElement> tableKey = new ArrayList<>();
        KeySchemaElement key = KeySchemaElement.builder()
            .attributeName("year")
            .keyType(KeyType.HASH)
            .build();

        KeySchemaElement key2 = KeySchemaElement.builder()
            .attributeName("title")
            .keyType(KeyType.RANGE)
            .build();

        // Add KeySchemaElement objects to the list.
        tableKey.add(key);
        tableKey.add(key2);

        CreateTableRequest request = CreateTableRequest.builder()
            .keySchema(tableKey)
            .billingMode(BillingMode.PAY_PER_REQUEST) //  DynamoDB automatically
scales based on traffic.
            .attributeDefinitions(attributeDefinitions)
            .tableName(tableName)
            .build();

        try {
            CreateTableResponse response = ddb.createTable(request);
            DescribeTableRequest tableRequest = DescribeTableRequest.builder()
                .tableName(tableName)
                .build();

            // Wait until the Amazon DynamoDB table is created.
            WaiterResponse<DescribeTableResponse> waiterResponse =
dbWaiter.waitUntilTableExists(tableRequest);
            waiterResponse.matched().response().ifPresent(System.out::println);
            String newTable = response.tableDescription().tableName();
            System.out.println("The " + newTable + " was successfully created.");

        } catch (DynamoDbException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
```

```
    }
```

Erstellen Sie eine Helper-Funktion zum Herunterladen und Extrahieren der JSON-Beispieldatei.

```
   // Load data into the table.
   public static void loadData(DynamoDbClient ddb, String tableName, String
fileName) throws IOException {
       DynamoDbEnhancedClient enhancedClient = DynamoDbEnhancedClient.builder()
           .dynamoDbClient(ddb)
           .build();

       DynamoDbTable<Movies> mappedTable = enhancedClient.table("Movies",
TableSchema.fromBean(Movies.class));
       JsonParser parser = new JsonFactory().createParser(new File(fileName));
       com.fasterxml.jackson.databind.JsonNode rootNode = new
ObjectMapper().readTree(parser);
       Iterator<JsonNode> iter = rootNode.iterator();
       ObjectNode currentNode;
       int t = 0;
       while (iter.hasNext()) {
           // Only add 200 Movies to the table.
           if (t == 200)
               break;
           currentNode = (ObjectNode) iter.next();

           int year = currentNode.path("year").asInt();
           String title = currentNode.path("title").asText();
           String info = currentNode.path("info").toString();

           Movies movies = new Movies();
           movies.setYear(year);
           movies.setTitle(title);
           movies.setInfo(info);

           // Put the data into the Amazon DynamoDB Movie table.
           mappedTable.putItem(movies);
           t++;
       }
   }
```

Rufen Sie ein Element aus einer Tabelle ab.

```
    public static void getItem(DynamoDbClient ddb) {

        HashMap<String, AttributeValue> keyToGet = new HashMap<>();
        keyToGet.put("year", AttributeValue.builder()
            .n("1933")
            .build());

        keyToGet.put("title", AttributeValue.builder()
            .s("King Kong")
            .build());

        GetItemRequest request = GetItemRequest.builder()
            .key(keyToGet)
            .tableName("Movies")
            .build();

        try {
            Map<String, AttributeValue> returnedItem =
ddb.getItem(request).item();

            if (returnedItem != null) {
                Set<String> keys = returnedItem.keySet();
                System.out.println("Amazon DynamoDB table attributes: \n");

                for (String key1 : keys) {
                    System.out.format("%s: %s\n", key1,
returnedItem.get(key1).toString());
                }
            } else {
                System.out.format("No item found with the key %s!\n", "year");
            }

        } catch (DynamoDbException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
    }
```

Vollständiges Beispiel.

```
/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 * <p>
 * For more information, see the following documentation topic:
 * <p>
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 * <p>
 * This Java example performs these tasks:
 * <p>
 * 1. Creates the Amazon DynamoDB Movie table with partition and sort key.
 * 2. Puts data into the Amazon DynamoDB table from a JSON document using the
 * Enhanced client.
 * 3. Gets data from the Movie table.
 * 4. Adds a new item.
 * 5. Updates an item.
 * 6. Uses a Scan to query items using the Enhanced client.
 * 7. Queries all items where the year is 2013 using the Enhanced Client.
 * 8. Deletes the table.
 */

public class Scenario {
    public static final String DASHES = new String(new char[80]).replace("\0",
 "-");

    public static void main(String[] args) throws IOException {
        String tableName = "Movies";
        String fileName = "../../../resources/sample_files/movies.json";
        Region region = Region.US_EAST_1;
        DynamoDbClient ddb = DynamoDbClient.builder()
            .region(region)
            .build();

        System.out.println(DASHES);
        System.out.println("Welcome to the Amazon DynamoDB example scenario.");
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println(
            "1. Creating an Amazon DynamoDB table named Movies with a key named
 year and a sort key named title.");
```

```
        createTable(ddb, tableName);
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("2. Loading data into the Amazon DynamoDB table.");
        loadData(ddb, tableName, fileName);
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("3. Getting data from the Movie table.");
        getItem(ddb);
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("4. Putting a record into the Amazon DynamoDB
table.");
        putRecord(ddb);
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("5. Updating a record.");
        updateTableItem(ddb, tableName);
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("6. Scanning the Amazon DynamoDB table.");
        scanMovies(ddb, tableName);
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("7. Querying the Movies released in 2013.");
        queryTable(ddb);
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("8. Deleting the Amazon DynamoDB table.");
        deleteDynamoDBTable(ddb, tableName);
        System.out.println(DASHES);

        ddb.close();
    }

    // Create a table with a Sort key.
    public static void createTable(DynamoDbClient ddb, String tableName) {
```

```
        DynamoDbWaiter dbWaiter = ddb.waiter();
        ArrayList<AttributeDefinition> attributeDefinitions = new ArrayList<>();

        // Define attributes.
        attributeDefinitions.add(AttributeDefinition.builder()
            .attributeName("year")
            .attributeType("N")
            .build());

        attributeDefinitions.add(AttributeDefinition.builder()
            .attributeName("title")
            .attributeType("S")
            .build());

        ArrayList<KeySchemaElement> tableKey = new ArrayList<>();
        KeySchemaElement key = KeySchemaElement.builder()
            .attributeName("year")
            .keyType(KeyType.HASH)
            .build();

        KeySchemaElement key2 = KeySchemaElement.builder()
            .attributeName("title")
            .keyType(KeyType.RANGE)
            .build();

        // Add KeySchemaElement objects to the list.
        tableKey.add(key);
        tableKey.add(key2);

        CreateTableRequest request = CreateTableRequest.builder()
            .keySchema(tableKey)
            .billingMode(BillingMode.PAY_PER_REQUEST) //  DynamoDB automatically
scales based on traffic.
            .attributeDefinitions(attributeDefinitions)
            .tableName(tableName)
            .build();

        try {
            CreateTableResponse response = ddb.createTable(request);
            DescribeTableRequest tableRequest = DescribeTableRequest.builder()
                .tableName(tableName)
                .build();

            // Wait until the Amazon DynamoDB table is created.
```

```
            WaiterResponse<DescribeTableResponse> waiterResponse =
dbWaiter.waitUntilTableExists(tableRequest);
            waiterResponse.matched().response().ifPresent(System.out::println);
            String newTable = response.tableDescription().tableName();
            System.out.println("The " + newTable + " was successfully created.");

        } catch (DynamoDbException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
    }

    // Query the table.
    public static void queryTable(DynamoDbClient ddb) {
        try {
            DynamoDbEnhancedClient enhancedClient =
DynamoDbEnhancedClient.builder()
                .dynamoDbClient(ddb)
                .build();

            DynamoDbTable<Movies> custTable = enhancedClient.table("Movies",
TableSchema.fromBean(Movies.class));
            QueryConditional queryConditional = QueryConditional
                .keyEqualTo(Key.builder()
                    .partitionValue(2013)
                    .build());

            // Get items in the table and write out the ID value.
            Iterator<Movies> results =
custTable.query(queryConditional).items().iterator();
            String result = "";

            while (results.hasNext()) {
                Movies rec = results.next();
                System.out.println("The title of the movie is " +
rec.getTitle());
                System.out.println("The movie information  is " + rec.getInfo());
            }

        } catch (DynamoDbException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
    }
```

```
   // Scan the table.
   public static void scanMovies(DynamoDbClient ddb, String tableName) {
       System.out.println("******* Scanning all movies.\n");
       try {
           DynamoDbEnhancedClient enhancedClient =
DynamoDbEnhancedClient.builder()
               .dynamoDbClient(ddb)
               .build();

           DynamoDbTable<Movies> custTable = enhancedClient.table("Movies",
TableSchema.fromBean(Movies.class));
           Iterator<Movies> results = custTable.scan().items().iterator();
           while (results.hasNext()) {
               Movies rec = results.next();
               System.out.println("The movie title is " + rec.getTitle());
               System.out.println("The movie year is " + rec.getYear());
           }

       } catch (DynamoDbException e) {
           System.err.println(e.getMessage());
           System.exit(1);
       }
   }

   // Load data into the table.
   public static void loadData(DynamoDbClient ddb, String tableName, String
fileName) throws IOException {
       DynamoDbEnhancedClient enhancedClient = DynamoDbEnhancedClient.builder()
           .dynamoDbClient(ddb)
           .build();

       DynamoDbTable<Movies> mappedTable = enhancedClient.table("Movies",
TableSchema.fromBean(Movies.class));
       JsonParser parser = new JsonFactory().createParser(new File(fileName));
       com.fasterxml.jackson.databind.JsonNode rootNode = new
ObjectMapper().readTree(parser);
       Iterator<JsonNode> iter = rootNode.iterator();
       ObjectNode currentNode;
       int t = 0;
       while (iter.hasNext()) {
           // Only add 200 Movies to the table.
           if (t == 200)
               break;
```

```
            currentNode = (ObjectNode) iter.next();

            int year = currentNode.path("year").asInt();
            String title = currentNode.path("title").asText();
            String info = currentNode.path("info").toString();

            Movies movies = new Movies();
            movies.setYear(year);
            movies.setTitle(title);
            movies.setInfo(info);

            // Put the data into the Amazon DynamoDB Movie table.
            mappedTable.putItem(movies);
            t++;
        }
    }

    // Update the record to include show only directors.
    public static void updateTableItem(DynamoDbClient ddb, String tableName) {
        HashMap<String, AttributeValue> itemKey = new HashMap<>();
        itemKey.put("year", AttributeValue.builder().n("1933").build());
        itemKey.put("title", AttributeValue.builder().s("King Kong").build());

        HashMap<String, AttributeValueUpdate> updatedValues = new HashMap<>();
        updatedValues.put("info", AttributeValueUpdate.builder()
            .value(AttributeValue.builder().s("{\"directors\":[\"Merian C. Cooper
\",\"Ernest B. Schoedsack\"]")
                .build())
            .action(AttributeAction.PUT)
            .build());

        UpdateItemRequest request = UpdateItemRequest.builder()
            .tableName(tableName)
            .key(itemKey)
            .attributeUpdates(updatedValues)
            .build();

        try {
            ddb.updateItem(request);
        } catch (ResourceNotFoundException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        } catch (DynamoDbException e) {
            System.err.println(e.getMessage());
```

```
            System.exit(1);
        }

        System.out.println("Item was updated!");
    }

    public static void deleteDynamoDBTable(DynamoDbClient ddb, String tableName)
{
        DeleteTableRequest request = DeleteTableRequest.builder()
            .tableName(tableName)
            .build();

        try {
            ddb.deleteTable(request);

        } catch (DynamoDbException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
        System.out.println(tableName + " was successfully deleted!");
    }

    public static void putRecord(DynamoDbClient ddb) {
        try {
            DynamoDbEnhancedClient enhancedClient =
DynamoDbEnhancedClient.builder()
                .dynamoDbClient(ddb)
                .build();

            DynamoDbTable<Movies> table = enhancedClient.table("Movies",
TableSchema.fromBean(Movies.class));

            // Populate the Table.
            Movies record = new Movies();
            record.setYear(2020);
            record.setTitle("My Movie2");
            record.setInfo("no info");
            table.putItem(record);

        } catch (DynamoDbException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
        System.out.println("Added a new movie to the table.");
```

```
    }

    public static void getItem(DynamoDbClient ddb) {

        HashMap<String, AttributeValue> keyToGet = new HashMap<>();
        keyToGet.put("year", AttributeValue.builder()
            .n("1933")
            .build());

        keyToGet.put("title", AttributeValue.builder()
            .s("King Kong")
            .build());

        GetItemRequest request = GetItemRequest.builder()
            .key(keyToGet)
            .tableName("Movies")
            .build();

        try {
            Map<String, AttributeValue> returnedItem =
 ddb.getItem(request).item();

            if (returnedItem != null) {
                Set<String> keys = returnedItem.keySet();
                System.out.println("Amazon DynamoDB table attributes: \n");

                for (String key1 : keys) {
                    System.out.format("%s: %s\n", key1,
 returnedItem.get(key1).toString());
                }
            } else {
                System.out.format("No item found with the key %s!\n", "year");
            }

        } catch (DynamoDbException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
    }
}
```

- Weitere API-Informationen finden Sie in den folgenden Themen der AWS SDK for Java 2.x - API-Referenz.
  - BatchWriteItem
  - CreateTable
  - DeleteItem
  - DeleteTable
  - DescribeTable
  - GetItem
  - PutItem
  - Query
  - Scan
  - UpdateItem

JavaScript

SDK für JavaScript (v3)

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
import { readFileSync } from "node:fs";
import {
  BillingMode,
  CreateTableCommand,
  DeleteTableCommand,
  DynamoDBClient,
  waitUntilTableExists,
} from "@aws-sdk/client-dynamodb";

/**
 * This module is a convenience library. It abstracts Amazon DynamoDB's data type
 * descriptors (such as S, N, B, and BOOL) by marshalling JavaScript objects into
 * AttributeValue shapes.
 */
```

```
import {
  BatchWriteCommand,
  DeleteCommand,
  DynamoDBDocumentClient,
  GetCommand,
  PutCommand,
  UpdateCommand,
  paginateQuery,
  paginateScan,
} from "@aws-sdk/lib-dynamodb";

// These modules are local to our GitHub repository. We recommend cloning
// the project from GitHub if you want to run this example.
// For more information, see https://github.com/awsdocs/aws-doc-sdk-examples.
import { getUniqueName } from "@aws-doc-sdk-examples/lib/utils/util-string.js";
import { dirnameFromMetaUrl } from "@aws-doc-sdk-examples/lib/utils/util-fs.js";
import { chunkArray } from "@aws-doc-sdk-examples/lib/utils/util-array.js";

const dirname = dirnameFromMetaUrl(import.meta.url);
const tableName = getUniqueName("Movies");
const client = new DynamoDBClient({});
const docClient = DynamoDBDocumentClient.from(client);

const log = (msg) => console.log(`[SCENARIO] ${msg}`);

export const main = async () => {
  /**
   * Create a table.
   */

  const createTableCommand = new CreateTableCommand({
    TableName: tableName,
    // This example performs a large write to the database.
    // Set the billing mode to PAY_PER_REQUEST to
    // avoid throttling the large write.
    BillingMode: BillingMode.PAY_PER_REQUEST,
    // Define the attributes that are necessary for the key schema.
    AttributeDefinitions: [
      {
        AttributeName: "year",
        // 'N' is a data type descriptor that represents a number type.
        // For a list of all data type descriptors, see the following link.
        // https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/
Programming.LowLevelAPI.html#Programming.LowLevelAPI.DataTypeDescriptors
```

```
        AttributeType: "N",
      },
      { AttributeName: "title", AttributeType: "S" },
    ],
    // The KeySchema defines the primary key. The primary key can be
    // a partition key, or a combination of a partition key and a sort key.
    // Key schema design is important. For more info, see
    // https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/best-
practices.html
    KeySchema: [
      // The way your data is accessed determines how you structure your keys.
      // The movies table will be queried for movies by year. It makes sense
      // to make year our partition (HASH) key.
      { AttributeName: "year", KeyType: "HASH" },
      { AttributeName: "title", KeyType: "RANGE" },
    ],
  });

  log("Creating a table.");
  const createTableResponse = await client.send(createTableCommand);
  log(`Table created: ${JSON.stringify(createTableResponse.TableDescription)}`);

  // This polls with DescribeTableCommand until the requested table is 'ACTIVE'.
  // You can't write to a table before it's active.
  log("Waiting for the table to be active.");
  await waitUntilTableExists({ client }, { TableName: tableName });
  log("Table active.");

  /**
   * Add a movie to the table.
   */

  log("Adding a single movie to the table.");
  // PutCommand is the first example usage of 'lib-dynamodb'.
  const putCommand = new PutCommand({
    TableName: tableName,
    Item: {
      // In 'client-dynamodb', the AttributeValue would be required (`year: { N:
 1981 }`)
      // 'lib-dynamodb' simplifies the usage ( `year: 1981` )
      year: 1981,
      // The preceding KeySchema defines 'title' as our sort (RANGE) key, so
 'title'
      // is required.
```

```
      title: "The Evil Dead",
      // Every other attribute is optional.
      info: {
        genres: ["Horror"],
      },
    },
  });
  await docClient.send(putCommand);
  log("The movie was added.");

  /**
   * Get a movie from the table.
   */

  log("Getting a single movie from the table.");
  const getCommand = new GetCommand({
    TableName: tableName,
    // Requires the complete primary key. For the movies table, the primary key
    // is only the id (partition key).
    Key: {
      year: 1981,
      title: "The Evil Dead",
    },
    // Set this to make sure that recent writes are reflected.
    // For more information, see https://docs.aws.amazon.com/amazondynamodb/
latest/developerguide/HowItWorks.ReadConsistency.html.
    ConsistentRead: true,
  });
  const getResponse = await docClient.send(getCommand);
  log(`Got the movie: ${JSON.stringify(getResponse.Item)}`);

  /**
   * Update a movie in the table.
   */

  log("Updating a single movie in the table.");
  const updateCommand = new UpdateCommand({
    TableName: tableName,
    Key: { year: 1981, title: "The Evil Dead" },
    // This update expression appends "Comedy" to the list of genres.
    // For more information on update expressions, see
    // https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/
Expressions.UpdateExpressions.html
    UpdateExpression: "set #i.#g = list_append(#i.#g, :vals)",
```

```
  ExpressionAttributeNames: { "#i": "info", "#g": "genres" },
  ExpressionAttributeValues: {
    ":vals": ["Comedy"],
  },
  ReturnValues: "ALL_NEW",
});
const updateResponse = await docClient.send(updateCommand);
log(`Movie updated: ${JSON.stringify(updateResponse.Attributes)}`);

/**
 * Delete a movie from the table.
 */

log("Deleting a single movie from the table.");
const deleteCommand = new DeleteCommand({
  TableName: tableName,
  Key: { year: 1981, title: "The Evil Dead" },
});
await docClient.send(deleteCommand);
log("Movie deleted.");

/**
 * Upload a batch of movies.
 */

log("Adding movies from local JSON file.");
const file = readFileSync(
  `${dirname}../../../../resources/sample_files/movies.json`,
);
const movies = JSON.parse(file.toString());
// chunkArray is a local convenience function. It takes an array and returns
// a generator function. The generator function yields every N items.
const movieChunks = chunkArray(movies, 25);
// For every chunk of 25 movies, make one BatchWrite request.
for (const chunk of movieChunks) {
  const putRequests = chunk.map((movie) => ({
    PutRequest: {
      Item: movie,
    },
  }));

  const command = new BatchWriteCommand({
    RequestItems: {
      [tableName]: putRequests,
```

```
      },
    });

    await docClient.send(command);
  }
  log("Movies added.");

  /**
   * Query for movies by year.
   */

  log("Querying for all movies from 1981.");
  const paginatedQuery = paginateQuery(
    { client: docClient },
    {
      TableName: tableName,
      //For more information about query expressions, see
      // https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/
Query.html#Query.KeyConditionExpressions
      KeyConditionExpression: "#y = :y",
      // 'year' is a reserved word in DynamoDB. Indicate that it's an attribute
      // name by using an expression attribute name.
      ExpressionAttributeNames: { "#y": "year" },
      ExpressionAttributeValues: { ":y": 1981 },
      ConsistentRead: true,
    },
  );
  /**
   * @type { Record<string, any>[] };
   */
  const movies1981 = [];
  for await (const page of paginatedQuery) {
    movies1981.push(...page.Items);
  }
  log(`Movies: ${movies1981.map((m) => m.title).join(", ")}`);

  /**
   * Scan the table for movies between 1980 and 1990.
   */

  log("Scan for movies released between 1980 and 1990");
  // A 'Scan' operation always reads every item in the table. If your design
 requires
  // the use of 'Scan', consider indexing your table or changing your design.
```

```
  // https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/bp-query-
scan.html
  const paginatedScan = paginateScan(
    { client: docClient },
    {
      TableName: tableName,
      // Scan uses a filter expression instead of a key condition expression.
 Scan will
      // read the entire table and then apply the filter.
      FilterExpression: "#y between :y1 and :y2",
      ExpressionAttributeNames: { "#y": "year" },
      ExpressionAttributeValues: { ":y1": 1980, ":y2": 1990 },
      ConsistentRead: true,
    },
  );
  /**
   * @type { Record<string, any>[] };
   */
  const movies1980to1990 = [];
  for await (const page of paginatedScan) {
    movies1980to1990.push(...page.Items);
  }
  log(
    `Movies: ${movies1980to1990
      .map((m) => `${m.title} (${m.year})`)
      .join(", ")}`,
  );

  /**
   * Delete the table.
   */

  const deleteTableCommand = new DeleteTableCommand({ TableName: tableName });
  log(`Deleting table ${tableName}.`);
  await client.send(deleteTableCommand);
  log("Table deleted.");
};
```

- Weitere API-Informationen finden Sie in den folgenden Themen der AWS SDK für JavaScript -API-Referenz.
  - BatchWriteItem

- CreateTable
- DeleteItem
- DeleteTable
- DescribeTable
- GetItem
- PutItem
- Query
- Scan
- UpdateItem

Kotlin

SDK für Kotlin

> **Note**
>
> Es gibt noch mehr GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

Erstellen Sie eine DynamoDB-Tabelle.

```
suspend fun createScenarioTable(
    tableNameVal: String,
    key: String,
) {
    val attDef =
        AttributeDefinition {
            attributeName = key
            attributeType = ScalarAttributeType.N
        }

    val attDef1 =
        AttributeDefinition {
            attributeName = "title"
            attributeType = ScalarAttributeType.S
        }
```

```
    val keySchemaVal =
        KeySchemaElement {
            attributeName = key
            keyType = KeyType.Hash
        }

    val keySchemaVal1 =
        KeySchemaElement {
            attributeName = "title"
            keyType = KeyType.Range
        }

    val request =
        CreateTableRequest {
            attributeDefinitions = listOf(attDef, attDef1)
            keySchema = listOf(keySchemaVal, keySchemaVal1)
            billingMode = BillingMode.PayPerRequest
            tableName = tableNameVal
        }

    DynamoDbClient.fromEnvironment { region = "us-east-1" }.use { ddb ->
        val response = ddb.createTable(request)
        ddb.waitUntilTableExists {
            // suspend call
            tableName = tableNameVal
        }
        println("The table was successfully created
 ${response.tableDescription?.tableArn}")
    }
}
```

Erstellen Sie eine Helper-Funktion zum Herunterladen und Extrahieren der JSON-Beispieldatei.

```
// Load data into the table.
suspend fun loadData(
    tableName: String,
    fileName: String,
) {
    val parser = JsonFactory().createParser(File(fileName))
    val rootNode = ObjectMapper().readTree<JsonNode>(parser)
    val iter: Iterator<JsonNode> = rootNode.iterator()
```

```
    var currentNode: ObjectNode

    var t = 0
    while (iter.hasNext()) {
        if (t == 50) {
            break
        }

        currentNode = iter.next() as ObjectNode
        val year = currentNode.path("year").asInt()
        val title = currentNode.path("title").asText()
        val info = currentNode.path("info").toString()
        putMovie(tableName, year, title, info)
        t++
    }
}

suspend fun putMovie(
    tableNameVal: String,
    year: Int,
    title: String,
    info: String,
) {
    val itemValues = mutableMapOf<String, AttributeValue>()
    val strVal = year.toString()
    // Add all content to the table.
    itemValues["year"] = AttributeValue.N(strVal)
    itemValues["title"] = AttributeValue.S(title)
    itemValues["info"] = AttributeValue.S(info)

    val request =
        PutItemRequest {
            tableName = tableNameVal
            item = itemValues
        }

    DynamoDbClient.fromEnvironment { region = "us-east-1" }.use { ddb ->
        ddb.putItem(request)
        println("Added $title to the Movie table.")
    }
}
```

Rufen Sie ein Element aus einer Tabelle ab.

```
suspend fun getMovie(
    tableNameVal: String,
    keyName: String,
    keyVal: String,
) {
    val keyToGet = mutableMapOf<String, AttributeValue>()
    keyToGet[keyName] = AttributeValue.N(keyVal)
    keyToGet["title"] = AttributeValue.S("King Kong")

    val request =
        GetItemRequest {
            key = keyToGet
            tableName = tableNameVal
        }

    DynamoDbClient.fromEnvironment { region = "us-east-1" }.use { ddb ->
        val returnedItem = ddb.getItem(request)
        val numbersMap = returnedItem.item
        numbersMap?.forEach { key1 ->
            println(key1.key)
            println(key1.value)
        }
    }
}
```

Vollständiges Beispiel.

```
suspend fun main() {
    val tableName = "Movies"
    val fileName = "../../../resources/sample_files/movies.json"
    val partitionAlias = "#a"

    println("Creating an Amazon DynamoDB table named Movies with a key named id
 and a sort key named title.")
    createScenarioTable(tableName, "year")
    loadData(tableName, fileName)
    getMovie(tableName, "year", "1933")
    scanMovies(tableName)
    val count = queryMovieTable(tableName, "year", partitionAlias)
    println("There are $count Movies released in 2013.")
```

```
    deletIssuesTable(tableName)
}

suspend fun createScenarioTable(
    tableNameVal: String,
    key: String,
) {
    val attDef =
        AttributeDefinition {
            attributeName = key
            attributeType = ScalarAttributeType.N
        }

    val attDef1 =
        AttributeDefinition {
            attributeName = "title"
            attributeType = ScalarAttributeType.S
        }

    val keySchemaVal =
        KeySchemaElement {
            attributeName = key
            keyType = KeyType.Hash
        }

    val keySchemaVal1 =
        KeySchemaElement {
            attributeName = "title"
            keyType = KeyType.Range
        }

    val request =
        CreateTableRequest {
            attributeDefinitions = listOf(attDef, attDef1)
            keySchema = listOf(keySchemaVal, keySchemaVal1)
            billingMode = BillingMode.PayPerRequest
            tableName = tableNameVal
        }

    DynamoDbClient.fromEnvironment { region = "us-east-1" }.use { ddb ->
        val response = ddb.createTable(request)
        ddb.waitUntilTableExists {
            // suspend call
            tableName = tableNameVal
```

```
        }
        println("The table was successfully created
 ${response.tableDescription?.tableArn}")
    }
}

// Load data into the table.
suspend fun loadData(
    tableName: String,
    fileName: String,
) {
    val parser = JsonFactory().createParser(File(fileName))
    val rootNode = ObjectMapper().readTree<JsonNode>(parser)
    val iter: Iterator<JsonNode> = rootNode.iterator()
    var currentNode: ObjectNode

    var t = 0
    while (iter.hasNext()) {
        if (t == 50) {
            break
        }

        currentNode = iter.next() as ObjectNode
        val year = currentNode.path("year").asInt()
        val title = currentNode.path("title").asText()
        val info = currentNode.path("info").toString()
        putMovie(tableName, year, title, info)
        t++
    }
}

suspend fun putMovie(
    tableNameVal: String,
    year: Int,
    title: String,
    info: String,
) {
    val itemValues = mutableMapOf<String, AttributeValue>()
    val strVal = year.toString()
    // Add all content to the table.
    itemValues["year"] = AttributeValue.N(strVal)
    itemValues["title"] = AttributeValue.S(title)
    itemValues["info"] = AttributeValue.S(info)
```

```
    val request =
        PutItemRequest {
            tableName = tableNameVal
            item = itemValues
        }

    DynamoDbClient.fromEnvironment { region = "us-east-1" }.use { ddb ->
        ddb.putItem(request)
        println("Added $title to the Movie table.")
    }
}

suspend fun getMovie(
    tableNameVal: String,
    keyName: String,
    keyVal: String,
) {
    val keyToGet = mutableMapOf<String, AttributeValue>()
    keyToGet[keyName] = AttributeValue.N(keyVal)
    keyToGet["title"] = AttributeValue.S("King Kong")

    val request =
        GetItemRequest {
            key = keyToGet
            tableName = tableNameVal
        }

    DynamoDbClient.fromEnvironment { region = "us-east-1" }.use { ddb ->
        val returnedItem = ddb.getItem(request)
        val numbersMap = returnedItem.item
        numbersMap?.forEach { key1 ->
            println(key1.key)
            println(key1.value)
        }
    }
}

suspend fun deletIssuesTable(tableNameVal: String) {
    val request =
        DeleteTableRequest {
            tableName = tableNameVal
        }

    DynamoDbClient.fromEnvironment { region = "us-east-1" }.use { ddb ->
```

```
        ddb.deleteTable(request)
        println("$tableNameVal was deleted")
    }
}

suspend fun queryMovieTable(
    tableNameVal: String,
    partitionKeyName: String,
    partitionAlias: String,
): Int {
    val attrNameAlias = mutableMapOf<String, String>()
    attrNameAlias[partitionAlias] = "year"

    // Set up mapping of the partition name with the value.
    val attrValues = mutableMapOf<String, AttributeValue>()
    attrValues[":$partitionKeyName"] = AttributeValue.N("2013")

    val request =
        QueryRequest {
            tableName = tableNameVal
            keyConditionExpression = "$partitionAlias = :$partitionKeyName"
            expressionAttributeNames = attrNameAlias
            this.expressionAttributeValues = attrValues
        }

    DynamoDbClient.fromEnvironment { region = "us-east-1" }.use { ddb ->
        val response = ddb.query(request)
        return response.count
    }
}

suspend fun scanMovies(tableNameVal: String) {
    val request =
        ScanRequest {
            tableName = tableNameVal
        }

    DynamoDbClient.fromEnvironment { region = "us-east-1" }.use { ddb ->
        val response = ddb.scan(request)
        response.items?.forEach { item ->
            item.keys.forEach { key ->
                println("The key name is $key\n")
                println("The value is ${item[key]}")
            }
```

```
        }
    }
}
```

- Weitere API-Informationen finden Sie in den folgenden Themen der API-Referenz zum AWS -SDK für Kotlin.
  - BatchWriteItem
  - CreateTable
  - DeleteItem
  - DeleteTable
  - DescribeTable
  - GetItem
  - PutItem
  - Query
  - Scan
  - UpdateItem

PHP

SDK für PHP

> **Note**
>
> Es gibt noch mehr GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

Da in diesem Beispiel Hilfsdateien verwendet werden, lesen Sie die Anleitungen in der README.md-Datei für die PHP-Beispiele.

```
namespace DynamoDb\Basics;

use Aws\DynamoDb\Marshaler;
use DynamoDb;
use DynamoDb\DynamoDBAttribute;
use DynamoDb\DynamoDBService;
```

```
use function AwsUtilities\loadMovieData;
use function AwsUtilities\testable_readline;

class GettingStartedWithDynamoDB
{
    public function run()
    {
        echo("\n");
        echo("---------------------------------------\n");
        print("Welcome to the Amazon DynamoDB getting started demo using PHP!
\n");
        echo("---------------------------------------\n");

        $uuid = uniqid();
        $service = new DynamoDBService();

        $tableName = "ddb_demo_table_$uuid";
        $service->createTable(
            $tableName,
            [
                new DynamoDBAttribute('year', 'N', 'HASH'),
                new DynamoDBAttribute('title', 'S', 'RANGE')
            ]
        );

        echo "Waiting for table...";
        $service->dynamoDbClient->waitUntil("TableExists", ['TableName' =>
 $tableName]);
        echo "table $tableName found!\n";

        echo "What's the name of the last movie you watched?\n";
        while (empty($movieName)) {
            $movieName = testable_readline("Movie name: ");
        }
        echo "And what year was it released?\n";
        $movieYear = "year";
        while (!is_numeric($movieYear) || intval($movieYear) != $movieYear) {
            $movieYear = testable_readline("Year released: ");
        }

        $service->putItem([
            'Item' => [
                'year' => [
```

```
                    'N' => "$movieYear",
                ],
                'title' => [
                    'S' => $movieName,
                ],
            ],
            'TableName' => $tableName,
        ]);

        echo "How would you rate the movie from 1-10?\n";
        $rating = 0;
        while (!is_numeric($rating) || intval($rating) != $rating || $rating < 1
|| $rating > 10) {
            $rating = testable_readline("Rating (1-10): ");
        }
        echo "What was the movie about?\n";
        while (empty($plot)) {
            $plot = testable_readline("Plot summary: ");
        }
        $key = [
            'Item' => [
                'title' => [
                    'S' => $movieName,
                ],
                'year' => [
                    'N' => $movieYear,
                ],
            ]
        ];
        $attributes = ["rating" =>
            [
                'AttributeName' => 'rating',
                'AttributeType' => 'N',
                'Value' => $rating,
            ],
            'plot' => [
                'AttributeName' => 'plot',
                'AttributeType' => 'S',
                'Value' => $plot,
            ]
        ];
        $service->updateItemAttributesByKey($tableName, $key, $attributes);
        echo "Movie added and updated.";
```

```
        $batch = json_decode(loadMovieData());

        $service->writeBatch($tableName, $batch);


        $movie = $service->getItemByKey($tableName, $key);
        echo "\nThe movie {$movie['Item']['title']['S']} was released in
 {$movie['Item']['year']['N']}.\n";
        echo "What rating would you like to give {$movie['Item']['title']['S']}?
\n";
        $rating = 0;
        while (!is_numeric($rating) || intval($rating) != $rating || $rating < 1
 || $rating > 10) {
            $rating = testable_readline("Rating (1-10): ");
        }
        $service->updateItemAttributeByKey($tableName, $key, 'rating', 'N',
 $rating);

        $movie = $service->getItemByKey($tableName, $key);
        echo "Ok, you have rated {$movie['Item']['title']['S']} as a
 {$movie['Item']['rating']['N']}\n";

        $service->deleteItemByKey($tableName, $key);
        echo "But, bad news, this was a trap. That movie has now been deleted
 because of your rating...harsh.\n";

        echo "That's okay though. The book was better. Now, for something
 lighter, in what year were you born?\n";
        $birthYear = "not a number";
        while (!is_numeric($birthYear) || $birthYear >= date("Y")) {
            $birthYear = testable_readline("Birth year: ");
        }
        $birthKey = [
            'Key' => [
                'year' => [
                    'N' => "$birthYear",
                ],
            ],
        ];
        $result = $service->query($tableName, $birthKey);
        $marshal = new Marshaler();
        echo "Here are the movies in our collection released the year you were
 born:\n";
```

```
        $oops = "Oops! There were no movies released in that year (that we know
 of).\n";
        $display = "";
        foreach ($result['Items'] as $movie) {
            $movie = $marshal->unmarshalItem($movie);
            $display .= $movie['title'] . "\n";
        }
        echo ($display) ?: $oops;

        $yearsKey = [
            'Key' => [
                'year' => [
                    'N' => [
                        'minRange' => 1990,
                        'maxRange' => 1999,
                    ],
                ],
            ],
        ];
        $filter = "year between 1990 and 1999";
        echo "\nHere's a list of all the movies released in the 90s:\n";
        $result = $service->scan($tableName, $yearsKey, $filter);
        foreach ($result['Items'] as $movie) {
            $movie = $marshal->unmarshalItem($movie);
            echo $movie['title'] . "\n";
        }

        echo "\nCleaning up this demo by deleting table $tableName...\n";
        $service->deleteTable($tableName);
    }
}
```

- Weitere API-Informationen finden Sie in den folgenden Themen der AWS SDK für PHP - API-Referenz.
  - BatchWriteItem
  - CreateTable
  - DeleteItem
  - DeleteTable
  - DescribeTable
  - GetItem

- PutItem
- Query
- Scan
- UpdateItem

Python

SDK für Python (Boto3)

> **Note**
>
> Es gibt noch mehr GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

Erstellen Sie eine Klasse, die eine DynamoDB-Tabelle enthält.

```
from decimal import Decimal
from io import BytesIO
import json
import logging
import os
from pprint import pprint
import requests
from zipfile import ZipFile
import boto3
from boto3.dynamodb.conditions import Key
from botocore.exceptions import ClientError
from question import Question

logger = logging.getLogger(__name__)

class Movies:
    """Encapsulates an Amazon DynamoDB table of movie data.

    Example data structure for a movie record in this table:
        {
            "year": 1999,
            "title": "For Love of the Game",
            "info": {
```

```
                "directors": ["Sam Raimi"],
                "release_date": "1999-09-15T00:00:00Z",
                "rating": 6.3,
                "plot": "A washed up pitcher flashes through his career.",
                "rank": 4987,
                "running_time_secs": 8220,
                "actors": [
                    "Kevin Costner",
                    "Kelly Preston",
                    "John C. Reilly"
                ]
            }
        }
    """

    def __init__(self, dyn_resource):
        """
        :param dyn_resource: A Boto3 DynamoDB resource.
        """
        self.dyn_resource = dyn_resource
        # The table variable is set during the scenario in the call to
        # 'exists' if the table exists. Otherwise, it is set by 'create_table'.
        self.table = None


    def exists(self, table_name):
        """
        Determines whether a table exists. As a side effect, stores the table in
        a member variable.

        :param table_name: The name of the table to check.
        :return: True when the table exists; otherwise, False.
        """
        try:
            table = self.dyn_resource.Table(table_name)
            table.load()
            exists = True
        except ClientError as err:
            if err.response["Error"]["Code"] == "ResourceNotFoundException":
                exists = False
            else:
                logger.error(
                    "Couldn't check for existence of %s. Here's why: %s: %s",
                    table_name,
```

```
                    err.response["Error"]["Code"],
                    err.response["Error"]["Message"],
                )
                raise
        else:
            self.table = table
        return exists


    def create_table(self, table_name):
        """
        Creates an Amazon DynamoDB table that can be used to store movie data.
        The table uses the release year of the movie as the partition key and the
        title as the sort key.

        :param table_name: The name of the table to create.
        :return: The newly created table.
        """
        try:
            self.table = self.dyn_resource.create_table(
                TableName=table_name,
                KeySchema=[
                    {"AttributeName": "year", "KeyType": "HASH"},  # Partition
key
                    {"AttributeName": "title", "KeyType": "RANGE"},  # Sort key
                ],
                AttributeDefinitions=[
                    {"AttributeName": "year", "AttributeType": "N"},
                    {"AttributeName": "title", "AttributeType": "S"},
                ],
                BillingMode='PAY_PER_REQUEST',
            )
            self.table.wait_until_exists()
        except ClientError as err:
            logger.error(
                "Couldn't create table %s. Here's why: %s: %s",
                table_name,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return self.table
```

```
    def list_tables(self):
        """
        Lists the Amazon DynamoDB tables for the current account.

        :return: The list of tables.
        """
        try:
            tables = []
            for table in self.dyn_resource.tables.all():
                print(table.name)
                tables.append(table)
        except ClientError as err:
            logger.error(
                "Couldn't list tables. Here's why: %s: %s",
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return tables


    def write_batch(self, movies):
        """
        Fills an Amazon DynamoDB table with the specified data, using the Boto3
        Table.batch_writer() function to put the items in the table.
        Inside the context manager, Table.batch_writer builds a list of
        requests. On exiting the context manager, Table.batch_writer starts
sending
        batches of write requests to Amazon DynamoDB and automatically
        handles chunking, buffering, and retrying.

        :param movies: The data to put in the table. Each item must contain at
least
                       the keys required by the schema that was specified when
the
                       table was created.
        """
        try:
            with self.table.batch_writer() as writer:
                for movie in movies:
                    writer.put_item(Item=movie)
        except ClientError as err:
```

```
            logger.error(
                "Couldn't load data into table %s. Here's why: %s: %s",
                self.table.name,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise


    def add_movie(self, title, year, plot, rating):
        """
        Adds a movie to the table.

        :param title: The title of the movie.
        :param year: The release year of the movie.
        :param plot: The plot summary of the movie.
        :param rating: The quality rating of the movie.
        """
        try:
            self.table.put_item(
                Item={
                    "year": year,
                    "title": title,
                    "info": {"plot": plot, "rating": Decimal(str(rating))},
                }
            )
        except ClientError as err:
            logger.error(
                "Couldn't add movie %s to table %s. Here's why: %s: %s",
                title,
                self.table.name,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise


    def get_movie(self, title, year):
        """
        Gets movie data from the table for a specific movie.

        :param title: The title of the movie.
        :param year: The release year of the movie.
        :return: The data about the requested movie.
```

```
        """
        try:
            response = self.table.get_item(Key={"year": year, "title": title})
        except ClientError as err:
            logger.error(
                "Couldn't get movie %s from table %s. Here's why: %s: %s",
                title,
                self.table.name,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return response["Item"]


    def update_movie(self, title, year, rating, plot):
        """
        Updates rating and plot data for a movie in the table.

        :param title: The title of the movie to update.
        :param year: The release year of the movie to update.
        :param rating: The updated rating to the give the movie.
        :param plot: The updated plot summary to give the movie.
        :return: The fields that were updated, with their new values.
        """
        try:
            response = self.table.update_item(
                Key={"year": year, "title": title},
                UpdateExpression="set info.rating=:r, info.plot=:p",
                ExpressionAttributeValues={":r": Decimal(str(rating)), ":p":
plot},
                ReturnValues="UPDATED_NEW",
            )
        except ClientError as err:
            logger.error(
                "Couldn't update movie %s in table %s. Here's why: %s: %s",
                title,
                self.table.name,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
```

```
            return response["Attributes"]


    def query_movies(self, year):
        """
        Queries for movies that were released in the specified year.

        :param year: The year to query.
        :return: The list of movies that were released in the specified year.
        """
        try:
            response =
self.table.query(KeyConditionExpression=Key("year").eq(year))
        except ClientError as err:
            logger.error(
                "Couldn't query for movies released in %s. Here's why: %s: %s",
                year,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return response["Items"]


    def scan_movies(self, year_range):
        """
        Scans for movies that were released in a range of years.
        Uses a projection expression to return a subset of data for each movie.

        :param year_range: The range of years to retrieve.
        :return: The list of movies released in the specified years.
        """
        movies = []
        scan_kwargs = {
            "FilterExpression": Key("year").between(
                year_range["first"], year_range["second"]
            ),
            "ProjectionExpression": "#yr, title, info.rating",
            "ExpressionAttributeNames": {"#yr": "year"},
        }
        try:
            done = False
            start_key = None
```

```
            while not done:
                if start_key:
                    scan_kwargs["ExclusiveStartKey"] = start_key
                response = self.table.scan(**scan_kwargs)
                movies.extend(response.get("Items", []))
                start_key = response.get("LastEvaluatedKey", None)
                done = start_key is None
        except ClientError as err:
            logger.error(
                "Couldn't scan for movies. Here's why: %s: %s",
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise

        return movies


    def delete_movie(self, title, year):
        """
        Deletes a movie from the table.

        :param title: The title of the movie to delete.
        :param year: The release year of the movie to delete.
        """
        try:
            self.table.delete_item(Key={"year": year, "title": title})
        except ClientError as err:
            logger.error(
                "Couldn't delete movie %s. Here's why: %s: %s",
                title,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise


    def delete_table(self):
        """
        Deletes the table.
        """
        try:
            self.table.delete()
            self.table = None
```

```
        except ClientError as err:
            logger.error(
                "Couldn't delete table. Here's why: %s: %s",
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
```

Erstellen Sie eine Helper-Funktion zum Herunterladen und Extrahieren der JSON-Beispieldatei.

```
def get_sample_movie_data(movie_file_name):
    """
    Gets sample movie data, either from a local file or by first downloading it
 from
    the Amazon DynamoDB developer guide.

    :param movie_file_name: The local file name where the movie data is stored in
 JSON format.
    :return: The movie data as a dict.
    """
    if not os.path.isfile(movie_file_name):
        print(f"Downloading {movie_file_name}...")
        movie_content = requests.get(
            "https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/
samples/moviedata.zip"
        )
        movie_zip = ZipFile(BytesIO(movie_content.content))
        movie_zip.extractall()

    try:
        with open(movie_file_name) as movie_file:
            movie_data = json.load(movie_file, parse_float=Decimal)
    except FileNotFoundError:
        print(
            f"File {movie_file_name} not found. You must first download the file
 to "
            "run this demo. See the README for instructions."
        )
```

```
            raise
        else:
            # The sample file lists over 4000 movies, return only the first 250.
            return movie_data[:250]
```

Führen Sie ein interaktives Szenario aus, um die Tabelle zu erstellen und Aktionen darauf auszuführen.

```
def run_scenario(table_name, movie_file_name, dyn_resource):
    logging.basicConfig(level=logging.INFO, format="%(levelname)s: %(message)s")

    print("-" * 88)
    print("Welcome to the Amazon DynamoDB getting started demo.")
    print("-" * 88)

    movies = Movies(dyn_resource)
    movies_exists = movies.exists(table_name)
    if not movies_exists:
        print(f"\nCreating table {table_name}...")
        movies.create_table(table_name)
        print(f"\nCreated table {movies.table.name}.")

    my_movie = Question.ask_questions(
        [
            Question(
                "title", "Enter the title of a movie you want to add to the
 table: "
            ),
            Question("year", "What year was it released? ", Question.is_int),
            Question(
                "rating",
                "On a scale of 1 - 10, how do you rate it? ",
                Question.is_float,
                Question.in_range(1, 10),
            ),
            Question("plot", "Summarize the plot for me: "),
        ]
    )
    movies.add_movie(**my_movie)
    print(f"\nAdded '{my_movie['title']}' to '{movies.table.name}'.")
```

```
    print("-" * 88)

    movie_update = Question.ask_questions(
        [
            Question(
                "rating",
                f"\nLet's update your movie.\nYou rated it {my_movie['rating']},
what new "
                f"rating would you give it? ",
                Question.is_float,
                Question.in_range(1, 10),
            ),
            Question(
                "plot",
                f"You summarized the plot as '{my_movie['plot']}'.\nWhat would
you say now? ",
            ),
        ]
    )
    my_movie.update(movie_update)
    updated = movies.update_movie(**my_movie)
    print(f"\nUpdated '{my_movie['title']}' with new attributes:")
    pprint(updated)
    print("-" * 88)

    if not movies_exists:
        movie_data = get_sample_movie_data(movie_file_name)
        print(f"\nReading data from '{movie_file_name}' into your table.")
        movies.write_batch(movie_data)
        print(f"\nWrote {len(movie_data)} movies into {movies.table.name}.")
    print("-" * 88)

    title = "The Lord of the Rings: The Fellowship of the Ring"
    if Question.ask_question(
        f"Let's move on...do you want to get info about '{title}'? (y/n) ",
        Question.is_yesno,
    ):
        movie = movies.get_movie(title, 2001)
        print("\nHere's what I found:")
        pprint(movie)
    print("-" * 88)

    ask_for_year = True
    while ask_for_year:
```

```
        release_year = Question.ask_question(
            f"\nLet's get a list of movies released in a given year. Enter a year
between "
            f"1972 and 2018: ",
            Question.is_int,
            Question.in_range(1972, 2018),
        )
        releases = movies.query_movies(release_year)
        if releases:
            print(f"There were {len(releases)} movies released in
{release_year}:")
            for release in releases:
                print(f"\t{release['title']}")
            ask_for_year = False
        else:
            print(f"I don't know about any movies released in {release_year}!")
            ask_for_year = Question.ask_question(
                "Try another year? (y/n) ", Question.is_yesno
            )
    print("-" * 88)

    years = Question.ask_questions(
        [
            Question(
                "first",
                f"\nNow let's scan for movies released in a range of years. Enter
a year: ",
                Question.is_int,
                Question.in_range(1972, 2018),
            ),
            Question(
                "second",
                "Now enter another year: ",
                Question.is_int,
                Question.in_range(1972, 2018),
            ),
        ]
    )
    releases = movies.scan_movies(years)
    if releases:
        count = Question.ask_question(
            f"\nFound {len(releases)} movies. How many do you want to see? ",
            Question.is_int,
            Question.in_range(1, len(releases)),
```

```
        )
        print(f"\nHere are your {count} movies:\n")
        pprint(releases[:count])
    else:
        print(
            f"I don't know about any movies released between {years['first']} "
            f"and {years['second']}."
        )
    print("-" * 88)

    if Question.ask_question(
        f"\nLet's remove your movie from the table. Do you want to remove "
        f"'{my_movie['title']}'? (y/n)",
        Question.is_yesno,
    ):
        movies.delete_movie(my_movie["title"], my_movie["year"])
        print(f"\nRemoved '{my_movie['title']}' from the table.")
    print("-" * 88)

    if Question.ask_question(f"\nDelete the table? (y/n) ", Question.is_yesno):
        movies.delete_table()
        print(f"Deleted {table_name}.")
    else:
        print(
            "Don't forget to delete the table when you're done or you might incur
 "
            "charges on your account."
        )

    print("\nThanks for watching!")
    print("-" * 88)


if __name__ == "__main__":
    try:
        run_scenario(
            "doc-example-table-movies", "moviedata.json",
 boto3.resource("dynamodb")
        )
    except Exception as e:
        print(f"Something went wrong with the demo! Here's what: {e}")
```

In diesem Szenario wird die folgende Helper-Klasse verwendet, um Fragen an einer Eingabeaufforderung zu stellen.

```
class Question:
    """
    A helper class to ask questions at a command prompt and validate and convert
    the answers.
    """

    def __init__(self, key, question, *validators):
        """
        :param key: The key that is used for storing the answer in a dict, when
                    multiple questions are asked in a set.
        :param question: The question to ask.
        :param validators: The answer is passed through the list of validators
 until
                           one fails or they all pass. Validators may also
 convert the
                           answer to another form, such as from a str to an int.
        """
        self.key = key
        self.question = question
        self.validators = Question.non_empty, *validators

    @staticmethod
    def ask_questions(questions):
        """
        Asks a set of questions and stores the answers in a dict.

        :param questions: The list of questions to ask.
        :return: A dict of answers.
        """
        answers = {}
        for question in questions:
            answers[question.key] = Question.ask_question(
                question.question, *question.validators
            )
        return answers

    @staticmethod
    def ask_question(question, *validators):
        """
        Asks a single question and validates it against a list of validators.
```

```
        When an answer fails validation, the complaint is printed and the
question
        is asked again.

        :param question: The question to ask.
        :param validators: The list of validators that the answer must pass.
        :return: The answer, converted to its final form by the validators.
        """
        answer = None
        while answer is None:
            answer = input(question)
            for validator in validators:
                answer, complaint = validator(answer)
                if answer is None:
                    print(complaint)
                    break
        return answer

    @staticmethod
    def non_empty(answer):
        """
        Validates that the answer is not empty.
        :return: The non-empty answer, or None.
        """
        return answer if answer != "" else None, "I need an answer. Please?"

    @staticmethod
    def is_yesno(answer):
        """
        Validates a yes/no answer.
        :return: True when the answer is 'y'; otherwise, False.
        """
        return answer.lower() == "y", ""

    @staticmethod
    def is_int(answer):
        """
        Validates that the answer can be converted to an int.
        :return: The int answer; otherwise, None.
        """
        try:
            int_answer = int(answer)
        except ValueError:
            int_answer = None
```

```
        return int_answer, f"{answer} must be a valid integer."

    @staticmethod
    def is_letter(answer):
        """
        Validates that the answer is a letter.
        :return The letter answer, converted to uppercase; otherwise, None.
        """
        return (
            answer.upper() if answer.isalpha() else None,
            f"{answer} must be a single letter.",
        )

    @staticmethod
    def is_float(answer):
        """
        Validate that the answer can be converted to a float.
        :return The float answer; otherwise, None.
        """
        try:
            float_answer = float(answer)
        except ValueError:
            float_answer = None
        return float_answer, f"{answer} must be a valid float."

    @staticmethod
    def in_range(lower, upper):
        """
        Validate that the answer is within a range. The answer must be of a type
that can
        be compared to the lower and upper bounds.
        :return: The answer, if it is within the range; otherwise, None.
        """

        def _validate(answer):
            return (
                answer if lower <= answer <= upper else None,
                f"{answer} must be between {lower} and {upper}.",
            )

        return _validate
```

- Weitere API-Informationen finden Sie in den folgenden Themen der API-Referenz zum AWS SDK für Python (Boto3).
  - BatchWriteItem
  - CreateTable
  - DeleteItem
  - DeleteTable
  - DescribeTable
  - GetItem
  - PutItem
  - Query
  - Scan
  - UpdateItem

Ruby

SDK für Ruby

> **Note**
>
> Es gibt noch mehr GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

Erstellen Sie eine Klasse, die eine DynamoDB-Tabelle enthält.

```
# Creates an Amazon DynamoDB table that can be used to store movie data.
# The table uses the release year of the movie as the partition key and the
# title as the sort key.
#
# @param table_name [String] The name of the table to create.
# @return [Aws::DynamoDB::Table] The newly created table.
def create_table(table_name)
  @table = @dynamo_resource.create_table(
    table_name: table_name,
    key_schema: [
```

```
        { attribute_name: 'year', key_type: 'HASH' }, # Partition key
        { attribute_name: 'title', key_type: 'RANGE' } # Sort key
      ],
      attribute_definitions: [
        { attribute_name: 'year', attribute_type: 'N' },
        { attribute_name: 'title', attribute_type: 'S' }
      ],
      billing_mode: 'PAY_PER_REQUEST'
    )
    @dynamo_resource.client.wait_until(:table_exists, table_name: table_name)
    @table
  rescue Aws::DynamoDB::Errors::ServiceError => e
    @logger.error("Failed create table #{table_name}:\n#{e.code}: #{e.message}")
    raise
  end
```

Erstellen Sie eine Helper-Funktion zum Herunterladen und Extrahieren der JSON-Beispieldatei.

```
  # Gets sample movie data, either from a local file or by first downloading it
 from
  # the Amazon DynamoDB Developer Guide.
  #
  # @param movie_file_name [String] The local file name where the movie data is
 stored in JSON format.
  # @return [Hash] The movie data as a Hash.
  def fetch_movie_data(movie_file_name)
    if !File.file?(movie_file_name)
      @logger.debug("Downloading #{movie_file_name}...")
      movie_content = URI.open(
        'https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/
samples/moviedata.zip'
      )
      movie_json = ''
      Zip::File.open_buffer(movie_content) do |zip|
        zip.each do |entry|
          movie_json = entry.get_input_stream.read
        end
      end
    else
      movie_json = File.read(movie_file_name)
    end
```

```
    movie_data = JSON.parse(movie_json)
    # The sample file lists over 4000 movies. This returns only the first 250.
    movie_data.slice(0, 250)
  rescue StandardError => e
    puts("Failure downloading movie data:\n#{e}")
    raise
  end
```

Führen Sie ein interaktives Szenario aus, um die Tabelle zu erstellen und Aktionen darauf auszuführen.

```
 table_name = "doc-example-table-movies-#{rand(10**4)}"
 scaffold = Scaffold.new(table_name)
 dynamodb_wrapper = DynamoDBBasics.new(table_name)

 new_step(1, 'Create a new DynamoDB table if none already exists.')
 unless scaffold.exists?(table_name)
   puts("\nNo such table: #{table_name}. Creating it...")
   scaffold.create_table(table_name)
   print "Done!\n".green
 end

 new_step(2, 'Add a new record to the DynamoDB table.')
 my_movie = {}
 my_movie[:title] = CLI::UI::Prompt.ask('Enter the title of a movie to add to
the table. E.g. The Matrix')
 my_movie[:year] = CLI::UI::Prompt.ask('What year was it released? E.g.
1989').to_i
 my_movie[:rating] = CLI::UI::Prompt.ask('On a scale of 1 - 10, how do you rate
it? E.g. 7').to_i
 my_movie[:plot] = CLI::UI::Prompt.ask('Enter a brief summary of the plot. E.g.
A man awakens to a new reality.')
 dynamodb_wrapper.add_item(my_movie)
 puts("\nNew record added:")
 puts JSON.pretty_generate(my_movie).green
 print "Done!\n".green

 new_step(3, 'Update a record in the DynamoDB table.')
 my_movie[:rating] = CLI::UI::Prompt.ask("Let's update the movie you added with
a new rating, e.g. 3:").to_i
 response = dynamodb_wrapper.update_item(my_movie)
 puts("Updated '#{my_movie[:title]}' with new attributes:")
```

```
 puts JSON.pretty_generate(response).green
 print "Done!\n".green

 new_step(4, 'Get a record from the DynamoDB table.')
 puts("Searching for #{my_movie[:title]} (#{my_movie[:year]})...")
 response = dynamodb_wrapper.get_item(my_movie[:title], my_movie[:year])
 puts JSON.pretty_generate(response).green
 print "Done!\n".green

 new_step(5, 'Write a batch of items into the DynamoDB table.')
 download_file = 'moviedata.json'
 puts("Downloading movie database to #{download_file}...")
 movie_data = scaffold.fetch_movie_data(download_file)
 puts("Writing movie data from #{download_file} into your table...")
 scaffold.write_batch(movie_data)
 puts("Records added: #{movie_data.length}.")
 print "Done!\n".green

 new_step(5, 'Query for a batch of items by key.')
 loop do
   release_year = CLI::UI::Prompt.ask('Enter a year between 1972 and 2018, e.g.
1999:').to_i
   results = dynamodb_wrapper.query_items(release_year)
   if results.any?
     puts("There were #{results.length} movies released in #{release_year}:")
     results.each do |movie|
       print "\t #{movie['title']}".green
     end
     break
   else
     continue = CLI::UI::Prompt.ask("Found no movies released in
#{release_year}! Try another year? (y/n)")
     break unless continue.eql?('y')
   end
 end
 print "\nDone!\n".green

 new_step(6, 'Scan for a batch of items using a filter expression.')
 years = {}
 years[:start] = CLI::UI::Prompt.ask('Enter a starting year between 1972 and
2018:')
 years[:end] = CLI::UI::Prompt.ask('Enter an ending year between 1972 and
2018:')
 releases = dynamodb_wrapper.scan_items(years)
```

```
  if !releases.empty?
    puts("Found #{releases.length} movies.")
    count = Question.ask(
      'How many do you want to see? ', method(:is_int), in_range(1,
 releases.length)
    )
    puts("Here are your #{count} movies:")
    releases.take(count).each do |release|
      puts("\t#{release['title']}")
    end
  else
    puts("I don't know about any movies released between #{years[:start]} "\
         "and #{years[:end]}.")
  end
  print "\nDone!\n".green

  new_step(7, 'Delete an item from the DynamoDB table.')
  answer = CLI::UI::Prompt.ask("Do you want to remove '#{my_movie[:title]}'? (y/
n) ")
  if answer.eql?('y')
    dynamodb_wrapper.delete_item(my_movie[:title], my_movie[:year])
    puts("Removed '#{my_movie[:title]}' from the table.")
    print "\nDone!\n".green
  end

  new_step(8, 'Delete the DynamoDB table.')
  answer = CLI::UI::Prompt.ask('Delete the table? (y/n)')
  if answer.eql?('y')
    scaffold.delete_table
    puts("Deleted #{table_name}.")
  else
    puts("Don't forget to delete the table when you're done!")
  end
  print "\nThanks for watching!\n".green
rescue Aws::Errors::ServiceError
  puts('Something went wrong with the demo.')
rescue Errno::ENOENT
  true
end
```

- Weitere API-Informationen finden Sie in den folgenden Themen der AWS SDK für Ruby - API-Referenz.

- BatchWriteItem
- CreateTable
- DeleteItem
- DeleteTable
- DescribeTable
- GetItem
- PutItem
- Query
- Scan
- UpdateItem

SAP ABAP

SDK für SAP ABAP

> **Note**
>
> Es gibt noch mehr dazu. GitHub Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
    " Create an Amazon Dynamo DB table.

    TRY.
        DATA(lo_session) = /aws1/cl_rt_session_aws=>create( cv_pfl ).
        DATA(lo_dyn) = /aws1/cl_dyn_factory=>create( lo_session ).
        DATA(lt_keyschema) = VALUE /aws1/cl_dynkeyschemaelement=>tt_keyschema(
          ( NEW /aws1/cl_dynkeyschemaelement( iv_attributename = 'year'
                                              iv_keytype = 'HASH' ) )
          ( NEW /aws1/cl_dynkeyschemaelement( iv_attributename = 'title'
                                              iv_keytype = 'RANGE' ) ) ).
        DATA(lt_attributedefinitions) = VALUE /aws1/
cl_dynattributedefn=>tt_attributedefinitions(
          ( NEW /aws1/cl_dynattributedefn( iv_attributename = 'year'
                                           iv_attributetype = 'N' ) )
          ( NEW /aws1/cl_dynattributedefn( iv_attributename = 'title'
                                           iv_attributetype = 'S' ) ) ).
```

```
        " Adjust read/write capacities as desired.
        DATA(lo_dynprovthroughput)  = NEW /aws1/cl_dynprovthroughput(
          iv_readcapacityunits = 5
          iv_writecapacityunits = 5 ).
        DATA(oo_result) = lo_dyn->createtable(
          it_keyschema = lt_keyschema
          iv_tablename = iv_table_name
          it_attributedefinitions = lt_attributedefinitions
          io_provisionedthroughput = lo_dynprovthroughput ).
        " Table creation can take some time. Wait till table exists before
 returning.
        lo_dyn->get_waiter( )->tableexists(
          iv_max_wait_time = 200
          iv_tablename     = iv_table_name ).
        MESSAGE 'DynamoDB Table' && iv_table_name && 'created.' TYPE 'I'.
        " It throws exception if the table already exists.
      CATCH /aws1/cx_dynresourceinuseex INTO DATA(lo_resourceinuseex).
        DATA(lv_error) = |"{ lo_resourceinuseex->av_err_code }" -
{ lo_resourceinuseex->av_err_msg }|.
        MESSAGE lv_error TYPE 'E'.
    ENDTRY.

    " Describe table
    TRY.
        DATA(lo_table) = lo_dyn->describetable( iv_tablename = iv_table_name ).
        DATA(lv_tablename) = lo_table->get_table( )->ask_tablename( ).
        MESSAGE 'The table name is ' && lv_tablename TYPE 'I'.
      CATCH /aws1/cx_dynresourcenotfoundex.
        MESSAGE 'The table does not exist' TYPE 'E'.
    ENDTRY.

    " Put items into the table.
    TRY.
        DATA(lo_resp_putitem) = lo_dyn->putitem(
          iv_tablename = iv_table_name
          it_item      = VALUE /aws1/
cl_dynattributevalue=>tt_putiteminputattributemap(
            ( VALUE /aws1/cl_dynattributevalue=>ts_putiteminputattrmap_maprow(
              key = 'title' value = NEW /aws1/cl_dynattributevalue( iv_s =
 'Jaws' ) ) )
            ( VALUE /aws1/cl_dynattributevalue=>ts_putiteminputattrmap_maprow(
              key = 'year' value = NEW /aws1/cl_dynattributevalue( iv_n = |
{ '1975' }| ) ) )
```

```
            ( VALUE /aws1/cl_dynattributevalue=>ts_putiteminputattrmap_maprow(
              key = 'rating' value = NEW /aws1/cl_dynattributevalue( iv_n = |
{ '7.5' }| ) ) )
          ) ).
        lo_resp_putitem = lo_dyn->putitem(
          iv_tablename = iv_table_name
          it_item      = VALUE /aws1/
cl_dynattributevalue=>tt_putiteminputattributemap(
            ( VALUE /aws1/cl_dynattributevalue=>ts_putiteminputattrmap_maprow(
              key = 'title' value = NEW /aws1/cl_dynattributevalue( iv_s = 'Star
 Wars' ) ) )
            ( VALUE /aws1/cl_dynattributevalue=>ts_putiteminputattrmap_maprow(
              key = 'year' value = NEW /aws1/cl_dynattributevalue( iv_n = |
{ '1978' }| ) ) )
            ( VALUE /aws1/cl_dynattributevalue=>ts_putiteminputattrmap_maprow(
              key = 'rating' value = NEW /aws1/cl_dynattributevalue( iv_n = |
{ '8.1' }| ) ) )
          ) ).
        lo_resp_putitem = lo_dyn->putitem(
          iv_tablename = iv_table_name
          it_item      = VALUE /aws1/
cl_dynattributevalue=>tt_putiteminputattributemap(
            ( VALUE /aws1/cl_dynattributevalue=>ts_putiteminputattrmap_maprow(
              key = 'title' value = NEW /aws1/cl_dynattributevalue( iv_s =
 'Speed' ) ) )
            ( VALUE /aws1/cl_dynattributevalue=>ts_putiteminputattrmap_maprow(
              key = 'year' value = NEW /aws1/cl_dynattributevalue( iv_n = |
{ '1994' }| ) ) )
            ( VALUE /aws1/cl_dynattributevalue=>ts_putiteminputattrmap_maprow(
              key = 'rating' value = NEW /aws1/cl_dynattributevalue( iv_n = |
{ '7.9' }| ) ) )
          ) ).
        " TYPE REF TO /AWSEX/CL_AWS1_dyn_PUT_ITEM_OUTPUT
        MESSAGE '3 rows inserted into DynamoDB Table' && iv_table_name TYPE 'I'.
      CATCH /aws1/cx_dyncondalcheckfaile00.
        MESSAGE 'A condition specified in the operation could not be evaluated.'
 TYPE 'E'.
      CATCH /aws1/cx_dynresourcenotfoundex.
        MESSAGE 'The table or index does not exist' TYPE 'E'.
      CATCH /aws1/cx_dyntransactconflictex.
        MESSAGE 'Another transaction is using the item' TYPE 'E'.
    ENDTRY.

    " Get item from table.
```

```
    TRY.
        DATA(lo_resp_getitem) = lo_dyn->getitem(
          iv_tablename               = iv_table_name
          it_key                     = VALUE /aws1/cl_dynattributevalue=>tt_key(
           ( VALUE /aws1/cl_dynattributevalue=>ts_key_maprow(
             key = 'title' value = NEW /aws1/cl_dynattributevalue( iv_s =
 'Jaws' ) ) )
           ( VALUE /aws1/cl_dynattributevalue=>ts_key_maprow(
             key = 'year' value = NEW /aws1/cl_dynattributevalue( iv_n =
 '1975' ) ) )
          ) ).
        DATA(lt_attr) = lo_resp_getitem->get_item( ).
        DATA(lo_title) = lt_attr[ key = 'title' ]-value.
        DATA(lo_year) = lt_attr[ key = 'year' ]-value.
        DATA(lo_rating) = lt_attr[ key = 'year' ]-value.
        MESSAGE 'Movie name is: ' && lo_title->get_s( ) TYPE 'I'.
        MESSAGE 'Movie year is: ' && lo_year->get_n( ) TYPE 'I'.
        MESSAGE 'Movie rating is: ' && lo_rating->get_n( ) TYPE 'I'.
      CATCH /aws1/cx_dynresourcenotfoundex.
        MESSAGE 'The table or index does not exist' TYPE 'E'.
    ENDTRY.

    " Query item from table.
    TRY.
        DATA(lt_attributelist) = VALUE /aws1/
cl_dynattributevalue=>tt_attributevaluelist(
             ( NEW /aws1/cl_dynattributevalue( iv_n = '1975' ) ) ).
        DATA(lt_keyconditions) = VALUE /aws1/cl_dyncondition=>tt_keyconditions(
          ( VALUE /aws1/cl_dyncondition=>ts_keyconditions_maprow(
          key = 'year'
          value = NEW /aws1/cl_dyncondition(
            it_attributevaluelist = lt_attributelist
            iv_comparisonoperator = |EQ|
          ) ) ) ).
        DATA(lo_query_result) = lo_dyn->query(
          iv_tablename = iv_table_name
          it_keyconditions = lt_keyconditions ).
        DATA(lt_items) = lo_query_result->get_items( ).
        READ TABLE lo_query_result->get_items( ) INTO DATA(lt_item) INDEX 1.
        lo_title = lt_item[ key = 'title' ]-value.
        lo_year = lt_item[ key = 'year' ]-value.
        lo_rating = lt_item[ key = 'rating' ]-value.
        MESSAGE 'Movie name is: ' && lo_title->get_s( ) TYPE 'I'.
        MESSAGE 'Movie year is: ' && lo_year->get_n( ) TYPE 'I'.
```

```
        MESSAGE 'Movie rating is: ' && lo_rating->get_n( ) TYPE 'I'.
      CATCH /aws1/cx_dynresourcenotfoundex.
        MESSAGE 'The table or index does not exist' TYPE 'E'.
    ENDTRY.

    " Scan items from table.
    TRY.
        DATA(lo_scan_result) = lo_dyn->scan( iv_tablename = iv_table_name ).
        lt_items = lo_scan_result->get_items( ).
        " Read the first item and display the attributes.
        READ TABLE lo_query_result->get_items( ) INTO lt_item INDEX 1.
        lo_title = lt_item[ key = 'title' ]-value.
        lo_year = lt_item[ key = 'year' ]-value.
        lo_rating = lt_item[ key = 'rating' ]-value.
        MESSAGE 'Movie name is: ' && lo_title->get_s( ) TYPE 'I'.
        MESSAGE 'Movie year is: ' && lo_year->get_n( ) TYPE 'I'.
        MESSAGE 'Movie rating is: ' && lo_rating->get_n( ) TYPE 'I'.
      CATCH /aws1/cx_dynresourcenotfoundex.
        MESSAGE 'The table or index does not exist' TYPE 'E'.
    ENDTRY.

    " Update items from table.
    TRY.
        DATA(lt_attributeupdates) = VALUE /aws1/
cl_dynattrvalueupdate=>tt_attributeupdates(
          ( VALUE /aws1/cl_dynattrvalueupdate=>ts_attributeupdates_maprow(
          key = 'rating' value = NEW /aws1/cl_dynattrvalueupdate(
            io_value  = NEW /aws1/cl_dynattributevalue( iv_n = '7.6' )
            iv_action = |PUT| ) ) ) ).
        DATA(lt_key) = VALUE /aws1/cl_dynattributevalue=>tt_key(
          ( VALUE /aws1/cl_dynattributevalue=>ts_key_maprow(
            key = 'year' value = NEW /aws1/cl_dynattributevalue( iv_n =
 '1975' ) ) )
          ( VALUE /aws1/cl_dynattributevalue=>ts_key_maprow(
            key = 'title' value = NEW /aws1/cl_dynattributevalue( iv_s =
 '1980' ) ) ) ).
        DATA(lo_resp) = lo_dyn->updateitem(
          iv_tablename        = iv_table_name
          it_key              = lt_key
          it_attributeupdates = lt_attributeupdates ).
        MESSAGE '1 item updated in DynamoDB Table' && iv_table_name TYPE 'I'.
      CATCH /aws1/cx_dyncondalcheckfaile00.
        MESSAGE 'A condition specified in the operation could not be evaluated.'
 TYPE 'E'.
```

```
      CATCH /aws1/cx_dynresourcenotfoundex.
        MESSAGE 'The table or index does not exist' TYPE 'E'.
      CATCH /aws1/cx_dyntransactconflictex.
        MESSAGE 'Another transaction is using the item' TYPE 'E'.
    ENDTRY.

    " Delete table.
    TRY.
        lo_dyn->deletetable( iv_tablename = iv_table_name ).
        lo_dyn->get_waiter( )->tablenotexists(
          iv_max_wait_time = 200
          iv_tablename     = iv_table_name ).
        MESSAGE 'DynamoDB Table deleted.' TYPE 'I'.
      CATCH /aws1/cx_dynresourcenotfoundex.
        MESSAGE 'The table or index does not exist' TYPE 'E'.
      CATCH /aws1/cx_dynresourceinuseex.
        MESSAGE 'The table cannot be deleted as it is in use' TYPE 'E'.
    ENDTRY.
```

- Weitere API-Informationen finden Sie in den folgenden Themen der API-Referenz zum AWS -SDK für SAP ABAP.
  - BatchWriteItem
  - CreateTable
  - DeleteItem
  - DeleteTable
  - DescribeTable
  - GetItem
  - PutItem
  - Query
  - Scan
  - UpdateItem

Swift

SDK für Swift

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

Eine Swift-Klasse, die DynamoDB-Aufrufe an das SDK für Swift verarbeitet

```
import AWSDynamoDB
import Foundation

/// An enumeration of error codes representing issues that can arise when using
/// the `MovieTable` class.
enum MoviesError: Error {
    /// The specified table wasn't found or couldn't be created.
    case TableNotFound
    /// The specified item wasn't found or couldn't be created.
    case ItemNotFound
    /// The Amazon DynamoDB client is not properly initialized.
    case UninitializedClient
    /// The table status reported by Amazon DynamoDB is not recognized.
    case StatusUnknown
    /// One or more specified attribute values are invalid or missing.
    case InvalidAttributes
}

/// A class representing an Amazon DynamoDB table containing movie
/// information.
public class MovieTable {
    var ddbClient: DynamoDBClient?
    let tableName: String

    /// Create an object representing a movie table in an Amazon DynamoDB
    /// database.
    ///
    /// - Parameters:
    ///   - region: The optional Amazon Region to create the database in.
    ///   - tableName: The name to assign to the table. If not specified, a
    ///     random table name is generated automatically.
```

```
    ///
    /// > Note: The table is not necessarily available when this function
    /// returns. Use `tableExists()` to check for its availability, or
    /// `awaitTableActive()` to wait until the table's status is reported as
    /// ready to use by Amazon DynamoDB.
    ///
    init(region: String? = nil, tableName: String) async throws {
        do {
            let config = try await DynamoDBClient.DynamoDBClientConfiguration()
            if let region = region {
                config.region = region
            }

            self.ddbClient = DynamoDBClient(config: config)
            self.tableName = tableName

            try await self.createTable()
        } catch {
            print("ERROR: ", dump(error, name: "Initializing Amazon
DynamoDBClient client"))
            throw error
        }
    }

    ///
    /// Create a movie table in the Amazon DynamoDB data store.
    ///
    private func createTable() async throws {
        do {
            guard let client = self.ddbClient else {
                throw MoviesError.UninitializedClient
            }

            let input = CreateTableInput(
                attributeDefinitions: [
                    DynamoDBClientTypes.AttributeDefinition(attributeName:
"year", attributeType: .n),
                    DynamoDBClientTypes.AttributeDefinition(attributeName:
"title", attributeType: .s)
                ],
                billingMode: DynamoDBClientTypes.BillingMode.payPerRequest,
                keySchema: [
                    DynamoDBClientTypes.KeySchemaElement(attributeName: "year",
keyType: .hash),
```

```
                    DynamoDBClientTypes.KeySchemaElement(attributeName: "title",
keyType: .range)
                ],
                tableName: self.tableName
            )
            let output = try await client.createTable(input: input)
            if output.tableDescription == nil {
                throw MoviesError.TableNotFound
            }
        } catch {
            print("ERROR: createTable:", dump(error))
            throw error
        }
    }


    /// Check to see if the table exists online yet.
    ///
    /// - Returns: `true` if the table exists, or `false` if not.
    ///
    func tableExists() async throws -> Bool {
        do {
            guard let client = self.ddbClient else {
                throw MoviesError.UninitializedClient
            }

            let input = DescribeTableInput(
                tableName: tableName
            )
            let output = try await client.describeTable(input: input)
            guard let description = output.table else {
                throw MoviesError.TableNotFound
            }

            return description.tableName == self.tableName
        } catch {
            print("ERROR: tableExists:", dump(error))
            throw error
        }
    }


    ///
    /// Waits for the table to exist and for its status to be active.
```

```
    ///
    func awaitTableActive() async throws {
        while try (await self.tableExists() == false) {
            do {
                let duration = UInt64(0.25 * 1_000_000_000) // Convert .25
seconds to nanoseconds.
                try await Task.sleep(nanoseconds: duration)
            } catch {
                print("Sleep error:", dump(error))
            }
        }

        while try (await self.getTableStatus() != .active) {
            do {
                let duration = UInt64(0.25 * 1_000_000_000) // Convert .25
seconds to nanoseconds.
                try await Task.sleep(nanoseconds: duration)
            } catch {
                print("Sleep error:", dump(error))
            }
        }
    }


    ///
    /// Deletes the table from Amazon DynamoDB.
    ///
    func deleteTable() async throws {
        do {
            guard let client = self.ddbClient else {
                throw MoviesError.UninitializedClient
            }

            let input = DeleteTableInput(
                tableName: self.tableName
            )
            _ = try await client.deleteTable(input: input)
        } catch {
            print("ERROR: deleteTable:", dump(error))
            throw error
        }
    }
```

```
    /// Get the table's status.
    ///
    /// - Returns: The table status, as defined by the
    ///   `DynamoDBClientTypes.TableStatus` enum.
    ///
    func getTableStatus() async throws -> DynamoDBClientTypes.TableStatus {
        do {
            guard let client = self.ddbClient else {
                throw MoviesError.UninitializedClient
            }

            let input = DescribeTableInput(
                tableName: self.tableName
            )
            let output = try await client.describeTable(input: input)
            guard let description = output.table else {
                throw MoviesError.TableNotFound
            }
            guard let status = description.tableStatus else {
                throw MoviesError.StatusUnknown
            }
            return status
        } catch {
            print("ERROR: getTableStatus:", dump(error))
            throw error
        }
    }


    /// Populate the movie database from the specified JSON file.
    ///
    /// - Parameter jsonPath: Path to a JSON file containing movie data.
    ///
    func populate(jsonPath: String) async throws {
        do {
            guard let client = self.ddbClient else {
                throw MoviesError.UninitializedClient
            }

            // Create a Swift `URL` and use it to load the file into a `Data`
            // object. Then decode the JSON into an array of `Movie` objects.

            let fileUrl = URL(fileURLWithPath: jsonPath)
            let jsonData = try Data(contentsOf: fileUrl)
```

```
            var movieList = try JSONDecoder().decode([Movie].self, from:
jsonData)

            // Truncate the list to the first 200 entries or so for this example.

            if movieList.count > 200 {
                movieList = Array(movieList[...199])
            }

            // Before sending records to the database, break the movie list into
            // 25-entry chunks, which is the maximum size of a batch item
request.

            let count = movieList.count
            let chunks = stride(from: 0, to: count, by: 25).map {
                Array(movieList[$0 ..< Swift.min($0 + 25, count)])
            }

            // For each chunk, create a list of write request records and
populate
            // them with `PutRequest` requests, each specifying one movie from
the
            // chunk. Once the chunk's items are all in the `PutRequest` list,
            // send them to Amazon DynamoDB using the
            // `DynamoDBClient.batchWriteItem()` function.

            for chunk in chunks {
                var requestList: [DynamoDBClientTypes.WriteRequest] = []

                for movie in chunk {
                    let item = try await movie.getAsItem()
                    let request = DynamoDBClientTypes.WriteRequest(
                        putRequest: .init(
                            item: item
                        )
                    )
                    requestList.append(request)
                }

                let input = BatchWriteItemInput(requestItems: [tableName:
requestList])
                _ = try await client.batchWriteItem(input: input)
            }
```

```
        } catch {
            print("ERROR: populate:", dump(error))
            throw error
        }
    }


    /// Add a movie specified as a `Movie` structure to the Amazon DynamoDB
    /// table.
    ///
    /// - Parameter movie: The `Movie` to add to the table.
    ///
    func add(movie: Movie) async throws {
        do {
            guard let client = self.ddbClient else {
                throw MoviesError.UninitializedClient
            }

            // Get a DynamoDB item containing the movie data.
            let item = try await movie.getAsItem()

            // Send the `PutItem` request to Amazon DynamoDB.

            let input = PutItemInput(
                item: item,
                tableName: self.tableName
            )
            _ = try await client.putItem(input: input)
        } catch {
            print("ERROR: add movie:", dump(error))
            throw error
        }
    }


    /// Given a movie's details, add a movie to the Amazon DynamoDB table.
    ///
    /// - Parameters:
    ///   - title: The movie's title as a `String`.
    ///   - year: The release year of the movie (`Int`).
    ///   - rating: The movie's rating if available (`Double`; default is
    ///     `nil`).
    ///   - plot: A summary of the movie's plot (`String`; default is `nil`,
    ///     indicating no plot summary is available).
```

```
    ///
    func add(title: String, year: Int, rating: Double? = nil,
             plot: String? = nil) async throws
    {
        do {
            let movie = Movie(title: title, year: year, rating: rating, plot:
plot)
            try await self.add(movie: movie)
        } catch {
            print("ERROR: add with fields:", dump(error))
            throw error
        }
    }


    /// Return a `Movie` record describing the specified movie from the Amazon
    /// DynamoDB table.
    ///
    /// - Parameters:
    ///   - title: The movie's title (`String`).
    ///   - year: The movie's release year (`Int`).
    ///
    /// - Throws: `MoviesError.ItemNotFound` if the movie isn't in the table.
    ///
    /// - Returns: A `Movie` record with the movie's details.
    func get(title: String, year: Int) async throws -> Movie {
        do {
            guard let client = self.ddbClient else {
                throw MoviesError.UninitializedClient
            }

            let input = GetItemInput(
                key: [
                    "year": .n(String(year)),
                    "title": .s(title)
                ],
                tableName: self.tableName
            )
            let output = try await client.getItem(input: input)
            guard let item = output.item else {
                throw MoviesError.ItemNotFound
            }

            let movie = try Movie(withItem: item)
```

```
            return movie
        } catch {
            print("ERROR: get:", dump(error))
            throw error
        }
    }


    /// Get all the movies released in the specified year.
    ///
    /// - Parameter year: The release year of the movies to return.
    ///
    /// - Returns: An array of `Movie` objects describing each matching movie.
    ///
    func getMovies(fromYear year: Int) async throws -> [Movie] {
        do {
            guard let client = self.ddbClient else {
                throw MoviesError.UninitializedClient
            }

            let input = QueryInput(
                expressionAttributeNames: [
                    "#y": "year"
                ],
                expressionAttributeValues: [
                    ":y": .n(String(year))
                ],
                keyConditionExpression: "#y = :y",
                tableName: self.tableName
            )
            // Use "Paginated" to get all the movies.
            // This lets the SDK handle the 'lastEvaluatedKey' property in
"QueryOutput".

            let pages = client.queryPaginated(input: input)

            var movieList: [Movie] = []
            for try await page in pages {
                guard let items = page.items else {
                    print("Error: no items returned.")
                    continue
                }
```

```
                // Convert the found movies into `Movie` objects and return an
array
                // of them.

                for item in items {
                    let movie = try Movie(withItem: item)
                    movieList.append(movie)
                }
            }
            return movieList
        } catch {
            print("ERROR: getMovies:", dump(error))
            throw error
        }
    }


    /// Return an array of `Movie` objects released in the specified range of
    /// years.
    ///
    /// - Parameters:
    ///   - firstYear: The first year of movies to return.
    ///   - lastYear: The last year of movies to return.
    ///   - startKey: A starting point to resume processing; always use `nil`.
    ///
    /// - Returns: An array of `Movie` objects describing the matching movies.
    ///
    /// > Note: The `startKey` parameter is used by this function when
    ///   recursively calling itself, and should always be `nil` when calling
    ///   directly.
    ///
    func getMovies(firstYear: Int, lastYear: Int,
                   startKey: [Swift.String: DynamoDBClientTypes.AttributeValue]?
= nil)
        async throws -> [Movie]
    {
        do {
            var movieList: [Movie] = []

            guard let client = self.ddbClient else {
                throw MoviesError.UninitializedClient
            }

            let input = ScanInput(
```

```
                consistentRead: true,
                exclusiveStartKey: startKey,
                expressionAttributeNames: [
                    "#y": "year" // `year` is a reserved word, so use `#y`
instead.
                ],
                expressionAttributeValues: [
                    ":y1": .n(String(firstYear)),
                    ":y2": .n(String(lastYear))
                ],
                filterExpression: "#y BETWEEN :y1 AND :y2",
                tableName: self.tableName
            )

            let pages = client.scanPaginated(input: input)

            for try await page in pages {
                guard let items = page.items else {
                    print("Error: no items returned.")
                    continue
                }

                // Build an array of `Movie` objects for the returned items.

                for item in items {
                    let movie = try Movie(withItem: item)
                    movieList.append(movie)
                }
            }
            return movieList

        } catch {
            print("ERROR: getMovies with scan:", dump(error))
            throw error
        }
    }


    /// Update the specified movie with new `rating` and `plot` information.
    ///
    /// - Parameters:
    ///   - title: The title of the movie to update.
    ///   - year: The release year of the movie to update.
    ///   - rating: The new rating for the movie.
```

```
    ///   - plot: The new plot summary string for the movie.
    ///
    /// - Returns: An array of mappings of attribute names to their new
    ///   listing each item actually changed. Items that didn't need to change
    ///   aren't included in this list. `nil` if no changes were made.
    ///
    func update(title: String, year: Int, rating: Double? = nil, plot: String? =
nil) async throws
        -> [Swift.String: DynamoDBClientTypes.AttributeValue]?
    {
        do {
            guard let client = self.ddbClient else {
                throw MoviesError.UninitializedClient
            }

            // Build the update expression and the list of expression attribute
            // values. Include only the information that's changed.

            var expressionParts: [String] = []
            var attrValues: [Swift.String: DynamoDBClientTypes.AttributeValue] =
[:]

            if rating != nil {
                expressionParts.append("info.rating=:r")
                attrValues[":r"] = .n(String(rating!))
            }
            if plot != nil {
                expressionParts.append("info.plot=:p")
                attrValues[":p"] = .s(plot!)
            }
            let expression = "set \(expressionParts.joined(separator: ", "))"

            let input = UpdateItemInput(
                // Create substitution tokens for the attribute values, to ensure
                // no conflicts in expression syntax.
                expressionAttributeValues: attrValues,
                // The key identifying the movie to update consists of the
release
                // year and title.
                key: [
                    "year": .n(String(year)),
                    "title": .s(title)
                ],
                returnValues: .updatedNew,
```

```
                tableName: self.tableName,
                updateExpression: expression
            )
            let output = try await client.updateItem(input: input)

            guard let attributes: [Swift.String:
 DynamoDBClientTypes.AttributeValue] = output.attributes else {
                throw MoviesError.InvalidAttributes
            }
            return attributes
        } catch {
            print("ERROR: update:", dump(error))
            throw error
        }
    }


    /// Delete a movie, given its title and release year.
    ///
    /// - Parameters:
    ///   - title: The movie's title.
    ///   - year: The movie's release year.
    ///
    func delete(title: String, year: Int) async throws {
        do {
            guard let client = self.ddbClient else {
                throw MoviesError.UninitializedClient
            }

            let input = DeleteItemInput(
                key: [
                    "year": .n(String(year)),
                    "title": .s(title)
                ],
                tableName: self.tableName
            )
            _ = try await client.deleteItem(input: input)
        } catch {
            print("ERROR: delete:", dump(error))
            throw error
        }
    }
}
```

Die Strukturen, die von der MovieTable Klasse verwendet werden, um Filme darzustellen.

```
import Foundation
import AWSDynamoDB

/// The optional details about a movie.
public struct Details: Codable {
    /// The movie's rating, if available.
    var rating: Double?
    /// The movie's plot, if available.
    var plot: String?
}

/// A structure describing a movie. The `year` and `title` properties are
/// required and are used as the key for Amazon DynamoDB operations. The
/// `info` sub-structure's two properties, `rating` and `plot`, are optional.
public struct Movie: Codable {
    /// The year in which the movie was released.
    var year: Int
    /// The movie's title.
    var title: String
    /// A `Details` object providing the optional movie rating and plot
    /// information.
    var info: Details

    /// Create a `Movie` object representing a movie, given the movie's
    /// details.
    ///
    /// - Parameters:
    ///   - title: The movie's title (`String`).
    ///   - year: The year in which the movie was released (`Int`).
    ///   - rating: The movie's rating (optional `Double`).
    ///   - plot: The movie's plot (optional `String`)
    init(title: String, year: Int, rating: Double? = nil, plot: String? = nil) {
        self.title = title
        self.year = year

        self.info = Details(rating: rating, plot: plot)
    }

    /// Create a `Movie` object representing a movie, given the movie's
```

```
    /// details.
    ///
    /// - Parameters:
    ///   - title: The movie's title (`String`).
    ///   - year: The year in which the movie was released (`Int`).
    ///   - info: The optional rating and plot information for the movie in a
    ///     `Details` object.
    init(title: String, year: Int, info: Details?){
        self.title = title
        self.year = year

        if info != nil {
            self.info = info!
        } else {
            self.info = Details(rating: nil, plot: nil)
        }
    }

    ///
    /// Return a new `MovieTable` object, given an array mapping string to Amazon
    /// DynamoDB attribute values.
    ///
    /// - Parameter item: The item information provided to the form used by
    ///   DynamoDB. This is an array of strings mapped to
    ///   `DynamoDBClientTypes.AttributeValue` values.
    init(withItem item: [Swift.String:DynamoDBClientTypes.AttributeValue]) throws
{
        // Read the attributes.

        guard let titleAttr = item["title"],
              let yearAttr = item["year"] else {
            throw MoviesError.ItemNotFound
        }
        let infoAttr = item["info"] ?? nil

        // Extract the values of the title and year attributes.

        if case .s(let titleVal) = titleAttr {
            self.title = titleVal
        } else {
            throw MoviesError.InvalidAttributes
        }

        if case .n(let yearVal) = yearAttr {
```

```
            self.year = Int(yearVal)!
        } else {
            throw MoviesError.InvalidAttributes
        }

        // Extract the rating and/or plot from the `info` attribute, if
        // they're present.

        var rating: Double? = nil
        var plot: String? = nil

        if infoAttr != nil, case .m(let infoVal) = infoAttr {
            let ratingAttr = infoVal["rating"] ?? nil
            let plotAttr = infoVal["plot"] ?? nil

            if ratingAttr != nil, case .n(let ratingVal) = ratingAttr {
                rating = Double(ratingVal) ?? nil
            }
            if plotAttr != nil, case .s(let plotVal) = plotAttr {
                plot = plotVal
            }
        }

        self.info = Details(rating: rating, plot: plot)
    }

    ///
    /// Return an array mapping attribute names to Amazon DynamoDB attribute
    /// values, representing the contents of the `Movie` record as a DynamoDB
    /// item.
    ///
    /// - Returns: The movie item as an array of type
    ///   `[Swift.String:DynamoDBClientTypes.AttributeValue]`.
    ///
    func getAsItem() async throws ->
[Swift.String:DynamoDBClientTypes.AttributeValue]  {
        // Build the item record, starting with the year and title, which are
        // always present.

        var item: [Swift.String:DynamoDBClientTypes.AttributeValue] = [
            "year": .n(String(self.year)),
            "title": .s(self.title)
        ]
```

```
        // Add the `info` field with the rating and/or plot if they're
        // available.

        var details: [Swift.String:DynamoDBClientTypes.AttributeValue] = [:]
        if (self.info.rating != nil || self.info.plot != nil) {
            if self.info.rating != nil {
                details["rating"] = .n(String(self.info.rating!))
            }
            if self.info.plot != nil {
                details["plot"] = .s(self.info.plot!)
            }
        }
        item["info"] = .m(details)

        return item
    }
 }
```

Ein Programm, das die MovieTable Klasse verwendet, um auf eine DynamoDB-Datenbank zuzugreifen.

```
import ArgumentParser
import ClientRuntime
import Foundation

import AWSDynamoDB


@testable import MovieList

extension String {
    // Get the directory if the string is a file path.
    func directory() -> String {
        guard let lastIndex = lastIndex(of: "/") else {
            print("Error: String directory separator not found.")
            return ""
        }
        return String(self[...lastIndex])
    }
}

struct ExampleCommand: ParsableCommand {
```

```
    @Argument(help: "The path of the sample movie data JSON file.")
    var jsonPath: String = #file.directory() + "../../../../../resources/
sample_files/movies.json"

    @Option(help: "The AWS Region to run AWS API calls in.")
    var awsRegion: String?

    @Option(
        help: ArgumentHelp("The level of logging for the Swift SDK to perform."),
        completion: .list([
            "critical",
            "debug",
            "error",
            "info",
            "notice",
            "trace",
            "warning"
        ])
    )
    var logLevel: String = "error"

    /// Configuration details for the command.
    static var configuration = CommandConfiguration(
        commandName: "basics",
        abstract: "A basic scenario demonstrating the usage of Amazon DynamoDB.",
        discussion: """
        An example showing how to use Amazon DynamoDB to perform a series of
        common database activities on a simple movie database.
        """
    )

    /// Called by ``main()`` to asynchronously run the AWS example.
    func runAsync() async throws {
        print("Welcome to the AWS SDK for Swift basic scenario for Amazon
 DynamoDB!")

        //========================================================================
        // 1. Create the table. The Amazon DynamoDB table is represented by
        //    the `MovieTable` class.
        //========================================================================

        let tableName = "ddb-movies-sample-\(Int.random(in: 1 ... Int.max))"

        print("Creating table \"\(tableName)\"...")
```

```
        let movieDatabase = try await MovieTable(region: awsRegion,
                                                 tableName: tableName)

        print("\nWaiting for table to be ready to use...")
        try await movieDatabase.awaitTableActive()

        //========================================================================
        // 2. Add a movie to the table.
        //========================================================================

        print("\nAdding a movie...")
        try await movieDatabase.add(title: "Avatar: The Way of Water", year:
2022)
        try await movieDatabase.add(title: "Not a Real Movie", year: 2023)

        //========================================================================
        // 3. Update the plot and rating of the movie using an update
        //    expression.
        //========================================================================

        print("\nAdding details to the added movie...")
        _ = try await movieDatabase.update(title: "Avatar: The Way of Water",
year: 2022,
                                           rating: 9.2, plot: "It's a sequel.")

        //========================================================================
        // 4. Populate the table from the JSON file.
        //========================================================================

        print("\nPopulating the movie database from JSON...")
        try await movieDatabase.populate(jsonPath: jsonPath)

        //========================================================================
        // 5. Get a specific movie by key. In this example, the key is a
        //    combination of `title` and `year`.
        //========================================================================

        print("\nLooking for a movie in the table...")
        let gotMovie = try await movieDatabase.get(title: "This Is the End",
year: 2013)

        print("Found the movie \"\(gotMovie.title)\", released in
\(gotMovie.year).")
```

```
        print("Rating: \(gotMovie.info.rating ?? 0.0).")
        print("Plot summary: \(gotMovie.info.plot ?? "None.")")

        //========================================================================
        // 6. Delete a movie.
        //========================================================================

        print("\nDeleting the added movie...")
        try await movieDatabase.delete(title: "Avatar: The Way of Water", year:
 2022)

        //========================================================================
        // 7. Use a query with a key condition expression to return all movies
        //    released in a given year.
        //========================================================================

        print("\nGetting movies released in 1994...")
        let movieList = try await movieDatabase.getMovies(fromYear: 1994)
        for movie in movieList {
            print("    \(movie.title)")
        }

        //========================================================================
        // 8. Use `scan()` to return movies released in a range of years.
        //========================================================================

        print("\nGetting movies released between 1993 and 1997...")
        let scannedMovies = try await movieDatabase.getMovies(firstYear: 1993,
 lastYear: 1997)
        for movie in scannedMovies {
            print("    \(movie.title) (\(movie.year))")
        }

        //========================================================================
        // 9. Delete the table.
        //========================================================================

        print("\nDeleting the table...")
        try await movieDatabase.deleteTable()
    }
}

@main
struct Main {
```

```
    static func main() async {
        let args = Array(CommandLine.arguments.dropFirst())

        do {
            let command = try ExampleCommand.parse(args)
            try await command.runAsync()
        } catch {
            ExampleCommand.exit(withError: error)
        }
    }
}
```

- Weitere API-Informationen finden Sie in den folgenden Themen der API-Referenz zum AWS SDK für Swift.
  - BatchWriteItem
  - CreateTable
  - DeleteItem
  - DeleteTable
  - DescribeTable
  - GetItem
  - PutItem
  - Query
  - Scan
  - UpdateItem

Eine vollständige Liste der AWS SDK-Entwicklerhandbücher und Codebeispiele finden Sie unter. DynamoDB mit einem SDK verwenden AWS Dieses Thema enthält auch Informationen zu den ersten Schritten und Details zu früheren SDK-Versionen.

Aktionen für DynamoDB mit AWS SDKs

Die folgenden Codebeispiele zeigen, wie einzelne DynamoDB-Aktionen mit ausgeführt werden. AWS SDKs Jedes Beispiel enthält einen Link zu GitHub, wo Sie Anweisungen zum Einrichten und Ausführen des Codes finden.

Diese Auszüge rufen die DynamoDB-API auf und sind Codeauszüge aus größeren Programmen, die im Kontext ausgeführt werden müssen. Sie können Aktionen im Kontext unter Szenarien für die Verwendung von DynamoDB AWS SDKs anzeigen.

Die folgenden Beispiele enthalten nur die am häufigsten verwendeten Aktionen. Eine vollständige Liste finden Sie in der API-Referenz zu Amazon DynamoDB.

Beispiele

- BatchExecuteStatementMit einem AWS SDK verwenden
- Verwendung BatchGetItem mit einem AWS SDK oder CLI
- Verwendung BatchWriteItem mit einem AWS SDK oder CLI
- Verwendung CreateTable mit einem AWS SDK oder CLI
- Verwendung DeleteItem mit einem AWS SDK oder CLI
- Verwendung DeleteTable mit einem AWS SDK oder CLI
- Verwendung DescribeTable mit einem AWS SDK oder CLI
- Verwendung DescribeTimeToLive mit einem AWS SDK oder CLI
- ExecuteStatementMit einem AWS SDK verwenden
- Verwendung GetItem mit einem AWS SDK oder CLI
- Verwendung ListTables mit einem AWS SDK oder CLI
- Verwendung PutItem mit einem AWS SDK oder CLI
- Verwendung Query mit einem AWS SDK oder CLI
- Verwendung Scan mit einem AWS SDK oder CLI
- Verwendung UpdateItem mit einem AWS SDK oder CLI
- Verwendung UpdateTable mit einem AWS SDK oder CLI
- Verwendung UpdateTimeToLive mit einem AWS SDK oder CLI

`BatchExecuteStatement`Mit einem AWS SDK verwenden

Die folgenden Code-Beispiele zeigen, wie `BatchExecuteStatement` verwendet wird.

Aktionsbeispiele sind Codeauszüge aus größeren Programmen und müssen im Kontext ausgeführt werden. Sie können diese Aktion in den folgenden Codebeispielen im Kontext sehen:

- Löschen von Daten mit PartiQL DELETE

- Einfügen von Daten mithilfe von PartiQL INSERT
- Abfragen einer Tabelle mithilfe von Stapeln von PartiQL-Anweisungen
- Abfragen von Daten mit PartiQL SELECT
- Aktualisieren von Daten mit PartiQL UPDATE

.NET

SDK für .NET

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

Verwenden Sie Stapel von INSERT-Anweisungen, um Elemente hinzuzufügen.

```
        /// <summary>
        /// Inserts movies imported from a JSON file into the movie table by
        /// using an Amazon DynamoDB PartiQL INSERT statement.
        /// </summary>
        /// <param name="tableName">The name of the table into which the movie
        /// information will be inserted.</param>
        /// <param name="movieFileName">The name of the JSON file that contains
        /// movie information.</param>
        /// <returns>A Boolean value that indicates the success or failure of
        /// the insert operation.</returns>
        public static async Task<bool> InsertMovies(string tableName, string
movieFileName)
        {
            // Get the list of movies from the JSON file.
            var movies = ImportMovies(movieFileName);

            var success = false;

            if (movies is not null)
            {
                // Insert the movies in a batch using PartiQL. Because the
                // batch can contain a maximum of 25 items, insert 25 movies
                // at a time.
```

```
                string insertBatch = $"INSERT INTO {tableName} VALUE
{{'title': ?, 'year': ?}}";
                var statements = new List<BatchStatementRequest>();

                try
                {
                    for (var indexOffset = 0; indexOffset < 250; indexOffset +=
25)
                    {
                        for (var i = indexOffset; i < indexOffset + 25; i++)
                        {
                            statements.Add(new BatchStatementRequest
                            {
                                Statement = insertBatch,
                                Parameters = new List<AttributeValue>
                                {
                                    new AttributeValue { S = movies[i].Title },
                                    new AttributeValue { N =
movies[i].Year.ToString() },
                                },
                            });
                        }

                        var response = await
Client.BatchExecuteStatementAsync(new BatchExecuteStatementRequest
                        {
                            Statements = statements,
                        });

                        // Wait between batches for movies to be successfully
added.
                        System.Threading.Thread.Sleep(3000);

                        success = response.HttpStatusCode ==
System.Net.HttpStatusCode.OK;

                        // Clear the list of statements for the next batch.
                        statements.Clear();
                    }
                }
                catch (AmazonDynamoDBException ex)
                {
                    Console.WriteLine(ex.Message);
                }
```

```
            }

            return success;
        }

        /// <summary>
        /// Loads the contents of a JSON file into a list of movies to be
        /// added to the DynamoDB table.
        /// </summary>
        /// <param name="movieFileName">The full path to the JSON file.</param>
        /// <returns>A generic list of movie objects.</returns>
        public static List<Movie> ImportMovies(string movieFileName)
        {
            if (!File.Exists(movieFileName))
            {
                return null!;
            }

            using var sr = new StreamReader(movieFileName);
            string json = sr.ReadToEnd();
            var allMovies = JsonConvert.DeserializeObject<List<Movie>>(json);

            if (allMovies is not null)
            {
                // Return the first 250 entries.
                return allMovies.GetRange(0, 250);
            }
            else
            {
                return null!;
            }
        }
```

Verwenden Sie Stapel von SELECT-Anweisungen, um Elemente abzurufen.

```
        /// <summary>
        /// Gets movies from the movie table by
        /// using an Amazon DynamoDB PartiQL SELECT statement.
        /// </summary>
        /// <param name="tableName">The name of the table.</param>
        /// <param name="title1">The title of the first movie.</param>
        /// <param name="title2">The title of the second movie.</param>
```

```
        /// <param name="year1">The year of the first movie.</param>
        /// <param name="year2">The year of the second movie.</param>
        /// <returns>True if successful.</returns>
        public static async Task<bool> GetBatch(
            string tableName,
            string title1,
            string title2,
            int year1,
            int year2)
        {
            var getBatch = $"SELECT * FROM {tableName} WHERE title = ? AND year
= ?";
            var statements = new List<BatchStatementRequest>
            {
                new BatchStatementRequest
                {
                    Statement = getBatch,
                    Parameters = new List<AttributeValue>
                    {
                        new AttributeValue { S = title1 },
                        new AttributeValue { N = year1.ToString() },
                    },
                },

                new BatchStatementRequest
                {
                    Statement = getBatch,
                    Parameters = new List<AttributeValue>
                    {
                        new AttributeValue { S = title2 },
                        new AttributeValue { N = year2.ToString() },
                    },
                }
            };

            var response = await Client.BatchExecuteStatementAsync(new
BatchExecuteStatementRequest
            {
                Statements = statements,
            });

            if (response.Responses.Count > 0)
            {
                response.Responses.ForEach(r =>
```

```
                {
                    if (r.Item.Any())
                    {

 Console.WriteLine($"{r.Item["title"]}\t{r.Item["year"]}");
                    }
                });
                return true;
            }
            else
            {
                Console.WriteLine($"Couldn't find either {title1} or {title2}.");
                return false;
            }

        }
```

Verwenden Sie Stapel von UPDATE-Anweisungen, um Elemente zu aktualisieren.

```
        /// <summary>
        /// Updates information for multiple movies.
        /// </summary>
        /// <param name="tableName">The name of the table containing the
        /// movies to be updated.</param>
        /// <param name="producer1">The producer name for the first movie
        /// to update.</param>
        /// <param name="title1">The title of the first movie.</param>
        /// <param name="year1">The year that the first movie was released.</
param>
        /// <param name="producer2">The producer name for the second
        /// movie to update.</param>
        /// <param name="title2">The title of the second movie.</param>
        /// <param name="year2">The year that the second movie was released.</
param>
        /// <returns>A Boolean value that indicates the success of the update.</
returns>
        public static async Task<bool> UpdateBatch(
            string tableName,
            string producer1,
            string title1,
            int year1,
            string producer2,
```

```
            string title2,
            int year2)
        {

            string updateBatch = $"UPDATE {tableName} SET Producer=? WHERE title
= ? AND year = ?";
            var statements = new List<BatchStatementRequest>
            {
                new BatchStatementRequest
                {
                    Statement = updateBatch,
                    Parameters = new List<AttributeValue>
                    {
                        new AttributeValue { S = producer1 },
                        new AttributeValue { S = title1 },
                        new AttributeValue { N = year1.ToString() },
                    },
                },

                new BatchStatementRequest
                {
                    Statement = updateBatch,
                    Parameters = new List<AttributeValue>
                    {
                        new AttributeValue { S = producer2 },
                        new AttributeValue { S = title2 },
                        new AttributeValue { N = year2.ToString() },
                    },
                }
            };

            var response = await Client.BatchExecuteStatementAsync(new
BatchExecuteStatementRequest
            {
                Statements = statements,
            });

            return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
        }
```

Verwenden Sie Stapel von DELETE-Anweisungen, um Elemente zu löschen.

```
        /// <summary>
        /// Deletes multiple movies using a PartiQL BatchExecuteAsync
        /// statement.
        /// </summary>
        /// <param name="tableName">The name of the table containing the
        /// moves that will be deleted.</param>
        /// <param name="title1">The title of the first movie.</param>
        /// <param name="year1">The year the first movie was released.</param>
        /// <param name="title2">The title of the second movie.</param>
        /// <param name="year2">The year the second movie was released.</param>
        /// <returns>A Boolean value indicating the success of the operation.</
returns>
        public static async Task<bool> DeleteBatch(
            string tableName,
            string title1,
            int year1,
            string title2,
            int year2)
        {

            string updateBatch = $"DELETE FROM {tableName} WHERE title = ? AND
 year = ?";
            var statements = new List<BatchStatementRequest>
            {
                new BatchStatementRequest
                {
                    Statement = updateBatch,
                    Parameters = new List<AttributeValue>
                    {
                        new AttributeValue { S = title1 },
                        new AttributeValue { N = year1.ToString() },
                    },
                },

                new BatchStatementRequest
                {
                    Statement = updateBatch,
                    Parameters = new List<AttributeValue>
                    {
                        new AttributeValue { S = title2 },
                        new AttributeValue { N = year2.ToString() },
                    },
                }
```

```
            };

            var response = await Client.BatchExecuteStatementAsync(new
BatchExecuteStatementRequest
            {
                Statements = statements,
            });

            return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
        }
```

- Einzelheiten zur API finden Sie BatchExecuteStatementin der AWS SDK für .NET API-Referenz.

C++

SDK für C++

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

Verwenden Sie Stapel von INSERT-Anweisungen, um Elemente hinzuzufügen.

```
    // 2. Add multiple movies using "Insert" statements. (BatchExecuteStatement)
    Aws::DynamoDB::DynamoDBClient dynamoClient(clientConfiguration);

    std::vector<Aws::String> titles;
    std::vector<float> ratings;
    std::vector<int> years;
    std::vector<Aws::String> plots;
    Aws::String doAgain = "n";
    do {
        Aws::String aTitle = askQuestion(
                "Enter the title of a movie you want to add to the table: ");
        titles.push_back(aTitle);
        int aYear = askQuestionForInt("What year was it released? ");
        years.push_back(aYear);
        float aRating = askQuestionForFloatRange(
```

```
                "On a scale of 1 - 10, how do you rate it? ",
                1, 10);
        ratings.push_back(aRating);
        Aws::String aPlot = askQuestion("Summarize the plot for me: ");
        plots.push_back(aPlot);

        doAgain = askQuestion(Aws::String("Would you like to add more movies? (y/
n) "));
    } while (doAgain == "y");

    std::cout << "Adding " << titles.size()
              << (titles.size() == 1 ? " movie " : " movies ")
              << "to the table using a batch \"INSERT\" statement." << std::endl;

    {
        Aws::Vector<Aws::DynamoDB::Model::BatchStatementRequest> statements(
                titles.size());

        std::stringstream sqlStream;
        sqlStream << "INSERT INTO \"" << MOVIE_TABLE_NAME << "\" VALUE {'"
                  << TITLE_KEY << "': ?, '" << YEAR_KEY << "': ?, '"
                  << INFO_KEY << "': ?}";

        std::string sql(sqlStream.str());

        for (size_t i = 0; i < statements.size(); ++i) {
            statements[i].SetStatement(sql);

            Aws::Vector<Aws::DynamoDB::Model::AttributeValue> attributes;
            attributes.push_back(
                    Aws::DynamoDB::Model::AttributeValue().SetS(titles[i]));

 attributes.push_back(Aws::DynamoDB::Model::AttributeValue().SetN(years[i]));

            // Create attribute for the info map.
            Aws::DynamoDB::Model::AttributeValue infoMapAttribute;

            std::shared_ptr<Aws::DynamoDB::Model::AttributeValue> ratingAttribute
 = Aws::MakeShared<Aws::DynamoDB::Model::AttributeValue>(
                    ALLOCATION_TAG.c_str());
            ratingAttribute->SetN(ratings[i]);
            infoMapAttribute.AddMEntry(RATING_KEY, ratingAttribute);
```

```
            std::shared_ptr<Aws::DynamoDB::Model::AttributeValue> plotAttribute =
Aws::MakeShared<Aws::DynamoDB::Model::AttributeValue>(
                    ALLOCATION_TAG.c_str());
            plotAttribute->SetS(plots[i]);
            infoMapAttribute.AddMEntry(PLOT_KEY, plotAttribute);
            attributes.push_back(infoMapAttribute);
            statements[i].SetParameters(attributes);
        }

        Aws::DynamoDB::Model::BatchExecuteStatementRequest request;

        request.SetStatements(statements);

        Aws::DynamoDB::Model::BatchExecuteStatementOutcome outcome =
dynamoClient.BatchExecuteStatement(
                request);
        if (!outcome.IsSuccess()) {
            std::cerr << "Failed to add the movies: " <<
outcome.GetError().GetMessage()
                      << std::endl;
            return false;
        }
    }
```

Verwenden Sie Stapel von SELECT-Anweisungen, um Elemente abzurufen.

```
    // 3. Get the data for multiple movies using "Select" statements.
(BatchExecuteStatement)
    {
        Aws::Vector<Aws::DynamoDB::Model::BatchStatementRequest> statements(
                titles.size());
        std::stringstream sqlStream;
        sqlStream << "SELECT * FROM  \"" << MOVIE_TABLE_NAME << "\" WHERE "
                  << TITLE_KEY << "=? and " << YEAR_KEY << "=?";

        std::string sql(sqlStream.str());

        for (size_t i = 0; i < statements.size(); ++i) {
            statements[i].SetStatement(sql);
            Aws::Vector<Aws::DynamoDB::Model::AttributeValue> attributes;
            attributes.push_back(
                    Aws::DynamoDB::Model::AttributeValue().SetS(titles[i]));
```

```
attributes.push_back(Aws::DynamoDB::Model::AttributeValue().SetN(years[i]));
            statements[i].SetParameters(attributes);
        }

        Aws::DynamoDB::Model::BatchExecuteStatementRequest request;

        request.SetStatements(statements);

        Aws::DynamoDB::Model::BatchExecuteStatementOutcome outcome =
dynamoClient.BatchExecuteStatement(
                request);
        if (outcome.IsSuccess()) {
            const Aws::DynamoDB::Model::BatchExecuteStatementResult &result =
outcome.GetResult();

            const Aws::Vector<Aws::DynamoDB::Model::BatchStatementResponse>
&responses = result.GetResponses();

            for (const Aws::DynamoDB::Model::BatchStatementResponse &response:
responses) {
                const Aws::Map<Aws::String, Aws::DynamoDB::Model::AttributeValue>
&item = response.GetItem();

                printMovieInfo(item);
            }
        }
        else {
            std::cerr << "Failed to retrieve the movie information: "
                      << outcome.GetError().GetMessage() << std::endl;
            return false;
        }
    }
```

Verwenden Sie Stapel von UPDATE-Anweisungen, um Elemente zu aktualisieren.

```
    // 4. Update the data for multiple movies using "Update" statements.
(BatchExecuteStatement)

    for (size_t i = 0; i < titles.size(); ++i) {
        ratings[i] = askQuestionForFloatRange(
                Aws::String("\nLet's update your the movie, \"") + titles[i] +
```

```
                ".\nYou rated it  " + std::to_string(ratings[i])
                + ", what new rating would you give it? ", 1, 10);
    }

    std::cout << "Updating the movie with a batch \"UPDATE\" statement." <<
std::endl;

    {
        Aws::Vector<Aws::DynamoDB::Model::BatchStatementRequest> statements(
                titles.size());

        std::stringstream sqlStream;
        sqlStream << "UPDATE \"" << MOVIE_TABLE_NAME << "\" SET "
                  << INFO_KEY << "." << RATING_KEY << "=? WHERE "
                  << TITLE_KEY << "=? AND " << YEAR_KEY << "=?";


        std::string sql(sqlStream.str());

        for (size_t i = 0; i < statements.size(); ++i) {
            statements[i].SetStatement(sql);

            Aws::Vector<Aws::DynamoDB::Model::AttributeValue> attributes;
            attributes.push_back(
                    Aws::DynamoDB::Model::AttributeValue().SetN(ratings[i]));
            attributes.push_back(
                    Aws::DynamoDB::Model::AttributeValue().SetS(titles[i]));

attributes.push_back(Aws::DynamoDB::Model::AttributeValue().SetN(years[i]));
            statements[i].SetParameters(attributes);
        }

        Aws::DynamoDB::Model::BatchExecuteStatementRequest request;

        request.SetStatements(statements);
        Aws::DynamoDB::Model::BatchExecuteStatementOutcome outcome =
dynamoClient.BatchExecuteStatement(
                request);
        if (!outcome.IsSuccess()) {
            std::cerr << "Failed to update movie information: "
                      << outcome.GetError().GetMessage() << std::endl;
            return false;
        }
    }
```

Verwenden Sie Stapel von DELETE-Anweisungen, um Elemente zu löschen.

```
   // 6. Delete multiple movies using "Delete" statements.
(BatchExecuteStatement)
   {
       Aws::Vector<Aws::DynamoDB::Model::BatchStatementRequest> statements(
               titles.size());
       std::stringstream sqlStream;
       sqlStream << "DELETE FROM  \"" << MOVIE_TABLE_NAME << "\" WHERE "
                 << TITLE_KEY << "=? and " << YEAR_KEY << "=?";

       std::string sql(sqlStream.str());

       for (size_t i = 0; i < statements.size(); ++i) {
           statements[i].SetStatement(sql);
           Aws::Vector<Aws::DynamoDB::Model::AttributeValue> attributes;
           attributes.push_back(
                   Aws::DynamoDB::Model::AttributeValue().SetS(titles[i]));

attributes.push_back(Aws::DynamoDB::Model::AttributeValue().SetN(years[i]));
           statements[i].SetParameters(attributes);
       }

       Aws::DynamoDB::Model::BatchExecuteStatementRequest request;

       request.SetStatements(statements);

       Aws::DynamoDB::Model::BatchExecuteStatementOutcome outcome =
dynamoClient.BatchExecuteStatement(
               request);

       if (!outcome.IsSuccess()) {
           std::cerr << "Failed to delete the movies: "
                     << outcome.GetError().GetMessage() << std::endl;
           return false;
       }
   }
```

- Einzelheiten zur API finden Sie BatchExecuteStatementin der AWS SDK für C++ API-Referenz.

Go

SDK für Go V2

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

Definieren Sie eine Funktionsempfängerstruktur für das Beispiel.

```
import (
 "context"
 "fmt"
 "log"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/attributevalue"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb/types"
)

// PartiQLRunner encapsulates the Amazon DynamoDB service actions used in the
// PartiQL examples. It contains a DynamoDB service client that is used to act on
 the
// specified table.
type PartiQLRunner struct {
 DynamoDbClient *dynamodb.Client
 TableName      string
}
```

Verwenden Sie Stapel von INSERT-Anweisungen, um Elemente hinzuzufügen.

```
// AddMovieBatch runs a batch of PartiQL INSERT statements to add multiple movies
 to the
// DynamoDB table.
```

```
func (runner PartiQLRunner) AddMovieBatch(ctx context.Context, movies []Movie)
 error {
 statementRequests := make([]types.BatchStatementRequest, len(movies))
 for index, movie := range movies {
  params, err := attributevalue.MarshalList([]interface{}{movie.Title,
 movie.Year, movie.Info})
  if err != nil {
   panic(err)
  }
  statementRequests[index] = types.BatchStatementRequest{
   Statement: aws.String(fmt.Sprintf(
    "INSERT INTO \"%v\" VALUE {'title': ?, 'year': ?, 'info': ?}",
 runner.TableName)),
   Parameters: params,
  }
 }

 _, err := runner.DynamoDbClient.BatchExecuteStatement(ctx,
 &dynamodb.BatchExecuteStatementInput{
  Statements: statementRequests,
 })
 if err != nil {
  log.Printf("Couldn't insert a batch of items with PartiQL. Here's why: %v\n",
 err)
 }
 return err
}
```

Verwenden Sie Stapel von SELECT-Anweisungen, um Elemente abzurufen.

```
// GetMovieBatch runs a batch of PartiQL SELECT statements to get multiple movies
 from
// the DynamoDB table by title and year.
func (runner PartiQLRunner) GetMovieBatch(ctx context.Context, movies []Movie)
 ([]Movie, error) {
 statementRequests := make([]types.BatchStatementRequest, len(movies))
 for index, movie := range movies {
  params, err := attributevalue.MarshalList([]interface{}{movie.Title,
 movie.Year})
  if err != nil {
```

```
   panic(err)
  }
  statementRequests[index] = types.BatchStatementRequest{
   Statement: aws.String(
    fmt.Sprintf("SELECT * FROM \"%v\" WHERE title=? AND year=?",
 runner.TableName)),
   Parameters: params,
  }
 }

 output, err := runner.DynamoDbClient.BatchExecuteStatement(ctx,
 &dynamodb.BatchExecuteStatementInput{
  Statements: statementRequests,
 })
 var outMovies []Movie
 if err != nil {
  log.Printf("Couldn't get a batch of items with PartiQL. Here's why: %v\n", err)
 } else {
  for _, response := range output.Responses {
   var movie Movie
   err = attributevalue.UnmarshalMap(response.Item, &movie)
   if err != nil {
    log.Printf("Couldn't unmarshal response. Here's why: %v\n", err)
   } else {
    outMovies = append(outMovies, movie)
   }
  }
 }
 return outMovies, err
}
```

Verwenden Sie Stapel von UPDATE-Anweisungen, um Elemente zu aktualisieren.

```
// UpdateMovieBatch runs a batch of PartiQL UPDATE statements to update the
 rating of
// multiple movies that already exist in the DynamoDB table.
func (runner PartiQLRunner) UpdateMovieBatch(ctx context.Context, movies []Movie,
 ratings []float64) error {
 statementRequests := make([]types.BatchStatementRequest, len(movies))
 for index, movie := range movies {
```

```
  params, err := attributevalue.MarshalList([]interface{}{ratings[index],
 movie.Title, movie.Year})
  if err != nil {
   panic(err)
  }
  statementRequests[index] = types.BatchStatementRequest{
   Statement: aws.String(
    fmt.Sprintf("UPDATE \"%v\" SET info.rating=? WHERE title=? AND year=?",
 runner.TableName)),
   Parameters: params,
  }
 }

 _, err := runner.DynamoDbClient.BatchExecuteStatement(ctx,
 &dynamodb.BatchExecuteStatementInput{
  Statements: statementRequests,
 })
 if err != nil {
  log.Printf("Couldn't update the batch of movies. Here's why: %v\n", err)
 }
 return err
}
```

Verwenden Sie Stapel von DELETE-Anweisungen, um Elemente zu löschen.

```
// DeleteMovieBatch runs a batch of PartiQL DELETE statements to remove multiple
 movies
// from the DynamoDB table.
func (runner PartiQLRunner) DeleteMovieBatch(ctx context.Context, movies []Movie)
 error {
 statementRequests := make([]types.BatchStatementRequest, len(movies))
 for index, movie := range movies {
  params, err := attributevalue.MarshalList([]interface{}{movie.Title,
 movie.Year})
  if err != nil {
   panic(err)
  }
  statementRequests[index] = types.BatchStatementRequest{
   Statement: aws.String(
```

```
    fmt.Sprintf("DELETE FROM \"%v\" WHERE title=? AND year=?",
 runner.TableName)),
   Parameters: params,
  }
 }

 _, err := runner.DynamoDbClient.BatchExecuteStatement(ctx,
 &dynamodb.BatchExecuteStatementInput{
  Statements: statementRequests,
 })
 if err != nil {
  log.Printf("Couldn't delete the batch of movies. Here's why: %v\n", err)
 }
 return err
}
```

Definieren Sie eine Movie-Struktur, die in diesem Beispiel verwendet wird.

```
import (
 "archive/zip"
 "bytes"
 "encoding/json"
 "fmt"
 "io"
 "log"
 "net/http"

 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/attributevalue"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb/types"
)

// Movie encapsulates data about a movie. Title and Year are the composite
 primary key
// of the movie in Amazon DynamoDB. Title is the sort key, Year is the partition
 key,
// and Info is additional data.
type Movie struct {
 Title string                 `dynamodbav:"title"`
 Year  int                    `dynamodbav:"year"`
 Info  map[string]interface{} `dynamodbav:"info"`
```

```
}

// GetKey returns the composite primary key of the movie in a format that can be
// sent to DynamoDB.
func (movie Movie) GetKey() map[string]types.AttributeValue {
 title, err := attributevalue.Marshal(movie.Title)
 if err != nil {
  panic(err)
 }
 year, err := attributevalue.Marshal(movie.Year)
 if err != nil {
  panic(err)
 }
 return map[string]types.AttributeValue{"title": title, "year": year}
}

// String returns the title, year, rating, and plot of a movie, formatted for the
 example.
func (movie Movie) String() string {
 return fmt.Sprintf("%v\n\tReleased: %v\n\tRating: %v\n\tPlot: %v\n",
  movie.Title, movie.Year, movie.Info["rating"], movie.Info["plot"])
}
```

- Einzelheiten zur API finden Sie BatchExecuteStatementin der AWS SDK für Go API-Referenz.

JavaScript

SDK für JavaScript (v3)

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

Erstellen Sie mithilfe von PartiQL einen Stapel von Elementen.

```
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";
```

```
import {
  DynamoDBDocumentClient,
  BatchExecuteStatementCommand,
} from "@aws-sdk/lib-dynamodb";

const client = new DynamoDBClient({});
const docClient = DynamoDBDocumentClient.from(client);

export const main = async () => {
  const breakfastFoods = ["Eggs", "Bacon", "Sausage"];
  const command = new BatchExecuteStatementCommand({
    Statements: breakfastFoods.map((food) => ({
      Statement: `INSERT INTO BreakfastFoods value {'Name':?}`,
      Parameters: [food],
    })),
  });

  const response = await docClient.send(command);
  console.log(response);
  return response;
};
```

Rufen Sie mithilfe von PartiQL einen Stapel von Elementen ab.

```
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";

import {
  DynamoDBDocumentClient,
  BatchExecuteStatementCommand,
} from "@aws-sdk/lib-dynamodb";

const client = new DynamoDBClient({});
const docClient = DynamoDBDocumentClient.from(client);

export const main = async () => {
  const command = new BatchExecuteStatementCommand({
    Statements: [
      {
        Statement: "SELECT * FROM PepperMeasurements WHERE Unit=?",
        Parameters: ["Teaspoons"],
        ConsistentRead: true,
      },
```

```
      {
        Statement: "SELECT * FROM PepperMeasurements WHERE Unit=?",
        Parameters: ["Grams"],
        ConsistentRead: true,
      },
    ],
  });

  const response = await docClient.send(command);
  console.log(response);
  return response;
};
```

Aktualisieren Sie mithilfe von PartiQL einen Stapel von Elementen.

```
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";

import {
  DynamoDBDocumentClient,
  BatchExecuteStatementCommand,
} from "@aws-sdk/lib-dynamodb";

const client = new DynamoDBClient({});
const docClient = DynamoDBDocumentClient.from(client);

export const main = async () => {
  const eggUpdates = [
    ["duck", "fried"],
    ["chicken", "omelette"],
  ];
  const command = new BatchExecuteStatementCommand({
    Statements: eggUpdates.map((change) => ({
      Statement: "UPDATE Eggs SET Style=? where Variety=?",
      Parameters: [change[1], change[0]],
    })),
  });

  const response = await docClient.send(command);
  console.log(response);
  return response;
};
```

Löschen Sie mithilfe von PartiQL einen Stapel von Elementen.

```
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";

import {
  DynamoDBDocumentClient,
  BatchExecuteStatementCommand,
} from "@aws-sdk/lib-dynamodb";

const client = new DynamoDBClient({});
const docClient = DynamoDBDocumentClient.from(client);

export const main = async () => {
  const command = new BatchExecuteStatementCommand({
    Statements: [
      {
        Statement: "DELETE FROM Flavors where Name=?",
        Parameters: ["Grape"],
      },
      {
        Statement: "DELETE FROM Flavors where Name=?",
        Parameters: ["Strawberry"],
      },
    ],
  });

  const response = await docClient.send(command);
  console.log(response);
  return response;
};
```

- Einzelheiten zur API finden Sie BatchExecuteStatementin der AWS SDK für JavaScript API-Referenz.

PHP

SDK für PHP

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
    public function getItemByPartiQLBatch(string $tableName, array $keys): Result
    {
        $statements = [];
        foreach ($keys as $key) {
            list($statement, $parameters) = $this-
>buildStatementAndParameters("SELECT", $tableName, $key['Item']);
            $statements[] = [
                'Statement' => "$statement",
                'Parameters' => $parameters,
            ];
        }

        return $this->dynamoDbClient->batchExecuteStatement([
            'Statements' => $statements,
        ]);
    }

    public function insertItemByPartiQLBatch(string $statement, array
 $parameters)
    {
        $this->dynamoDbClient->batchExecuteStatement([
            'Statements' => [
                [
                    'Statement' => "$statement",
                    'Parameters' => $parameters,
                ],
            ],
        ]);
    }

    public function updateItemByPartiQLBatch(string $statement, array
 $parameters)
```

```
    {
        $this->dynamoDbClient->batchExecuteStatement([
            'Statements' => [
                [
                    'Statement' => "$statement",
                    'Parameters' => $parameters,
                ],
            ],
        ]);
    }

    public function deleteItemByPartiQLBatch(string $statement, array
$parameters)
    {
        $this->dynamoDbClient->batchExecuteStatement([
            'Statements' => [
                [
                    'Statement' => "$statement",
                    'Parameters' => $parameters,
                ],
            ],
        ]);
    }
```

- Einzelheiten zur API finden Sie BatchExecuteStatementin der AWS SDK für PHP API-Referenz.

Python

SDK für Python (Boto3)

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
class PartiQLBatchWrapper:
    """
    Encapsulates a DynamoDB resource to run PartiQL statements.
```

```
    """

    def __init__(self, dyn_resource):
        """
        :param dyn_resource: A Boto3 DynamoDB resource.
        """
        self.dyn_resource = dyn_resource


    def run_partiql(self, statements, param_list):
        """
        Runs a PartiQL statement. A Boto3 resource is used even though
        `execute_statement` is called on the underlying `client` object because
the
        resource transforms input and output from plain old Python objects
(POPOs) to
        the DynamoDB format. If you create the client directly, you must do these
        transforms yourself.

        :param statements: The batch of PartiQL statements.
        :param param_list: The batch of PartiQL parameters that are associated
with
                           each statement. This list must be in the same order as
the
                           statements.
        :return: The responses returned from running the statements, if any.
        """
        try:
            output = self.dyn_resource.meta.client.batch_execute_statement(
                Statements=[
                    {"Statement": statement, "Parameters": params}
                    for statement, params in zip(statements, param_list)
                ]
            )
        except ClientError as err:
            if err.response["Error"]["Code"] == "ResourceNotFoundException":
                logger.error(
                    "Couldn't execute batch of PartiQL statements because the
table "
                    "does not exist."
                )
            else:
                logger.error(
```

```
                    "Couldn't execute batch of PartiQL statements. Here's why:
%s: %s",
                    err.response["Error"]["Code"],
                    err.response["Error"]["Message"],
                )
            raise
        else:
            return output
```

- Einzelheiten zur API finden Sie BatchExecuteStatementin AWS SDK for Python (Boto3) API Reference.

Ruby

SDK für Ruby

> **Note**
>
> Es gibt noch mehr dazu. GitHub Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

Lesen Sie einen Stapel von Elementen mithilfe von PartiQL.

```
class DynamoDBPartiQLBatch
  attr_reader :dynamo_resource, :table

  def initialize(table_name)
    client = Aws::DynamoDB::Client.new(region: 'us-east-1')
    @dynamodb = Aws::DynamoDB::Resource.new(client: client)
    @table = @dynamodb.table(table_name)
  end

  # Selects a batch of items from a table using PartiQL
  #
  # @param batch_titles [Array] Collection of movie titles
  # @return [Aws::DynamoDB::Types::BatchExecuteStatementOutput]
  def batch_execute_select(batch_titles)
    request_items = batch_titles.map do |title, year|
```

```
      {
        statement: "SELECT * FROM \"#{@table.name}\" WHERE title=? and year=?",
        parameters: [title, year]
      }
    end
    @dynamodb.client.batch_execute_statement({ statements: request_items })
  end
```

Löschen Sie mithilfe von PartiQL einen Stapel von Elementen.

```
class DynamoDBPartiQLBatch
  attr_reader :dynamo_resource, :table

  def initialize(table_name)
    client = Aws::DynamoDB::Client.new(region: 'us-east-1')
    @dynamodb = Aws::DynamoDB::Resource.new(client: client)
    @table = @dynamodb.table(table_name)
  end

  # Deletes a batch of items from a table using PartiQL
  #
  # @param batch_titles [Array] Collection of movie titles
  # @return [Aws::DynamoDB::Types::BatchExecuteStatementOutput]
  def batch_execute_write(batch_titles)
    request_items = batch_titles.map do |title, year|
      {
        statement: "DELETE FROM \"#{@table.name}\" WHERE title=? and year=?",
        parameters: [title, year]
      }
    end
    @dynamodb.client.batch_execute_statement({ statements: request_items })
  end
```

- Einzelheiten zur API finden Sie BatchExecuteStatementin der AWS SDK für Ruby API-Referenz.

Eine vollständige Liste der AWS SDK-Entwicklerhandbücher und Codebeispiele finden Sie unterDynamoDB mit einem SDK verwenden AWS. Dieses Thema enthält auch Informationen zu den ersten Schritten und Details zu früheren SDK-Versionen.

Verwendung **`BatchGetItem`** mit einem AWS SDK oder CLI

Die folgenden Code-Beispiele zeigen, wie `BatchGetItem` verwendet wird.

.NET

SDK für .NET

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
using System;
using System.Collections.Generic;
using Amazon.DynamoDBv2;
using Amazon.DynamoDBv2.Model;

namespace LowLevelBatchGet
{
    public class LowLevelBatchGet
    {
        private static readonly string _table1Name = "Forum";
        private static readonly string _table2Name = "Thread";

        public static async void
 RetrieveMultipleItemsBatchGet(AmazonDynamoDBClient client)
        {
            var request = new BatchGetItemRequest
            {
                RequestItems = new Dictionary<string, KeysAndAttributes>()
            {
                { _table1Name,
                  new KeysAndAttributes
                  {
                      Keys = new List<Dictionary<string, AttributeValue> >()
                      {
                          new Dictionary<string, AttributeValue>()
                          {
                              { "Name", new AttributeValue {
                            S = "Amazon DynamoDB"
```

```
                        } }
                          },
                          new Dictionary<string, AttributeValue>()
                          {
                              { "Name", new AttributeValue {
                            S = "Amazon S3"
                        } }
                          }
                      }
                  }},
                {
                    _table2Name,
                    new KeysAndAttributes
                    {
                        Keys = new List<Dictionary<string, AttributeValue> >()
                        {
                            new Dictionary<string, AttributeValue>()
                            {
                                { "ForumName", new AttributeValue {
                                      S = "Amazon DynamoDB"
                                  } },
                                { "Subject", new AttributeValue {
                                      S = "DynamoDB Thread 1"
                                  } }
                            },
                            new Dictionary<string, AttributeValue>()
                            {
                                { "ForumName", new AttributeValue {
                                      S = "Amazon DynamoDB"
                                  } },
                                { "Subject", new AttributeValue {
                                      S = "DynamoDB Thread 2"
                                  } }
                            },
                            new Dictionary<string, AttributeValue>()
                            {
                                { "ForumName", new AttributeValue {
                                      S = "Amazon S3"
                                  } },
                                { "Subject", new AttributeValue {
                                      S = "S3 Thread 1"
                                  } }
                            }
                        }
```

```
                }
            }
        }
        };

        BatchGetItemResponse response;
        do
        {
            Console.WriteLine("Making request");
            response = await client.BatchGetItemAsync(request);

            // Check the response.
            var responses = response.Responses; // Attribute list in the
response.

            foreach (var tableResponse in responses)
            {
                var tableResults = tableResponse.Value;
                Console.WriteLine("Items retrieved from table {0}",
tableResponse.Key);
                foreach (var item1 in tableResults)
                {
                    PrintItem(item1);
                }
            }

            // Any unprocessed keys? could happen if you exceed
ProvisionedThroughput or some other error.
            Dictionary<string, KeysAndAttributes> unprocessedKeys =
response.UnprocessedKeys;
            foreach (var unprocessedTableKeys in unprocessedKeys)
            {
                // Print table name.
                Console.WriteLine(unprocessedTableKeys.Key);
                // Print unprocessed primary keys.
                foreach (var key in unprocessedTableKeys.Value.Keys)
                {
                    PrintItem(key);
                }
            }

            request.RequestItems = unprocessedKeys;
        } while (response.UnprocessedKeys.Count > 0);
    }
```

```
        private static void PrintItem(Dictionary<string, AttributeValue>
 attributeList)
        {
            foreach (KeyValuePair<string, AttributeValue> kvp in attributeList)
            {
                string attributeName = kvp.Key;
                AttributeValue value = kvp.Value;

                Console.WriteLine(
                    attributeName + " " +
                    (value.S == null ? "" : "S=[" + value.S + "]") +
                    (value.N == null ? "" : "N=[" + value.N + "]") +
                    (value.SS == null ? "" : "SS=[" + string.Join(",",
 value.SS.ToArray()) + "]") +
                    (value.NS == null ? "" : "NS=[" + string.Join(",",
 value.NS.ToArray()) + "]")
                    );
            }

 Console.WriteLine("************************************************");
        }

        static void Main()
        {
            var client = new AmazonDynamoDBClient();

            RetrieveMultipleItemsBatchGet(client);
        }
    }
}
```

- Einzelheiten zur API finden Sie BatchGetItemin der AWS SDK für .NET API-Referenz.

Bash

AWS CLI mit Bash-Skript

> **Note**
>
> Es gibt noch mehr dazu. GitHub Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
###############################################################################
# function dynamodb_batch_get_item
#
# This function gets a batch of items from a DynamoDB table.
#
# Parameters:
#       -i item  -- Path to json file containing the keys of the items to get.
#
#  Returns:
#       The items as json output.
#  And:
#       0 - If successful.
#       1 - If it fails.
###############################################################################
function dynamodb_batch_get_item() {
  local item response
  local option OPTARG # Required to use getopts command in a function.

  ######################################
  # Function usage explanation
  ######################################
  function usage() {
    echo "function dynamodb_batch_get_item"
    echo "Get a batch of items from a DynamoDB table."
    echo " -i item  -- Path to json file containing the keys of the items to
 get."
    echo ""
  }

  while getopts "i:h" option; do
    case "${option}" in
      i) item="${OPTARG}" ;;
```

```
      h)
        usage
        return 0
        ;;
      \?)
        echo "Invalid parameter"
        usage
        return 1
        ;;
    esac
  done
  export OPTIND=1

  if [[ -z "$item" ]]; then
    errecho "ERROR: You must provide an item with the -i parameter."
    usage
    return 1
  fi

  response=$(aws dynamodb batch-get-item \
    --request-items file://"$item")
  local error_code=${?}

  if [[ $error_code -ne 0 ]]; then
    aws_cli_error_log $error_code
    errecho "ERROR: AWS reports batch-get-item operation failed.$response"
    return 1
  fi

  echo "$response"

  return 0
}
```

Die in diesem Beispiel verwendeten Hilfsfunktionen.

```
###############################################################################
# function errecho
#
# This function outputs everything sent to it to STDERR (standard error output).
###############################################################################
function errecho() {
```

```
  printf "%s\n" "$*" 1>&2
}

###############################################################################
# function aws_cli_error_log()
#
# This function is used to log the error messages from the AWS CLI.
#
# See https://docs.aws.amazon.com/cli/latest/topic/return-codes.html#cli-aws-
help-return-codes.
#
# The function expects the following argument:
#         $1 - The error code returned by the AWS CLI.
#
#  Returns:
#          0: - Success.
#
###############################################################################
function aws_cli_error_log() {
  local err_code=$1
  errecho "Error code : $err_code"
  if [ "$err_code" == 1 ]; then
    errecho "  One or more S3 transfers failed."
  elif [ "$err_code" == 2 ]; then
    errecho "  Command line failed to parse."
  elif [ "$err_code" == 130 ]; then
    errecho "  Process received SIGINT."
  elif [ "$err_code" == 252 ]; then
    errecho "  Command syntax invalid."
  elif [ "$err_code" == 253 ]; then
    errecho "  The system environment or configuration was invalid."
  elif [ "$err_code" == 254 ]; then
    errecho "  The service returned an error."
  elif [ "$err_code" == 255 ]; then
    errecho "  255 is a catch-all error."
  fi

  return 0
}
```

- Einzelheiten zur API finden Sie BatchGetItemin der AWS CLI Befehlsreferenz.

C++

SDK für C++

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
//! Batch get items from different Amazon DynamoDB tables.
/*!
  \sa batchGetItem()
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
 */
bool AwsDoc::DynamoDB::batchGetItem(
        const Aws::Client::ClientConfiguration &clientConfiguration) {
    Aws::DynamoDB::DynamoDBClient dynamoClient(clientConfiguration);

    Aws::DynamoDB::Model::BatchGetItemRequest request;

    // Table1: Forum.
    Aws::String table1Name = "Forum";
    Aws::DynamoDB::Model::KeysAndAttributes table1KeysAndAttributes;

    // Table1: Projection expression.
    table1KeysAndAttributes.SetProjectionExpression("#n, Category, Messages,
 #v");

    // Table1: Expression attribute names.
    Aws::Http::HeaderValueCollection headerValueCollection;
    headerValueCollection.emplace("#n", "Name");
    headerValueCollection.emplace("#v", "Views");
    table1KeysAndAttributes.SetExpressionAttributeNames(headerValueCollection);

    // Table1: Set key name, type, and value to search.
    std::vector<Aws::String> nameValues = {"Amazon DynamoDB", "Amazon S3"};
    for (const Aws::String &name: nameValues) {
        Aws::Map<Aws::String, Aws::DynamoDB::Model::AttributeValue> keys;
        Aws::DynamoDB::Model::AttributeValue key;
        key.SetS(name);
```

```
        keys.emplace("Name", key);
        table1KeysAndAttributes.AddKeys(keys);
    }

    Aws::Map<Aws::String, Aws::DynamoDB::Model::KeysAndAttributes> requestItems;
    requestItems.emplace(table1Name, table1KeysAndAttributes);

    // Table2: ProductCatalog.
    Aws::String table2Name = "ProductCatalog";
    Aws::DynamoDB::Model::KeysAndAttributes table2KeysAndAttributes;
    table2KeysAndAttributes.SetProjectionExpression("Title, Price, Color");

    // Table2: Set key name, type, and value to search.
    std::vector<Aws::String> idValues = {"102", "103", "201"};
    for (const Aws::String &id: idValues) {
        Aws::Map<Aws::String, Aws::DynamoDB::Model::AttributeValue> keys;
        Aws::DynamoDB::Model::AttributeValue key;
        key.SetN(id);
        keys.emplace("Id", key);
        table2KeysAndAttributes.AddKeys(keys);
    }

    requestItems.emplace(table2Name, table2KeysAndAttributes);

    bool result = true;
    do {  // Use a do loop to handle pagination.
        request.SetRequestItems(requestItems);
        const Aws::DynamoDB::Model::BatchGetItemOutcome &outcome =
dynamoClient.BatchGetItem(
                request);

        if (outcome.IsSuccess()) {
            for (const auto &responsesMapEntry:
outcome.GetResult().GetResponses()) {
                Aws::String tableName = responsesMapEntry.first;
                const Aws::Vector<Aws::Map<Aws::String,
Aws::DynamoDB::Model::AttributeValue>> &tableResults = responsesMapEntry.second;
                std::cout << "Retrieved " << tableResults.size()
                          << " responses for table '" << tableName << "'.\n"
                          << std::endl;
                if (tableName == "Forum") {

                    std::cout << "Name | Category | Message | Views" <<
std::endl;
```

```
                    for (const Aws::Map<Aws::String,
 Aws::DynamoDB::Model::AttributeValue> &item: tableResults) {
                        std::cout << item.at("Name").GetS() << " | ";
                        std::cout << item.at("Category").GetS() << " | ";
                        std::cout << (item.count("Message") == 0 ? "" : item.at(
                                "Messages").GetN()) << " | ";
                        std::cout << (item.count("Views") == 0 ? "" : item.at(
                                "Views").GetN()) << std::endl;
                    }
                }
                else {
                    std::cout << "Title | Price | Color" << std::endl;
                    for (const Aws::Map<Aws::String,
 Aws::DynamoDB::Model::AttributeValue> &item: tableResults) {
                        std::cout << item.at("Title").GetS() << " | ";
                        std::cout << (item.count("Price") == 0 ? "" : item.at(
                                "Price").GetN());
                        if (item.count("Color")) {
                            std::cout << " | ";
                            for (const
 std::shared_ptr<Aws::DynamoDB::Model::AttributeValue> &listItem: item.at(
                                    "Color").GetL())
                                std::cout << listItem->GetS() << " ";
                        }
                        std::cout << std::endl;
                    }
                }
                std::cout << std::endl;
            }

            // If necessary, repeat request for remaining items.
            requestItems = outcome.GetResult().GetUnprocessedKeys();
        }
        else {
            std::cerr << "Batch get item failed: " <<
 outcome.GetError().GetMessage()
                      << std::endl;
            result = false;
            break;
        }
    } while (!requestItems.empty());

    return result;
}
```

- Einzelheiten zur API finden Sie BatchGetItemin der AWS SDK für C++ API-Referenz.

CLI

AWS CLI

So rufen Sie mehrere Elemente aus einer Tabelle ab

Im folgenden Beispiel für `batch-get-items` werden mithilfe eines Batches von drei `GetItem`-Anfragen mehrere Elemente aus der `MusicCollection`-Tabelle gelesen und die Anzahl der durch den Vorgang verbrauchten Lesekapazitätseinheiten wird abgefragt. Der Befehl gibt nur das `AlbumTitle`-Attribut zurück.

```
aws dynamodb batch-get-item \
    --request-items file://request-items.json \
    --return-consumed-capacity TOTAL
```

Inhalt von `request-items.json`:

```
{
    "MusicCollection": {
        "Keys": [
            {
                "Artist": {"S": "No One You Know"},
                "SongTitle": {"S": "Call Me Today"}
            },
            {
                "Artist": {"S": "Acme Band"},
                "SongTitle": {"S": "Happy Day"}
            },
            {
                "Artist": {"S": "No One You Know"},
                "SongTitle": {"S": "Scared of My Shadow"}
            }
        ],
        "ProjectionExpression":"AlbumTitle"
    }
}
```

Ausgabe:

```
{
    "Responses": {
        "MusicCollection": [
            {
                "AlbumTitle": {
                    "S": "Somewhat Famous"
                }
            },
            {
                "AlbumTitle": {
                    "S": "Blue Sky Blues"
                }
            },
            {
                "AlbumTitle": {
                    "S": "Louder Than Ever"
                }
            }
        ]
    },
    "UnprocessedKeys": {},
    "ConsumedCapacity": [
        {
            "TableName": "MusicCollection",
            "CapacityUnits": 1.5
        }
    ]
}
```

Weitere Informationen finden Sie unter Batch-Vorgänge im Entwicklerhandbuch zu Amazon DynamoDB.

- Einzelheiten zur API finden Sie BatchGetItemin der AWS CLI Befehlsreferenz.

Java

SDK für Java 2.x

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

Zeigt, wie Batch-Elemente mithilfe des Service-Clients abgerufen werden.

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.AttributeValue;
import software.amazon.awssdk.services.dynamodb.model.BatchGetItemRequest;
import software.amazon.awssdk.services.dynamodb.model.BatchGetItemResponse;
import software.amazon.awssdk.services.dynamodb.model.KeysAndAttributes;
import java.util.HashMap;
import java.util.List;
import java.util.Map;

/**
 * Before running this Java V2 code example, set up your development environment,
 including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 */
public class BatchReadItems {
    public static void main(String[] args){
        final String usage = """

                Usage:
                    <tableName>

                Where:
                    tableName - The Amazon DynamoDB table (for example, Music).\s
                """;

        String tableName = "Music";
```

```
        Region region = Region.US_EAST_1;
        DynamoDbClient dynamoDbClient = DynamoDbClient.builder()
            .region(region)
            .build();

        getBatchItems(dynamoDbClient, tableName);
    }

    public static void getBatchItems(DynamoDbClient dynamoDbClient, String
tableName) {
        // Define the primary key values for the items you want to retrieve.
        Map<String, AttributeValue> key1 = new HashMap<>();
        key1.put("Artist", AttributeValue.builder().s("Artist1").build());

        Map<String, AttributeValue> key2 = new HashMap<>();
        key2.put("Artist", AttributeValue.builder().s("Artist2").build());

        // Construct the batchGetItem request.
        Map<String, KeysAndAttributes> requestItems = new HashMap<>();
        requestItems.put(tableName, KeysAndAttributes.builder()
            .keys(List.of(key1, key2))
            .projectionExpression("Artist, SongTitle")
            .build());

        BatchGetItemRequest batchGetItemRequest = BatchGetItemRequest.builder()
            .requestItems(requestItems)
            .build();

        // Make the batchGetItem request.
        BatchGetItemResponse batchGetItemResponse =
dynamoDbClient.batchGetItem(batchGetItemRequest);

        // Extract and print the retrieved items.
        Map<String, List<Map<String, AttributeValue>>> responses =
batchGetItemResponse.responses();
        if (responses.containsKey(tableName)) {
            List<Map<String, AttributeValue>> musicItems =
responses.get(tableName);
            for (Map<String, AttributeValue> item : musicItems) {
                System.out.println("Artist: " + item.get("Artist").s() +
                    ", SongTitle: " + item.get("SongTitle").s());
            }
        } else {
            System.out.println("No items retrieved.");
```

```
        }
    }
}
```

Zeigt, wie Batch-Elemente mithilfe des Service-Clients und eines Paginators abgerufen werden.

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.AttributeValue;
import software.amazon.awssdk.services.dynamodb.model.BatchGetItemRequest;
import software.amazon.awssdk.services.dynamodb.model.KeysAndAttributes;
import java.util.Collections;
import java.util.HashMap;
import java.util.List;
import java.util.Map;

public class BatchGetItemsPaginator {

    public static void main(String[] args){
        final String usage = """

                Usage:
                    <tableName>

                Where:
                    tableName - The Amazon DynamoDB table (for example, Music).\s
                """;

        String tableName = "Music";
        Region region = Region.US_EAST_1;
        DynamoDbClient dynamoDbClient = DynamoDbClient.builder()
            .region(region)
            .build();

        getBatchItemsPaginator(dynamoDbClient, tableName) ;
    }

    public static void getBatchItemsPaginator(DynamoDbClient dynamoDbClient,
 String tableName) {
        // Define the primary key values for the items you want to retrieve.
        Map<String, AttributeValue> key1 = new HashMap<>();
```

```
        key1.put("Artist", AttributeValue.builder().s("Artist1").build());

        Map<String, AttributeValue> key2 = new HashMap<>();
        key2.put("Artist", AttributeValue.builder().s("Artist2").build());

        // Construct the batchGetItem request.
        Map<String, KeysAndAttributes> requestItems = new HashMap<>();
        requestItems.put(tableName, KeysAndAttributes.builder()
            .keys(List.of(key1, key2))
            .projectionExpression("Artist, SongTitle")
            .build());

        BatchGetItemRequest batchGetItemRequest = BatchGetItemRequest.builder()
            .requestItems(requestItems)
            .build();

        // Use batchGetItemPaginator for paginated requests.
        dynamoDbClient.batchGetItemPaginator(batchGetItemRequest).stream()
            .flatMap(response -> response.responses().getOrDefault(tableName,
 Collections.emptyList()).stream())
            .forEach(item -> {
                System.out.println("Artist: " + item.get("Artist").s() +
                    ", SongTitle: " + item.get("SongTitle").s());
            });
    }
}
```

- Einzelheiten zur API finden Sie BatchGetItemin der AWS SDK for Java 2.x API-Referenz.

JavaScript

SDK für JavaScript (v3)

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

In diesem Beispiel wird der Dokument-Client verwendet, um die Arbeit mit Elementen in DynamoDB zu vereinfachen. Einzelheiten zur API finden Sie unter BatchGet.

```
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";
import { BatchGetCommand, DynamoDBDocumentClient } from "@aws-sdk/lib-dynamodb";

const client = new DynamoDBClient({});
const docClient = DynamoDBDocumentClient.from(client);

export const main = async () => {
  const command = new BatchGetCommand({
    // Each key in this object is the name of a table. This example refers
    // to a Books table.
    RequestItems: {
      Books: {
        // Each entry in Keys is an object that specifies a primary key.
        Keys: [
          {
            Title: "How to AWS",
          },
          {
            Title: "DynamoDB for DBAs",
          },
        ],
        // Only return the "Title" and "PageCount" attributes.
        ProjectionExpression: "Title, PageCount",
      },
    },
  });

  const response = await docClient.send(command);
  console.log(response.Responses.Books);
  return response;
};
```

- Weitere Informationen finden Sie im AWS SDK für JavaScript -Entwicklerhandbuch.
- Einzelheiten zur API finden Sie BatchGetItemin der AWS SDK für JavaScript API-Referenz.

SDK für JavaScript (v2)

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create DynamoDB service object
var ddb = new AWS.DynamoDB({ apiVersion: "2012-08-10" });

var params = {
  RequestItems: {
    TABLE_NAME: {
      Keys: [
        { KEY_NAME: { N: "KEY_VALUE_1" } },
        { KEY_NAME: { N: "KEY_VALUE_2" } },
        { KEY_NAME: { N: "KEY_VALUE_3" } },
      ],
      ProjectionExpression: "KEY_NAME, ATTRIBUTE",
    },
  },
};

ddb.batchGetItem(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    data.Responses.TABLE_NAME.forEach(function (element, index, array) {
      console.log(element);
    });
  }
});
```

- Weitere Informationen finden Sie im AWS SDK für JavaScript -Entwicklerhandbuch.
- Einzelheiten zur API finden Sie BatchGetItemin der AWS SDK für JavaScript API-Referenz.

PowerShell

Tools für PowerShell V4

Beispiel 1: Ruft das Element mit dem Namen SongTitle „Somewhere Down The Road“ aus den DynamoDB-Tabellen „Music“ und „Songs“ ab.

```
$key = @{
    SongTitle = 'Somewhere Down The Road'
    Artist = 'No One You Know'
} | ConvertTo-DDBItem

$keysAndAttributes = New-Object Amazon.DynamoDBv2.Model.KeysAndAttributes
$list = New-Object
 'System.Collections.Generic.List[System.Collections.Generic.Dictionary[String,
 Amazon.DynamoDBv2.Model.AttributeValue]]'
$list.Add($key)
$keysAndAttributes.Keys = $list

$requestItem = @{
    'Music' = [Amazon.DynamoDBv2.Model.KeysAndAttributes]$keysAndAttributes
    'Songs' = [Amazon.DynamoDBv2.Model.KeysAndAttributes]$keysAndAttributes
}

$batchItems = Get-DDBBatchItem -RequestItem $requestItem
$batchItems.GetEnumerator() | ForEach-Object {$PSItem.Value} | ConvertFrom-
DDBItem
```

Ausgabe:

```
Name                           Value
----                           -----
Artist                         No One You Know
SongTitle                      Somewhere Down The Road
AlbumTitle                     Somewhat Famous
CriticRating                   10
Genre                          Country
Price                          1.94
Artist                         No One You Know
SongTitle                      Somewhere Down The Road
AlbumTitle                     Somewhat Famous
CriticRating                   10
Genre                          Country
Price                          1.94
```

- Einzelheiten zur API finden Sie unter AWS -Tools für PowerShell Cmdlet-Referenz (BatchGetItemV4).

Tools für V5 PowerShell

Beispiel 1: Ruft das Element mit dem Namen SongTitle „Somewhere Down The Road“ aus den DynamoDB-Tabellen „Music“ und „Songs“ ab.

```
$key = @{
    SongTitle = 'Somewhere Down The Road'
    Artist = 'No One You Know'
} | ConvertTo-DDBItem

$keysAndAttributes = New-Object Amazon.DynamoDBv2.Model.KeysAndAttributes
$list = New-Object
 'System.Collections.Generic.List[System.Collections.Generic.Dictionary[String,
 Amazon.DynamoDBv2.Model.AttributeValue]]'
$list.Add($key)
$keysAndAttributes.Keys = $list

$requestItem = @{
    'Music' = [Amazon.DynamoDBv2.Model.KeysAndAttributes]$keysAndAttributes
    'Songs' = [Amazon.DynamoDBv2.Model.KeysAndAttributes]$keysAndAttributes
}

$batchItems = Get-DDBBatchItem -RequestItem $requestItem
$batchItems.GetEnumerator() | ForEach-Object {$PSItem.Value} | ConvertFrom-
DDBItem
```

Ausgabe:

```
Name                           Value
----                           -----
Artist                         No One You Know
SongTitle                      Somewhere Down The Road
AlbumTitle                     Somewhat Famous
CriticRating                   10
Genre                          Country
Price                          1.94
Artist                         No One You Know
SongTitle                      Somewhere Down The Road
AlbumTitle                     Somewhat Famous
CriticRating                   10
Genre                          Country
Price                          1.94
```

- Einzelheiten zur API finden Sie unter AWS -Tools für PowerShell Cmdlet-Referenz (BatchGetItemV5).

Python

SDK für Python (Boto3)

> **Note**
>
> Es gibt noch mehr dazu. GitHub Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
import decimal
import json
import logging
import os
import pprint
import time
import boto3
from botocore.exceptions import ClientError

logger = logging.getLogger(__name__)
dynamodb = boto3.resource("dynamodb")

MAX_GET_SIZE = 100  # Amazon DynamoDB rejects a get batch larger than 100 items.


def do_batch_get(batch_keys):
    """
    Gets a batch of items from Amazon DynamoDB. Batches can contain keys from
    more than one table.

    When Amazon DynamoDB cannot process all items in a batch, a set of
 unprocessed
    keys is returned. This function uses an exponential backoff algorithm to
 retry
    getting the unprocessed keys until all are retrieved or the specified
    number of tries is reached.
```

```
    :param batch_keys: The set of keys to retrieve. A batch can contain at most
100
                       keys. Otherwise, Amazon DynamoDB returns an error.
    :return: The dictionary of retrieved items grouped under their respective
             table names.
    """
    tries = 0
    max_tries = 5
    sleepy_time = 1  # Start with 1 second of sleep, then exponentially increase.
    retrieved = {key: [] for key in batch_keys}
    while tries < max_tries:
        response = dynamodb.batch_get_item(RequestItems=batch_keys)
        # Collect any retrieved items and retry unprocessed keys.
        for key in response.get("Responses", []):
            retrieved[key] += response["Responses"][key]
        unprocessed = response["UnprocessedKeys"]
        if len(unprocessed) > 0:
            batch_keys = unprocessed
            unprocessed_count = sum(
                [len(batch_key["Keys"]) for batch_key in batch_keys.values()]
            )
            logger.info(
                "%s unprocessed keys returned. Sleep, then retry.",
unprocessed_count
            )
            tries += 1
            if tries < max_tries:
                logger.info("Sleeping for %s seconds.", sleepy_time)
                time.sleep(sleepy_time)
                sleepy_time = min(sleepy_time * 2, 32)
        else:
            break

    return retrieved
```

- Einzelheiten zur API finden Sie BatchGetItemin AWS SDK for Python (Boto3) API Reference.

Swift

SDK für Swift

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
import AWSDynamoDB


    /// Gets an array of `Movie` objects describing all the movies in the
    /// specified list. Any movies that aren't found in the list have no
    /// corresponding entry in the resulting array.
    ///
    /// - Parameters
    ///     - keys: An array of tuples, each of which specifies the title and
    ///       release year of a movie to fetch from the table.
    ///
    /// - Returns:
    ///     - An array of `Movie` objects describing each match found in the
    ///     table.
    ///
    /// - Throws:
    ///     - `MovieError.ClientUninitialized` if the DynamoDB client has not
    ///     been initialized.
    ///     - DynamoDB errors are thrown without change.
    func batchGet(keys: [(title: String, year: Int)]) async throws -> [Movie] {
        do {
            guard let client = self.ddbClient else {
                throw MovieError.ClientUninitialized
            }

            var movieList: [Movie] = []
            var keyItems: [[Swift.String: DynamoDBClientTypes.AttributeValue]] =
 []

            // Convert the list of keys into the form used by DynamoDB.

            for key in keys {
```

```
                let item: [Swift.String: DynamoDBClientTypes.AttributeValue] = [
                    "title": .s(key.title),
                    "year": .n(String(key.year))
                ]
                keyItems.append(item)
            }

            // Create the input record for `batchGetItem()`. The list of
requested
            // items is in the `requestItems` property. This array contains one
            // entry for each table from which items are to be fetched. In this
            // example, there's only one table containing the movie data.
            //
            // If we wanted this program to also support searching for matches
            // in a table of book data, we could add a second `requestItem`
            // mapping the name of the book table to the list of items we want to
            // find in it.
            let input = BatchGetItemInput(
                requestItems: [
                    self.tableName: .init(
                        consistentRead: true,
                        keys: keyItems
                    )
                ]
            )

            // Fetch the matching movies from the table.

            let output = try await client.batchGetItem(input: input)

            // Get the set of responses. If there aren't any, return the empty
            // movie list.

            guard let responses = output.responses else {
                return movieList
            }

            // Get the list of matching items for the table with the name
            // `tableName`.

            guard let responseList = responses[self.tableName] else {
                return movieList
            }
```

```
            // Create `Movie` items for each of the matching movies in the table
            // and add them to the `MovieList` array.

            for response in responseList {
                try movieList.append(Movie(withItem: response))
            }

            return movieList
        } catch {
            print("ERROR: batchGet", dump(error))
            throw error
        }
    }
```

- Einzelheiten zur API finden Sie BatchGetItemin der API-Referenz zum AWS SDK für Swift.

Eine vollständige Liste der AWS SDK-Entwicklerhandbücher und Codebeispiele finden Sie unterDynamoDB mit einem SDK verwenden AWS. Dieses Thema enthält auch Informationen zu den ersten Schritten und Details zu früheren SDK-Versionen.

Verwendung **`BatchWriteItem`** mit einem AWS SDK oder CLI

Die folgenden Code-Beispiele zeigen, wie `BatchWriteItem` verwendet wird.

Beispiele für Aktionen sind Codeauszüge aus größeren Programmen und müssen im Kontext ausgeführt werden. Im folgenden Codebeispiel können Sie diese Aktion im Kontext sehen:

- Kennenlernen der Grundlagen

.NET

SDK für .NET (v4)

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

Schreibt einen Stapel von Elementen in die Filmtabelle.

```
    /// <summary>
    /// Loads the contents of a JSON file into a list of movies to be
    /// added to the DynamoDB table.
    /// </summary>
    /// <param name="movieFileName">The name of the JSON file.</param>
    /// <returns>A generic list of movie objects.</returns>
    public List<Movie> ImportMovies(string movieFileName)
    {
        var moviesList = new List<Movie>();
        if (!File.Exists(movieFileName))
        {
            return moviesList;
        }

        using var sr = new StreamReader(movieFileName);
        string json = sr.ReadToEnd();
        var allMovies = JsonSerializer.Deserialize<List<Movie>>(
            json,
            new JsonSerializerOptions
            {
                PropertyNameCaseInsensitive = true
            });

        // Now return the first 250 entries.
        if (allMovies != null && allMovies.Any())
        {
            moviesList = allMovies.GetRange(0, 250);
        }
        return moviesList;
    }

    /// <summary>
    /// Writes 250 items to the movie table.
    /// </summary>
    /// <param name="movieFileName">A string containing the full path to
    /// the JSON file containing movie data.</param>
    /// <param name="tableName">The name of the table to write items to.</param>
    /// <returns>A long integer value representing the number of movies
    /// imported from the JSON file.</returns>
    public async Task<long> BatchWriteItemsAsync(
        string movieFileName, string tableName)
    {
```

```
        try
        {
            var movies = ImportMovies(movieFileName);
            if (!movies.Any())
            {
                Console.WriteLine("Couldn't find the JSON file with movie
data.");
                return 0;
            }

            var context = new DynamoDBContextBuilder()
                // Optional call to provide a specific instance of
IAmazonDynamoDB
                .WithDynamoDBClient(() => _amazonDynamoDB)
                .Build();

            var movieBatch = context.CreateBatchWrite<Movie>(
                new BatchWriteConfig()
                {
                    OverrideTableName = tableName
                });
            movieBatch.AddPutItems(movies);

            Console.WriteLine("Adding imported movies to the table.");
            await movieBatch.ExecuteAsync();

            return movies.Count;
        }
        catch (ResourceNotFoundException ex)
        {
            Console.WriteLine($"Table was not found during batch write operation.
{ex.Message}");
            throw;
        }
        catch (AmazonDynamoDBException ex)
        {
            Console.WriteLine($"An Amazon DynamoDB error occurred during batch
write operation. {ex.Message}");
            throw;
        }
        catch (Exception ex)
        {
            Console.WriteLine($"An error occurred during batch write operation.
{ex.Message}");
```

```
            throw;
        }
    }
```

- Einzelheiten zur API finden Sie BatchWriteItemin der AWS SDK für .NET API-Referenz.

Bash

AWS CLI mit Bash-Skript

> **Note**
>
> Es gibt noch mehr dazu. GitHub Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
###############################################################################
# function dynamodb_batch_write_item
#
# This function writes a batch of items into a DynamoDB table.
#
# Parameters:
#       -i item  -- Path to json file containing the items to write.
#
#  Returns:
#       0 - If successful.
#       1 - If it fails.
###############################################################################
function dynamodb_batch_write_item() {
  local item response
  local option OPTARG # Required to use getopts command in a function.

  ######################################
  # Function usage explanation
  ######################################
  function usage() {
    echo "function dynamodb_batch_write_item"
    echo "Write a batch of items into a DynamoDB table."
    echo " -i item  -- Path to json file containing the items to write."
    echo ""
```

```
  }
  while getopts "i:h" option; do
    case "${option}" in
      i) item="${OPTARG}" ;;
      h)
        usage
        return 0
        ;;
      \?)
        echo "Invalid parameter"
        usage
        return 1
        ;;
    esac
  done
  export OPTIND=1

  if [[ -z "$item" ]]; then
    errecho "ERROR: You must provide an item with the -i parameter."
    usage
    return 1
  fi

  iecho "Parameters:\n"
  iecho "    table_name:   $table_name"
  iecho "    item:   $item"
  iecho ""

  response=$(aws dynamodb batch-write-item \
    --request-items file://"$item")

  local error_code=${?}

  if [[ $error_code -ne 0 ]]; then
    aws_cli_error_log $error_code
    errecho "ERROR: AWS reports batch-write-item operation failed.$response"
    return 1
  fi

  return 0
}
```

Die in diesem Beispiel verwendeten Hilfsfunktionen.

```
###############################################################################
# function iecho
#
# This function enables the script to display the specified text only if
# the global variable $VERBOSE is set to true.
###############################################################################
function iecho() {
  if [[ $VERBOSE == true ]]; then
    echo "$@"
  fi
}

###############################################################################
# function errecho
#
# This function outputs everything sent to it to STDERR (standard error output).
###############################################################################
function errecho() {
  printf "%s\n" "$*" 1>&2
}

###############################################################################
# function aws_cli_error_log()
#
# This function is used to log the error messages from the AWS CLI.
#
# See https://docs.aws.amazon.com/cli/latest/topic/return-codes.html#cli-aws-
help-return-codes.
#
# The function expects the following argument:
#         $1 - The error code returned by the AWS CLI.
#
#  Returns:
#          0: - Success.
#
###############################################################################
function aws_cli_error_log() {
  local err_code=$1
  errecho "Error code : $err_code"
  if [ "$err_code" == 1 ]; then
    errecho "  One or more S3 transfers failed."
  elif [ "$err_code" == 2 ]; then
```

```
    errecho "  Command line failed to parse."
  elif [ "$err_code" == 130 ]; then
    errecho "  Process received SIGINT."
  elif [ "$err_code" == 252 ]; then
    errecho "  Command syntax invalid."
  elif [ "$err_code" == 253 ]; then
    errecho "  The system environment or configuration was invalid."
  elif [ "$err_code" == 254 ]; then
    errecho "  The service returned an error."
  elif [ "$err_code" == 255 ]; then
    errecho "  255 is a catch-all error."
  fi

  return 0
}
```

- Einzelheiten zur API finden Sie BatchWriteItemin der AWS CLI Befehlsreferenz.

C++

SDK für C++

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
//! Batch write items from a JSON file.
/*!
  \sa batchWriteItem()
  \param jsonFilePath: JSON file path.
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
 */

/*
 * The input for this routine is a JSON file that you can download from the
 following URL:
 * https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/
SampleData.html.
```

```
 *
 * The JSON data uses the BatchWriteItem API request syntax. The JSON strings are
 * converted to AttributeValue objects. These AttributeValue objects will then
 generate
 * JSON strings when constructing the BatchWriteItem request, essentially
 outputting
 * their input.
 *
 * This is perhaps an artificial example, but it demonstrates the APIs.
 */

bool AwsDoc::DynamoDB::batchWriteItem(const Aws::String &jsonFilePath,
                                      const Aws::Client::ClientConfiguration
 &clientConfiguration) {
    std::ifstream fileStream(jsonFilePath);

    if (!fileStream) {
        std::cerr << "Error: could not open file '" << jsonFilePath << "'."
                  << std::endl;
    }

    std::stringstream stringStream;
    stringStream << fileStream.rdbuf();
    Aws::Utils::Json::JsonValue jsonValue(stringStream);

    Aws::DynamoDB::Model::BatchWriteItemRequest batchWriteItemRequest;
    Aws::Map<Aws::String, Aws::Utils::Json::JsonView> level1Map =
 jsonValue.View().GetAllObjects();
    for (const auto &level1Entry: level1Map) {
        const Aws::Utils::Json::JsonView &entriesView = level1Entry.second;
        const Aws::String &tableName = level1Entry.first;
        // The JSON entries at this level are as follows:
        //  key - table name
        //  value - list of request objects
        if (!entriesView.IsListType()) {
            std::cerr << "Error: JSON file entry '"
                      << tableName << "' is not a list." << std::endl;
            continue;
        }

        Aws::Utils::Array<Aws::Utils::Json::JsonView> entries =
 entriesView.AsArray();

        Aws::Vector<Aws::DynamoDB::Model::WriteRequest> writeRequests;
```

```
        if (AwsDoc::DynamoDB::addWriteRequests(tableName, entries,
                                               writeRequests)) {
            batchWriteItemRequest.AddRequestItems(tableName, writeRequests);
        }
    }

    Aws::DynamoDB::DynamoDBClient dynamoClient(clientConfiguration);

    Aws::DynamoDB::Model::BatchWriteItemOutcome outcome =
 dynamoClient.BatchWriteItem(
            batchWriteItemRequest);

    if (outcome.IsSuccess()) {
        std::cout << "DynamoDB::BatchWriteItem was successful." << std::endl;
    }
    else {
        std::cerr << "Error with DynamoDB::BatchWriteItem. "
                  << outcome.GetError().GetMessage()
                  << std::endl;
        return false;
    }

    return outcome.IsSuccess();
}

//! Convert requests in JSON format to a vector of WriteRequest objects.
/*!
  \sa addWriteRequests()
  \param tableName: Name of the table for the write operations.
  \param requestsJson: Request data in JSON format.
  \param writeRequests: Vector to receive the WriteRequest objects.
  \return bool: Function succeeded.
 */
bool AwsDoc::DynamoDB::addWriteRequests(const Aws::String &tableName,
                                        const
 Aws::Utils::Array<Aws::Utils::Json::JsonView> &requestsJson,

 Aws::Vector<Aws::DynamoDB::Model::WriteRequest> &writeRequests) {
    for (size_t i = 0; i < requestsJson.GetLength(); ++i) {
        const Aws::Utils::Json::JsonView &requestsEntry = requestsJson[i];
        if (!requestsEntry.IsObject()) {
            std::cerr << "Error: incorrect requestsEntry type "
                      << requestsEntry.WriteReadable() << std::endl;
            return false;
```

```
        }

        Aws::Map<Aws::String, Aws::Utils::Json::JsonView> requestsMap =
 requestsEntry.GetAllObjects();

        for (const auto &request: requestsMap) {
            const Aws::String &requestType = request.first;
            const Aws::Utils::Json::JsonView &requestJsonView = request.second;

            if (requestType == "PutRequest") {
                if (!requestJsonView.ValueExists("Item")) {
                    std::cerr << "Error: item key missing for requests "
                              << requestJsonView.WriteReadable() << std::endl;
                    return false;
                }
                Aws::Map<Aws::String, Aws::DynamoDB::Model::AttributeValue>
 attributes;
                if (!getAttributeObjectsMap(requestJsonView.GetObject("Item"),
                                            attributes)) {
                    std::cerr << "Error getting attributes "
                              << requestJsonView.WriteReadable() << std::endl;
                    return false;
                }

                Aws::DynamoDB::Model::PutRequest putRequest;
                putRequest.SetItem(attributes);
                writeRequests.push_back(
                        Aws::DynamoDB::Model::WriteRequest().WithPutRequest(
                                putRequest));
            }
            else {
                std::cerr << "Error: unimplemented request type '" << requestType
                          << "'." << std::endl;
            }
        }
    }

    return true;
}

//! Generate a map of AttributeValue objects from JSON records.
/*!
  \sa getAttributeObjectsMap()
  \param jsonView: JSONView of attribute records.
```

```
  \param writeRequests: Map to receive the AttributeValue objects.
  \return bool: Function succeeded.
 */
bool
AwsDoc::DynamoDB::getAttributeObjectsMap(const Aws::Utils::Json::JsonView
 &jsonView,
                                         Aws::Map<Aws::String,
 Aws::DynamoDB::Model::AttributeValue> &attributes) {
    Aws::Map<Aws::String, Aws::Utils::Json::JsonView> objectsMap =
 jsonView.GetAllObjects();
    for (const auto &entry: objectsMap) {
        const Aws::String &attributeKey = entry.first;
        const Aws::Utils::Json::JsonView &attributeJsonView = entry.second;

        if (!attributeJsonView.IsObject()) {
            std::cerr << "Error: attribute not an object "
                      << attributeJsonView.WriteReadable() << std::endl;
            return false;
        }

        attributes.emplace(attributeKey,

 Aws::DynamoDB::Model::AttributeValue(attributeJsonView));
    }

    return true;
}
```

- Einzelheiten zur API finden Sie BatchWriteItemin der AWS SDK für C++ API-Referenz.

CLI

AWS CLI

So fügen Sie mehrere Elemente einer Tabelle hinzu

Im folgenden Beispiel für `batch-write-item` werden der `MusicCollection`-Tabelle drei neue Elemente hinzugefügt, wobei ein Batch von drei `PutItem`-Anforderungen verwendet wird. Außerdem werden Informationen zur Anzahl der Schreibkapazitätseinheiten abgefragt, die von dem Vorgang verbraucht werden, sowie zu allen Elementsammlungen, die durch den Vorgang geändert werden.

```
aws dynamodb batch-write-item \
    --request-items file://request-items.json \
    --return-consumed-capacity INDEXES \
    --return-item-collection-metrics SIZE
```

Inhalt von `request-items.json`:

```
{
    "MusicCollection": [
        {
            "PutRequest": {
                "Item": {
                    "Artist": {"S": "No One You Know"},
                    "SongTitle": {"S": "Call Me Today"},
                    "AlbumTitle": {"S": "Somewhat Famous"}
                }
            }
        },
        {
            "PutRequest": {
                "Item": {
                    "Artist": {"S": "Acme Band"},
                    "SongTitle": {"S": "Happy Day"},
                    "AlbumTitle": {"S": "Songs About Life"}
                }
            }
        },
        {
            "PutRequest": {
                "Item": {
                    "Artist": {"S": "No One You Know"},
                    "SongTitle": {"S": "Scared of My Shadow"},
                    "AlbumTitle": {"S": "Blue Sky Blues"}
                }
            }
        }
    ]
}
```

Ausgabe:

```
{
```

```
    "UnprocessedItems": {},
    "ItemCollectionMetrics": {
        "MusicCollection": [
            {
                "ItemCollectionKey": {
                    "Artist": {
                        "S": "No One You Know"
                    }
                },
                "SizeEstimateRangeGB": [
                    0.0,
                    1.0
                ]
            },
            {
                "ItemCollectionKey": {
                    "Artist": {
                        "S": "Acme Band"
                    }
                },
                "SizeEstimateRangeGB": [
                    0.0,
                    1.0
                ]
            }
        ]
    },
    "ConsumedCapacity": [
        {
            "TableName": "MusicCollection",
            "CapacityUnits": 6.0,
            "Table": {
                "CapacityUnits": 3.0
            },
            "LocalSecondaryIndexes": {
                "AlbumTitleIndex": {
                    "CapacityUnits": 3.0
                }
            }
        }
    ]
}
```

Weitere Informationen finden Sie unter Batch-Vorgänge im Entwicklerhandbuch zu Amazon DynamoDB.

- Einzelheiten zur API finden Sie BatchWriteItemin der AWS CLI Befehlsreferenz.

Go

SDK für Go V2

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
import (
 "context"
 "errors"
 "log"
 "time"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/attributevalue"
 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/expression"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb/types"
)

// TableBasics encapsulates the Amazon DynamoDB service actions used in the
 examples.
// It contains a DynamoDB service client that is used to act on the specified
 table.
type TableBasics struct {
 DynamoDbClient *dynamodb.Client
 TableName      string
}



// AddMovieBatch adds a slice of movies to the DynamoDB table. The function sends
// batches of 25 movies to DynamoDB until all movies are added or it reaches the
```

```
// specified maximum.
func (basics TableBasics) AddMovieBatch(ctx context.Context, movies []Movie,
 maxMovies int) (int, error) {
 var err error
 var item map[string]types.AttributeValue
 written := 0
 batchSize := 25 // DynamoDB allows a maximum batch size of 25 items.
 start := 0
 end := start + batchSize
 for start < maxMovies && start < len(movies) {
  var writeReqs []types.WriteRequest
  if end > len(movies) {
   end = len(movies)
  }
  for _, movie := range movies[start:end] {
   item, err = attributevalue.MarshalMap(movie)
   if err != nil {
    log.Printf("Couldn't marshal movie %v for batch writing. Here's why: %v\n",
 movie.Title, err)
   } else {
    writeReqs = append(
     writeReqs,
     types.WriteRequest{PutRequest: &types.PutRequest{Item: item}},
    )
   }
  }
  _, err = basics.DynamoDbClient.BatchWriteItem(ctx,
 &dynamodb.BatchWriteItemInput{
   RequestItems: map[string][]types.WriteRequest{basics.TableName: writeReqs}})
  if err != nil {
   log.Printf("Couldn't add a batch of movies to %v. Here's why: %v\n",
 basics.TableName, err)
  } else {
   written += len(writeReqs)
  }
  start = end
  end += batchSize
 }

 return written, err
}
```

Definieren Sie eine Movie-Struktur, die in diesem Beispiel verwendet wird.

```
import (
 "archive/zip"
 "bytes"
 "encoding/json"
 "fmt"
 "io"
 "log"
 "net/http"

 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/attributevalue"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb/types"
)

// Movie encapsulates data about a movie. Title and Year are the composite
 primary key
// of the movie in Amazon DynamoDB. Title is the sort key, Year is the partition
 key,
// and Info is additional data.
type Movie struct {
 Title string                 `dynamodbav:"title"`
 Year  int                    `dynamodbav:"year"`
 Info  map[string]interface{} `dynamodbav:"info"`
}

// GetKey returns the composite primary key of the movie in a format that can be
// sent to DynamoDB.
func (movie Movie) GetKey() map[string]types.AttributeValue {
 title, err := attributevalue.Marshal(movie.Title)
 if err != nil {
  panic(err)
 }
 year, err := attributevalue.Marshal(movie.Year)
 if err != nil {
  panic(err)
 }
 return map[string]types.AttributeValue{"title": title, "year": year}
}

// String returns the title, year, rating, and plot of a movie, formatted for the
 example.
func (movie Movie) String() string {
```

```
 return fmt.Sprintf("%v\n\tReleased: %v\n\tRating: %v\n\tPlot: %v\n",
  movie.Title, movie.Year, movie.Info["rating"], movie.Info["plot"])
}
```

- Einzelheiten zur API finden Sie BatchWriteItemin der AWS SDK für Go API-Referenz.

Java

SDK für Java 2.x

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

Fügt mithilfe des Service-Clients viele Elemente in eine Tabelle ein.

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.AttributeValue;
import software.amazon.awssdk.services.dynamodb.model.BatchWriteItemRequest;
import software.amazon.awssdk.services.dynamodb.model.BatchWriteItemResponse;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.model.PutRequest;
import software.amazon.awssdk.services.dynamodb.model.WriteRequest;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;

/**
 * Before running this Java V2 code example, set up your development environment,
 including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 */
```

```
public class BatchWriteItems {
    public static void main(String[] args){
        final String usage = """

                Usage:
                    <tableName>

                Where:
                    tableName - The Amazon DynamoDB table (for example, Music).\s
                """;

        String tableName = "Music";
        Region region = Region.US_EAST_1;
        DynamoDbClient dynamoDbClient = DynamoDbClient.builder()
            .region(region)
            .build();

        addBatchItems(dynamoDbClient, tableName);
    }

    public static void addBatchItems(DynamoDbClient dynamoDbClient, String
 tableName) {
        // Specify the updates you want to perform.
        List<WriteRequest> writeRequests = new ArrayList<>();

        // Set item 1.
        Map<String, AttributeValue> item1Attributes = new HashMap<>();
        item1Attributes.put("Artist",
 AttributeValue.builder().s("Artist1").build());
        item1Attributes.put("Rating", AttributeValue.builder().s("5").build());
        item1Attributes.put("Comments", AttributeValue.builder().s("Great
 song!").build());
        item1Attributes.put("SongTitle",
 AttributeValue.builder().s("SongTitle1").build());

 writeRequests.add(WriteRequest.builder().putRequest(PutRequest.builder().item(item1Attri

        // Set item 2.
        Map<String, AttributeValue> item2Attributes = new HashMap<>();
        item2Attributes.put("Artist",
 AttributeValue.builder().s("Artist2").build());
        item2Attributes.put("Rating", AttributeValue.builder().s("4").build());
        item2Attributes.put("Comments", AttributeValue.builder().s("Nice
 melody.").build());
```

```
        item2Attributes.put("SongTitle",
 AttributeValue.builder().s("SongTitle2").build());

 writeRequests.add(WriteRequest.builder().putRequest(PutRequest.builder().item(item2Attri

        try {
            // Create the BatchWriteItemRequest.
            BatchWriteItemRequest batchWriteItemRequest =
 BatchWriteItemRequest.builder()
                .requestItems(Map.of(tableName, writeRequests))
                .build();

            // Execute the BatchWriteItem operation.
            BatchWriteItemResponse batchWriteItemResponse =
 dynamoDbClient.batchWriteItem(batchWriteItemRequest);

            // Process the response.
            System.out.println("Batch write successful: " +
 batchWriteItemResponse);

        } catch (DynamoDbException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
    }
}
```

Fügt mithilfe des erweiterten Clients viele Elemente in eine Tabelle ein.

```
import com.example.dynamodb.Customer;
import com.example.dynamodb.Music;
import software.amazon.awssdk.enhanced.dynamodb.DynamoDbEnhancedClient;
import software.amazon.awssdk.enhanced.dynamodb.DynamoDbTable;
import software.amazon.awssdk.enhanced.dynamodb.Key;
import software.amazon.awssdk.enhanced.dynamodb.TableSchema;
import
 software.amazon.awssdk.enhanced.dynamodb.model.BatchWriteItemEnhancedRequest;
import software.amazon.awssdk.enhanced.dynamodb.model.WriteBatch;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import java.time.Instant;
```

```
import java.time.LocalDate;
import java.time.LocalDateTime;
import java.time.ZoneOffset;

/*
 * Before running this code example, create an Amazon DynamoDB table named
 Customer with these columns:
 *   - id - the id of the record that is the key
 *   - custName - the customer name
 *   - email - the email value
 *   - registrationDate - an instant value when the item was added to the table
 *
 * Also, ensure that you have set up your development environment, including your
 credentials.
 *
 * For information, see this documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 */
public class EnhancedBatchWriteItems {
        public static void main(String[] args) {
                Region region = Region.US_EAST_1;
                DynamoDbClient ddb = DynamoDbClient.builder()
                                .region(region)
                                .build();
                DynamoDbEnhancedClient enhancedClient =
 DynamoDbEnhancedClient.builder()
                                .dynamoDbClient(ddb)
                                .build();
                putBatchRecords(enhancedClient);
                ddb.close();
        }

        public static void putBatchRecords(DynamoDbEnhancedClient enhancedClient)
 {
                try {
                        DynamoDbTable<Customer> customerMappedTable =
 enhancedClient.table("Customer",
                                        TableSchema.fromBean(Customer.class));
                        DynamoDbTable<Music> musicMappedTable =
 enhancedClient.table("Music",
                                        TableSchema.fromBean(Music.class));
                        LocalDate localDate = LocalDate.parse("2020-04-07");
```

```
                        LocalDateTime localDateTime = localDate.atStartOfDay();
                        Instant instant =
localDateTime.toInstant(ZoneOffset.UTC);

                        Customer record2 = new Customer();
                        record2.setCustName("Fred Pink");
                        record2.setId("id110");
                        record2.setEmail("fredp@noserver.com");
                        record2.setRegistrationDate(instant);

                        Customer record3 = new Customer();
                        record3.setCustName("Susan Pink");
                        record3.setId("id120");
                        record3.setEmail("spink@noserver.com");
                        record3.setRegistrationDate(instant);

                        Customer record4 = new Customer();
                        record4.setCustName("Jerry orange");
                        record4.setId("id101");
                        record4.setEmail("jorange@noserver.com");
                        record4.setRegistrationDate(instant);

                        BatchWriteItemEnhancedRequest
batchWriteItemEnhancedRequest = BatchWriteItemEnhancedRequest
                                        .builder()
                                        .writeBatches(

WriteBatch.builder(Customer.class) // add items to the Customer

         // table

.mappedTableResource(customerMappedTable)

.addPutItem(builder -> builder.item(record2))

.addPutItem(builder -> builder.item(record3))

.addPutItem(builder -> builder.item(record4))
                                                                        .build(),

WriteBatch.builder(Music.class) // delete an item from the Music

      // table
```

```
 .mappedTableResource(musicMappedTable)

 .addDeleteItem(builder -> builder.key(

       Key.builder().partitionValue(

                        "Famous Band")

                        .build()))
                                                                    .build())
                                     .build();

                         // Add three items to the Customer table and delete one
 item from the Music
                         // table.

 enhancedClient.batchWriteItem(batchWriteItemEnhancedRequest);
                         System.out.println("done");

                } catch (DynamoDbException e) {
                        System.err.println(e.getMessage());
                        System.exit(1);
                }
        }
}
```

- Einzelheiten zur API finden Sie BatchWriteItemin der AWS SDK for Java 2.x API-Referenz.

JavaScript

SDK für JavaScript (v3)

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

In diesem Beispiel wird der Dokument-Client verwendet, um die Arbeit mit Elementen in DynamoDB zu vereinfachen. Einzelheiten zur API finden Sie unter BatchWrite.

```
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";
import {
  BatchWriteCommand,
  DynamoDBDocumentClient,
} from "@aws-sdk/lib-dynamodb";
import { readFileSync } from "node:fs";

// These modules are local to our GitHub repository. We recommend cloning
// the project from GitHub if you want to run this example.
// For more information, see https://github.com/awsdocs/aws-doc-sdk-examples.
import { dirnameFromMetaUrl } from "@aws-doc-sdk-examples/lib/utils/util-fs.js";
import { chunkArray } from "@aws-doc-sdk-examples/lib/utils/util-array.js";

const dirname = dirnameFromMetaUrl(import.meta.url);

const client = new DynamoDBClient({});
const docClient = DynamoDBDocumentClient.from(client);

export const main = async () => {
  const file = readFileSync(
    `${dirname}../../../../../resources/sample_files/movies.json`,
  );

  const movies = JSON.parse(file.toString());

  // chunkArray is a local convenience function. It takes an array and returns
  // a generator function. The generator function yields every N items.
  const movieChunks = chunkArray(movies, 25);

  // For every chunk of 25 movies, make one BatchWrite request.
  for (const chunk of movieChunks) {
    const putRequests = chunk.map((movie) => ({
      PutRequest: {
        Item: movie,
      },
    }));

    const command = new BatchWriteCommand({
      RequestItems: {
        // An existing table is required. A composite key of 'title' and 'year'
 is recommended
        // to account for duplicate titles.
        BatchWriteMoviesTable: putRequests,
```

```
      },
    });

    await docClient.send(command);
  }
};
```

- Einzelheiten zur API finden Sie BatchWriteItemin der AWS SDK für JavaScript API-Referenz.

SDK für JavaScript (v2)

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create DynamoDB service object
var ddb = new AWS.DynamoDB({ apiVersion: "2012-08-10" });

var params = {
  RequestItems: {
    TABLE_NAME: [
      {
        PutRequest: {
          Item: {
            KEY: { N: "KEY_VALUE" },
            ATTRIBUTE_1: { S: "ATTRIBUTE_1_VALUE" },
            ATTRIBUTE_2: { N: "ATTRIBUTE_2_VALUE" },
          },
        },
      },
      {
        PutRequest: {
          Item: {
            KEY: { N: "KEY_VALUE" },
```

```
            ATTRIBUTE_1: { S: "ATTRIBUTE_1_VALUE" },
            ATTRIBUTE_2: { N: "ATTRIBUTE_2_VALUE" },
          },
        },
      },
    ],
  },
};

ddb.batchWriteItem(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data);
  }
});
```

- Weitere Informationen finden Sie im AWS SDK für JavaScript -Entwicklerhandbuch.
- Einzelheiten zur API finden Sie BatchWriteItemin der AWS SDK für JavaScript API-Referenz.

PHP

SDK für PHP

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
    public function writeBatch(string $TableName, array $Batch, int $depth = 2)
    {
        if (--$depth <= 0) {
            throw new Exception("Max depth exceeded. Please try with fewer batch
 items or increase depth.");
        }

        $marshal = new Marshaler();
        $total = 0;
```

```
        foreach (array_chunk($Batch, 25) as $Items) {
            foreach ($Items as $Item) {
                $BatchWrite['RequestItems'][$TableName][] = ['PutRequest' =>
 ['Item' => $marshal->marshalItem($Item)]];
            }
            try {
                echo "Batching another " . count($Items) . " for a total of " .
 ($total += count($Items)) . " items!\n";
                $response = $this->dynamoDbClient->batchWriteItem($BatchWrite);
                $BatchWrite = [];
            } catch (Exception $e) {
                echo "uh oh...";
                echo $e->getMessage();
                die();
            }
            if ($total >= 250) {
                echo "250 movies is probably enough. Right? We can stop there.
\n";
                break;
            }
        }
    }
```

- Einzelheiten zur API finden Sie BatchWriteItemin der AWS SDK für PHP API-Referenz.

PowerShell

Tools für PowerShell V4

Beispiel 1: Erstellt ein neues Element oder ersetzt ein vorhandenes durch ein neues Element in den DynamoDB-Tabellen „Music“ und „Songs“.

```
$item = @{
    SongTitle = 'Somewhere Down The Road'
    Artist = 'No One You Know'
    AlbumTitle = 'Somewhat Famous'
    Price = 1.94
    Genre = 'Country'
    CriticRating = 10.0
} | ConvertTo-DDBItem

$writeRequest = New-Object Amazon.DynamoDBv2.Model.WriteRequest
```

```
$writeRequest.PutRequest = [Amazon.DynamoDBv2.Model.PutRequest]$item

$requestItem = @{
    'Music' = [Amazon.DynamoDBv2.Model.WriteRequest]($writeRequest)
    'Songs' = [Amazon.DynamoDBv2.Model.WriteRequest]($writeRequest)
}

Set-DDBBatchItem -RequestItem $requestItem
```

- Einzelheiten zur API finden Sie unter BatchWriteItem AWS -Tools für PowerShellCmdlet-Referenz (V4).

Tools für V5 PowerShell

Beispiel 1: Erstellt ein neues Element oder ersetzt ein vorhandenes durch ein neues Element in den DynamoDB-Tabellen „Music“ und „Songs“.

```
$item = @{
    SongTitle = 'Somewhere Down The Road'
    Artist = 'No One You Know'
    AlbumTitle = 'Somewhat Famous'
    Price = 1.94
    Genre = 'Country'
    CriticRating = 10.0
} | ConvertTo-DDBItem

$writeRequest = New-Object Amazon.DynamoDBv2.Model.WriteRequest
$writeRequest.PutRequest = [Amazon.DynamoDBv2.Model.PutRequest]$item

$requestItem = @{
    'Music' = [Amazon.DynamoDBv2.Model.WriteRequest]($writeRequest)
    'Songs' = [Amazon.DynamoDBv2.Model.WriteRequest]($writeRequest)
}

Set-DDBBatchItem -RequestItem $requestItem
```

- Einzelheiten zur API finden Sie unter BatchWriteItem AWS -Tools für PowerShellCmdlet-Referenz (V5).

Python

SDK für Python (Boto3)

> **Note**
>
> Es gibt noch mehr dazu. GitHub Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
class Movies:
    """Encapsulates an Amazon DynamoDB table of movie data.

    Example data structure for a movie record in this table:
        {
            "year": 1999,
            "title": "For Love of the Game",
            "info": {
                "directors": ["Sam Raimi"],
                "release_date": "1999-09-15T00:00:00Z",
                "rating": 6.3,
                "plot": "A washed up pitcher flashes through his career.",
                "rank": 4987,
                "running_time_secs": 8220,
                "actors": [
                    "Kevin Costner",
                    "Kelly Preston",
                    "John C. Reilly"
                ]
            }
        }
    """

    def __init__(self, dyn_resource):
        """
        :param dyn_resource: A Boto3 DynamoDB resource.
        """
        self.dyn_resource = dyn_resource
        # The table variable is set during the scenario in the call to
        # 'exists' if the table exists. Otherwise, it is set by 'create_table'.
        self.table = None
```

```
   def write_batch(self, movies):
        """
        Fills an Amazon DynamoDB table with the specified data, using the Boto3
        Table.batch_writer() function to put the items in the table.
        Inside the context manager, Table.batch_writer builds a list of
        requests. On exiting the context manager, Table.batch_writer starts
sending
        batches of write requests to Amazon DynamoDB and automatically
        handles chunking, buffering, and retrying.

        :param movies: The data to put in the table. Each item must contain at
least
                       the keys required by the schema that was specified when
the
                       table was created.
        """
        try:
            with self.table.batch_writer() as writer:
                for movie in movies:
                    writer.put_item(Item=movie)
        except ClientError as err:
            logger.error(
                "Couldn't load data into table %s. Here's why: %s: %s",
                self.table.name,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
```

- Einzelheiten zur API finden Sie BatchWriteItemin AWS SDK for Python (Boto3) API Reference.

Ruby

SDK für Ruby

> **Note**
>
> Es gibt noch mehr dazu. GitHub Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
class DynamoDBBasics
  attr_reader :dynamo_resource, :table

  def initialize(table_name)
    client = Aws::DynamoDB::Client.new(region: 'us-east-1')
    @dynamo_resource = Aws::DynamoDB::Resource.new(client: client)
    @table = @dynamo_resource.table(table_name)
  end

  # Fills an Amazon DynamoDB table with the specified data. Items are sent in
  # batches of 25 until all items are written.
  #
  # @param movies [Enumerable] The data to put in the table. Each item must
 contain at least
  #                            the keys required by the schema that was specified
 when the
  #                            table was created.
  def write_batch(movies)
    index = 0
    slice_size = 25
    while index < movies.length
      movie_items = []
      movies[index, slice_size].each do |movie|
        movie_items.append({ put_request: { item: movie } })
      end
      @dynamo_resource.client.batch_write_item({ request_items: { @table.name =>
 movie_items } })
      index += slice_size
    end
  rescue Aws::DynamoDB::Errors::ServiceError => e
    puts(
      "Couldn't load data into table #{@table.name}. Here's why:"
```

```
    )
    puts("\t#{e.code}: #{e.message}")
    raise
  end
```

- Einzelheiten zur API finden Sie BatchWriteItemin der AWS SDK für Ruby API-Referenz.

Swift

SDK für Swift

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
import AWSDynamoDB


    /// Populate the movie database from the specified JSON file.
    ///
    /// - Parameter jsonPath: Path to a JSON file containing movie data.
    ///
    func populate(jsonPath: String) async throws {
        do {
            guard let client = self.ddbClient else {
                throw MoviesError.UninitializedClient
            }

            // Create a Swift `URL` and use it to load the file into a `Data`
            // object. Then decode the JSON into an array of `Movie` objects.

            let fileUrl = URL(fileURLWithPath: jsonPath)
            let jsonData = try Data(contentsOf: fileUrl)

            var movieList = try JSONDecoder().decode([Movie].self, from:
 jsonData)

            // Truncate the list to the first 200 entries or so for this example.
```

```
            if movieList.count > 200 {
                movieList = Array(movieList[...199])
            }

            // Before sending records to the database, break the movie list into
            // 25-entry chunks, which is the maximum size of a batch item
request.

            let count = movieList.count
            let chunks = stride(from: 0, to: count, by: 25).map {
                Array(movieList[$0 ..< Swift.min($0 + 25, count)])
            }

            // For each chunk, create a list of write request records and
populate
            // them with `PutRequest` requests, each specifying one movie from
the
            // chunk. Once the chunk's items are all in the `PutRequest` list,
            // send them to Amazon DynamoDB using the
            // `DynamoDBClient.batchWriteItem()` function.

            for chunk in chunks {
                var requestList: [DynamoDBClientTypes.WriteRequest] = []

                for movie in chunk {
                    let item = try await movie.getAsItem()
                    let request = DynamoDBClientTypes.WriteRequest(
                        putRequest: .init(
                            item: item
                        )
                    )
                    requestList.append(request)
                }

                let input = BatchWriteItemInput(requestItems: [tableName:
requestList])
                _ = try await client.batchWriteItem(input: input)
            }
        } catch {
            print("ERROR: populate:", dump(error))
            throw error
        }
    }
```

- Einzelheiten zur API finden Sie BatchWriteItemin der API-Referenz zum AWS SDK für Swift.

Eine vollständige Liste der AWS SDK-Entwicklerhandbücher und Codebeispiele finden Sie unterDynamoDB mit einem SDK verwenden AWS. Dieses Thema enthält auch Informationen zu den ersten Schritten und Details zu früheren SDK-Versionen.

Verwendung **`CreateTable`** mit einem AWS SDK oder CLI

Die folgenden Code-Beispiele zeigen, wie `CreateTable` verwendet wird.

Aktionsbeispiele sind Codeauszüge aus größeren Programmen und müssen im Kontext ausgeführt werden. Sie können diese Aktion in den folgenden Codebeispielen im Kontext sehen:

- Kennenlernen der Grundlagen
- Beschleunigen von Lesevorgängen mit DAX
- Erweiterte Szenarien für globale sekundäre Indizes
- Erstellen einer Tabelle mit einem globalen sekundären Index
- Erstellen einer Tabelle mit aktiviertem Warmdurchsatz
- Erstellen und Verwalten globaler Tabellen zur Demonstration von MREC
- Erstellen und Verwalten von globalen MRSC-Tabellen
- Verwalten globaler sekundärer Indizes
- Verwalten ressourcenbasierter Richtlinien
- Einrichten der attributbasierten Zugriffskontrolle
- Arbeiten mit globalen Tabellen und letztendlicher multiregionaler Konsistenz (Multi-Region Eventual Consistency, MREC)
- Arbeiten mit lokalen sekundären Indizes
- Arbeiten mit Ressourcen-Tagging
- Arbeite mit Streams und Time-to-Live
- Arbeiten mit der Tabellenverschlüsselung

.NET

SDK für .NET (v4)

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
    /// <summary>
    /// Creates a new Amazon DynamoDB table and then waits for the new
    /// table to become active.
    /// </summary>
    /// <param name="tableName">The name of the table to create.</param>
    /// <returns>A Boolean value indicating the success of the operation.</
returns>
    public async Task<bool> CreateMovieTableAsync(string tableName)
    {
        try
        {
            var response = await _amazonDynamoDB.CreateTableAsync(new
 CreateTableRequest
            {
                TableName = tableName,
                AttributeDefinitions = new List<AttributeDefinition>()
                {
                    new AttributeDefinition
                    {
                        AttributeName = "title",
                        AttributeType = ScalarAttributeType.S,
                    },
                    new AttributeDefinition
                    {
                        AttributeName = "year",
                        AttributeType = ScalarAttributeType.N,
                    },
                },
                KeySchema = new List<KeySchemaElement>()
                {
                    new KeySchemaElement
```

```
                    {
                        AttributeName = "year",
                        KeyType = KeyType.HASH,
                    },
                    new KeySchemaElement
                    {
                        AttributeName = "title",
                        KeyType = KeyType.RANGE,
                    },
                },
                BillingMode = BillingMode.PAY_PER_REQUEST,
            });

            // Wait until the table is ACTIVE and then report success.
            Console.Write("Waiting for table to become active...");

            var request = new DescribeTableRequest
            {
                TableName = response.TableDescription.TableName,
            };

            TableStatus status;

            int sleepDuration = 2000;

            do
            {
                Thread.Sleep(sleepDuration);

                var describeTableResponse = await
_amazonDynamoDB.DescribeTableAsync(request);
                status = describeTableResponse.Table.TableStatus;

                Console.Write(".");
            }
            while (status != "ACTIVE");

            return status == TableStatus.ACTIVE;
        }
        catch (ResourceInUseException ex)
        {
            Console.WriteLine($"Table {tableName} already exists. {ex.Message}");
            throw;
        }
```

```
        catch (AmazonDynamoDBException ex)
        {
            Console.WriteLine($"An Amazon DynamoDB error occurred while creating
table {tableName}. {ex.Message}");
            throw;
        }
        catch (Exception ex)
        {
            Console.WriteLine($"An error occurred while creating table
{tableName}. {ex.Message}");
            throw;
        }
    }
```

- Einzelheiten zur API finden Sie CreateTablein der AWS SDK für .NET API-Referenz.

Bash

AWS CLI mit Bash-Skript

> **Note**
>
> Es gibt noch mehr dazu. GitHub Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
###############################################################################
# function dynamodb_create_table
#
# This function creates an Amazon DynamoDB table.
#
# Parameters:
#       -n table_name  -- The name of the table to create.
#       -a attribute_definitions -- JSON file path of a list of attributes and
 their types.
#       -k key_schema -- JSON file path of a list of attributes and their key
 types.
#
#  Returns:
#       0 - If successful.
```

```
#       1 - If it fails.
###############################################################################
function dynamodb_create_table() {
  local table_name attribute_definitions key_schema response
  local option OPTARG # Required to use getopts command in a function.

  ######################################
  # Function usage explanation
  ######################################
  function usage() {
    echo "function dynamodb_create_table"
    echo "Creates an Amazon DynamoDB table with on-demand billing."
    echo " -n table_name  -- The name of the table to create."
    echo " -a attribute_definitions -- JSON file path of a list of attributes and
 their types."
    echo " -k key_schema -- JSON file path of a list of attributes and their key
 types."
    echo ""
  }

  # Retrieve the calling parameters.
  while getopts "n:a:k:h" option; do
    case "${option}" in
      n) table_name="${OPTARG}" ;;
      a) attribute_definitions="${OPTARG}" ;;
      k) key_schema="${OPTARG}" ;;
      h)
        usage
        return 0
        ;;
      \?)
        echo "Invalid parameter"
        usage
        return 1
        ;;
    esac
  done
  export OPTIND=1

  if [[ -z "$table_name" ]]; then
    errecho "ERROR: You must provide a table name with the -n parameter."
    usage
    return 1
  fi
```

```
  if [[ -z "$attribute_definitions" ]]; then
    errecho "ERROR: You must provide an attribute definitions json file path the
 -a parameter."
    usage
    return 1
  fi

  if [[ -z "$key_schema" ]]; then
    errecho "ERROR: You must provide a key schema json file path the -k
 parameter."
    usage
    return 1
  fi

  iecho "Parameters:\n"
  iecho "    table_name:   $table_name"
  iecho "    attribute_definitions:   $attribute_definitions"
  iecho "    key_schema:   $key_schema"
  iecho ""

  response=$(aws dynamodb create-table \
    --table-name "$table_name" \
    --attribute-definitions file://"$attribute_definitions" \
    --billing-mode PAY_PER_REQUEST \
    --key-schema file://"$key_schema" )

  local error_code=${?}

  if [[ $error_code -ne 0 ]]; then
    aws_cli_error_log $error_code
    errecho "ERROR: AWS reports create-table operation failed.$response"
    return 1
  fi

  return 0
}
```

Die in diesem Beispiel verwendeten Hilfsfunktionen.

```
###############################################################################
# function iecho
```

```
#
# This function enables the script to display the specified text only if
# the global variable $VERBOSE is set to true.
#############################################################################
function iecho() {
  if [[ $VERBOSE == true ]]; then
    echo "$@"
  fi
}

#############################################################################
# function errecho
#
# This function outputs everything sent to it to STDERR (standard error output).
#############################################################################
function errecho() {
  printf "%s\n" "$*" 1>&2
}

#############################################################################
# function aws_cli_error_log()
#
# This function is used to log the error messages from the AWS CLI.
#
# See https://docs.aws.amazon.com/cli/latest/topic/return-codes.html#cli-aws-
help-return-codes.
#
# The function expects the following argument:
#         $1 - The error code returned by the AWS CLI.
#
#  Returns:
#          0: - Success.
#
#############################################################################
function aws_cli_error_log() {
  local err_code=$1
  errecho "Error code : $err_code"
  if [ "$err_code" == 1 ]; then
    errecho "  One or more S3 transfers failed."
  elif [ "$err_code" == 2 ]; then
    errecho "  Command line failed to parse."
  elif [ "$err_code" == 130 ]; then
    errecho "  Process received SIGINT."
  elif [ "$err_code" == 252 ]; then
```

```
    errecho "  Command syntax invalid."
  elif [ "$err_code" == 253 ]; then
    errecho "  The system environment or configuration was invalid."
  elif [ "$err_code" == 254 ]; then
    errecho "  The service returned an error."
  elif [ "$err_code" == 255 ]; then
    errecho "  255 is a catch-all error."
  fi

  return 0
}
```

- Einzelheiten zur API finden Sie CreateTablein der AWS CLI Befehlsreferenz.

C++

SDK für C++

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
//! Create an Amazon DynamoDB table.
/*!
  \sa createTable()
  \param tableName: Name for the DynamoDB table.
  \param primaryKey: Primary key for the DynamoDB table.
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
 */
bool AwsDoc::DynamoDB::createTable(const Aws::String &tableName,
                                   const Aws::String &primaryKey,
                                   const Aws::Client::ClientConfiguration
 &clientConfiguration) {
    Aws::DynamoDB::DynamoDBClient dynamoClient(clientConfiguration);

    std::cout << "Creating table " << tableName <<
              " with a simple primary key: \"" << primaryKey << "\"." <<
 std::endl;
```

```
    Aws::DynamoDB::Model::CreateTableRequest request;

    Aws::DynamoDB::Model::AttributeDefinition hashKey;
    hashKey.SetAttributeName(primaryKey);
    hashKey.SetAttributeType(Aws::DynamoDB::Model::ScalarAttributeType::S);
    request.AddAttributeDefinitions(hashKey);

    Aws::DynamoDB::Model::KeySchemaElement keySchemaElement;
    keySchemaElement.WithAttributeName(primaryKey).WithKeyType(
            Aws::DynamoDB::Model::KeyType::HASH);
    request.AddKeySchema(keySchemaElement);

    Aws::DynamoDB::Model::ProvisionedThroughput throughput;
    throughput.WithReadCapacityUnits(5).WithWriteCapacityUnits(5);
    request.SetProvisionedThroughput(throughput);
    request.SetTableName(tableName);

    const Aws::DynamoDB::Model::CreateTableOutcome &outcome =
 dynamoClient.CreateTable(
            request);
    if (outcome.IsSuccess()) {
        std::cout << "Table \""
                  << outcome.GetResult().GetTableDescription().GetTableName() <<
                  " created!" << std::endl;
    }
    else {
        std::cerr << "Failed to create table: " <<
 outcome.GetError().GetMessage()
                  << std::endl;
        return false;
    }

    return waitTableActive(tableName, dynamoClient);
}
```

Code, der darauf wartet, dass die Tabelle aktiv wird.

```
//! Query a newly created DynamoDB table until it is active.
/*!
  \sa waitTableActive()
  \param waitTableActive: The DynamoDB table's name.
```

```
  \param dynamoClient: A DynamoDB client.
  \return bool: Function succeeded.
*/
bool AwsDoc::DynamoDB::waitTableActive(const Aws::String &tableName,
                                       const Aws::DynamoDB::DynamoDBClient
 &dynamoClient) {

    // Repeatedly call DescribeTable until table is ACTIVE.
    const int MAX_QUERIES = 20;
    Aws::DynamoDB::Model::DescribeTableRequest request;
    request.SetTableName(tableName);

    int count = 0;
    while (count < MAX_QUERIES) {
        const Aws::DynamoDB::Model::DescribeTableOutcome &result =
 dynamoClient.DescribeTable(
                request);
        if (result.IsSuccess()) {
            Aws::DynamoDB::Model::TableStatus status =
 result.GetResult().GetTable().GetTableStatus();

            if (Aws::DynamoDB::Model::TableStatus::ACTIVE != status) {
                std::this_thread::sleep_for(std::chrono::seconds(1));
            }
            else {
                return true;
            }
        }
        else {
            std::cerr << "Error DynamoDB::waitTableActive "
                      << result.GetError().GetMessage() << std::endl;
            return false;
        }
        count++;
    }
    return false;
}
```

- Einzelheiten zur API finden Sie CreateTablein der AWS SDK für C++ API-Referenz.

CLI

AWS CLI

Beispiel 1: So erstellen Sie eine Tabelle mit Tags

Im folgenden Beispiel für `create-table` werden die angegebenen Attribute und das angegebene Schlüsselschema verwendet, um eine Tabelle mit dem Namen `MusicCollection` zu erstellen. Diese Tabelle verwendet den bereitgestellten Durchsatz und wird im Ruhezustand mit dem standardmäßigen AWS eigenen CMK verschlüsselt. Der Befehl weist der Tabelle außerdem ein Tag mit dem Schlüssel `Owner` und dem Wert `blueTeam` zu.

```
aws dynamodb create-table \
    --table-name MusicCollection \
    --attribute-
definitions AttributeName=Artist,AttributeType=S AttributeName=SongTitle,AttributeType=S
 \
    --key-
schema AttributeName=Artist,KeyType=HASH AttributeName=SongTitle,KeyType=RANGE \
    --provisioned-throughput ReadCapacityUnits=5,WriteCapacityUnits=5 \
    --tags Key=Owner,Value=blueTeam
```

Ausgabe:

```
{
    "TableDescription": {
        "AttributeDefinitions": [
            {
                "AttributeName": "Artist",
                "AttributeType": "S"
            },
            {
                "AttributeName": "SongTitle",
                "AttributeType": "S"
            }
        ],
        "ProvisionedThroughput": {
            "NumberOfDecreasesToday": 0,
            "WriteCapacityUnits": 5,
            "ReadCapacityUnits": 5
        },
        "TableSizeBytes": 0,
```

```
        "TableName": "MusicCollection",
        "TableStatus": "CREATING",
        "KeySchema": [
            {
                "KeyType": "HASH",
                "AttributeName": "Artist"
            },
            {
                "KeyType": "RANGE",
                "AttributeName": "SongTitle"
            }
        ],
        "ItemCount": 0,
        "CreationDateTime": "2020-05-26T16:04:41.627000-07:00",
        "TableArn": "arn:aws:dynamodb:us-west-2:123456789012:table/
MusicCollection",
        "TableId": "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111"
    }
}
```

Weitere Informationen finden Sie unter Grundlegende Operationen für Tabellen im Entwicklerhandbuch zu Amazon DynamoDB.

Beispiel 2: So erstellen Sie eine Tabelle im On-Demand-Modus

Im folgenden Beispiel wird mit dem On-Demand-Modus anstelle des bereitgestellten Durchsatzmodus eine Tabelle mit dem Namen `MusicCollection` erstellt. Dies ist nützlich für Tabellen mit unvorhersehbaren Workloads.

```
aws dynamodb create-table \
    --table-name MusicCollection \
    --attribute-
definitions AttributeName=Artist,AttributeType=S AttributeName=SongTitle,AttributeType=S
 \
    --key-
schema AttributeName=Artist,KeyType=HASH AttributeName=SongTitle,KeyType=RANGE \
    --billing-mode PAY_PER_REQUEST
```

Ausgabe:

```
{
    "TableDescription": {
        "AttributeDefinitions": [
```

```
            {
                "AttributeName": "Artist",
                "AttributeType": "S"
            },
            {
                "AttributeName": "SongTitle",
                "AttributeType": "S"
            }
        ],
        "TableName": "MusicCollection",
        "KeySchema": [
            {
                "AttributeName": "Artist",
                "KeyType": "HASH"
            },
            {
                "AttributeName": "SongTitle",
                "KeyType": "RANGE"
            }
        ],
        "TableStatus": "CREATING",
        "CreationDateTime": "2020-05-27T11:44:10.807000-07:00",
        "ProvisionedThroughput": {
            "NumberOfDecreasesToday": 0,
            "ReadCapacityUnits": 0,
            "WriteCapacityUnits": 0
        },
        "TableSizeBytes": 0,
        "ItemCount": 0,
        "TableArn": "arn:aws:dynamodb:us-west-2:123456789012:table/
MusicCollection",
        "TableId": "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
        "BillingModeSummary": {
            "BillingMode": "PAY_PER_REQUEST"
        }
    }
}
```

Weitere Informationen finden Sie unter Grundlegende Operationen für Tabellen im Entwicklerhandbuch zu Amazon DynamoDB.

Beispiel 3: So erstellen Sie eine Tabelle und verschlüsseln sie mit einem kundenseitig verwalteten Schlüssel (Customer Managed Key; CMK).

Im folgenden Beispiel wird eine Tabelle mit dem Namen `MusicCollection` erstellt und mithilfe eines kundenseitig verwalteten Schlüssels (Customer Managed Key; CMK) verschlüsselt.

```
aws dynamodb create-table \
    --table-name MusicCollection \
    --attribute-
definitions AttributeName=Artist,AttributeType=S AttributeName=SongTitle,AttributeType=S
 \
    --key-
schema AttributeName=Artist,KeyType=HASH AttributeName=SongTitle,KeyType=RANGE \
    --provisioned-throughput ReadCapacityUnits=5,WriteCapacityUnits=5 \
    --sse-specification Enabled=true,SSEType=KMS,KMSMasterKeyId=abcd1234-
abcd-1234-a123-ab1234a1b234
```

Ausgabe:

```
{
    "TableDescription": {
        "AttributeDefinitions": [
            {
                "AttributeName": "Artist",
                "AttributeType": "S"
            },
            {
                "AttributeName": "SongTitle",
                "AttributeType": "S"
            }
        ],
        "TableName": "MusicCollection",
        "KeySchema": [
            {
                "AttributeName": "Artist",
                "KeyType": "HASH"
            },
            {
                "AttributeName": "SongTitle",
                "KeyType": "RANGE"
            }
        ],
        "TableStatus": "CREATING",
        "CreationDateTime": "2020-05-27T11:12:16.431000-07:00",
        "ProvisionedThroughput": {
```

```
            "NumberOfDecreasesToday": 0,
            "ReadCapacityUnits": 5,
            "WriteCapacityUnits": 5
        },
        "TableSizeBytes": 0,
        "ItemCount": 0,
        "TableArn": "arn:aws:dynamodb:us-west-2:123456789012:table/
MusicCollection",
        "TableId": "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
        "SSEDescription": {
            "Status": "ENABLED",
            "SSEType": "KMS",
            "KMSMasterKeyArn": "arn:aws:kms:us-west-2:123456789012:key/abcd1234-
abcd-1234-a123-ab1234a1b234"
        }
    }
}
```

Weitere Informationen finden Sie unter Grundlegende Operationen für Tabellen im Entwicklerhandbuch zu Amazon DynamoDB.

Bespiel 4: So erstellen Sie eine Tabelle mit einem lokalen sekundären Index

Im folgenden Beispiel werden die angegebenen Attribute und das angegebene Schlüsselschema verwendet, um eine Tabelle namens `MusicCollection` mit dem lokalen sekundären Index `AlbumTitleIndex` zu erstellen.

```
aws dynamodb create-table \
    --table-name MusicCollection \
    --attribute-
definitions AttributeName=Artist,AttributeType=S AttributeName=SongTitle,AttributeType=S
 \
    --key-
schema AttributeName=Artist,KeyType=HASH AttributeName=SongTitle,KeyType=RANGE \
    --provisioned-throughput ReadCapacityUnits=10,WriteCapacityUnits=5 \
    --local-secondary-indexes \
        "[
            {
                \"IndexName\": \"AlbumTitleIndex\",
                \"KeySchema\": [
                    {\"AttributeName\": \"Artist\",\"KeyType\":\"HASH\"},
                    {\"AttributeName\": \"AlbumTitle\",\"KeyType\":\"RANGE\"}
                ],
```

```
                \"Projection\": {
                    \"ProjectionType\": \"INCLUDE\",
                    \"NonKeyAttributes\": [\"Genre\", \"Year\"]
                }
            }
        ]"
```

Ausgabe:

```
{
    "TableDescription": {
        "AttributeDefinitions": [
            {
                "AttributeName": "AlbumTitle",
                "AttributeType": "S"
            },
            {
                "AttributeName": "Artist",
                "AttributeType": "S"
            },
            {
                "AttributeName": "SongTitle",
                "AttributeType": "S"
            }
        ],
        "TableName": "MusicCollection",
        "KeySchema": [
            {
                "AttributeName": "Artist",
                "KeyType": "HASH"
            },
            {
                "AttributeName": "SongTitle",
                "KeyType": "RANGE"
            }
        ],
        "TableStatus": "CREATING",
        "CreationDateTime": "2020-05-26T15:59:49.473000-07:00",
        "ProvisionedThroughput": {
            "NumberOfDecreasesToday": 0,
            "ReadCapacityUnits": 10,
            "WriteCapacityUnits": 5
        },
```

```
        "TableSizeBytes": 0,
        "ItemCount": 0,
        "TableArn": "arn:aws:dynamodb:us-west-2:123456789012:table/
MusicCollection",
        "TableId": "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
        "LocalSecondaryIndexes": [
            {
                "IndexName": "AlbumTitleIndex",
                "KeySchema": [
                    {
                        "AttributeName": "Artist",
                        "KeyType": "HASH"
                    },
                    {
                        "AttributeName": "AlbumTitle",
                        "KeyType": "RANGE"
                    }
                ],
                "Projection": {
                    "ProjectionType": "INCLUDE",
                    "NonKeyAttributes": [
                        "Genre",
                        "Year"
                    ]
                },
                "IndexSizeBytes": 0,
                "ItemCount": 0,
                "IndexArn": "arn:aws:dynamodb:us-west-2:123456789012:table/
MusicCollection/index/AlbumTitleIndex"
            }
        ]
    }
}
```

Weitere Informationen finden Sie unter Grundlegende Operationen für Tabellen im Entwicklerhandbuch zu Amazon DynamoDB.

Beispiel 5: So erstellen Sie eine Tabelle mit einem globalen sekundären Index

Im folgenden Beispiel wird eine Tabelle mit dem Namen `GameScores` mit einem globalen sekundären Index namens `GameTitleIndex` erstellt. Die Basistabelle hat einen Partitionsschlüssel von `UserId` und einen Sortierschlüssel von `GameTitle`, mit dem Sie effizient die beste Punktzahl eines einzelnen Benutzers für ein bestimmtes

Spiel finden können, während die GSI einen Partitionsschlüssel von `GameTitle` und einen Sortierschlüssel von `TopScore` hat, mit dem Sie finden Sie schnell die höchste Gesamtpunktzahl für ein bestimmtes Spiel.

```
aws dynamodb create-table \
    --table-name GameScores \
    --attribute-
definitions AttributeName=UserId,AttributeType=S AttributeName=GameTitle,AttributeType=S
 \
    --key-schema AttributeName=UserId,KeyType=HASH \
                 AttributeName=GameTitle,KeyType=RANGE \
    --provisioned-throughput ReadCapacityUnits=10,WriteCapacityUnits=5 \
    --global-secondary-indexes \
        "[
            {
                \"IndexName\": \"GameTitleIndex\",
                \"KeySchema\": [
                    {\"AttributeName\":\"GameTitle\",\"KeyType\":\"HASH\"},
                    {\"AttributeName\":\"TopScore\",\"KeyType\":\"RANGE\"}
                ],
                \"Projection\": {
                    \"ProjectionType\":\"INCLUDE\",
                    \"NonKeyAttributes\":[\"UserId\"]
                },
                \"ProvisionedThroughput\": {
                    \"ReadCapacityUnits\": 10,
                    \"WriteCapacityUnits\": 5
                }
            }
        ]"
```

Ausgabe:

```
{
    "TableDescription": {
        "AttributeDefinitions": [
            {
                "AttributeName": "GameTitle",
                "AttributeType": "S"
            },
            {
                "AttributeName": "TopScore",
                "AttributeType": "N"
```

```
            },
            {
                "AttributeName": "UserId",
                "AttributeType": "S"
            }
        ],
        "TableName": "GameScores",
        "KeySchema": [
            {
                "AttributeName": "UserId",
                "KeyType": "HASH"
            },
            {
                "AttributeName": "GameTitle",
                "KeyType": "RANGE"
            }
        ],
        "TableStatus": "CREATING",
        "CreationDateTime": "2020-05-26T17:28:15.602000-07:00",
        "ProvisionedThroughput": {
            "NumberOfDecreasesToday": 0,
            "ReadCapacityUnits": 10,
            "WriteCapacityUnits": 5
        },
        "TableSizeBytes": 0,
        "ItemCount": 0,
        "TableArn": "arn:aws:dynamodb:us-west-2:123456789012:table/GameScores",
        "TableId": "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
        "GlobalSecondaryIndexes": [
            {
                "IndexName": "GameTitleIndex",
                "KeySchema": [
                    {
                        "AttributeName": "GameTitle",
                        "KeyType": "HASH"
                    },
                    {
                        "AttributeName": "TopScore",
                        "KeyType": "RANGE"
                    }
                ],
                "Projection": {
                    "ProjectionType": "INCLUDE",
                    "NonKeyAttributes": [
```

```
                        "UserId"
                    ]
                },
                "IndexStatus": "CREATING",
                "ProvisionedThroughput": {
                    "NumberOfDecreasesToday": 0,
                    "ReadCapacityUnits": 10,
                    "WriteCapacityUnits": 5
                },
                "IndexSizeBytes": 0,
                "ItemCount": 0,
                "IndexArn": "arn:aws:dynamodb:us-west-2:123456789012:table/
GameScores/index/GameTitleIndex"
            }
        ]
    }
}
```

Weitere Informationen finden Sie unter Grundlegende Operationen für Tabellen im Entwicklerhandbuch zu Amazon DynamoDB.

Beispiel 6: So erstellen Sie eine Tabelle mit mehreren globalen sekundären Indizes gleichzeitig

Im folgenden Beispiel wird eine Tabelle namens `GameScores` mit zwei globalen sekundären Indizes erstellt. Die GSI-Schemas werden über eine Datei und nicht über die Befehlszeile übergeben.

```
aws dynamodb create-table \
    --table-name GameScores \
    --attribute-
definitions AttributeName=UserId,AttributeType=S AttributeName=GameTitle,AttributeType=S
 \
    --key-
schema AttributeName=UserId,KeyType=HASH AttributeName=GameTitle,KeyType=RANGE \
    --provisioned-throughput ReadCapacityUnits=10,WriteCapacityUnits=5 \
    --global-secondary-indexes file://gsi.json
```

Inhalt von `gsi.json`:

```
[
    {
        "IndexName": "GameTitleIndex",
```

```
        "KeySchema": [
            {
                "AttributeName": "GameTitle",
                "KeyType": "HASH"
            },
            {
                "AttributeName": "TopScore",
                "KeyType": "RANGE"
            }
        ],
        "Projection": {
            "ProjectionType": "ALL"
        },
        "ProvisionedThroughput": {
            "ReadCapacityUnits": 10,
            "WriteCapacityUnits": 5
        }
    },
    {
        "IndexName": "GameDateIndex",
        "KeySchema": [
            {
                "AttributeName": "GameTitle",
                "KeyType": "HASH"
            },
            {
                "AttributeName": "Date",
                "KeyType": "RANGE"
            }
        ],
        "Projection": {
            "ProjectionType": "ALL"
        },
        "ProvisionedThroughput": {
            "ReadCapacityUnits": 5,
            "WriteCapacityUnits": 5
        }
    }
]
```

Ausgabe:

```
{
```

```
    "TableDescription": {
        "AttributeDefinitions": [
            {
                "AttributeName": "Date",
                "AttributeType": "S"
            },
            {
                "AttributeName": "GameTitle",
                "AttributeType": "S"
            },
            {
                "AttributeName": "TopScore",
                "AttributeType": "N"
            },
            {
                "AttributeName": "UserId",
                "AttributeType": "S"
            }
        ],
        "TableName": "GameScores",
        "KeySchema": [
            {
                "AttributeName": "UserId",
                "KeyType": "HASH"
            },
            {
                "AttributeName": "GameTitle",
                "KeyType": "RANGE"
            }
        ],
        "TableStatus": "CREATING",
        "CreationDateTime": "2020-08-04T16:40:55.524000-07:00",
        "ProvisionedThroughput": {
            "NumberOfDecreasesToday": 0,
            "ReadCapacityUnits": 10,
            "WriteCapacityUnits": 5
        },
        "TableSizeBytes": 0,
        "ItemCount": 0,
        "TableArn": "arn:aws:dynamodb:us-west-2:123456789012:table/GameScores",
        "TableId": "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
        "GlobalSecondaryIndexes": [
            {
                "IndexName": "GameTitleIndex",
```

```
                "KeySchema": [
                    {
                        "AttributeName": "GameTitle",
                        "KeyType": "HASH"
                    },
                    {
                        "AttributeName": "TopScore",
                        "KeyType": "RANGE"
                    }
                ],
                "Projection": {
                    "ProjectionType": "ALL"
                },
                "IndexStatus": "CREATING",
                "ProvisionedThroughput": {
                    "NumberOfDecreasesToday": 0,
                    "ReadCapacityUnits": 10,
                    "WriteCapacityUnits": 5
                },
                "IndexSizeBytes": 0,
                "ItemCount": 0,
                "IndexArn": "arn:aws:dynamodb:us-west-2:123456789012:table/
GameScores/index/GameTitleIndex"
            },
            {
                "IndexName": "GameDateIndex",
                "KeySchema": [
                    {
                        "AttributeName": "GameTitle",
                        "KeyType": "HASH"
                    },
                    {
                        "AttributeName": "Date",
                        "KeyType": "RANGE"
                    }
                ],
                "Projection": {
                    "ProjectionType": "ALL"
                },
                "IndexStatus": "CREATING",
                "ProvisionedThroughput": {
                    "NumberOfDecreasesToday": 0,
                    "ReadCapacityUnits": 5,
                    "WriteCapacityUnits": 5
```

```
                },
                "IndexSizeBytes": 0,
                "ItemCount": 0,
                "IndexArn": "arn:aws:dynamodb:us-west-2:123456789012:table/
GameScores/index/GameDateIndex"
            }
        ]
    }
}
```

Weitere Informationen finden Sie unter Grundlegende Operationen für Tabellen im Entwicklerhandbuch zu Amazon DynamoDB.

Beispiel 7: So erstellen Sie eine Tabelle mit aktivierten Streams

Im folgenden Beispiel wird die Tabelle `GameScores` mit aktiviertem DynamoDB Streams aufgerufen. Sowohl neue als auch alte Images von jedem Element werden in den Stream geschrieben.

```
aws dynamodb create-table \
    --table-name GameScores \
    --attribute-
definitions AttributeName=UserId,AttributeType=S AttributeName=GameTitle,AttributeType=S
 \
    --key-
schema AttributeName=UserId,KeyType=HASH AttributeName=GameTitle,KeyType=RANGE \
    --provisioned-throughput ReadCapacityUnits=10,WriteCapacityUnits=5 \
    --stream-specification StreamEnabled=TRUE,StreamViewType=NEW_AND_OLD_IMAGES
```

Ausgabe:

```
{
    "TableDescription": {
        "AttributeDefinitions": [
            {
                "AttributeName": "GameTitle",
                "AttributeType": "S"
            },
            {
                "AttributeName": "UserId",
                "AttributeType": "S"
            }
```

```
        ],
        "TableName": "GameScores",
        "KeySchema": [
            {
                "AttributeName": "UserId",
                "KeyType": "HASH"
            },
            {
                "AttributeName": "GameTitle",
                "KeyType": "RANGE"
            }
        ],
        "TableStatus": "CREATING",
        "CreationDateTime": "2020-05-27T10:49:34.056000-07:00",
        "ProvisionedThroughput": {
            "NumberOfDecreasesToday": 0,
            "ReadCapacityUnits": 10,
            "WriteCapacityUnits": 5
        },
        "TableSizeBytes": 0,
        "ItemCount": 0,
        "TableArn": "arn:aws:dynamodb:us-west-2:123456789012:table/GameScores",
        "TableId": "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
        "StreamSpecification": {
            "StreamEnabled": true,
            "StreamViewType": "NEW_AND_OLD_IMAGES"
        },
        "LatestStreamLabel": "2020-05-27T17:49:34.056",
        "LatestStreamArn": "arn:aws:dynamodb:us-west-2:123456789012:table/
GameScores/stream/2020-05-27T17:49:34.056"
    }
}
```

Weitere Informationen finden Sie unter Grundlegende Operationen für Tabellen im Entwicklerhandbuch zu Amazon DynamoDB.

Beispiel 8: So erstellen Sie eine Tabelle mit aktiviertem Keys-Only-Stream

Im folgenden Beispiel wird die Tabelle `GameScores` mit aktiviertem DynamoDB Streams aufgerufen. Nur die Schlüsselattribute von geänderten Elementen werden in den Stream geschrieben.

```
aws dynamodb create-table \
```

```
    --table-name GameScores \
    --attribute-
definitions AttributeName=UserId,AttributeType=S AttributeName=GameTitle,AttributeType=S
 \
    --key-
schema AttributeName=UserId,KeyType=HASH AttributeName=GameTitle,KeyType=RANGE \
    --provisioned-throughput ReadCapacityUnits=10,WriteCapacityUnits=5 \
    --stream-specification StreamEnabled=TRUE,StreamViewType=KEYS_ONLY
```

Ausgabe:

```
{
    "TableDescription": {
        "AttributeDefinitions": [
            {
                "AttributeName": "GameTitle",
                "AttributeType": "S"
            },
            {
                "AttributeName": "UserId",
                "AttributeType": "S"
            }
        ],
        "TableName": "GameScores",
        "KeySchema": [
            {
                "AttributeName": "UserId",
                "KeyType": "HASH"
            },
            {
                "AttributeName": "GameTitle",
                "KeyType": "RANGE"
            }
        ],
        "TableStatus": "CREATING",
        "CreationDateTime": "2023-05-25T18:45:34.140000+00:00",
        "ProvisionedThroughput": {
            "NumberOfDecreasesToday": 0,
            "ReadCapacityUnits": 10,
            "WriteCapacityUnits": 5
        },
        "TableSizeBytes": 0,
        "ItemCount": 0,
```

```
        "TableArn": "arn:aws:dynamodb:us-west-2:123456789012:table/GameScores",
        "TableId": "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
        "StreamSpecification": {
            "StreamEnabled": true,
            "StreamViewType": "KEYS_ONLY"
        },
        "LatestStreamLabel": "2023-05-25T18:45:34.140",
        "LatestStreamArn": "arn:aws:dynamodb:us-west-2:123456789012:table/
GameScores/stream/2023-05-25T18:45:34.140",
        "DeletionProtectionEnabled": false
    }
}
```

Weitere Informationen finden Sie unter Ändern der Datenerfassung für DynamoDB Streams im Entwicklerhandbuch zu Amazon DynamoDB.

Beispiel 9: So erstellen Sie eine Tabelle mit der Klasse Standard Infrequent Access

Im folgenden Beispiel wird eine Tabelle mit dem Namen `GameScores` erstellt und die Tabellenklasse Standard-Infrequent Access (DynamoDB Standard-IA) zugewiesen. Diese Tabellenklasse ist dafür optimiert, dass Speicher der Hauptkostenfaktor ist.

```
aws dynamodb create-table \
    --table-name GameScores \
    --attribute-
definitions AttributeName=UserId,AttributeType=S AttributeName=GameTitle,AttributeType=S
 \
    --key-
schema AttributeName=UserId,KeyType=HASH AttributeName=GameTitle,KeyType=RANGE \
    --provisioned-throughput ReadCapacityUnits=10,WriteCapacityUnits=5 \
    --table-class STANDARD_INFREQUENT_ACCESS
```

Ausgabe:

```
{
    "TableDescription": {
        "AttributeDefinitions": [
            {
                "AttributeName": "GameTitle",
                "AttributeType": "S"
            },
            {
```

```
                "AttributeName": "UserId",
                "AttributeType": "S"
            }
        ],
        "TableName": "GameScores",
        "KeySchema": [
            {
                "AttributeName": "UserId",
                "KeyType": "HASH"
            },
            {
                "AttributeName": "GameTitle",
                "KeyType": "RANGE"
            }
        ],
        "TableStatus": "CREATING",
        "CreationDateTime": "2023-05-25T18:33:07.581000+00:00",
        "ProvisionedThroughput": {
            "NumberOfDecreasesToday": 0,
            "ReadCapacityUnits": 10,
            "WriteCapacityUnits": 5
        },
        "TableSizeBytes": 0,
        "ItemCount": 0,
        "TableArn": "arn:aws:dynamodb:us-west-2:123456789012:table/GameScores",
        "TableId": "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
        "TableClassSummary": {
            "TableClass": "STANDARD_INFREQUENT_ACCESS"
        },
        "DeletionProtectionEnabled": false
    }
}
```

Weitere Informationen finden Sie unter Tabellenklassen im Entwicklerhandbuch zu Amazon DynamoDB.

Beispiel 10: So erstellen Sie eine Tabelle mit aktiviertem Löschschutz

Im folgenden Beispiel wird eine Tabelle mit dem Namen `GameScores` erstellt und der Löschschutz aktiviert.

```
aws dynamodb create-table \
    --table-name GameScores \
```

```
    --attribute-
definitions AttributeName=UserId,AttributeType=S AttributeName=GameTitle,AttributeType=S
 \
    --key-
schema AttributeName=UserId,KeyType=HASH AttributeName=GameTitle,KeyType=RANGE \
    --provisioned-throughput ReadCapacityUnits=10,WriteCapacityUnits=5 \
    --deletion-protection-enabled
```

Ausgabe:

```
{
    "TableDescription": {
        "AttributeDefinitions": [
            {
                "AttributeName": "GameTitle",
                "AttributeType": "S"
            },
            {
                "AttributeName": "UserId",
                "AttributeType": "S"
            }
        ],
        "TableName": "GameScores",
        "KeySchema": [
            {
                "AttributeName": "UserId",
                "KeyType": "HASH"
            },
            {
                "AttributeName": "GameTitle",
                "KeyType": "RANGE"
            }
        ],
        "TableStatus": "CREATING",
        "CreationDateTime": "2023-05-25T23:02:17.093000+00:00",
        "ProvisionedThroughput": {
            "NumberOfDecreasesToday": 0,
            "ReadCapacityUnits": 10,
            "WriteCapacityUnits": 5
        },
        "TableSizeBytes": 0,
        "ItemCount": 0,
        "TableArn": "arn:aws:dynamodb:us-west-2:123456789012:table/GameScores",
```

```
        "TableId": "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
        "DeletionProtectionEnabled": true
    }
}
```

Weitere Informationen finden Sie unter Verwenden des Löschschutzes im Entwicklerhandbuch zu Amazon DynamoDB.

- Einzelheiten zur API finden Sie unter CreateTable AWS CLIBefehlsreferenz.

Go

SDK für Go V2

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
import (
 "context"
 "errors"
 "log"
 "time"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/attributevalue"
 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/expression"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb/types"
)

// TableBasics encapsulates the Amazon DynamoDB service actions used in the
 examples.
// It contains a DynamoDB service client that is used to act on the specified
 table.
type TableBasics struct {
 DynamoDbClient *dynamodb.Client
 TableName      string
}
```

```
// CreateMovieTable creates a DynamoDB table with a composite primary key defined
 as
// a string sort key named `title`, and a numeric partition key named `year`.
// This function uses NewTableExistsWaiter to wait for the table to be created by
// DynamoDB before it returns.
func (basics TableBasics) CreateMovieTable(ctx context.Context)
 (*types.TableDescription, error) {
 var tableDesc *types.TableDescription
 table, err := basics.DynamoDbClient.CreateTable(ctx, &dynamodb.CreateTableInput{
  AttributeDefinitions: []types.AttributeDefinition{{
   AttributeName: aws.String("year"),
   AttributeType: types.ScalarAttributeTypeN,
  }, {
   AttributeName: aws.String("title"),
   AttributeType: types.ScalarAttributeTypeS,
  }},
  KeySchema: []types.KeySchemaElement{{
   AttributeName: aws.String("year"),
   KeyType:       types.KeyTypeHash,
  }, {
   AttributeName: aws.String("title"),
   KeyType:       types.KeyTypeRange,
  }},
  TableName:   aws.String(basics.TableName),
  BillingMode: types.BillingModePayPerRequest,
 })
 if err != nil {
  log.Printf("Couldn't create table %v. Here's why: %v\n", basics.TableName, err)
 } else {
  waiter := dynamodb.NewTableExistsWaiter(basics.DynamoDbClient)
  err = waiter.Wait(ctx, &dynamodb.DescribeTableInput{
   TableName: aws.String(basics.TableName)}, 5*time.Minute)
  if err != nil {
   log.Printf("Wait for table exists failed. Here's why: %v\n", err)
  }
  tableDesc = table.TableDescription
  log.Printf("Ccreating table test")
 }
 return tableDesc, err
}
```

- Einzelheiten zur API finden Sie CreateTablein der AWS SDK für Go API-Referenz.

Java

SDK für Java 2.x

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
import software.amazon.awssdk.core.waiters.WaiterResponse;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.AttributeDefinition;
import software.amazon.awssdk.services.dynamodb.model.BillingMode;
import software.amazon.awssdk.services.dynamodb.model.CreateTableRequest;
import software.amazon.awssdk.services.dynamodb.model.CreateTableResponse;
import software.amazon.awssdk.services.dynamodb.model.DescribeTableRequest;
import software.amazon.awssdk.services.dynamodb.model.DescribeTableResponse;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.model.KeySchemaElement;
import software.amazon.awssdk.services.dynamodb.model.KeyType;
import software.amazon.awssdk.services.dynamodb.model.OnDemandThroughput;
import software.amazon.awssdk.services.dynamodb.model.ProvisionedThroughput;
import software.amazon.awssdk.services.dynamodb.model.ScalarAttributeType;
import software.amazon.awssdk.services.dynamodb.waiters.DynamoDbWaiter;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 * <p>
 * For more information, see the following documentation topic:
 * <p>
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 */
public class CreateTable {
```

```
   public static void main(String[] args) {
       final String usage = """

           Usage:
               <tableName> <key>

           Where:
               tableName - The Amazon DynamoDB table to create (for example,
Music3).
               key - The key for the Amazon DynamoDB table (for example,
Artist).
           """;

       if (args.length != 2) {
           System.out.println(usage);
           System.exit(1);
       }

       String tableName = args[0];
       String key = args[1];
       System.out.println("Creating an Amazon DynamoDB table " + tableName + "
with a simple primary key: " + key);
       Region region = Region.US_EAST_1;
       DynamoDbClient ddb = DynamoDbClient.builder()
           .region(region)
           .build();

       String result = createTable(ddb, tableName, key);
       System.out.println("New table is " + result);
       ddb.close();
   }

   public static String createTable(DynamoDbClient ddb, String tableName, String
key) {
       DynamoDbWaiter dbWaiter = ddb.waiter();
       CreateTableRequest request = CreateTableRequest.builder()
           .attributeDefinitions(AttributeDefinition.builder()
               .attributeName(key)
               .attributeType(ScalarAttributeType.S)
               .build())
           .keySchema(KeySchemaElement.builder()
               .attributeName(key)
               .keyType(KeyType.HASH)
               .build())
```

```
            .billingMode(BillingMode.PAY_PER_REQUEST) //  DynamoDB automatically
 scales based on traffic.
            .tableName(tableName)
            .build();

        String newTable;
        try {
            CreateTableResponse response = ddb.createTable(request);
            DescribeTableRequest tableRequest = DescribeTableRequest.builder()
                .tableName(tableName)
                .build();

            // Wait until the Amazon DynamoDB table is created.
            WaiterResponse<DescribeTableResponse> waiterResponse =
 dbWaiter.waitUntilTableExists(tableRequest);
            waiterResponse.matched().response().ifPresent(System.out::println);
            newTable = response.tableDescription().tableName();
            return newTable;

        } catch (DynamoDbException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
        return "";
    }
}
```

- Einzelheiten zur API finden Sie CreateTablein der AWS SDK for Java 2.x API-Referenz.

JavaScript

SDK für JavaScript (v3)

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
import { CreateTableCommand, DynamoDBClient } from "@aws-sdk/client-dynamodb";
```

```
const client = new DynamoDBClient({});

export const main = async () => {
  const command = new CreateTableCommand({
    TableName: "EspressoDrinks",
    // For more information about data types,
    // see https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/
HowItWorks.NamingRulesDataTypes.html#HowItWorks.DataTypes and
    // https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/
Programming.LowLevelAPI.html#Programming.LowLevelAPI.DataTypeDescriptors
    AttributeDefinitions: [
      {
        AttributeName: "DrinkName",
        AttributeType: "S",
      },
    ],
    KeySchema: [
      {
        AttributeName: "DrinkName",
        KeyType: "HASH",
      },
    ],
    BillingMode: "PAY_PER_REQUEST",
  });

  const response = await client.send(command);
  console.log(response);
  return response;
};
```

- Weitere Informationen finden Sie im AWS SDK für JavaScript -Entwicklerhandbuch.
- Einzelheiten zur API finden Sie CreateTablein der AWS SDK für JavaScript API-Referenz.

SDK für JavaScript (v2)

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create the DynamoDB service object
var ddb = new AWS.DynamoDB({ apiVersion: "2012-08-10" });

var params = {
  AttributeDefinitions: [
    {
      AttributeName: "CUSTOMER_ID",
      AttributeType: "N",
    },
    {
      AttributeName: "CUSTOMER_NAME",
      AttributeType: "S",
    },
  ],
  KeySchema: [
    {
      AttributeName: "CUSTOMER_ID",
      KeyType: "HASH",
    },
    {
      AttributeName: "CUSTOMER_NAME",
      KeyType: "RANGE",
    },
  ],
  ProvisionedThroughput: {
    ReadCapacityUnits: 1,
    WriteCapacityUnits: 1,
  },
  TableName: "CUSTOMER_LIST",
  StreamSpecification: {
    StreamEnabled: false,
  },
};

// Call DynamoDB to create the table
ddb.createTable(params, function (err, data) {
  if (err) {
    console.log("Error", err);
```

```
  } else {
    console.log("Table Created", data);
  }
});
```

- Weitere Informationen finden Sie im AWS SDK für JavaScript -Entwicklerhandbuch.
- Einzelheiten zur API finden Sie CreateTablein der AWS SDK für JavaScript API-Referenz.

Kotlin

SDK für Kotlin

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
suspend fun createNewTable(
    tableNameVal: String,
    key: String,
): String? {
    val attDef =
        AttributeDefinition {
            attributeName = key
            attributeType = ScalarAttributeType.S
        }

    val keySchemaVal =
        KeySchemaElement {
            attributeName = key
            keyType = KeyType.Hash
        }

    val request =
        CreateTableRequest {
            attributeDefinitions = listOf(attDef)
            keySchema = listOf(keySchemaVal)
            billingMode = BillingMode.PayPerRequest
            tableName = tableNameVal
```

```
        }

    DynamoDbClient.fromEnvironment { region = "us-east-1" }.use { ddb ->
        var tableArn: String
        val response = ddb.createTable(request)
        ddb.waitUntilTableExists {
            // suspend call
            tableName = tableNameVal
        }
        tableArn = response.tableDescription!!.tableArn.toString()
        println("Table $tableArn is ready")
        return tableArn
    }
}
```

- Einzelheiten zur API finden Sie CreateTablein der API-Referenz zum AWS SDK für Kotlin.

PHP

SDK für PHP

> **Note**
>
> Es gibt noch mehr dazu. GitHub Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

Erstellen Sie eine -Tabelle.

```
        $tableName = "ddb_demo_table_$uuid";
        $service->createTable(
            $tableName,
            [
                new DynamoDBAttribute('year', 'N', 'HASH'),
                new DynamoDBAttribute('title', 'S', 'RANGE')
            ]
        );

    public function createTable(string $tableName, array $attributes)
    {
        $keySchema = [];
```

```
        $attributeDefinitions = [];
        foreach ($attributes as $attribute) {
            if (is_a($attribute, DynamoDBAttribute::class)) {
                $keySchema[] = ['AttributeName' => $attribute->AttributeName,
'KeyType' => $attribute->KeyType];
                $attributeDefinitions[] =
                    ['AttributeName' => $attribute->AttributeName,
'AttributeType' => $attribute->AttributeType];
            }
        }

        $this->dynamoDbClient->createTable([
            'TableName' => $tableName,
            'KeySchema' => $keySchema,
            'AttributeDefinitions' => $attributeDefinitions,
            'ProvisionedThroughput' => ['ReadCapacityUnits' => 10,
'WriteCapacityUnits' => 10],
        ]);
    }
```

- Einzelheiten zur API finden Sie CreateTablein der AWS SDK für PHP API-Referenz.

PowerShell

Tools für PowerShell V4

Beispiel 1: In diesem Beispiel wird eine Tabelle mit dem Namen Thread erstellt, deren Primärschlüssel aus 'ForumName' (Schlüsseltyp-Hash) und 'Subject' (Schlüsseltypbereich) besteht. Das zur Erstellung der Tabelle verwendete Schema kann wie gezeigt an jedes Cmdlet übergeben oder mit dem Parameter „-Schema“ angegeben werden.

```
$schema = New-DDBTableSchema
$schema | Add-DDBKeySchema -KeyName "ForumName" -KeyDataType "S"
$schema | Add-DDBKeySchema -KeyName "Subject" -KeyType RANGE -KeyDataType "S"
$schema | New-DDBTable -TableName "Thread" -ReadCapacity 10 -WriteCapacity 5
```

Ausgabe:

```
AttributeDefinitions   : {ForumName, Subject}
TableName              : Thread
```

```
KeySchema              : {ForumName, Subject}
TableStatus            : CREATING
CreationDateTime       : 10/28/2013 4:39:49 PM
ProvisionedThroughput  : Amazon.DynamoDBv2.Model.ProvisionedThroughputDescription
TableSizeBytes         : 0
ItemCount              : 0
LocalSecondaryIndexes  : {}
```

Beispiel 2: In diesem Beispiel wird eine Tabelle mit dem Namen Thread erstellt, deren Primärschlüssel aus 'ForumName' (Schlüsseltyp-Hash) und 'Subject' (Schlüsseltypbereich) besteht. Ein lokaler sekundärer Index wird ebenfalls definiert. Der Schlüssel des lokalen sekundären Indexes wird automatisch anhand des primären Hashschlüssels in der Tabelle (ForumName) festgelegt. Das zur Erstellung der Tabelle verwendete Schema kann wie gezeigt an jedes Cmdlet übergeben oder mit dem Parameter „-Schema“ angegeben werden.

```
$schema = New-DDBTableSchema
$schema | Add-DDBKeySchema -KeyName "ForumName" -KeyDataType "S"
$schema | Add-DDBKeySchema -KeyName "Subject" -KeyDataType "S"
$schema | Add-DDBIndexSchema -IndexName "LastPostIndex" -RangeKeyName
 "LastPostDateTime" -RangeKeyDataType "S" -ProjectionType "keys_only"
$schema | New-DDBTable -TableName "Thread" -ReadCapacity 10 -WriteCapacity 5
```

Ausgabe:

```
AttributeDefinitions   : {ForumName, LastPostDateTime, Subject}
TableName              : Thread
KeySchema              : {ForumName, Subject}
TableStatus            : CREATING
CreationDateTime       : 10/28/2013 4:39:49 PM
ProvisionedThroughput  : Amazon.DynamoDBv2.Model.ProvisionedThroughputDescription
TableSizeBytes         : 0
ItemCount              : 0
LocalSecondaryIndexes  : {LastPostIndex}
```

Beispiel 3: Dieses Beispiel zeigt, wie eine einzelne Pipeline verwendet wird, um eine Tabelle mit dem Namen Thread zu erstellen, deren Primärschlüssel aus 'ForumName' (Schlüsseltyp Hash) und 'Subject' (Schlüsseltypbereich) und einem lokalen sekundären Index besteht. Mit den Optionen „Add- DDBKey Schema“ und DDBIndex „Add- Schema“ wird ein neues TableSchema Objekt für Sie erstellt, falls keines über die Pipeline oder den Parameter -Schema bereitgestellt wird.

```
New-DDBTableSchema |
  Add-DDBKeySchema -KeyName "ForumName" -KeyDataType "S" |
  Add-DDBKeySchema -KeyName "Subject" -KeyDataType "S" |
  Add-DDBIndexSchema -IndexName "LastPostIndex" `
                     -RangeKeyName "LastPostDateTime" `
                     -RangeKeyDataType "S" `
                     -ProjectionType "keys_only" |
  New-DDBTable -TableName "Thread" -ReadCapacity 10 -WriteCapacity 5
```

Ausgabe:

```
AttributeDefinitions   : {ForumName, LastPostDateTime, Subject}
TableName              : Thread
KeySchema              : {ForumName, Subject}
TableStatus            : CREATING
CreationDateTime       : 10/28/2013 4:39:49 PM
ProvisionedThroughput  : Amazon.DynamoDBv2.Model.ProvisionedThroughputDescription
TableSizeBytes         : 0
ItemCount              : 0
LocalSecondaryIndexes  : {LastPostIndex}
```

- Einzelheiten zur API finden Sie unter CreateTable AWS -Tools für PowerShellCmdlet-Referenz (V4).

Tools für V5 PowerShell

Beispiel 1: In diesem Beispiel wird eine Tabelle mit dem Namen Thread erstellt, deren Primärschlüssel aus 'ForumName' (Schlüsseltyp-Hash) und 'Subject' (Schlüsseltypbereich) besteht. Das zur Erstellung der Tabelle verwendete Schema kann wie gezeigt an jedes Cmdlet übergeben oder mit dem Parameter „-Schema“ angegeben werden.

```
$schema = New-DDBTableSchema
$schema | Add-DDBKeySchema -KeyName "ForumName" -KeyDataType "S"
$schema | Add-DDBKeySchema -KeyName "Subject" -KeyType RANGE -KeyDataType "S"
$schema | New-DDBTable -TableName "Thread" -ReadCapacity 10 -WriteCapacity 5
```

Ausgabe:

```
AttributeDefinitions   : {ForumName, Subject}
TableName              : Thread
KeySchema              : {ForumName, Subject}
TableStatus            : CREATING
```

```
CreationDateTime       : 10/28/2013 4:39:49 PM
ProvisionedThroughput  : Amazon.DynamoDBv2.Model.ProvisionedThroughputDescription
TableSizeBytes         : 0
ItemCount              : 0
LocalSecondaryIndexes  : {}
```

Beispiel 2: In diesem Beispiel wird eine Tabelle mit dem Namen Thread erstellt, deren Primärschlüssel aus 'ForumName' (Schlüsseltyp-Hash) und 'Subject' (Schlüsseltypbereich) besteht. Ein lokaler sekundärer Index wird ebenfalls definiert. Der Schlüssel des lokalen sekundären Indexes wird automatisch anhand des primären Hashschlüssels in der Tabelle (ForumName) festgelegt. Das zur Erstellung der Tabelle verwendete Schema kann wie gezeigt an jedes Cmdlet übergeben oder mit dem Parameter „-Schema“ angegeben werden.

```
$schema = New-DDBTableSchema
$schema | Add-DDBKeySchema -KeyName "ForumName" -KeyDataType "S"
$schema | Add-DDBKeySchema -KeyName "Subject" -KeyDataType "S"
$schema | Add-DDBIndexSchema -IndexName "LastPostIndex" -RangeKeyName
 "LastPostDateTime" -RangeKeyDataType "S" -ProjectionType "keys_only"
$schema | New-DDBTable -TableName "Thread" -ReadCapacity 10 -WriteCapacity 5
```

Ausgabe:

```
AttributeDefinitions   : {ForumName, LastPostDateTime, Subject}
TableName              : Thread
KeySchema              : {ForumName, Subject}
TableStatus            : CREATING
CreationDateTime       : 10/28/2013 4:39:49 PM
ProvisionedThroughput  : Amazon.DynamoDBv2.Model.ProvisionedThroughputDescription
TableSizeBytes         : 0
ItemCount              : 0
LocalSecondaryIndexes  : {LastPostIndex}
```

Beispiel 3: Dieses Beispiel zeigt, wie eine einzelne Pipeline verwendet wird, um eine Tabelle mit dem Namen Thread zu erstellen, deren Primärschlüssel aus 'ForumName' (Schlüsseltyp Hash) und 'Subject' (Schlüsseltypbereich) und einem lokalen sekundären Index besteht. Mit den Optionen „Add- DDBKey Schema“ und DDBIndex „Add- Schema“ wird ein neues TableSchema Objekt für Sie erstellt, falls keines über die Pipeline oder den Parameter -Schema bereitgestellt wird.

```
New-DDBTableSchema |
```

```
  Add-DDBKeySchema -KeyName "ForumName" -KeyDataType "S" |
  Add-DDBKeySchema -KeyName "Subject" -KeyDataType "S" |
  Add-DDBIndexSchema -IndexName "LastPostIndex" `
                     -RangeKeyName "LastPostDateTime" `
                     -RangeKeyDataType "S" `
                     -ProjectionType "keys_only" |
  New-DDBTable -TableName "Thread" -ReadCapacity 10 -WriteCapacity 5
```

Ausgabe:

```
AttributeDefinitions   : {ForumName, LastPostDateTime, Subject}
TableName              : Thread
KeySchema              : {ForumName, Subject}
TableStatus            : CREATING
CreationDateTime       : 10/28/2013 4:39:49 PM
ProvisionedThroughput  : Amazon.DynamoDBv2.Model.ProvisionedThroughputDescription
TableSizeBytes         : 0
ItemCount              : 0
LocalSecondaryIndexes  : {LastPostIndex}
```

- Einzelheiten zur API finden Sie unter CreateTable AWS -Tools für PowerShellCmdlet-Referenz (V5).

Python

SDK für Python (Boto3)

> **Note**
>
> Es gibt noch mehr dazu. GitHub Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

Erstellen Sie eine Tabelle zum Speichern von Filmdaten.

```
class Movies:
    """Encapsulates an Amazon DynamoDB table of movie data.

    Example data structure for a movie record in this table:
        {
            "year": 1999,
```

```
            "title": "For Love of the Game",
            "info": {
                "directors": ["Sam Raimi"],
                "release_date": "1999-09-15T00:00:00Z",
                "rating": 6.3,
                "plot": "A washed up pitcher flashes through his career.",
                "rank": 4987,
                "running_time_secs": 8220,
                "actors": [
                    "Kevin Costner",
                    "Kelly Preston",
                    "John C. Reilly"
                ]
            }
        }
    """

    def __init__(self, dyn_resource):
        """
        :param dyn_resource: A Boto3 DynamoDB resource.
        """
        self.dyn_resource = dyn_resource
        # The table variable is set during the scenario in the call to
        # 'exists' if the table exists. Otherwise, it is set by 'create_table'.
        self.table = None


    def create_table(self, table_name):
        """
        Creates an Amazon DynamoDB table that can be used to store movie data.
        The table uses the release year of the movie as the partition key and the
        title as the sort key.

        :param table_name: The name of the table to create.
        :return: The newly created table.
        """
        try:
            self.table = self.dyn_resource.create_table(
                TableName=table_name,
                KeySchema=[
                    {"AttributeName": "year", "KeyType": "HASH"},  # Partition
key
                    {"AttributeName": "title", "KeyType": "RANGE"},  # Sort key
                ],
```

```
                AttributeDefinitions=[
                    {"AttributeName": "year", "AttributeType": "N"},
                    {"AttributeName": "title", "AttributeType": "S"},
                ],
                BillingMode='PAY_PER_REQUEST',
            )
            self.table.wait_until_exists()
        except ClientError as err:
            logger.error(
                "Couldn't create table %s. Here's why: %s: %s",
                table_name,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return self.table
```

- Einzelheiten zur API finden Sie CreateTablein AWS SDK for Python (Boto3) API Reference.

Ruby

SDK für Ruby

> **Note**
>
> Es gibt noch mehr dazu. GitHub Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
# Encapsulates an Amazon DynamoDB table of movie data.
class Scaffold
  attr_reader :dynamo_resource, :table_name, :table

  def initialize(table_name)
    client = Aws::DynamoDB::Client.new(region: 'us-east-1')
    @dynamo_resource = Aws::DynamoDB::Resource.new(client: client)
    @table_name = table_name
    @table = nil
    @logger = Logger.new($stdout)
```

```
    @logger.level = Logger::DEBUG
  end

  # Creates an Amazon DynamoDB table that can be used to store movie data.
  # The table uses the release year of the movie as the partition key and the
  # title as the sort key.
  #
  # @param table_name [String] The name of the table to create.
  # @return [Aws::DynamoDB::Table] The newly created table.
  def create_table(table_name)
    @table = @dynamo_resource.create_table(
      table_name: table_name,
      key_schema: [
        { attribute_name: 'year', key_type: 'HASH' }, # Partition key
        { attribute_name: 'title', key_type: 'RANGE' } # Sort key
      ],
      attribute_definitions: [
        { attribute_name: 'year', attribute_type: 'N' },
        { attribute_name: 'title', attribute_type: 'S' }
      ],
      billing_mode: 'PAY_PER_REQUEST'
    )
    @dynamo_resource.client.wait_until(:table_exists, table_name: table_name)
    @table
  rescue Aws::DynamoDB::Errors::ServiceError => e
    @logger.error("Failed create table #{table_name}:\n#{e.code}: #{e.message}")
    raise
  end
```

- Einzelheiten zur API finden Sie CreateTablein der AWS SDK für Ruby API-Referenz.

Rust

SDK für Rust

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
pub async fn create_table(
    client: &Client,
    table: &str,
    key: &str,
) -> Result<CreateTableOutput, Error> {
    let a_name: String = key.into();
    let table_name: String = table.into();

    let ad = AttributeDefinition::builder()
        .attribute_name(&a_name)
        .attribute_type(ScalarAttributeType::S)
        .build()
        .map_err(Error::BuildError)?;

    let ks = KeySchemaElement::builder()
        .attribute_name(&a_name)
        .key_type(KeyType::Hash)
        .build()
        .map_err(Error::BuildError)?;

    let create_table_response = client
        .create_table()
        .table_name(table_name)
        .key_schema(ks)
        .attribute_definitions(ad)
        .billing_mode(BillingMode::PayPerRequest)
        .send()
        .await;

    match create_table_response {
        Ok(out) => {
            println!("Added table {} with key {}", table, key);
            Ok(out)
        }
        Err(e) => {
            eprintln!("Got an error creating table:");
            eprintln!("{}", e);
            Err(Error::unhandled(e))
        }
    }
}
```

- Einzelheiten zur API finden Sie CreateTablein der API-Referenz zum AWS SDK für Rust.

SAP ABAP

SDK für SAP ABAP

> **Note**
>
> Es gibt noch mehr dazu. GitHub Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
    TRY.
        DATA(lt_keyschema) = VALUE /aws1/cl_dynkeyschemaelement=>tt_keyschema(
          ( NEW /aws1/cl_dynkeyschemaelement( iv_attributename = 'year'
                                              iv_keytype = 'HASH' ) )
          ( NEW /aws1/cl_dynkeyschemaelement( iv_attributename = 'title'
                                              iv_keytype = 'RANGE' ) ) ).
        DATA(lt_attributedefinitions) = VALUE /aws1/
cl_dynattributedefn=>tt_attributedefinitions(
          ( NEW /aws1/cl_dynattributedefn( iv_attributename = 'year'
                                           iv_attributetype = 'N' ) )
          ( NEW /aws1/cl_dynattributedefn( iv_attributename = 'title'
                                           iv_attributetype = 'S' ) ) ).

        " Adjust read/write capacities as desired.
        DATA(lo_dynprovthroughput)  = NEW /aws1/cl_dynprovthroughput(
          iv_readcapacityunits = 5
          iv_writecapacityunits = 5 ).
        oo_result = lo_dyn->createtable(
          it_keyschema = lt_keyschema
          iv_tablename = iv_table_name
          it_attributedefinitions = lt_attributedefinitions
          io_provisionedthroughput = lo_dynprovthroughput ).
        " Table creation can take some time. Wait till table exists before
 returning.
        lo_dyn->get_waiter( )->tableexists(
          iv_max_wait_time = 200
          iv_tablename     = iv_table_name ).
        MESSAGE 'DynamoDB Table' && iv_table_name && 'created.' TYPE 'I'.
        " This exception can happen if the table already exists.
      CATCH /aws1/cx_dynresourceinuseex INTO DATA(lo_resourceinuseex).
```

```
        DATA(lv_error) = |"{ lo_resourceinuseex->av_err_code }" -
{ lo_resourceinuseex->av_err_msg }|.
        MESSAGE lv_error TYPE 'E'.
    ENDTRY.
```

- Einzelheiten zur API finden Sie CreateTablein der API-Referenz zum AWS SDK für SAP ABAP.

Swift

SDK für Swift

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
import AWSDynamoDB


    ///
    /// Create a movie table in the Amazon DynamoDB data store.
    ///
    private func createTable() async throws {
        do {
            guard let client = self.ddbClient else {
                throw MoviesError.UninitializedClient
            }

            let input = CreateTableInput(
                attributeDefinitions: [
                    DynamoDBClientTypes.AttributeDefinition(attributeName:
 "year", attributeType: .n),
                    DynamoDBClientTypes.AttributeDefinition(attributeName:
 "title", attributeType: .s)
                ],
                billingMode: DynamoDBClientTypes.BillingMode.payPerRequest,
                keySchema: [
```

```
                    DynamoDBClientTypes.KeySchemaElement(attributeName: "year",
keyType: .hash),
                    DynamoDBClientTypes.KeySchemaElement(attributeName: "title",
keyType: .range)
                ],
                tableName: self.tableName
            )
            let output = try await client.createTable(input: input)
            if output.tableDescription == nil {
                throw MoviesError.TableNotFound
            }
        } catch {
            print("ERROR: createTable:", dump(error))
            throw error
        }
    }
```

- Einzelheiten zur API finden Sie CreateTablein der API-Referenz zum AWS SDK für Swift.

Eine vollständige Liste der AWS SDK-Entwicklerhandbücher und Codebeispiele finden Sie unterDynamoDB mit einem SDK verwenden AWS. Dieses Thema enthält auch Informationen zu den ersten Schritten und Details zu früheren SDK-Versionen.

Verwendung **`DeleteItem`** mit einem AWS SDK oder CLI

Die folgenden Code-Beispiele zeigen, wie `DeleteItem` verwendet wird.

Aktionsbeispiele sind Codeauszüge aus größeren Programmen und müssen im Kontext ausgeführt werden. Sie können diese Aktion in den folgenden Codebeispielen im Kontext sehen:

- Kennenlernen der Grundlagen
- Verwenden von bedingten Operationen

.NET

SDK für .NET (v4)

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
    /// <summary>
    /// Deletes a single item from a DynamoDB table.
    /// </summary>
    /// <param name="tableName">The name of the table from which the item
    /// will be deleted.</param>
    /// <param name="movieToDelete">A movie object containing the title and
    /// year of the movie to delete.</param>
    /// <returns>A Boolean value indicating the success or failure of the
    /// delete operation.</returns>
    public async Task<bool> DeleteItemAsync(
        string tableName,
        Movie movieToDelete)
    {
        try
        {
            var key = new Dictionary<string, AttributeValue>
            {
                ["title"] = new AttributeValue { S = movieToDelete.Title },
                ["year"] = new AttributeValue { N =
movieToDelete.Year.ToString() },
            };

            var request = new DeleteItemRequest { TableName = tableName, Key =
key, };

            await _amazonDynamoDB.DeleteItemAsync(request);
            return true;
        }
        catch (ResourceNotFoundException ex)
        {
            Console.WriteLine($"Table {tableName} was not found. {ex.Message}");
```

```
            return false;
        }
        catch (AmazonDynamoDBException ex)
        {
            Console.WriteLine($"An Amazon DynamoDB error occurred while deleting
item. {ex.Message}");
            throw;
        }
        catch (Exception ex)
        {
            Console.WriteLine($"An error occurred while deleting item.
{ex.Message}");
            throw;
        }
    }
```

- Einzelheiten zur API finden Sie DeleteItemin der AWS SDK für .NET API-Referenz.

Bash

AWS CLI mit Bash-Skript

> **Note**
>
> Es gibt noch mehr dazu. GitHub Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
###############################################################################
# function dynamodb_delete_item
#
# This function deletes an item from a DynamoDB table.
#
# Parameters:
#       -n table_name  -- The name of the table.
#       -k keys  -- Path to json file containing the keys that identify the item
 to delete.
#
#  Returns:
#       0 - If successful.
```

```
#       1 - If it fails.
###############################################################################
function dynamodb_delete_item() {
  local table_name keys response
  local option OPTARG # Required to use getopts command in a function.

  # ######################################
  # Function usage explanation
  ######################################
  function usage() {
    echo "function dynamodb_delete_item"
    echo "Delete an item from a DynamoDB table."
    echo " -n table_name  -- The name of the table."
    echo " -k keys  -- Path to json file containing the keys that identify the
 item to delete."
    echo ""
  }
  while getopts "n:k:h" option; do
    case "${option}" in
      n) table_name="${OPTARG}" ;;
      k) keys="${OPTARG}" ;;
      h)
        usage
        return 0
        ;;
      \?)
        echo "Invalid parameter"
        usage
        return 1
        ;;
    esac
  done
  export OPTIND=1

  if [[ -z "$table_name" ]]; then
    errecho "ERROR: You must provide a table name with the -n parameter."
    usage
    return 1
  fi

  if [[ -z "$keys" ]]; then
    errecho "ERROR: You must provide a keys json file path the -k parameter."
    usage
    return 1
```

```
  fi

  iecho "Parameters:\n"
  iecho "    table_name:   $table_name"
  iecho "    keys:   $keys"
  iecho ""

  response=$(aws dynamodb delete-item \
    --table-name "$table_name" \
    --key file://"$keys")

  local error_code=${?}

  if [[ $error_code -ne 0 ]]; then
    aws_cli_error_log $error_code
    errecho "ERROR: AWS reports delete-item operation failed.$response"
    return 1
  fi

  return 0

}
```

Die in diesem Beispiel verwendeten Hilfsfunktionen.

```
###############################################################################
# function iecho
#
# This function enables the script to display the specified text only if
# the global variable $VERBOSE is set to true.
###############################################################################
function iecho() {
  if [[ $VERBOSE == true ]]; then
    echo "$@"
  fi
}

###############################################################################
# function errecho
#
# This function outputs everything sent to it to STDERR (standard error output).
###############################################################################
```

```
function errecho() {
  printf "%s\n" "$*" 1>&2
}

###############################################################################
# function aws_cli_error_log()
#
# This function is used to log the error messages from the AWS CLI.
#
# See https://docs.aws.amazon.com/cli/latest/topic/return-codes.html#cli-aws-
help-return-codes.
#
# The function expects the following argument:
#         $1 - The error code returned by the AWS CLI.
#
#  Returns:
#          0: - Success.
#
###############################################################################
function aws_cli_error_log() {
  local err_code=$1
  errecho "Error code : $err_code"
  if [ "$err_code" == 1 ]; then
    errecho "  One or more S3 transfers failed."
  elif [ "$err_code" == 2 ]; then
    errecho "  Command line failed to parse."
  elif [ "$err_code" == 130 ]; then
    errecho "  Process received SIGINT."
  elif [ "$err_code" == 252 ]; then
    errecho "  Command syntax invalid."
  elif [ "$err_code" == 253 ]; then
    errecho "  The system environment or configuration was invalid."
  elif [ "$err_code" == 254 ]; then
    errecho "  The service returned an error."
  elif [ "$err_code" == 255 ]; then
    errecho "  255 is a catch-all error."
  fi

  return 0
}
```

- Einzelheiten zur API finden Sie DeleteItemin der AWS CLI Befehlsreferenz.

C++

SDK für C++

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
//! Delete an item from an Amazon DynamoDB table.
/*!
  \sa deleteItem()
  \param tableName: The table name.
  \param partitionKey: The partition key.
  \param partitionValue: The value for the partition key.
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
 */

bool AwsDoc::DynamoDB::deleteItem(const Aws::String &tableName,
                                  const Aws::String &partitionKey,
                                  const Aws::String &partitionValue,
                                  const Aws::Client::ClientConfiguration
 &clientConfiguration) {
    Aws::DynamoDB::DynamoDBClient dynamoClient(clientConfiguration);

    Aws::DynamoDB::Model::DeleteItemRequest request;

    request.AddKey(partitionKey,
                   Aws::DynamoDB::Model::AttributeValue().SetS(partitionValue));
    request.SetTableName(tableName);

    const Aws::DynamoDB::Model::DeleteItemOutcome &outcome =
 dynamoClient.DeleteItem(
            request);
    if (outcome.IsSuccess()) {
        std::cout << "Item \"" << partitionValue << "\" deleted!" << std::endl;
    }
    else {
        std::cerr << "Failed to delete item: " << outcome.GetError().GetMessage()
                  << std::endl;
```

```
        return false;
    }

    return waitTableActive(tableName, dynamoClient);
}
```

Code, der darauf wartet, dass die Tabelle aktiv wird.

```
//! Query a newly created DynamoDB table until it is active.
/*!
  \sa waitTableActive()
  \param waitTableActive: The DynamoDB table's name.
  \param dynamoClient: A DynamoDB client.
  \return bool: Function succeeded.
*/
bool AwsDoc::DynamoDB::waitTableActive(const Aws::String &tableName,
                                       const Aws::DynamoDB::DynamoDBClient
 &dynamoClient) {

    // Repeatedly call DescribeTable until table is ACTIVE.
    const int MAX_QUERIES = 20;
    Aws::DynamoDB::Model::DescribeTableRequest request;
    request.SetTableName(tableName);

    int count = 0;
    while (count < MAX_QUERIES) {
        const Aws::DynamoDB::Model::DescribeTableOutcome &result =
 dynamoClient.DescribeTable(
                request);
        if (result.IsSuccess()) {
            Aws::DynamoDB::Model::TableStatus status =
 result.GetResult().GetTable().GetTableStatus();

            if (Aws::DynamoDB::Model::TableStatus::ACTIVE != status) {
                std::this_thread::sleep_for(std::chrono::seconds(1));
            }
            else {
                return true;
            }
        }
        else {
            std::cerr << "Error DynamoDB::waitTableActive "
```

```
                        << result.GetError().GetMessage() << std::endl;
            return false;
        }
        count++;
    }
    return false;
}
```

- Einzelheiten zur API finden Sie DeleteItemin der AWS SDK für C++ API-Referenz.

CLI

AWS CLI

Beispiel 1: So löschen Sie ein Element

Im folgenden Beispiel für `delete-item` wird ein Element aus der `MusicCollection`-Tabelle gelöscht und es werden Details zu dem gelöschten Element und der von der Anforderung verwendeten Kapazität angefordert.

```
aws dynamodb delete-item \
    --table-name MusicCollection \
    --key file://key.json \
    --return-values ALL_OLD \
    --return-consumed-capacity TOTAL \
    --return-item-collection-metrics SIZE
```

Inhalt von `key.json`:

```
{
    "Artist": {"S": "No One You Know"},
    "SongTitle": {"S": "Scared of My Shadow"}
}
```

Ausgabe:

```
{
    "Attributes": {
        "AlbumTitle": {
            "S": "Blue Sky Blues"
        },
```

```
        "Artist": {
            "S": "No One You Know"
        },
        "SongTitle": {
            "S": "Scared of My Shadow"
        }
    },
    "ConsumedCapacity": {
        "TableName": "MusicCollection",
        "CapacityUnits": 2.0
    },
    "ItemCollectionMetrics": {
        "ItemCollectionKey": {
            "Artist": {
                "S": "No One You Know"
            }
        },
        "SizeEstimateRangeGB": [
            0.0,
            1.0
        ]
    }
}
```

Weitere Informationen finden Sie unter Schreiben eines Elements im Entwicklerhandbuch zu Amazon DynamoDB.

Beispiel 2: So löschen Sie ein Element bedingungsabhängig

Im folgenden Beispiel wird ein Element nur dann aus der `ProductCatalog`-Tabelle gelöscht, wenn `ProductCategory` entweder `Sporting Goods` oder `Gardening Supplies` ist und der Preis zwischen 500 und 600 liegt. Es werden Details zu dem gelöschten Element zurückgegeben.

```
aws dynamodb delete-item \
    --table-name ProductCatalog \
    --key '{"Id":{"N":"456"}}' \
    --condition-expression "(ProductCategory IN (:cat1, :cat2)) and (#P
 between :lo and :hi)" \
    --expression-attribute-names file://names.json \
    --expression-attribute-values file://values.json \
    --return-values ALL_OLD
```

Inhalt von `names.json`:

```
{
    "#P": "Price"
}
```

Inhalt von `values.json`:

```
{
    ":cat1": {"S": "Sporting Goods"},
    ":cat2": {"S": "Gardening Supplies"},
    ":lo": {"N": "500"},
    ":hi": {"N": "600"}
}
```

Ausgabe:

```
{
    "Attributes": {
        "Id": {
            "N": "456"
        },
        "Price": {
            "N": "550"
        },
        "ProductCategory": {
            "S": "Sporting Goods"
        }
    }
}
```

Weitere Informationen finden Sie unter Schreiben eines Elements im Entwicklerhandbuch zu Amazon DynamoDB.

- Einzelheiten zur API finden Sie DeleteItemin der AWS CLI Befehlsreferenz.

Go

SDK für Go V2

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
import (
 "context"
 "errors"
 "log"
 "time"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/attributevalue"
 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/expression"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb/types"
)

// TableBasics encapsulates the Amazon DynamoDB service actions used in the
 examples.
// It contains a DynamoDB service client that is used to act on the specified
 table.
type TableBasics struct {
 DynamoDbClient *dynamodb.Client
 TableName      string
}



// DeleteMovie removes a movie from the DynamoDB table.
func (basics TableBasics) DeleteMovie(ctx context.Context, movie Movie) error {
 _, err := basics.DynamoDbClient.DeleteItem(ctx, &dynamodb.DeleteItemInput{
  TableName: aws.String(basics.TableName), Key: movie.GetKey(),
 })
 if err != nil {
```

```
  log.Printf("Couldn't delete %v from the table. Here's why: %v\n", movie.Title,
 err)
 }
 return err
}
```

Definieren Sie eine Movie-Struktur, die in diesem Beispiel verwendet wird.

```
import (
 "archive/zip"
 "bytes"
 "encoding/json"
 "fmt"
 "io"
 "log"
 "net/http"

 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/attributevalue"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb/types"
)

// Movie encapsulates data about a movie. Title and Year are the composite
 primary key
// of the movie in Amazon DynamoDB. Title is the sort key, Year is the partition
 key,
// and Info is additional data.
type Movie struct {
 Title string                `dynamodbav:"title"`
 Year  int                   `dynamodbav:"year"`
 Info  map[string]interface{} `dynamodbav:"info"`
}

// GetKey returns the composite primary key of the movie in a format that can be
// sent to DynamoDB.
func (movie Movie) GetKey() map[string]types.AttributeValue {
 title, err := attributevalue.Marshal(movie.Title)
 if err != nil {
  panic(err)
 }
 year, err := attributevalue.Marshal(movie.Year)
```

```
 if err != nil {
  panic(err)
 }
 return map[string]types.AttributeValue{"title": title, "year": year}
}

// String returns the title, year, rating, and plot of a movie, formatted for the
 example.
func (movie Movie) String() string {
 return fmt.Sprintf("%v\n\tReleased: %v\n\tRating: %v\n\tPlot: %v\n",
  movie.Title, movie.Year, movie.Info["rating"], movie.Info["plot"])
}
```

- Einzelheiten zur API finden Sie DeleteItemin der AWS SDK für Go API-Referenz.

Java

SDK für Java 2.x

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.AttributeValue;
import software.amazon.awssdk.services.dynamodb.model.DeleteItemRequest;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import java.util.HashMap;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
```

```
 */
public class DeleteItem {
    public static void main(String[] args) {
        final String usage = """

                Usage:
                    <tableName> <key> <keyval>

                Where:
                    tableName - The Amazon DynamoDB table to delete the item from
 (for example, Music3).
                    key - The key used in the Amazon DynamoDB table (for example,
 Artist).\s
                    keyval - The key value that represents the item to delete
 (for example, Famous Band).
                """;

        if (args.length != 3) {
            System.out.println(usage);
            System.exit(1);
        }

        String tableName = args[0];
        String key = args[1];
        String keyVal = args[2];
        System.out.format("Deleting item \"%s\" from %s\n", keyVal, tableName);
        Region region = Region.US_EAST_1;
        DynamoDbClient ddb = DynamoDbClient.builder()
                .region(region)
                .build();

        deleteDynamoDBItem(ddb, tableName, key, keyVal);
        ddb.close();
    }

    public static void deleteDynamoDBItem(DynamoDbClient ddb, String tableName,
 String key, String keyVal) {
        HashMap<String, AttributeValue> keyToGet = new HashMap<>();
        keyToGet.put(key, AttributeValue.builder()
                .s(keyVal)
                .build());

        DeleteItemRequest deleteReq = DeleteItemRequest.builder()
                .tableName(tableName)
```

```
                .key(keyToGet)
                .build();

        try {
            ddb.deleteItem(deleteReq);
        } catch (DynamoDbException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
    }
}
```

- Einzelheiten zur API finden Sie DeleteItemin der AWS SDK for Java 2.x API-Referenz.

JavaScript

SDK für JavaScript (v3)

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

In diesem Beispiel wird der Dokument-Client verwendet, um die Arbeit mit Elementen in DynamoDB zu vereinfachen. Einzelheiten zur API finden Sie unter DeleteCommand.

```
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";
import { DynamoDBDocumentClient, DeleteCommand } from "@aws-sdk/lib-dynamodb";

const client = new DynamoDBClient({});
const docClient = DynamoDBDocumentClient.from(client);

export const main = async () => {
  const command = new DeleteCommand({
    TableName: "Sodas",
    Key: {
      Flavor: "Cola",
    },
  });
```

```
  const response = await docClient.send(command);
  console.log(response);
  return response;
};
```

- Weitere Informationen finden Sie im AWS SDK für JavaScript -Entwicklerhandbuch.
- Einzelheiten zur API finden Sie DeleteItemin der AWS SDK für JavaScript API-Referenz.

SDK für JavaScript (v2)

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

Löschen Sie ein Element aus einer Tabelle.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create the DynamoDB service object
var ddb = new AWS.DynamoDB({ apiVersion: "2012-08-10" });

var params = {
  TableName: "TABLE",
  Key: {
    KEY_NAME: { N: "VALUE" },
  },
};

// Call DynamoDB to delete the item from the table
ddb.deleteItem(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data);
  }
```

```
});
```

Löschen Sie ein Element mithilfe des DynamoDB-Dokument-Clients aus einer Tabelle.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create DynamoDB document client
var docClient = new AWS.DynamoDB.DocumentClient({ apiVersion: "2012-08-10" });

var params = {
  Key: {
    HASH_KEY: VALUE,
  },
  TableName: "TABLE",
};

docClient.delete(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data);
  }
});
```

- Weitere Informationen finden Sie im AWS SDK für JavaScript -Entwicklerhandbuch.
- Einzelheiten zur API finden Sie DeleteItemin der AWS SDK für JavaScript API-Referenz.

Kotlin

SDK für Kotlin

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
suspend fun deleteDynamoDBItem(
    tableNameVal: String,
    keyName: String,
    keyVal: String,
) {
    val keyToGet = mutableMapOf<String, AttributeValue>()
    keyToGet[keyName] = AttributeValue.S(keyVal)

    val request =
        DeleteItemRequest {
            tableName = tableNameVal
            key = keyToGet
        }

    DynamoDbClient.fromEnvironment { region = "us-east-1" }.use { ddb ->
        ddb.deleteItem(request)
        println("Item with key matching $keyVal was deleted")
    }
}
```

- Einzelheiten zur API finden Sie DeleteItemin der API-Referenz zum AWS SDK für Kotlin.

PHP

SDK für PHP

> **Note**
>
> Es gibt noch mehr dazu. GitHub Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
        $key = [
            'Item' => [
                'title' => [
                    'S' => $movieName,
                ],
                'year' => [
                    'N' => $movieYear,
                ],
```

```
            ]
        ];

        $service->deleteItemByKey($tableName, $key);
        echo "But, bad news, this was a trap. That movie has now been deleted
 because of your rating...harsh.\n";

    public function deleteItemByKey(string $tableName, array $key)
    {
        $this->dynamoDbClient->deleteItem([
            'Key' => $key['Item'],
            'TableName' => $tableName,
        ]);
    }
```

- Einzelheiten zur API finden Sie DeleteItemin der AWS SDK für PHP API-Referenz.

PowerShell

Tools für PowerShell V4

Beispiel 1: Entfernt das DynamoDB-Element, das dem angegebenen Schlüssel entspricht.

```
$key = @{
    SongTitle = 'Somewhere Down The Road'
    Artist = 'No One You Know'
} | ConvertTo-DDBItem
Remove-DDBItem -TableName 'Music' -Key $key -Confirm:$false
```

- Einzelheiten zur API finden Sie unter DeleteItem AWS -Tools für PowerShellCmdlet-Referenz (V4).

Tools für V5 PowerShell

Beispiel 1: Entfernt das DynamoDB-Element, das dem angegebenen Schlüssel entspricht.

```
$key = @{
    SongTitle = 'Somewhere Down The Road'
    Artist = 'No One You Know'
} | ConvertTo-DDBItem
Remove-DDBItem -TableName 'Music' -Key $key -Confirm:$false
```

- Einzelheiten zur API finden Sie unter DeleteItem AWS -Tools für PowerShellCmdlet-Referenz (V5).

Python

SDK für Python (Boto3)

> **Note**
>
> Es gibt noch mehr dazu. GitHub Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
class Movies:
    """Encapsulates an Amazon DynamoDB table of movie data.

    Example data structure for a movie record in this table:
        {
            "year": 1999,
            "title": "For Love of the Game",
            "info": {
                "directors": ["Sam Raimi"],
                "release_date": "1999-09-15T00:00:00Z",
                "rating": 6.3,
                "plot": "A washed up pitcher flashes through his career.",
                "rank": 4987,
                "running_time_secs": 8220,
                "actors": [
                    "Kevin Costner",
                    "Kelly Preston",
                    "John C. Reilly"
                ]
            }
        }
    """

    def __init__(self, dyn_resource):
        """
        :param dyn_resource: A Boto3 DynamoDB resource.
        """
        self.dyn_resource = dyn_resource
```

```
        # The table variable is set during the scenario in the call to
        # 'exists' if the table exists. Otherwise, it is set by 'create_table'.
        self.table = None


    def delete_movie(self, title, year):
        """
        Deletes a movie from the table.

        :param title: The title of the movie to delete.
        :param year: The release year of the movie to delete.
        """
        try:
            self.table.delete_item(Key={"year": year, "title": title})
        except ClientError as err:
            logger.error(
                "Couldn't delete movie %s. Here's why: %s: %s",
                title,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
```

Sie können eine Bedingung angeben, damit ein Element nur gelöscht wird, wenn es bestimmte Kriterien erfüllt.

```
class UpdateQueryWrapper:
    def __init__(self, table):
        self.table = table


    def delete_underrated_movie(self, title, year, rating):
        """
        Deletes a movie only if it is rated below a specified value. By using a
        condition expression in a delete operation, you can specify that an item
 is
        deleted only when it meets certain criteria.

        :param title: The title of the movie to delete.
        :param year: The release year of the movie to delete.
        :param rating: The rating threshold to check before deleting the movie.
```

```
        """
        try:
            self.table.delete_item(
                Key={"year": year, "title": title},
                ConditionExpression="info.rating <= :val",
                ExpressionAttributeValues={":val": Decimal(str(rating))},
            )
        except ClientError as err:
            if err.response["Error"]["Code"] ==
 "ConditionalCheckFailedException":
                logger.warning(
                    "Didn't delete %s because its rating is greater than %s.",
                    title,
                    rating,
                )
            else:
                logger.error(
                    "Couldn't delete movie %s. Here's why: %s: %s",
                    title,
                    err.response["Error"]["Code"],
                    err.response["Error"]["Message"],
                )
            raise
```

- Einzelheiten zur API finden Sie DeleteItemin AWS SDK for Python (Boto3) API Reference.

Ruby

SDK für Ruby

> **Note**
>
> Es gibt noch mehr dazu. GitHub Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
class DynamoDBBasics
  attr_reader :dynamo_resource, :table

  def initialize(table_name)
```

```
    client = Aws::DynamoDB::Client.new(region: 'us-east-1')
    @dynamo_resource = Aws::DynamoDB::Resource.new(client: client)
    @table = @dynamo_resource.table(table_name)
  end

  # Deletes a movie from the table.
  #
  # @param title [String] The title of the movie to delete.
  # @param year [Integer] The release year of the movie to delete.
  def delete_item(title, year)
    @table.delete_item(key: { 'year' => year, 'title' => title })
  rescue Aws::DynamoDB::Errors::ServiceError => e
    puts("Couldn't delete movie #{title}. Here's why:")
    puts("\t#{e.code}: #{e.message}")
    raise
  end
```

- Einzelheiten zur API finden Sie DeleteItemin der AWS SDK für Ruby API-Referenz.

Rust

SDK für Rust

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
pub async fn delete_item(
    client: &Client,
    table: &str,
    key: &str,
    value: &str,
) -> Result<DeleteItemOutput, Error> {
    match client
        .delete_item()
        .table_name(table)
        .key(key, AttributeValue::S(value.into()))
        .send()
        .await
```

```
    {
        Ok(out) => {
            println!("Deleted item from table");
            Ok(out)
        }
        Err(e) => Err(Error::unhandled(e)),
    }
}
```

- Einzelheiten zur API finden Sie DeleteItemin der API-Referenz zum AWS SDK für Rust.

SAP ABAP

SDK für SAP ABAP

> **Note**
>
> Es gibt noch mehr dazu. GitHub Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
    TRY.
        DATA(lo_resp) = lo_dyn->deleteitem(
          iv_tablename                = iv_table_name
          it_key                      = it_key_input ).
        MESSAGE 'Deleted one item.' TYPE 'I'.
      CATCH /aws1/cx_dyncondalcheckfaile00.
        MESSAGE 'A condition specified in the operation could not be evaluated.'
TYPE 'E'.
      CATCH /aws1/cx_dynresourcenotfoundex.
        MESSAGE 'The table or index does not exist' TYPE 'E'.
      CATCH /aws1/cx_dyntransactconflictex.
        MESSAGE 'Another transaction is using the item' TYPE 'E'.
    ENDTRY.
```

- Einzelheiten zur API finden Sie DeleteItemin der API-Referenz zum AWS SDK für SAP ABAP.

Swift

SDK für Swift

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
import AWSDynamoDB


    /// Delete a movie, given its title and release year.
    ///
    /// - Parameters:
    ///   - title: The movie's title.
    ///   - year: The movie's release year.
    ///
    func delete(title: String, year: Int) async throws {
        do {
            guard let client = self.ddbClient else {
                throw MoviesError.UninitializedClient
            }

            let input = DeleteItemInput(
                key: [
                    "year": .n(String(year)),
                    "title": .s(title)
                ],
                tableName: self.tableName
            )
            _ = try await client.deleteItem(input: input)
        } catch {
            print("ERROR: delete:", dump(error))
            throw error
        }
    }
```

- Einzelheiten zur API finden Sie DeleteItemin der API-Referenz zum AWS SDK für Swift.

Eine vollständige Liste der AWS SDK-Entwicklerhandbücher und Codebeispiele finden Sie unterDynamoDB mit einem SDK verwenden AWS. Dieses Thema enthält auch Informationen zu den ersten Schritten und Details zu früheren SDK-Versionen.

Verwendung **`DeleteTable`** mit einem AWS SDK oder CLI

Die folgenden Code-Beispiele zeigen, wie `DeleteTable` verwendet wird.

Aktionsbeispiele sind Codeauszüge aus größeren Programmen und müssen im Kontext ausgeführt werden. Sie können diese Aktion in den folgenden Codebeispielen im Kontext sehen:

- Kennenlernen der Grundlagen
- Beschleunigen von Lesevorgängen mit DAX
- Erstellen und Verwalten von globalen MRSC-Tabellen
- Verwalten globaler sekundärer Indizes
- Arbeiten mit globalen Tabellen und letztendlicher multiregionaler Konsistenz (Multi-Region Eventual Consistency, MREC)

.NET

SDK für .NET (v4)

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
    /// <summary>
    /// Deletes a DynamoDB table.
    /// </summary>
    /// <param name="tableName">The name of the table to delete.</param>
    /// <returns>A Boolean value indicating the success of the operation.</
returns>
    public async Task<bool> DeleteTableAsync(string tableName)
    {
        try
        {
            var request = new DeleteTableRequest
```

```
            {
                TableName = tableName,
            };

            var response = await _amazonDynamoDB.DeleteTableAsync(request);

            Console.WriteLine($"Table {response.TableDescription.TableName}
successfully deleted.");
            return true;

        }
        catch (ResourceNotFoundException ex)
        {
            Console.WriteLine($"Table {tableName} was not found and cannot be
deleted. {ex.Message}");
            return false;
        }
        catch (AmazonDynamoDBException ex)
        {
            Console.WriteLine($"An Amazon DynamoDB error occurred while deleting
table {tableName}. {ex.Message}");
            return false;
        }
        catch (Exception ex)
        {
            Console.WriteLine($"An error occurred while deleting table
{tableName}. {ex.Message}");
            return false;
        }
    }
```

- Einzelheiten zur API finden Sie DeleteTablein der AWS SDK für .NET API-Referenz.

Bash

AWS CLI mit Bash-Skript

> **Note**
>
> Es gibt noch mehr dazu. GitHub Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
###############################################################################
# function dynamodb_delete_table
#
# This function deletes a DynamoDB table.
#
# Parameters:
#       -n table_name  -- The name of the table to delete.
#
#  Returns:
#       0 - If successful.
#       1 - If it fails.
###############################################################################
function dynamodb_delete_table() {
  local table_name response
  local option OPTARG # Required to use getopts command in a function.

  # bashsupport disable=BP5008
  function usage() {
    echo "function dynamodb_delete_table"
    echo "Deletes an Amazon DynamoDB table."
    echo " -n table_name  -- The name of the table to delete."
    echo ""
  }

  # Retrieve the calling parameters.
  while getopts "n:h" option; do
    case "${option}" in
      n) table_name="${OPTARG}" ;;
      h)
        usage
        return 0
        ;;
```

```
      \?)
        echo "Invalid parameter"
        usage
        return 1
        ;;
    esac
  done
  export OPTIND=1

  if [[ -z "$table_name" ]]; then
    errecho "ERROR: You must provide a table name with the -n parameter."
    usage
    return 1
  fi

  iecho "Parameters:\n"
  iecho "    table_name:   $table_name"
  iecho ""

  response=$(aws dynamodb delete-table \
    --table-name "$table_name")

  local error_code=${?}

  if [[ $error_code -ne 0 ]]; then
    aws_cli_error_log $error_code
    errecho "ERROR: AWS reports delete-table operation failed.$response"
    return 1
  fi

  return 0
}
```

Die in diesem Beispiel verwendeten Hilfsfunktionen.

```
###############################################################################
# function iecho
#
# This function enables the script to display the specified text only if
# the global variable $VERBOSE is set to true.
###############################################################################
function iecho() {
```

```
  if [[ $VERBOSE == true ]]; then
    echo "$@"
  fi
}

################################################################################
# function errecho
#
# This function outputs everything sent to it to STDERR (standard error output).
################################################################################
function errecho() {
  printf "%s\n" "$*" 1>&2
}

################################################################################
# function aws_cli_error_log()
#
# This function is used to log the error messages from the AWS CLI.
#
# See https://docs.aws.amazon.com/cli/latest/topic/return-codes.html#cli-aws-
help-return-codes.
#
# The function expects the following argument:
#         $1 - The error code returned by the AWS CLI.
#
#  Returns:
#          0: - Success.
#
################################################################################
function aws_cli_error_log() {
  local err_code=$1
  errecho "Error code : $err_code"
  if [ "$err_code" == 1 ]; then
    errecho "  One or more S3 transfers failed."
  elif [ "$err_code" == 2 ]; then
    errecho "  Command line failed to parse."
  elif [ "$err_code" == 130 ]; then
    errecho "  Process received SIGINT."
  elif [ "$err_code" == 252 ]; then
    errecho "  Command syntax invalid."
  elif [ "$err_code" == 253 ]; then
    errecho "  The system environment or configuration was invalid."
  elif [ "$err_code" == 254 ]; then
    errecho "  The service returned an error."
```

```
  elif [ "$err_code" == 255 ]; then
    errecho "  255 is a catch-all error."
  fi

  return 0
}
```

- Einzelheiten zur API finden Sie DeleteTablein der AWS CLI Befehlsreferenz.

C++

SDK für C++

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
//! Delete an Amazon DynamoDB table.
/*!
  \sa deleteTable()
  \param tableName: The DynamoDB table name.
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
*/
bool AwsDoc::DynamoDB::deleteTable(const Aws::String &tableName,
                                   const Aws::Client::ClientConfiguration
 &clientConfiguration) {
    Aws::DynamoDB::DynamoDBClient dynamoClient(clientConfiguration);

    Aws::DynamoDB::Model::DeleteTableRequest request;
    request.SetTableName(tableName);

    const Aws::DynamoDB::Model::DeleteTableOutcome &result =
 dynamoClient.DeleteTable(
            request);
    if (result.IsSuccess()) {
        std::cout << "Your table \""
                  << result.GetResult().GetTableDescription().GetTableName()
                  << " was deleted.\n";
```

```
    }
    else {
        std::cerr << "Failed to delete table: " << result.GetError().GetMessage()
                  << std::endl;
    }

    return result.IsSuccess();
}
```

- Einzelheiten zur API finden Sie DeleteTablein der AWS SDK für C++ API-Referenz.

CLI

AWS CLI

So löschen Sie eine Tabelle

Im folgenden Beispiel für `delete-table` wird die Tabelle `MusicCollection` gelöscht.

```
aws dynamodb delete-table \
    --table-name MusicCollection
```

Ausgabe:

```
{
    "TableDescription": {
        "TableStatus": "DELETING",
        "TableSizeBytes": 0,
        "ItemCount": 0,
        "TableName": "MusicCollection",
        "ProvisionedThroughput": {
            "NumberOfDecreasesToday": 0,
            "WriteCapacityUnits": 5,
            "ReadCapacityUnits": 5
        }
    }
}
```

Weitere Informationen finden Sie unter Löschen einer Tabelle im Entwicklerhandbuch zu Amazon DynamoDB.

- Einzelheiten zur API finden Sie DeleteTablein der AWS CLI Befehlsreferenz.

Go

SDK für Go V2

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
import (
 "context"
 "errors"
 "log"
 "time"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/attributevalue"
 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/expression"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb/types"
)

// TableBasics encapsulates the Amazon DynamoDB service actions used in the
 examples.
// It contains a DynamoDB service client that is used to act on the specified
 table.
type TableBasics struct {
 DynamoDbClient *dynamodb.Client
 TableName      string
}



// DeleteTable deletes the DynamoDB table and all of its data.
func (basics TableBasics) DeleteTable(ctx context.Context) error {
 _, err := basics.DynamoDbClient.DeleteTable(ctx, &dynamodb.DeleteTableInput{
  TableName: aws.String(basics.TableName)})
 if err != nil {
```

```
  log.Printf("Couldn't delete table %v. Here's why: %v\n", basics.TableName, err)
 }
 return err
}
```

- Einzelheiten zur API finden Sie DeleteTablein der AWS SDK für Go API-Referenz.

Java

SDK für Java 2.x

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.DeleteTableRequest;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 */

public class DeleteTable {
    public static void main(String[] args) {
        final String usage = """

                Usage:
                    <tableName>

                Where:
```

```
                    tableName - The Amazon DynamoDB table to delete (for example,
 Music3).

                **Warning** This program will delete the table that you specify!
                """;

        if (args.length != 1) {
            System.out.println(usage);
            System.exit(1);
        }

        String tableName = args[0];
        System.out.format("Deleting the Amazon DynamoDB table %s...\n",
 tableName);
        Region region = Region.US_EAST_1;
        DynamoDbClient ddb = DynamoDbClient.builder()
                .region(region)
                .build();

        deleteDynamoDBTable(ddb, tableName);
        ddb.close();
    }

    public static void deleteDynamoDBTable(DynamoDbClient ddb, String tableName)
 {
        DeleteTableRequest request = DeleteTableRequest.builder()
                .tableName(tableName)
                .build();

        try {
            ddb.deleteTable(request);

        } catch (DynamoDbException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
        System.out.println(tableName + " was successfully deleted!");
    }
}
```

- Einzelheiten zur API finden Sie DeleteTablein der AWS SDK for Java 2.x API-Referenz.

JavaScript

SDK für JavaScript (v3)

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
import { DeleteTableCommand, DynamoDBClient } from "@aws-sdk/client-dynamodb";

const client = new DynamoDBClient({});

export const main = async () => {
  const command = new DeleteTableCommand({
    TableName: "DecafCoffees",
  });

  const response = await client.send(command);
  console.log(response);
  return response;
};
```

- Einzelheiten zur API finden Sie DeleteTablein der AWS SDK für JavaScript API-Referenz.

SDK für JavaScript (v2)

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create the DynamoDB service object
```

```
var ddb = new AWS.DynamoDB({ apiVersion: "2012-08-10" });

var params = {
  TableName: process.argv[2],
};

// Call DynamoDB to delete the specified table
ddb.deleteTable(params, function (err, data) {
  if (err && err.code === "ResourceNotFoundException") {
    console.log("Error: Table not found");
  } else if (err && err.code === "ResourceInUseException") {
    console.log("Error: Table in use");
  } else {
    console.log("Success", data);
  }
});
```

- Weitere Informationen finden Sie im AWS SDK für JavaScript -Entwicklerhandbuch.
- Einzelheiten zur API finden Sie DeleteTablein der AWS SDK für JavaScript API-Referenz.

Kotlin

SDK für Kotlin

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
suspend fun deleteDynamoDBTable(tableNameVal: String) {
    val request =
        DeleteTableRequest {
            tableName = tableNameVal
        }

    DynamoDbClient.fromEnvironment { region = "us-east-1" }.use { ddb ->
        ddb.deleteTable(request)
        println("$tableNameVal was deleted")
    }
```

```
}
```

- Einzelheiten zur API finden Sie DeleteTablein der API-Referenz zum AWS SDK für Kotlin.

PHP

SDK für PHP

> **Note**
>
> Es gibt noch mehr dazu. GitHub Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
    public function deleteTable(string $TableName)
    {
        $this->customWaiter(function () use ($TableName) {
            return $this->dynamoDbClient->deleteTable([
                'TableName' => $TableName,
            ]);
        });
    }
```

- Einzelheiten zur API finden Sie DeleteTablein der AWS SDK für PHP API-Referenz.

PowerShell

Tools für PowerShell V4

Beispiel 1: Löscht die angegebene Tabelle. Sie werden um eine Bestätigung gebeten, bevor der Vorgang fortgesetzt wird.

```
Remove-DDBTable -TableName "myTable"
```

Beispiel 2: Löscht die angegebene Tabelle. Sie werden nicht zum Bestätigen aufgefordert, bevor der Vorgang fortgesetzt wird.

```
Remove-DDBTable -TableName "myTable" -Force
```

- Einzelheiten zur API finden Sie unter DeleteTable AWS -Tools für PowerShellCmdlet-Referenz (V4).

Tools für V5 PowerShell

Beispiel 1: Löscht die angegebene Tabelle. Sie werden um eine Bestätigung gebeten, bevor der Vorgang fortgesetzt wird.

```
Remove-DDBTable -TableName "myTable"
```

Beispiel 2: Löscht die angegebene Tabelle. Sie werden nicht zum Bestätigen aufgefordert, bevor der Vorgang fortgesetzt wird.

```
Remove-DDBTable -TableName "myTable" -Force
```

- Einzelheiten zur API finden Sie unter DeleteTable AWS -Tools für PowerShellCmdlet-Referenz (V5).

Python

SDK für Python (Boto3)

> **Note**
>
> Es gibt noch mehr dazu. GitHub Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
class Movies:
    """Encapsulates an Amazon DynamoDB table of movie data.

    Example data structure for a movie record in this table:
        {
            "year": 1999,
            "title": "For Love of the Game",
            "info": {
                "directors": ["Sam Raimi"],
```

```
                "release_date": "1999-09-15T00:00:00Z",
                "rating": 6.3,
                "plot": "A washed up pitcher flashes through his career.",
                "rank": 4987,
                "running_time_secs": 8220,
                "actors": [
                    "Kevin Costner",
                    "Kelly Preston",
                    "John C. Reilly"
                ]
            }
        }
    """

    def __init__(self, dyn_resource):
        """
        :param dyn_resource: A Boto3 DynamoDB resource.
        """
        self.dyn_resource = dyn_resource
        # The table variable is set during the scenario in the call to
        # 'exists' if the table exists. Otherwise, it is set by 'create_table'.
        self.table = None


    def delete_table(self):
        """
        Deletes the table.
        """
        try:
            self.table.delete()
            self.table = None
        except ClientError as err:
            logger.error(
                "Couldn't delete table. Here's why: %s: %s",
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
```

- Einzelheiten zur API finden Sie DeleteTablein AWS SDK for Python (Boto3) API Reference.

Ruby

SDK für Ruby

> **Note**
>
> Es gibt noch mehr dazu. GitHub Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
# Encapsulates an Amazon DynamoDB table of movie data.
class Scaffold
  attr_reader :dynamo_resource, :table_name, :table

  def initialize(table_name)
    client = Aws::DynamoDB::Client.new(region: 'us-east-1')
    @dynamo_resource = Aws::DynamoDB::Resource.new(client: client)
    @table_name = table_name
    @table = nil
    @logger = Logger.new($stdout)
    @logger.level = Logger::DEBUG
  end

  # Deletes the table.
  def delete_table
    @table.delete
    @table = nil
  rescue Aws::DynamoDB::Errors::ServiceError => e
    puts("Couldn't delete table. Here's why:")
    puts("\t#{e.code}: #{e.message}")
    raise
  end
```

- Einzelheiten zur API finden Sie DeleteTablein der AWS SDK für Ruby API-Referenz.

Rust

SDK für Rust

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
pub async fn delete_table(client: &Client, table: &str) ->
 Result<DeleteTableOutput, Error> {
    let resp = client.delete_table().table_name(table).send().await;

    match resp {
        Ok(out) => {
            println!("Deleted table");
            Ok(out)
        }
        Err(e) => Err(Error::Unhandled(e.into())),
    }
}
```

- Einzelheiten zur API finden Sie DeleteTablein der API-Referenz zum AWS SDK für Rust.

SAP ABAP

SDK für SAP ABAP

> **Note**
>
> Es gibt noch mehr dazu. GitHub Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
    TRY.
        lo_dyn->deletetable( iv_tablename = iv_table_name ).
        " Wait till the table is actually deleted.
        lo_dyn->get_waiter( )->tablenotexists(
```

```
          iv_max_wait_time = 200
          iv_tablename     = iv_table_name ).
        MESSAGE 'Table ' && iv_table_name && ' deleted.' TYPE 'I'.
      CATCH /aws1/cx_dynresourcenotfoundex.
        MESSAGE 'The table ' && iv_table_name && ' does not exist' TYPE 'E'.
      CATCH /aws1/cx_dynresourceinuseex.
        MESSAGE 'The table cannot be deleted since it is in use' TYPE 'E'.
    ENDTRY.
```

- Einzelheiten zur API finden Sie DeleteTablein der API-Referenz zum AWS SDK für SAP ABAP.

Swift

SDK für Swift

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
import AWSDynamoDB


    ///
    /// Deletes the table from Amazon DynamoDB.
    ///
    func deleteTable() async throws {
        do {
            guard let client = self.ddbClient else {
                throw MoviesError.UninitializedClient
            }

            let input = DeleteTableInput(
                tableName: self.tableName
            )
            _ = try await client.deleteTable(input: input)
        } catch {
            print("ERROR: deleteTable:", dump(error))
```

```
            throw error
        }
    }
```

- Einzelheiten zur API finden Sie DeleteTablein der API-Referenz zum AWS SDK für Swift.

Eine vollständige Liste der AWS SDK-Entwicklerhandbücher und Codebeispiele finden Sie unterDynamoDB mit einem SDK verwenden AWS. Dieses Thema enthält auch Informationen zu den ersten Schritten und Details zu früheren SDK-Versionen.

Verwendung **`DescribeTable`** mit einem AWS SDK oder CLI

Die folgenden Code-Beispiele zeigen, wie `DescribeTable` verwendet wird.

Aktionsbeispiele sind Codeauszüge aus größeren Programmen und müssen im Kontext ausgeführt werden. Sie können diese Aktion in den folgenden Codebeispielen im Kontext sehen:

- Kennenlernen der Grundlagen
- Erstellen und Verwalten globaler Tabellen zur Demonstration von MREC
- Erstellen und Verwalten von globalen MRSC-Tabellen
- Arbeiten mit globalen Tabellen und letztendlicher multiregionaler Konsistenz (Multi-Region Eventual Consistency, MREC)
- Arbeite mit Streams und Time-to-Live
- Arbeiten mit der Tabellenverschlüsselung

.NET

SDK für .NET

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
    private static async Task GetTableInformation()
```

```
    {
        Console.WriteLine("\n*** Retrieving table information ***");

        var response = await Client.DescribeTableAsync(new DescribeTableRequest
        {
            TableName = ExampleTableName
        });

        var table = response.Table;
        Console.WriteLine($"Name: {table.TableName}");
        Console.WriteLine($"# of items: {table.ItemCount}");

    }
```

- Einzelheiten zur API finden Sie DescribeTablein der AWS SDK für .NET API-Referenz.

Bash

AWS CLI mit Bash-Skript

> **Note**
>
> Es gibt noch mehr dazu. GitHub Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
###############################################################################
# function dynamodb_describe_table
#
# This function returns the status of a DynamoDB table.
#
# Parameters:
#       -n table_name  -- The name of the table.
#
#  Response:
#       - TableStatus:
#     And:
#       0 - Table is active.
#       1 - If it fails.
###############################################################################
function dynamodb_describe_table {
```

```
  local table_name
  local option OPTARG # Required to use getopts command in a function.

  ######################################
  # Function usage explanation
  ######################################
  function usage() {
    echo "function dynamodb_describe_table"
    echo "Describe the status of a DynamoDB table."
    echo "  -n table_name  -- The name of the table."
    echo ""
  }

  # Retrieve the calling parameters.
  while getopts "n:h" option; do
    case "${option}" in
      n) table_name="${OPTARG}" ;;
      h)
        usage
        return 0
        ;;
      \?)
        echo "Invalid parameter"
        usage
        return 1
        ;;
    esac
  done
  export OPTIND=1

  if [[ -z "$table_name" ]]; then
    errecho "ERROR: You must provide a table name with the -n parameter."
    usage
    return 1
  fi

  local table_status
    table_status=$(
      aws dynamodb describe-table \
        --table-name "$table_name" \
        --output text \
        --query 'Table.TableStatus'
    )
```

```
   local error_code=${?}

    if [[ $error_code -ne 0 ]]; then
      aws_cli_error_log "$error_code"
      errecho "ERROR: AWS reports describe-table operation failed.$table_status"
      return 1
    fi

  echo "$table_status"

  return 0
}
```

Die in diesem Beispiel verwendeten Hilfsfunktionen.

```
###############################################################################
# function errecho
#
# This function outputs everything sent to it to STDERR (standard error output).
###############################################################################
function errecho() {
  printf "%s\n" "$*" 1>&2
}

###############################################################################
# function aws_cli_error_log()
#
# This function is used to log the error messages from the AWS CLI.
#
# See https://docs.aws.amazon.com/cli/latest/topic/return-codes.html#cli-aws-
help-return-codes.
#
# The function expects the following argument:
#         $1 - The error code returned by the AWS CLI.
#
#  Returns:
#          0: - Success.
#
###############################################################################
function aws_cli_error_log() {
  local err_code=$1
  errecho "Error code : $err_code"
```

```
  if [ "$err_code" == 1 ]; then
    errecho "  One or more S3 transfers failed."
  elif [ "$err_code" == 2 ]; then
    errecho "  Command line failed to parse."
  elif [ "$err_code" == 130 ]; then
    errecho "  Process received SIGINT."
  elif [ "$err_code" == 252 ]; then
    errecho "  Command syntax invalid."
  elif [ "$err_code" == 253 ]; then
    errecho "  The system environment or configuration was invalid."
  elif [ "$err_code" == 254 ]; then
    errecho "  The service returned an error."
  elif [ "$err_code" == 255 ]; then
    errecho "  255 is a catch-all error."
  fi

  return 0
}
```

- Einzelheiten zur API finden Sie DescribeTablein der AWS CLI Befehlsreferenz.

C++

SDK für C++

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
//! Describe an Amazon DynamoDB table.
/*!
  \sa describeTable()
  \param tableName: The DynamoDB table name.
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
*/
bool AwsDoc::DynamoDB::describeTable(const Aws::String &tableName,
                                     const Aws::Client::ClientConfiguration
 &clientConfiguration) {
```

```
   Aws::DynamoDB::DynamoDBClient dynamoClient(clientConfiguration);

   Aws::DynamoDB::Model::DescribeTableRequest request;
   request.SetTableName(tableName);

   const Aws::DynamoDB::Model::DescribeTableOutcome &outcome =
dynamoClient.DescribeTable(
           request);

   if (outcome.IsSuccess()) {
       const Aws::DynamoDB::Model::TableDescription &td =
outcome.GetResult().GetTable();
       std::cout << "Table name  : " << td.GetTableName() << std::endl;
       std::cout << "Table ARN   : " << td.GetTableArn() << std::endl;
       std::cout << "Status      : "
                 <<
Aws::DynamoDB::Model::TableStatusMapper::GetNameForTableStatus(
                         td.GetTableStatus()) << std::endl;
       std::cout << "Item count  : " << td.GetItemCount() << std::endl;
       std::cout << "Size (bytes): " << td.GetTableSizeBytes() << std::endl;

       const Aws::DynamoDB::Model::ProvisionedThroughputDescription &ptd =
td.GetProvisionedThroughput();
       std::cout << "Throughput" << std::endl;
       std::cout << "  Read Capacity : " << ptd.GetReadCapacityUnits() <<
std::endl;
       std::cout << "  Write Capacity: " << ptd.GetWriteCapacityUnits() <<
std::endl;

       const Aws::Vector<Aws::DynamoDB::Model::AttributeDefinition> &ad =
td.GetAttributeDefinitions();
       std::cout << "Attributes" << std::endl;
       for (const auto &a: ad)
           std::cout << "  " << a.GetAttributeName() << " (" <<

Aws::DynamoDB::Model::ScalarAttributeTypeMapper::GetNameForScalarAttributeType(
                            a.GetAttributeType()) <<
                    ")" << std::endl;
   }
   else {
       std::cerr << "Failed to describe table: " <<
outcome.GetError().GetMessage();
   }
```

```
    return outcome.IsSuccess();
}
```

- Einzelheiten zur API finden Sie DescribeTablein der AWS SDK für C++ API-Referenz.

CLI

AWS CLI

So beschreiben Sie eine Tabelle

Im folgenden Beispiel für `describe-table` wird die `MusicCollection`-Tabelle beschrieben.

```
aws dynamodb describe-table \
    --table-name MusicCollection
```

Ausgabe:

```
{
    "Table": {
        "AttributeDefinitions": [
            {
                "AttributeName": "Artist",
                "AttributeType": "S"
            },
            {
                "AttributeName": "SongTitle",
                "AttributeType": "S"
            }
        ],
        "ProvisionedThroughput": {
            "NumberOfDecreasesToday": 0,
            "WriteCapacityUnits": 5,
            "ReadCapacityUnits": 5
        },
        "TableSizeBytes": 0,
        "TableName": "MusicCollection",
        "TableStatus": "ACTIVE",
        "KeySchema": [
            {
```

```
                "KeyType": "HASH",
                "AttributeName": "Artist"
            },
            {
                "KeyType": "RANGE",
                "AttributeName": "SongTitle"
            }
        ],
        "ItemCount": 0,
        "CreationDateTime": 1421866952.062
    }
}
```

Weitere Informationen finden Sie unter Beschreiben einer Tabelle im Entwicklerhandbuch zu Amazon DynamoDB.

- Einzelheiten zur API finden Sie DescribeTablein der AWS CLI Befehlsreferenz.

Go

SDK für Go V2

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
import (
 "context"
 "errors"
 "log"
 "time"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/attributevalue"
 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/expression"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb/types"
)
```

```
// TableBasics encapsulates the Amazon DynamoDB service actions used in the
 examples.
// It contains a DynamoDB service client that is used to act on the specified
 table.
type TableBasics struct {
 DynamoDbClient *dynamodb.Client
 TableName      string
}



// TableExists determines whether a DynamoDB table exists.
func (basics TableBasics) TableExists(ctx context.Context) (bool, error) {
 exists := true
 _, err := basics.DynamoDbClient.DescribeTable(
  ctx, &dynamodb.DescribeTableInput{TableName: aws.String(basics.TableName)},
 )
 if err != nil {
  var notFoundEx *types.ResourceNotFoundException
  if errors.As(err, &notFoundEx) {
   log.Printf("Table %v does not exist.\n", basics.TableName)
   err = nil
  } else {
   log.Printf("Couldn't determine existence of table %v. Here's why: %v\n",
 basics.TableName, err)
  }
  exists = false
 }
 return exists, err
}
```

- Einzelheiten zur API finden Sie DescribeTablein der AWS SDK für Go API-Referenz.

Java

SDK für Java 2.x

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.AttributeDefinition;
import software.amazon.awssdk.services.dynamodb.model.DescribeTableRequest;
import
 software.amazon.awssdk.services.dynamodb.model.ProvisionedThroughputDescription;
import software.amazon.awssdk.services.dynamodb.model.TableDescription;
import java.util.List;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 */
public class DescribeTable {
    public static void main(String[] args) {
        final String usage = """

                Usage:
                    <tableName>

                Where:
                    tableName - The Amazon DynamoDB table to get information
 about (for example, Music3).
                """;

        if (args.length != 1) {
```

```
            System.out.println(usage);
            System.exit(1);
        }

        String tableName = args[0];
        System.out.format("Getting description for %s\n\n", tableName);
        Region region = Region.US_EAST_1;
        DynamoDbClient ddb = DynamoDbClient.builder()
                .region(region)
                .build();

        describeDymamoDBTable(ddb, tableName);
        ddb.close();
    }

    public static void describeDymamoDBTable(DynamoDbClient ddb, String
tableName) {
        DescribeTableRequest request = DescribeTableRequest.builder()
                .tableName(tableName)
                .build();

        try {
            TableDescription tableInfo = ddb.describeTable(request).table();
            if (tableInfo != null) {
                System.out.format("Table name  : %s\n", tableInfo.tableName());
                System.out.format("Table ARN   : %s\n", tableInfo.tableArn());
                System.out.format("Status      : %s\n", tableInfo.tableStatus());
                System.out.format("Item count  : %d\n", tableInfo.itemCount());
                System.out.format("Size (bytes): %d\n",
tableInfo.tableSizeBytes());

                ProvisionedThroughputDescription throughputInfo =
tableInfo.provisionedThroughput();
                System.out.println("Throughput");
                System.out.format("  Read Capacity : %d\n",
throughputInfo.readCapacityUnits());
                System.out.format("  Write Capacity: %d\n",
throughputInfo.writeCapacityUnits());

                List<AttributeDefinition> attributes =
tableInfo.attributeDefinitions();
                System.out.println("Attributes");
                for (AttributeDefinition a : attributes) {
```

```
                    System.out.format("  %s (%s)\n", a.attributeName(),
 a.attributeType());
                }
            }

        } catch (DynamoDbException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
        System.out.println("\nDone!");
    }
}
```

- Einzelheiten zur API finden Sie DescribeTablein der AWS SDK for Java 2.x API-Referenz.

JavaScript

SDK für JavaScript (v3)

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
import { DescribeTableCommand, DynamoDBClient } from "@aws-sdk/client-dynamodb";

const client = new DynamoDBClient({});

export const main = async () => {
  const command = new DescribeTableCommand({
    TableName: "Pastries",
  });

  const response = await client.send(command);
  console.log(`TABLE NAME: ${response.Table.TableName}`);
  console.log(`TABLE ITEM COUNT: ${response.Table.ItemCount}`);
  return response;
};
```

- Weitere Informationen finden Sie im AWS SDK für JavaScript -Entwicklerhandbuch.
- Einzelheiten zur API finden Sie DescribeTablein der AWS SDK für JavaScript API-Referenz.

SDK für JavaScript (v2)

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create the DynamoDB service object
var ddb = new AWS.DynamoDB({ apiVersion: "2012-08-10" });

var params = {
  TableName: process.argv[2],
};

// Call DynamoDB to retrieve the selected table descriptions
ddb.describeTable(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data.Table.KeySchema);
  }
});
```

- Weitere Informationen finden Sie im AWS SDK für JavaScript -Entwicklerhandbuch.
- Einzelheiten zur API finden Sie DescribeTablein der AWS SDK für JavaScript API-Referenz.

PowerShell

Tools für PowerShell V4

Beispiel 1: Gibt Details der angegebenen Tabelle zurück.

```
Get-DDBTable -TableName "myTable"
```

- Einzelheiten zur API finden Sie unter DescribeTable AWS -Tools für PowerShellCmdlet-Referenz (V4).

Tools für V5 PowerShell

Beispiel 1: Gibt Details der angegebenen Tabelle zurück.

```
Get-DDBTable -TableName "myTable"
```

- Einzelheiten zur API finden Sie unter DescribeTable AWS -Tools für PowerShellCmdlet-Referenz (V5).

Python

SDK für Python (Boto3)

> **Note**
>
> Es gibt noch mehr dazu. GitHub Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
class Movies:
    """Encapsulates an Amazon DynamoDB table of movie data.

    Example data structure for a movie record in this table:
        {
            "year": 1999,
            "title": "For Love of the Game",
            "info": {
                "directors": ["Sam Raimi"],
                "release_date": "1999-09-15T00:00:00Z",
                "rating": 6.3,
                "plot": "A washed up pitcher flashes through his career.",
                "rank": 4987,
                "running_time_secs": 8220,
                "actors": [
                    "Kevin Costner",
                    "Kelly Preston",
```

```
                "John C. Reilly"
            ]
        }
    }
"""

def __init__(self, dyn_resource):
    """
    :param dyn_resource: A Boto3 DynamoDB resource.
    """
    self.dyn_resource = dyn_resource
    # The table variable is set during the scenario in the call to
    # 'exists' if the table exists. Otherwise, it is set by 'create_table'.
    self.table = None


def exists(self, table_name):
    """
    Determines whether a table exists. As a side effect, stores the table in
    a member variable.

    :param table_name: The name of the table to check.
    :return: True when the table exists; otherwise, False.
    """
    try:
        table = self.dyn_resource.Table(table_name)
        table.load()
        exists = True
    except ClientError as err:
        if err.response["Error"]["Code"] == "ResourceNotFoundException":
            exists = False
        else:
            logger.error(
                "Couldn't check for existence of %s. Here's why: %s: %s",
                table_name,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
    else:
        self.table = table
    return exists
```

- Einzelheiten zur API finden Sie DescribeTablein AWS SDK for Python (Boto3) API Reference.

Ruby

SDK für Ruby

> **Note**
>
> Es gibt noch mehr dazu. GitHub Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
# Encapsulates an Amazon DynamoDB table of movie data.
class Scaffold
  attr_reader :dynamo_resource, :table_name, :table

  def initialize(table_name)
    client = Aws::DynamoDB::Client.new(region: 'us-east-1')
    @dynamo_resource = Aws::DynamoDB::Resource.new(client: client)
    @table_name = table_name
    @table = nil
    @logger = Logger.new($stdout)
    @logger.level = Logger::DEBUG
  end

  # Determines whether a table exists. As a side effect, stores the table in
  # a member variable.
  #
  # @param table_name [String] The name of the table to check.
  # @return [Boolean] True when the table exists; otherwise, False.
  def exists?(table_name)
    @dynamo_resource.client.describe_table(table_name: table_name)
    @logger.debug("Table #{table_name} exists")
  rescue Aws::DynamoDB::Errors::ResourceNotFoundException
    @logger.debug("Table #{table_name} doesn't exist")
    false
  rescue Aws::DynamoDB::Errors::ServiceError => e
    puts("Couldn't check for existence of #{table_name}:\n")
```

```
    puts("\t#{e.code}: #{e.message}")
    raise
  end
```

- Einzelheiten zur API finden Sie DescribeTablein der AWS SDK für Ruby API-Referenz.

SAP ABAP

SDK für SAP ABAP

> **Note**
>
> Es gibt noch mehr dazu. GitHub Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
TRY.
    oo_result = lo_dyn->describetable( iv_tablename = iv_table_name ).
    DATA(lv_tablename) = oo_result->get_table( )->ask_tablename( ).
    DATA(lv_tablearn) = oo_result->get_table( )->ask_tablearn( ).
    DATA(lv_tablestatus) = oo_result->get_table( )->ask_tablestatus( ).
    DATA(lv_itemcount) = oo_result->get_table( )->ask_itemcount( ).
    MESSAGE 'The table name is ' && lv_tablename
        && '. The table ARN is ' && lv_tablearn
        && '. The tablestatus is ' && lv_tablestatus
        && '. Item count is ' && lv_itemcount TYPE 'I'.
  CATCH /aws1/cx_dynresourcenotfoundex.
    MESSAGE 'The table ' && lv_tablename && ' does not exist' TYPE 'E'.
ENDTRY.
```

- Einzelheiten zur API finden Sie DescribeTablein der API-Referenz zum AWS SDK für SAP ABAP.

Eine vollständige Liste der AWS SDK-Entwicklerhandbücher und Codebeispiele finden Sie unterDynamoDB mit einem SDK verwenden AWS. Dieses Thema enthält auch Informationen zu den ersten Schritten und Details zu früheren SDK-Versionen.

Verwendung **`DescribeTimeToLive`** mit einem AWS SDK oder CLI

Die folgenden Code-Beispiele zeigen, wie `DescribeTimeToLive` verwendet wird.

Beispiele für Aktionen sind Codeauszüge aus größeren Programmen und müssen im Kontext ausgeführt werden. Im folgenden Codebeispiel können Sie diese Aktion im Kontext sehen:

- Arbeite mit Streams und Time-to-Live

CLI

AWS CLI

So zeigen Sie Time-to-Live-Einstellungen für eine Tabelle an

Im folgenden Beispiel für `describe-time-to-live` werden die Time-to-Live-Einstellungen für die `MusicCollection`-Tabelle angezeigt.

```
aws dynamodb describe-time-to-live \
    --table-name MusicCollection
```

Ausgabe:

```
{
    "TimeToLiveDescription": {
        "TimeToLiveStatus": "ENABLED",
        "AttributeName": "ttl"
    }
}
```

Weitere Informationen finden Sie unter Time to Live im Entwicklerhandbuch zu Amazon DynamoDB.

- Einzelheiten zur API finden Sie DescribeTimeToLivein der AWS CLI Befehlsreferenz.

Java

SDK für Java 2.x

Beschreiben Sie die TTL-Konfiguration für eine bestehende DynamoDB-Tabelle mithilfe von AWS SDK for Java 2.x.

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.DescribeTimeToLiveRequest;
import software.amazon.awssdk.services.dynamodb.model.DescribeTimeToLiveResponse;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.model.ResourceNotFoundException;

import java.util.logging.Level;
import java.util.logging.Logger;

    public DescribeTimeToLiveResponse describeTTL(final String tableName, final
 Region region) {
        final DescribeTimeToLiveRequest request =
            DescribeTimeToLiveRequest.builder().tableName(tableName).build();

        try (DynamoDbClient ddb = dynamoDbClient != null
            ? dynamoDbClient
            : DynamoDbClient.builder().region(region).build()) {
            return ddb.describeTimeToLive(request);
        } catch (ResourceNotFoundException e) {
            System.err.format(TABLE_NOT_FOUND_ERROR, tableName);
            throw e;
        } catch (DynamoDbException e) {
            System.err.println(e.getMessage());
            throw e;
        }
    }
```

- Einzelheiten zur API finden Sie DescribeTimeToLivein der AWS SDK for Java 2.x API-Referenz.

JavaScript

SDK für JavaScript (v3)

Beschreiben Sie die TTL-Konfiguration für eine bestehende DynamoDB-Tabelle mithilfe von AWS SDK für JavaScript.

```
import { DynamoDBClient, DescribeTimeToLiveCommand } from "@aws-sdk/client-
dynamodb";
```

```
export const describeTTL = async (tableName, region) => {
    const client = new DynamoDBClient({
        region: region,
        endpoint: `https://dynamodb.${region}.amazonaws.com`
    });

    try {
        const ttlDescription = await client.send(new
 DescribeTimeToLiveCommand({ TableName: tableName }));

        if (ttlDescription.TimeToLiveDescription.TimeToLiveStatus === 'ENABLED')
 {
            console.log("TTL is enabled for table %s.", tableName);
        } else {
            console.log("TTL is not enabled for table %s.", tableName);
        }

        return ttlDescription;
    } catch (e) {
        console.error(`Error describing table: ${e}`);
        throw e;
    }
}

// Example usage (commented out for testing)
// describeTTL('your-table-name', 'us-east-1');
```

- Einzelheiten zur API finden Sie DescribeTimeToLivein der AWS SDK für JavaScript API-Referenz.

Python

SDK für Python (Boto3)

Beschreiben Sie die TTL-Konfiguration für eine bestehende DynamoDB-Tabelle mithilfe von AWS SDK für Python (Boto3).

```
import boto3


def describe_ttl(table_name, region):
    """
```

```
    Describes TTL on an existing table, as well as a region.

    :param table_name: String representing the name of the table
    :param region: AWS Region of the table - example `us-east-1`
    :return: Time to live description.
    """
    try:
        dynamodb = boto3.resource("dynamodb", region_name=region)
        ttl_description = dynamodb.describe_time_to_live(TableName=table_name)
        print(
            f"TimeToLive for table {table_name} is status
 {ttl_description['TimeToLiveDescription']['TimeToLiveStatus']}"
        )

        return ttl_description
    except Exception as e:
        print(f"Error describing table: {e}")
        raise


# Enter your own table name and AWS region
describe_ttl("your-table-name", "us-east-1")
```

- Einzelheiten zur API finden Sie DescribeTimeToLivein AWS SDK for Python (Boto3) API Reference.

Eine vollständige Liste der AWS SDK-Entwicklerhandbücher und Codebeispiele finden Sie unter. DynamoDB mit einem SDK verwenden AWS Dieses Thema enthält auch Informationen zu den ersten Schritten und Details zu früheren SDK-Versionen.

ExecuteStatementMit einem AWS SDK verwenden

Die folgenden Code-Beispiele zeigen, wie `ExecuteStatement` verwendet wird.

Aktionsbeispiele sind Codeauszüge aus größeren Programmen und müssen im Kontext ausgeführt werden. Sie können diese Aktion in den folgenden Codebeispielen im Kontext sehen:

- Löschen von Daten mit PartiQL DELETE
- Einfügen von Daten mithilfe von PartiQL INSERT
- Abfragen einer Tabelle mit PartiQL

- Abfragen von Daten mit PartiQL SELECT
- Aktualisieren von Daten mit PartiQL UPDATE

.NET

SDK für .NET

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

Verwenden Sie eine INSERT-Anweisung, um ein Element hinzuzufügen.

```
        /// <summary>
        /// Inserts a single movie into the movies table.
        /// </summary>
        /// <param name="tableName">The name of the table.</param>
        /// <param name="movieTitle">The title of the movie to insert.</param>
        /// <param name="year">The year that the movie was released.</param>
        /// <returns>A Boolean value that indicates the success or failure of
        /// the INSERT operation.</returns>
        public static async Task<bool> InsertSingleMovie(string tableName, string
movieTitle, int year)
        {
            string insertBatch = $"INSERT INTO {tableName} VALUE {{'title': ?,
'year': ?}}";

            var response = await Client.ExecuteStatementAsync(new
ExecuteStatementRequest
            {
                Statement = insertBatch,
                Parameters = new List<AttributeValue>
                {
                    new AttributeValue { S = movieTitle },
                    new AttributeValue { N = year.ToString() },
                },
            });
```

```
            return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
        }
```

Verwenden Sie eine SELECT-Anweisung, um ein Element abzurufen.

```
        /// <summary>
        /// Uses a PartiQL SELECT statement to retrieve a single movie from the
        /// movie database.
        /// </summary>
        /// <param name="tableName">The name of the movie table.</param>
        /// <param name="movieTitle">The title of the movie to retrieve.</param>
        /// <returns>A list of movie data. If no movie matches the supplied
        /// title, the list is empty.</returns>
        public static async Task<List<Dictionary<string, AttributeValue>>>
GetSingleMovie(string tableName, string movieTitle)
        {
            string selectSingle = $"SELECT * FROM {tableName} WHERE title = ?";
            var parameters = new List<AttributeValue>
            {
                new AttributeValue { S = movieTitle },
            };

            var response = await Client.ExecuteStatementAsync(new
ExecuteStatementRequest
            {
                Statement = selectSingle,
                Parameters = parameters,
            });

            return response.Items;
        }
```

Verwenden einer SELECT-Anweisung, um ein Liste an Elementen abzurufen.

```
        /// <summary>
        /// Retrieve multiple movies by year using a SELECT statement.
        /// </summary>
```

```
        /// <param name="tableName">The name of the movie table.</param>
        /// <param name="year">The year the movies were released.</param>
        /// <returns></returns>
        public static async Task<List<Dictionary<string, AttributeValue>>>
GetMovies(string tableName, int year)
        {
            string selectSingle = $"SELECT * FROM {tableName} WHERE year = ?";
            var parameters = new List<AttributeValue>
            {
                new AttributeValue { N = year.ToString() },
            };

            var response = await Client.ExecuteStatementAsync(new
ExecuteStatementRequest
            {
                Statement = selectSingle,
                Parameters = parameters,
            });

            return response.Items;
        }
```

Verwenden Sie eine UPDATE-Anweisung, um ein Element zu aktualisieren.

```
        /// <summary>
        /// Updates a single movie in the table, adding information for the
        /// producer.
        /// </summary>
        /// <param name="tableName">the name of the table.</param>
        /// <param name="producer">The name of the producer.</param>
        /// <param name="movieTitle">The movie title.</param>
        /// <param name="year">The year the movie was released.</param>
        /// <returns>A Boolean value that indicates the success of the
        /// UPDATE operation.</returns>
        public static async Task<bool> UpdateSingleMovie(string tableName, string
producer, string movieTitle, int year)
        {
            string insertSingle = $"UPDATE {tableName} SET Producer=? WHERE title
= ? AND year = ?";
```

```
            var response = await Client.ExecuteStatementAsync(new
ExecuteStatementRequest
            {
                Statement = insertSingle,
                Parameters = new List<AttributeValue>
                {
                    new AttributeValue { S = producer },
                    new AttributeValue { S = movieTitle },
                    new AttributeValue { N = year.ToString() },
                },
            });

            return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
        }
```

Verwenden einer DELETE-Anweisung, um einen einzelnen Film zu löschen.

```
        /// <summary>
        /// Deletes a single movie from the table.
        /// </summary>
        /// <param name="tableName">The name of the table.</param>
        /// <param name="movieTitle">The title of the movie to delete.</param>
        /// <param name="year">The year that the movie was released.</param>
        /// <returns>A Boolean value that indicates the success of the
        /// DELETE operation.</returns>
        public static async Task<bool> DeleteSingleMovie(string tableName, string
movieTitle, int year)
        {
            var deleteSingle = $"DELETE FROM {tableName} WHERE title = ? AND year
= ?";

            var response = await Client.ExecuteStatementAsync(new
ExecuteStatementRequest
            {
                Statement = deleteSingle,
                Parameters = new List<AttributeValue>
                {
                    new AttributeValue { S = movieTitle },
                    new AttributeValue { N = year.ToString() },
                },
```

```
            });

            return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
        }
```

- Einzelheiten zur API finden Sie ExecuteStatementin der AWS SDK für .NET API-Referenz.

C++

SDK für C++

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

Verwenden Sie eine INSERT-Anweisung, um ein Element hinzuzufügen.

```
    Aws::DynamoDB::DynamoDBClient dynamoClient(clientConfiguration);

    // 2. Add a new movie using an "Insert" statement. (ExecuteStatement)
    Aws::String title;
    float rating;
    int year;
    Aws::String plot;
    {
        title = askQuestion(
                "Enter the title of a movie you want to add to the table: ");
        year = askQuestionForInt("What year was it released? ");
        rating = askQuestionForFloatRange("On a scale of 1 - 10, how do you rate
it? ",
                                          1, 10);
        plot = askQuestion("Summarize the plot for me: ");

        Aws::DynamoDB::Model::ExecuteStatementRequest request;
        std::stringstream sqlStream;
        sqlStream << "INSERT INTO \"" << MOVIE_TABLE_NAME << "\" VALUE {'"
                  << TITLE_KEY << "': ?, '" << YEAR_KEY << "': ?, '"
                  << INFO_KEY << "': ?}";
```

```
        request.SetStatement(sqlStream.str());

        // Create the parameter attributes.
        Aws::Vector<Aws::DynamoDB::Model::AttributeValue> attributes;
        attributes.push_back(Aws::DynamoDB::Model::AttributeValue().SetS(title));
        attributes.push_back(Aws::DynamoDB::Model::AttributeValue().SetN(year));

        Aws::DynamoDB::Model::AttributeValue infoMapAttribute;

        std::shared_ptr<Aws::DynamoDB::Model::AttributeValue> ratingAttribute =
Aws::MakeShared<Aws::DynamoDB::Model::AttributeValue>(
                ALLOCATION_TAG.c_str());
        ratingAttribute->SetN(rating);
        infoMapAttribute.AddMEntry(RATING_KEY, ratingAttribute);

        std::shared_ptr<Aws::DynamoDB::Model::AttributeValue> plotAttribute =
Aws::MakeShared<Aws::DynamoDB::Model::AttributeValue>(
                ALLOCATION_TAG.c_str());
        plotAttribute->SetS(plot);
        infoMapAttribute.AddMEntry(PLOT_KEY, plotAttribute);
        attributes.push_back(infoMapAttribute);
        request.SetParameters(attributes);

        Aws::DynamoDB::Model::ExecuteStatementOutcome outcome =
dynamoClient.ExecuteStatement(
                request);

        if (!outcome.IsSuccess()) {
            std::cerr << "Failed to add a movie: " <<
outcome.GetError().GetMessage()
                      << std::endl;
            return false;
        }
    }
```

Verwenden Sie eine SELECT-Anweisung, um ein Element abzurufen.

```
    //  3. Get the data for the movie using a "Select" statement.
(ExecuteStatement)
    {
        Aws::DynamoDB::Model::ExecuteStatementRequest request;
```

```
        std::stringstream sqlStream;
        sqlStream << "SELECT * FROM  \"" << MOVIE_TABLE_NAME << "\" WHERE "
                  << TITLE_KEY << "=? and " << YEAR_KEY << "=?";

        request.SetStatement(sqlStream.str());

        Aws::Vector<Aws::DynamoDB::Model::AttributeValue> attributes;
        attributes.push_back(Aws::DynamoDB::Model::AttributeValue().SetS(title));
        attributes.push_back(Aws::DynamoDB::Model::AttributeValue().SetN(year));
        request.SetParameters(attributes);

        Aws::DynamoDB::Model::ExecuteStatementOutcome outcome =
dynamoClient.ExecuteStatement(
                request);

        if (!outcome.IsSuccess()) {
            std::cerr << "Failed to retrieve movie information: "
                      << outcome.GetError().GetMessage() << std::endl;
            return false;
        }
        else {
            // Print the retrieved movie information.
            const Aws::DynamoDB::Model::ExecuteStatementResult &result =
outcome.GetResult();

            const Aws::Vector<Aws::Map<Aws::String,
Aws::DynamoDB::Model::AttributeValue>> &items = result.GetItems();

            if (items.size() == 1) {
                printMovieInfo(items[0]);
            }
            else {
                std::cerr << "Error: " << items.size() << " movies were
retrieved. "
                          << " There should be only one movie." << std::endl;
            }
        }
    }
```

Verwenden Sie eine UPDATE-Anweisung, um ein Element zu aktualisieren.

```
   //  4. Update the data for the movie using an "Update" statement.
(ExecuteStatement)
   {
       rating = askQuestionForFloatRange(
               Aws::String("\nLet's update your movie.\nYou rated it  ") +
               std::to_string(rating)
               + ", what new rating would you give it? ", 1, 10);

       Aws::DynamoDB::Model::ExecuteStatementRequest request;
       std::stringstream sqlStream;
       sqlStream << "UPDATE \"" << MOVIE_TABLE_NAME << "\" SET "
                 << INFO_KEY << "." << RATING_KEY << "=? WHERE "
                 << TITLE_KEY << "=? AND " << YEAR_KEY << "=?";

       request.SetStatement(sqlStream.str());

       Aws::Vector<Aws::DynamoDB::Model::AttributeValue> attributes;

attributes.push_back(Aws::DynamoDB::Model::AttributeValue().SetN(rating));
       attributes.push_back(Aws::DynamoDB::Model::AttributeValue().SetS(title));
       attributes.push_back(Aws::DynamoDB::Model::AttributeValue().SetN(year));

       request.SetParameters(attributes);

       Aws::DynamoDB::Model::ExecuteStatementOutcome outcome =
dynamoClient.ExecuteStatement(
               request);

       if (!outcome.IsSuccess()) {
           std::cerr << "Failed to update a movie: "
                     << outcome.GetError().GetMessage();
           return false;
       }
   }
```

Verwenden Sie eine DELETE-Anweisung, um ein Element zu löschen.

```
   // 6. Delete the movie using a "Delete" statement. (ExecuteStatement)
   {
       Aws::DynamoDB::Model::ExecuteStatementRequest request;
       std::stringstream sqlStream;
       sqlStream << "DELETE FROM  \"" << MOVIE_TABLE_NAME << "\" WHERE "
```

```
                << TITLE_KEY << "=? and " << YEAR_KEY << "=?";

        request.SetStatement(sqlStream.str());

        Aws::Vector<Aws::DynamoDB::Model::AttributeValue> attributes;
        attributes.push_back(Aws::DynamoDB::Model::AttributeValue().SetS(title));
        attributes.push_back(Aws::DynamoDB::Model::AttributeValue().SetN(year));
        request.SetParameters(attributes);

        Aws::DynamoDB::Model::ExecuteStatementOutcome outcome =
 dynamoClient.ExecuteStatement(
                request);
        if (!outcome.IsSuccess()) {
            std::cerr << "Failed to delete the movie: "
                      << outcome.GetError().GetMessage() << std::endl;
            return false;
        }
    }
```

- Einzelheiten zur API finden Sie ExecuteStatementin der AWS SDK für C++ API-Referenz.

Go

SDK für Go V2

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

Definieren Sie eine Funktionsempfängerstruktur für das Beispiel.

```
import (
 "context"
 "fmt"
 "log"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/attributevalue"
```

```
 "github.com/aws/aws-sdk-go-v2/service/dynamodb"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb/types"
)

// PartiQLRunner encapsulates the Amazon DynamoDB service actions used in the
// PartiQL examples. It contains a DynamoDB service client that is used to act on
 the
// specified table.
type PartiQLRunner struct {
 DynamoDbClient *dynamodb.Client
 TableName      string
}
```

Verwenden Sie eine INSERT-Anweisung, um ein Element hinzuzufügen.

```
// AddMovie runs a PartiQL INSERT statement to add a movie to the DynamoDB table.
func (runner PartiQLRunner) AddMovie(ctx context.Context, movie Movie) error {
 params, err := attributevalue.MarshalList([]interface{}{movie.Title, movie.Year,
 movie.Info})
 if err != nil {
  panic(err)
 }
 _, err = runner.DynamoDbClient.ExecuteStatement(ctx,
 &dynamodb.ExecuteStatementInput{
  Statement: aws.String(
   fmt.Sprintf("INSERT INTO \"%v\" VALUE {'title': ?, 'year': ?, 'info': ?}",
    runner.TableName)),
  Parameters: params,
 })
 if err != nil {
  log.Printf("Couldn't insert an item with PartiQL. Here's why: %v\n", err)
 }
 return err
}
```

Verwenden Sie eine SELECT-Anweisung, um ein Element abzurufen.

```
// GetMovie runs a PartiQL SELECT statement to get a movie from the DynamoDB
 table by
// title and year.
func (runner PartiQLRunner) GetMovie(ctx context.Context, title string, year int)
 (Movie, error) {
 var movie Movie
 params, err := attributevalue.MarshalList([]interface{}{title, year})
 if err != nil {
  panic(err)
 }
 response, err := runner.DynamoDbClient.ExecuteStatement(ctx,
 &dynamodb.ExecuteStatementInput{
  Statement: aws.String(
   fmt.Sprintf("SELECT * FROM \"%v\" WHERE title=? AND year=?",
    runner.TableName)),
  Parameters: params,
 })
 if err != nil {
  log.Printf("Couldn't get info about %v. Here's why: %v\n", title, err)
 } else {
  err = attributevalue.UnmarshalMap(response.Items[0], &movie)
  if err != nil {
   log.Printf("Couldn't unmarshal response. Here's why: %v\n", err)
  }
 }
 return movie, err
}
```

Verwenden Sie eine SELECT-Anweisung, um eine Liste der Elemente abzurufen und die Ergebnisse zu projizieren.

```
// GetAllMovies runs a PartiQL SELECT statement to get all movies from the
 DynamoDB table.
// pageSize is not typically required and is used to show how to paginate the
 results.
// The results are projected to return only the title and rating of each movie.
func (runner PartiQLRunner) GetAllMovies(ctx context.Context, pageSize int32)
 ([]map[string]interface{}, error) {
 var output []map[string]interface{}
 var response *dynamodb.ExecuteStatementOutput
```

```
 var err error
 var nextToken *string
 for moreData := true; moreData; {
  response, err = runner.DynamoDbClient.ExecuteStatement(ctx,
 &dynamodb.ExecuteStatementInput{
   Statement: aws.String(
    fmt.Sprintf("SELECT title, info.rating FROM \"%v\"", runner.TableName)),
   Limit:     aws.Int32(pageSize),
   NextToken: nextToken,
  })
  if err != nil {
   log.Printf("Couldn't get movies. Here's why: %v\n", err)
   moreData = false
  } else {
   var pageOutput []map[string]interface{}
   err = attributevalue.UnmarshalListOfMaps(response.Items, &pageOutput)
   if err != nil {
    log.Printf("Couldn't unmarshal response. Here's why: %v\n", err)
   } else {
    log.Printf("Got a page of length %v.\n", len(response.Items))
    output = append(output, pageOutput...)
   }
   nextToken = response.NextToken
   moreData = nextToken != nil
  }
 }
 return output, err
}
```

Verwenden Sie eine UPDATE-Anweisung, um ein Element zu aktualisieren.

```
// UpdateMovie runs a PartiQL UPDATE statement to update the rating of a movie
 that
// already exists in the DynamoDB table.
func (runner PartiQLRunner) UpdateMovie(ctx context.Context, movie Movie, rating
 float64) error {
 params, err := attributevalue.MarshalList([]interface{}{rating, movie.Title,
 movie.Year})
 if err != nil {
  panic(err)
```

```
 }
 _, err = runner.DynamoDbClient.ExecuteStatement(ctx,
 &dynamodb.ExecuteStatementInput{
  Statement: aws.String(
   fmt.Sprintf("UPDATE \"%v\" SET info.rating=? WHERE title=? AND year=?",
    runner.TableName)),
  Parameters: params,
 })
 if err != nil {
  log.Printf("Couldn't update movie %v. Here's why: %v\n", movie.Title, err)
 }
 return err
}
```

Verwenden Sie eine DELETE-Anweisung, um ein Element zu löschen.

```
// DeleteMovie runs a PartiQL DELETE statement to remove a movie from the
 DynamoDB table.
func (runner PartiQLRunner) DeleteMovie(ctx context.Context, movie Movie) error {
 params, err := attributevalue.MarshalList([]interface{}{movie.Title,
 movie.Year})
 if err != nil {
  panic(err)
 }
 _, err = runner.DynamoDbClient.ExecuteStatement(ctx,
 &dynamodb.ExecuteStatementInput{
  Statement: aws.String(
   fmt.Sprintf("DELETE FROM \"%v\" WHERE title=? AND year=?",
    runner.TableName)),
  Parameters: params,
 })
 if err != nil {
  log.Printf("Couldn't delete %v from the table. Here's why: %v\n", movie.Title,
 err)
 }
 return err
}
```

Definieren Sie eine Movie-Struktur, die in diesem Beispiel verwendet wird.

```
import (
 "archive/zip"
 "bytes"
 "encoding/json"
 "fmt"
 "io"
 "log"
 "net/http"

 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/attributevalue"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb/types"
)

// Movie encapsulates data about a movie. Title and Year are the composite
 primary key
// of the movie in Amazon DynamoDB. Title is the sort key, Year is the partition
 key,
// and Info is additional data.
type Movie struct {
 Title string                 `dynamodbav:"title"`
 Year  int                    `dynamodbav:"year"`
 Info  map[string]interface{} `dynamodbav:"info"`
}

// GetKey returns the composite primary key of the movie in a format that can be
// sent to DynamoDB.
func (movie Movie) GetKey() map[string]types.AttributeValue {
 title, err := attributevalue.Marshal(movie.Title)
 if err != nil {
  panic(err)
 }
 year, err := attributevalue.Marshal(movie.Year)
 if err != nil {
  panic(err)
 }
 return map[string]types.AttributeValue{"title": title, "year": year}
}

// String returns the title, year, rating, and plot of a movie, formatted for the
 example.
func (movie Movie) String() string {
```

```
 return fmt.Sprintf("%v\n\tReleased: %v\n\tRating: %v\n\tPlot: %v\n",
  movie.Title, movie.Year, movie.Info["rating"], movie.Info["plot"])
}
```

- Einzelheiten zur API finden Sie ExecuteStatementin der AWS SDK für Go API-Referenz.

JavaScript

SDK für JavaScript (v3)

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

Erstellen Sie ein Element mithilfe von PartiQL.

```
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";

import {
  ExecuteStatementCommand,
  DynamoDBDocumentClient,
} from "@aws-sdk/lib-dynamodb";

const client = new DynamoDBClient({});
const docClient = DynamoDBDocumentClient.from(client);

export const main = async () => {
  const command = new ExecuteStatementCommand({
    Statement: `INSERT INTO Flowers value {'Name':?}`,
    Parameters: ["Rose"],
  });

  const response = await docClient.send(command);
  console.log(response);
  return response;
};
```

Rufen Sie ein Element mithilfe von PartiQL ab.

```
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";

import {
  ExecuteStatementCommand,
  DynamoDBDocumentClient,
} from "@aws-sdk/lib-dynamodb";

const client = new DynamoDBClient({});
const docClient = DynamoDBDocumentClient.from(client);

export const main = async () => {
  const command = new ExecuteStatementCommand({
    Statement: "SELECT * FROM CloudTypes WHERE IsStorm=?",
    Parameters: [false],
    ConsistentRead: true,
  });

  const response = await docClient.send(command);
  console.log(response);
  return response;
};
```

Aktualisieren Sie ein Element mithilfe von PartiQL.

```
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";

import {
  ExecuteStatementCommand,
  DynamoDBDocumentClient,
} from "@aws-sdk/lib-dynamodb";

const client = new DynamoDBClient({});
const docClient = DynamoDBDocumentClient.from(client);

export const main = async () => {
  const command = new ExecuteStatementCommand({
    Statement: "UPDATE EyeColors SET IsRecessive=? where Color=?",
    Parameters: [true, "blue"],
  });
```

```
  const response = await docClient.send(command);
  console.log(response);
  return response;
};
```

Löschen Sie ein Element mithilfe von PartiQL.

```
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";

import {
  ExecuteStatementCommand,
  DynamoDBDocumentClient,
} from "@aws-sdk/lib-dynamodb";

const client = new DynamoDBClient({});
const docClient = DynamoDBDocumentClient.from(client);

export const main = async () => {
  const command = new ExecuteStatementCommand({
    Statement: "DELETE FROM PaintColors where Name=?",
    Parameters: ["Purple"],
  });

  const response = await docClient.send(command);
  console.log(response);
  return response;
};
```

- Einzelheiten zur API finden Sie ExecuteStatementin der AWS SDK für JavaScript API-Referenz.

PHP

SDK für PHP

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
    public function insertItemByPartiQL(string $statement, array $parameters)
    {
        $this->dynamoDbClient->executeStatement([
            'Statement' => "$statement",
            'Parameters' => $parameters,
        ]);
    }

    public function getItemByPartiQL(string $tableName, array $key): Result
    {
        list($statement, $parameters) = $this-
>buildStatementAndParameters("SELECT", $tableName, $key['Item']);

        return $this->dynamoDbClient->executeStatement([
            'Parameters' => $parameters,
            'Statement' => $statement,
        ]);
    }

    public function updateItemByPartiQL(string $statement, array $parameters)
    {
        $this->dynamoDbClient->executeStatement([
            'Statement' => $statement,
            'Parameters' => $parameters,
        ]);
    }

    public function deleteItemByPartiQL(string $statement, array $parameters)
    {
        $this->dynamoDbClient->executeStatement([
            'Statement' => $statement,
            'Parameters' => $parameters,
        ]);
    }
```

- Einzelheiten zur API finden Sie ExecuteStatementin der AWS SDK für PHP API-Referenz.

Python

SDK für Python (Boto3)

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
class PartiQLWrapper:
    """
    Encapsulates a DynamoDB resource to run PartiQL statements.
    """

    def __init__(self, dyn_resource):
        """
        :param dyn_resource: A Boto3 DynamoDB resource.
        """
        self.dyn_resource = dyn_resource


    def run_partiql(self, statement, params):
        """
        Runs a PartiQL statement. A Boto3 resource is used even though
        `execute_statement` is called on the underlying `client` object because
 the
        resource transforms input and output from plain old Python objects
 (POPOs) to
        the DynamoDB format. If you create the client directly, you must do these
        transforms yourself.

        :param statement: The PartiQL statement.
        :param params: The list of PartiQL parameters. These are applied to the
                       statement in the order they are listed.
        :return: The items returned from the statement, if any.
        """
        try:
            output = self.dyn_resource.meta.client.execute_statement(
                Statement=statement, Parameters=params
            )
        except ClientError as err:
```

```
            if err.response["Error"]["Code"] == "ResourceNotFoundException":
                logger.error(
                    "Couldn't execute PartiQL '%s' because the table does not
 exist.",
                    statement,
                )
            else:
                logger.error(
                    "Couldn't execute PartiQL '%s'. Here's why: %s: %s",
                    statement,
                    err.response["Error"]["Code"],
                    err.response["Error"]["Message"],
                )
            raise
        else:
            return output
```

- Einzelheiten zur API finden Sie ExecuteStatementin AWS SDK for Python (Boto3) API Reference.

Ruby

SDK für Ruby

> **Note**
>
> Es gibt noch mehr dazu. GitHub Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

Wählen Sie ein einzelnes Element mithilfe von PartiQL aus.

```
class DynamoDBPartiQLSingle
  attr_reader :dynamo_resource, :table

  def initialize(table_name)
    client = Aws::DynamoDB::Client.new(region: 'us-east-1')
    @dynamodb = Aws::DynamoDB::Resource.new(client: client)
    @table = @dynamodb.table(table_name)
```

```
  end

  # Gets a single record from a table using PartiQL.
  # Note: To perform more fine-grained selects,
  # use the Client.query instance method instead.
  #
  # @param title [String] The title of the movie to search.
  # @return [Aws::DynamoDB::Types::ExecuteStatementOutput]
  def select_item_by_title(title)
    request = {
      statement: "SELECT * FROM \"#{@table.name}\" WHERE title=?",
      parameters: [title]
    }
    @dynamodb.client.execute_statement(request)
  end
```

Aktualisieren Sie ein einzelnes Element mithilfe von PartiQL.

```
class DynamoDBPartiQLSingle
  attr_reader :dynamo_resource, :table

  def initialize(table_name)
    client = Aws::DynamoDB::Client.new(region: 'us-east-1')
    @dynamodb = Aws::DynamoDB::Resource.new(client: client)
    @table = @dynamodb.table(table_name)
  end

  # Updates a single record from a table using PartiQL.
  #
  # @param title [String] The title of the movie to update.
  # @param year [Integer] The year the movie was released.
  # @param rating [Float] The new rating to assign the title.
  # @return [Aws::DynamoDB::Types::ExecuteStatementOutput]
  def update_rating_by_title(title, year, rating)
    request = {
      statement: "UPDATE \"#{@table.name}\" SET info.rating=? WHERE title=? and
 year=?",
      parameters: [{ "N": rating }, title, year]
    }
    @dynamodb.client.execute_statement(request)
  end
```

Fügen Sie ein einzelnes Element mithilfe von PartiQL hinzu.

```
class DynamoDBPartiQLSingle
  attr_reader :dynamo_resource, :table

  def initialize(table_name)
    client = Aws::DynamoDB::Client.new(region: 'us-east-1')
    @dynamodb = Aws::DynamoDB::Resource.new(client: client)
    @table = @dynamodb.table(table_name)
  end

  # Adds a single record to a table using PartiQL.
  #
  # @param title [String] The title of the movie to update.
  # @param year [Integer] The year the movie was released.
  # @param plot [String] The plot of the movie.
  # @param rating [Float] The new rating to assign the title.
  # @return [Aws::DynamoDB::Types::ExecuteStatementOutput]
  def insert_item(title, year, plot, rating)
    request = {
      statement: "INSERT INTO \"#{@table.name}\" VALUE {'title': ?, 'year': ?,
 'info': ?}",
      parameters: [title, year, { 'plot': plot, 'rating': rating }]
    }
    @dynamodb.client.execute_statement(request)
  end
```

Löschen Sie ein einzelnes Element mithilfe von PartiQL.

```
class DynamoDBPartiQLSingle
  attr_reader :dynamo_resource, :table

  def initialize(table_name)
    client = Aws::DynamoDB::Client.new(region: 'us-east-1')
    @dynamodb = Aws::DynamoDB::Resource.new(client: client)
    @table = @dynamodb.table(table_name)
  end

  # Deletes a single record from a table using PartiQL.
  #
  # @param title [String] The title of the movie to update.
  # @param year [Integer] The year the movie was released.
```

```
  # @return [Aws::DynamoDB::Types::ExecuteStatementOutput]
  def delete_item_by_title(title, year)
    request = {
      statement: "DELETE FROM \"#{@table.name}\" WHERE title=? and year=?",
      parameters: [title, year]
    }
    @dynamodb.client.execute_statement(request)
  end
```

- Einzelheiten zur API finden Sie ExecuteStatementin der AWS SDK für Ruby API-Referenz.

Eine vollständige Liste der AWS SDK-Entwicklerhandbücher und Codebeispiele finden Sie unterDynamoDB mit einem SDK verwenden AWS. Dieses Thema enthält auch Informationen zu den ersten Schritten und Details zu früheren SDK-Versionen.

Verwendung **`GetItem`** mit einem AWS SDK oder CLI

Die folgenden Code-Beispiele zeigen, wie `GetItem` verwendet wird.

Aktionsbeispiele sind Codeauszüge aus größeren Programmen und müssen im Kontext ausgeführt werden. Sie können diese Aktion in den folgenden Codebeispielen im Kontext sehen:

- Kennenlernen der Grundlagen
- Beschleunigen von Lesevorgängen mit DAX
- Erstellen und Verwalten von globalen MRSC-Tabellen
- Arbeiten mit globalen Tabellen und letztendlicher multiregionaler Konsistenz (Multi-Region Eventual Consistency, MREC)

.NET

SDK für .NET (v4)

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
    /// <summary>
    /// Gets information about an existing movie from the table.
    /// </summary>
    /// <param name="newMovie">A Movie object containing information about
    /// the movie to retrieve.</param>
    /// <param name="tableName">The name of the table containing the movie.</
param>
    /// <returns>A Dictionary object containing information about the item
    /// retrieved.</returns>
    public async Task<Dictionary<string, AttributeValue>> GetItemAsync(Movie
 newMovie, string tableName)
    {
        try
        {
            var key = new Dictionary<string, AttributeValue>
            {
                ["title"] = new AttributeValue { S = newMovie.Title },
                ["year"] = new AttributeValue { N = newMovie.Year.ToString() },
            };

            var request = new GetItemRequest
            {
                Key = key,
                TableName = tableName,
            };

            var response = await _amazonDynamoDB.GetItemAsync(request);
            return response.Item;
        }
        catch (ResourceNotFoundException ex)
        {
            Console.WriteLine($"Table {tableName} was not found. {ex.Message}");
            return new Dictionary<string, AttributeValue>();
        }
        catch (AmazonDynamoDBException ex)
        {
            Console.WriteLine($"An Amazon DynamoDB error occurred while getting
 item. {ex.Message}");
            throw;
        }
        catch (Exception ex)
        {
```

```
            Console.WriteLine($"An error occurred while getting item.
 {ex.Message}");
            throw;
        }
    }
```

- Einzelheiten zur API finden Sie GetItemin der AWS SDK für .NET API-Referenz.

Bash

AWS CLI mit Bash-Skript

> **Note**
>
> Es gibt noch mehr dazu. GitHub Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
###############################################################################
# function dynamodb_get_item
#
# This function gets an item from a DynamoDB table.
#
# Parameters:
#       -n table_name  -- The name of the table.
#       -k keys  -- Path to json file containing the keys that identify the item
 to get.
#       [-q query]  -- Optional JMESPath query expression.
#
#  Returns:
#       The item as text output.
#  And:
#       0 - If successful.
#       1 - If it fails.
###############################################################################
function dynamodb_get_item() {
  local table_name keys query response
  local option OPTARG # Required to use getopts command in a function.

  # ######################################
```

```
  # Function usage explanation
  ####################################
  function usage() {
    echo "function dynamodb_get_item"
    echo "Get an item from a DynamoDB table."
    echo " -n table_name  -- The name of the table."
    echo " -k keys  -- Path to json file containing the keys that identify the
 item to get."
    echo " [-q query]  -- Optional JMESPath query expression."
    echo ""
  }
  query=""
  while getopts "n:k:q:h" option; do
    case "${option}" in
      n) table_name="${OPTARG}" ;;
      k) keys="${OPTARG}" ;;
      q) query="${OPTARG}" ;;
      h)
        usage
        return 0
        ;;
      \?)
        echo "Invalid parameter"
        usage
        return 1
        ;;
    esac
  done
  export OPTIND=1

  if [[ -z "$table_name" ]]; then
    errecho "ERROR: You must provide a table name with the -n parameter."
    usage
    return 1
  fi

  if [[ -z "$keys" ]]; then
    errecho "ERROR: You must provide a keys json file path the -k parameter."
    usage
    return 1
  fi

  if [[ -n "$query" ]]; then
    response=$(aws dynamodb get-item \
```

```
      --table-name "$table_name" \
      --key file://"$keys" \
      --output text \
      --query "$query")
  else
    response=$(
      aws dynamodb get-item \
        --table-name "$table_name" \
        --key file://"$keys" \
        --output text
    )
  fi

  local error_code=${?}

  if [[ $error_code -ne 0 ]]; then
    aws_cli_error_log $error_code
    errecho "ERROR: AWS reports get-item operation failed.$response"
    return 1
  fi

  if [[ -n "$query" ]]; then
    echo "$response" | sed "/^\t/s/\t//1" # Remove initial tab that the JMSEPath
 query inserts on some strings.
  else
    echo "$response"
  fi

  return 0
}
```

Die in diesem Beispiel verwendeten Hilfsfunktionen.

```
###############################################################################
# function errecho
#
# This function outputs everything sent to it to STDERR (standard error output).
###############################################################################
function errecho() {
  printf "%s\n" "$*" 1>&2
}
```

```
###############################################################################
# function aws_cli_error_log()
#
# This function is used to log the error messages from the AWS CLI.
#
# See https://docs.aws.amazon.com/cli/latest/topic/return-codes.html#cli-aws-
help-return-codes.
#
# The function expects the following argument:
#         $1 - The error code returned by the AWS CLI.
#
#  Returns:
#          0: - Success.
#
###############################################################################
function aws_cli_error_log() {
  local err_code=$1
  errecho "Error code : $err_code"
  if [ "$err_code" == 1 ]; then
    errecho "  One or more S3 transfers failed."
  elif [ "$err_code" == 2 ]; then
    errecho "  Command line failed to parse."
  elif [ "$err_code" == 130 ]; then
    errecho "  Process received SIGINT."
  elif [ "$err_code" == 252 ]; then
    errecho "  Command syntax invalid."
  elif [ "$err_code" == 253 ]; then
    errecho "  The system environment or configuration was invalid."
  elif [ "$err_code" == 254 ]; then
    errecho "  The service returned an error."
  elif [ "$err_code" == 255 ]; then
    errecho "  255 is a catch-all error."
  fi

  return 0
}
```

- Einzelheiten zur API finden Sie GetItemin der AWS CLI Befehlsreferenz.

C++

SDK für C++

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
//! Get an item from an Amazon DynamoDB table.
/*!
  \sa getItem()
  \param tableName: The table name.
  \param partitionKey: The partition key.
  \param partitionValue: The value for the partition key.
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
 */

bool AwsDoc::DynamoDB::getItem(const Aws::String &tableName,
                               const Aws::String &partitionKey,
                               const Aws::String &partitionValue,
                               const Aws::Client::ClientConfiguration
 &clientConfiguration) {
    Aws::DynamoDB::DynamoDBClient dynamoClient(clientConfiguration);
    Aws::DynamoDB::Model::GetItemRequest request;

    // Set up the request.
    request.SetTableName(tableName);
    request.AddKey(partitionKey,
                   Aws::DynamoDB::Model::AttributeValue().SetS(partitionValue));

    // Retrieve the item's fields and values.
    const Aws::DynamoDB::Model::GetItemOutcome &outcome =
 dynamoClient.GetItem(request);
    if (outcome.IsSuccess()) {
        // Reference the retrieved fields/values.
        const Aws::Map<Aws::String, Aws::DynamoDB::Model::AttributeValue> &item =
 outcome.GetResult().GetItem();
        if (!item.empty()) {
            // Output each retrieved field and its value.
```

```
            for (const auto &i: item)
                std::cout << "Values: " << i.first << ": " << i.second.GetS()
                          << std::endl;
        }
        else {
            std::cout << "No item found with the key " << partitionKey <<
 std::endl;
        }
    }
    else {
        std::cout << "Failed to get item: " << outcome.GetError().GetMessage();
    }

    return outcome.IsSuccess();
}
```

- Einzelheiten zur API finden Sie GetItemin der AWS SDK für C++ API-Referenz.

CLI

AWS CLI

Beispiel 1: So lesen Sie ein Element in einer Tabelle

Das folgende Beispiel für `get-item` ruft ein Element aus der `MusicCollection`-Tabelle ab. Die Tabelle hat einen hash-and-range Primärschlüssel (`ArtistundSongTitle`), daher müssen Sie diese beiden Attribute angeben. Der Befehl fragt auch Informationen über die Lesekapazität ab, die von der Operation verbraucht wird.

```
aws dynamodb get-item \
    --table-name MusicCollection \
    --key file://key.json \
    --return-consumed-capacity TOTAL
```

Inhalt von `key.json`:

```
{
    "Artist": {"S": "Acme Band"},
    "SongTitle": {"S": "Happy Day"}
}
```

Ausgabe:

```
{
    "Item": {
        "AlbumTitle": {
            "S": "Songs About Life"
        },
        "SongTitle": {
            "S": "Happy Day"
        },
        "Artist": {
            "S": "Acme Band"
        }
    },
    "ConsumedCapacity": {
        "TableName": "MusicCollection",
        "CapacityUnits": 0.5
    }
}
```

Weitere Informationen finden Sie unter Lesen eines Elements im Entwicklerhandbuch zu Amazon DynamoDB.

Beispiel 2: So lesen Sie ein Element mit einem konsistenten Lesevorgang

Das folgende Beispiel ruft mithilfe von strikt konsistenten Lesevorgängen ein Element aus der `MusicCollection`-Tabelle ab.

```
aws dynamodb get-item \
    --table-name MusicCollection \
    --key file://key.json \
    --consistent-read \
    --return-consumed-capacity TOTAL
```

Inhalt von `key.json`:

```
{
    "Artist": {"S": "Acme Band"},
    "SongTitle": {"S": "Happy Day"}
}
```

Ausgabe:

```
{
    "Item": {
        "AlbumTitle": {
            "S": "Songs About Life"
        },
        "SongTitle": {
            "S": "Happy Day"
        },
        "Artist": {
            "S": "Acme Band"
        }
    },
    "ConsumedCapacity": {
        "TableName": "MusicCollection",
        "CapacityUnits": 1.0
    }
}
```

Weitere Informationen finden Sie unter Lesen eines Elements im Entwicklerhandbuch zu Amazon DynamoDB.

Beispiel 3: So rufen Sie bestimmte Attribute eines Elements ab

Im folgenden Beispiel wird ein Projektionsausdruck verwendet, um nur drei Attribute des gewünschten Elements abzurufen.

```
aws dynamodb get-item \
    --table-name ProductCatalog \
    --key '{"Id": {"N": "102"}}' \
    --projection-expression "#T, #C, #P" \
    --expression-attribute-names file://names.json
```

Inhalt von `names.json`:

```
{
    "#T": "Title",
    "#C": "ProductCategory",
    "#P": "Price"
}
```

Ausgabe:

```
{
    "Item": {
        "Price": {
            "N": "20"
        },
        "Title": {
            "S": "Book 102 Title"
        },
        "ProductCategory": {
            "S": "Book"
        }
    }
}
```

Weitere Informationen finden Sie unter Lesen eines Elements im Entwicklerhandbuch zu Amazon DynamoDB.

- Einzelheiten zur API finden Sie GetItemunter AWS CLI Befehlsreferenz.

Go

SDK für Go V2

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
import (
 "context"
 "errors"
 "log"
 "time"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/attributevalue"
 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/expression"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb/types"
```

```
)

// TableBasics encapsulates the Amazon DynamoDB service actions used in the
 examples.
// It contains a DynamoDB service client that is used to act on the specified
 table.
type TableBasics struct {
 DynamoDbClient *dynamodb.Client
 TableName      string
}



// GetMovie gets movie data from the DynamoDB table by using the primary
 composite key
// made of title and year.
func (basics TableBasics) GetMovie(ctx context.Context, title string, year int)
 (Movie, error) {
 movie := Movie{Title: title, Year: year}
 response, err := basics.DynamoDbClient.GetItem(ctx, &dynamodb.GetItemInput{
  Key: movie.GetKey(), TableName: aws.String(basics.TableName),
 })
 if err != nil {
  log.Printf("Couldn't get info about %v. Here's why: %v\n", title, err)
 } else {
  err = attributevalue.UnmarshalMap(response.Item, &movie)
  if err != nil {
   log.Printf("Couldn't unmarshal response. Here's why: %v\n", err)
  }
 }
 return movie, err
}
```

Definieren Sie eine Movie-Struktur, die in diesem Beispiel verwendet wird.

```
import (
 "archive/zip"
 "bytes"
 "encoding/json"
 "fmt"
```

```
 "io"
 "log"
 "net/http"

 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/attributevalue"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb/types"
)

// Movie encapsulates data about a movie. Title and Year are the composite
 primary key
// of the movie in Amazon DynamoDB. Title is the sort key, Year is the partition
 key,
// and Info is additional data.
type Movie struct {
 Title string                `dynamodbav:"title"`
 Year  int                   `dynamodbav:"year"`
 Info  map[string]interface{} `dynamodbav:"info"`
}

// GetKey returns the composite primary key of the movie in a format that can be
// sent to DynamoDB.
func (movie Movie) GetKey() map[string]types.AttributeValue {
 title, err := attributevalue.Marshal(movie.Title)
 if err != nil {
  panic(err)
 }
 year, err := attributevalue.Marshal(movie.Year)
 if err != nil {
  panic(err)
 }
 return map[string]types.AttributeValue{"title": title, "year": year}
}

// String returns the title, year, rating, and plot of a movie, formatted for the
 example.
func (movie Movie) String() string {
 return fmt.Sprintf("%v\n\tReleased: %v\n\tRating: %v\n\tPlot: %v\n",
  movie.Title, movie.Year, movie.Info["rating"], movie.Info["plot"])
}
```

- Einzelheiten zur API finden Sie GetItemin der AWS SDK für Go API-Referenz.

Java

SDK für Java 2.x

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

Ruft mithilfe von ein Element aus einer Tabelle ab DynamoDbClient.

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.AttributeValue;
import software.amazon.awssdk.services.dynamodb.model.GetItemRequest;
import java.util.HashMap;
import java.util.Map;
import java.util.Set;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 *
 * To get an item from an Amazon DynamoDB table using the AWS SDK for Java V2,
 * its better practice to use the
 * Enhanced Client, see the EnhancedGetItem example.
 */
public class GetItem {
    public static void main(String[] args) {
        final String usage = """

                Usage:
                    <tableName> <key> <keyVal>

                Where:
```

```
                    tableName - The Amazon DynamoDB table from which an item is
retrieved (for example, Music3).\s
                    key - The key used in the Amazon DynamoDB table (for example,
Artist).\s
                    keyval - The key value that represents the item to get (for
example, Famous Band).
                """";

        if (args.length != 3) {
            System.out.println(usage);
            System.exit(1);
        }

        String tableName = args[0];
        String key = args[1];
        String keyVal = args[2];
        System.out.format("Retrieving item \"%s\" from \"%s\"\n", keyVal,
tableName);
        Region region = Region.US_EAST_1;
        DynamoDbClient ddb = DynamoDbClient.builder()
                .region(region)
                .build();

        getDynamoDBItem(ddb, tableName, key, keyVal);
        ddb.close();
    }

    public static void getDynamoDBItem(DynamoDbClient ddb, String tableName,
String key, String keyVal) {
        HashMap<String, AttributeValue> keyToGet = new HashMap<>();
        keyToGet.put(key, AttributeValue.builder()
                .s(keyVal)
                .build());

        GetItemRequest request = GetItemRequest.builder()
                .key(keyToGet)
                .tableName(tableName)
                .build();

        try {
            // If there is no matching item, GetItem does not return any data.
            Map<String, AttributeValue> returnedItem =
ddb.getItem(request).item();
            if (returnedItem.isEmpty())
```

```
                System.out.format("No item found with the key %s!\n", key);
            else {
                Set<String> keys = returnedItem.keySet();
                System.out.println("Amazon DynamoDB table attributes: \n");
                for (String key1 : keys) {
                    System.out.format("%s: %s\n", key1,
 returnedItem.get(key1).toString());
                }
            }

        } catch (DynamoDbException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
    }
}
```

- Einzelheiten zur API finden Sie GetItemunter AWS SDK for Java 2.x API-Referenz.

JavaScript

SDK für JavaScript (v3)

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

In diesem Beispiel wird der Dokument-Client verwendet, um die Arbeit mit Elementen in DynamoDB zu vereinfachen. Einzelheiten zur API finden Sie unter GetCommand.

```
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";
import { DynamoDBDocumentClient, GetCommand } from "@aws-sdk/lib-dynamodb";

const client = new DynamoDBClient({});
const docClient = DynamoDBDocumentClient.from(client);

export const main = async () => {
  const command = new GetCommand({
    TableName: "AngryAnimals",
```

```
    Key: {
      CommonName: "Shoebill",
    },
  });

  const response = await docClient.send(command);
  console.log(response);
  return response;
};
```

- Einzelheiten zur API finden Sie GetItemin der AWS SDK für JavaScript API-Referenz.

SDK für JavaScript (v2)

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

Rufen Sie ein Element aus einer Tabelle ab.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create the DynamoDB service object
var ddb = new AWS.DynamoDB({ apiVersion: "2012-08-10" });

var params = {
  TableName: "TABLE",
  Key: {
    KEY_NAME: { N: "001" },
  },
  ProjectionExpression: "ATTRIBUTE_NAME",
};

// Call DynamoDB to read the item from the table
ddb.getItem(params, function (err, data) {
  if (err) {
    console.log("Error", err);
```

```
  } else {
    console.log("Success", data.Item);
  }
});
```

Rufen Sie ein Element mithilfe des DynamoDB-Dokument-Clients aus einer Tabelle ab.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create DynamoDB document client
var docClient = new AWS.DynamoDB.DocumentClient({ apiVersion: "2012-08-10" });

var params = {
  TableName: "EPISODES_TABLE",
  Key: { KEY_NAME: VALUE },
};

docClient.get(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data.Item);
  }
});
```

- Weitere Informationen finden Sie im AWS SDK für JavaScript -Entwicklerhandbuch.
- Einzelheiten zur API finden Sie GetItemin der AWS SDK für JavaScript API-Referenz.

Kotlin

SDK für Kotlin

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
suspend fun getSpecificItem(
    tableNameVal: String,
    keyName: String,
    keyVal: String,
) {
    val keyToGet = mutableMapOf<String, AttributeValue>()
    keyToGet[keyName] = AttributeValue.S(keyVal)

    val request =
        GetItemRequest {
            key = keyToGet
            tableName = tableNameVal
        }

    DynamoDbClient.fromEnvironment { region = "us-east-1" }.use { ddb ->
        val returnedItem = ddb.getItem(request)
        val numbersMap = returnedItem.item
        numbersMap?.forEach { key1 ->
            println(key1.key)
            println(key1.value)
        }
    }
}
```

- Einzelheiten zur API finden Sie GetItemin der API-Referenz zum AWS SDK für Kotlin.

PHP

SDK für PHP

> **Note**
>
> Es gibt noch mehr dazu. GitHub Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
        $movie = $service->getItemByKey($tableName, $key);
        echo "\nThe movie {$movie['Item']['title']['S']} was released in
{$movie['Item']['year']['N']}.\n";
```

```
    public function getItemByKey(string $tableName, array $key)
    {
        return $this->dynamoDbClient->getItem([
            'Key' => $key['Item'],
            'TableName' => $tableName,
        ]);
    }
```

- Einzelheiten zur API finden Sie GetItemin der AWS SDK für PHP API-Referenz.

PowerShell

Tools für PowerShell V4

Beispiel 1: Gibt das DynamoDB-Element mit dem Partitionsschlüssel SongTitle und dem Sortierschlüssel Artist zurück.

```
$key = @{
  SongTitle = 'Somewhere Down The Road'
  Artist = 'No One You Know'
} | ConvertTo-DDBItem

Get-DDBItem -TableName 'Music' -Key $key | ConvertFrom-DDBItem
```

Ausgabe:

```
Name                           Value
----                           -----
Genre                          Country
SongTitle                      Somewhere Down The Road
Price                          1.94
Artist                         No One You Know
CriticRating                   9
AlbumTitle                     Somewhat Famous
```

- Einzelheiten zur API finden Sie unter GetItem AWS -Tools für PowerShellCmdlet-Referenz (V4).

Tools für V5 PowerShell

Beispiel 1: Gibt das DynamoDB-Element mit dem Partitionsschlüssel SongTitle und dem Sortierschlüssel Artist zurück.

```
$key = @{
  SongTitle = 'Somewhere Down The Road'
  Artist = 'No One You Know'
} | ConvertTo-DDBItem

Get-DDBItem -TableName 'Music' -Key $key | ConvertFrom-DDBItem
```

Ausgabe:

```
Name                           Value
----                           -----
Genre                          Country
SongTitle                      Somewhere Down The Road
Price                          1.94
Artist                         No One You Know
CriticRating                   9
AlbumTitle                     Somewhat Famous
```

- Einzelheiten zur API finden Sie unter GetItem AWS -Tools für PowerShellCmdlet-Referenz (V5).

Python

SDK für Python (Boto3)

> **Note**
>
> Es gibt noch mehr dazu. GitHub Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
class Movies:
    """Encapsulates an Amazon DynamoDB table of movie data.

    Example data structure for a movie record in this table:
        {
```

```
            "year": 1999,
            "title": "For Love of the Game",
            "info": {
                "directors": ["Sam Raimi"],
                "release_date": "1999-09-15T00:00:00Z",
                "rating": 6.3,
                "plot": "A washed up pitcher flashes through his career.",
                "rank": 4987,
                "running_time_secs": 8220,
                "actors": [
                    "Kevin Costner",
                    "Kelly Preston",
                    "John C. Reilly"
                ]
            }
        }
    """

    def __init__(self, dyn_resource):
        """
        :param dyn_resource: A Boto3 DynamoDB resource.
        """
        self.dyn_resource = dyn_resource
        # The table variable is set during the scenario in the call to
        # 'exists' if the table exists. Otherwise, it is set by 'create_table'.
        self.table = None


    def get_movie(self, title, year):
        """
        Gets movie data from the table for a specific movie.

        :param title: The title of the movie.
        :param year: The release year of the movie.
        :return: The data about the requested movie.
        """
        try:
            response = self.table.get_item(Key={"year": year, "title": title})
        except ClientError as err:
            logger.error(
                "Couldn't get movie %s from table %s. Here's why: %s: %s",
                title,
                self.table.name,
                err.response["Error"]["Code"],
```

```
                err.response["Error"]["Message"],
            )
            raise
        else:
            return response["Item"]
```

- Einzelheiten zur API finden Sie GetItemin AWS SDK for Python (Boto3) API Reference.

Ruby

SDK für Ruby

> **Note**
>
> Es gibt noch mehr dazu. GitHub Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
class DynamoDBBasics
  attr_reader :dynamo_resource, :table

  def initialize(table_name)
    client = Aws::DynamoDB::Client.new(region: 'us-east-1')
    @dynamo_resource = Aws::DynamoDB::Resource.new(client: client)
    @table = @dynamo_resource.table(table_name)
  end

  # Gets movie data from the table for a specific movie.
  #
  # @param title [String] The title of the movie.
  # @param year [Integer] The release year of the movie.
  # @return [Hash] The data about the requested movie.
  def get_item(title, year)
    @table.get_item(key: { 'year' => year, 'title' => title })
  rescue Aws::DynamoDB::Errors::ServiceError => e
    puts("Couldn't get movie #{title} (#{year}) from table #{@table.name}:\n")
    puts("\t#{e.code}: #{e.message}")
    raise
  end
```

- Einzelheiten zur API finden Sie GetItemin der AWS SDK für Ruby API-Referenz.

SAP ABAP

SDK für SAP ABAP

> **Note**
>
> Es gibt noch mehr dazu. GitHub Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
TRY.
    oo_item = lo_dyn->getitem(
      iv_tablename               = iv_table_name
      it_key                     = it_key ).
    DATA(lt_attr) = oo_item->get_item( ).
    DATA(lo_title) = lt_attr[ key = 'title' ]-value.
    DATA(lo_year) = lt_attr[ key = 'year' ]-value.
    DATA(lo_rating) = lt_attr[ key = 'rating' ]-value.
    MESSAGE 'Movie name is: ' && lo_title->get_s( )
      && 'Movie year is: ' && lo_year->get_n( )
      && 'Moving rating is: ' && lo_rating->get_n( ) TYPE 'I'.
  CATCH /aws1/cx_dynresourcenotfoundex.
    MESSAGE 'The table or index does not exist' TYPE 'E'.
ENDTRY.
```

- Einzelheiten zur API finden Sie GetItemin der API-Referenz zum AWS SDK für SAP ABAP.

Swift

SDK für Swift

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
import AWSDynamoDB


    /// Return a `Movie` record describing the specified movie from the Amazon
    /// DynamoDB table.
    ///
    /// - Parameters:
    ///   - title: The movie's title (`String`).
    ///   - year: The movie's release year (`Int`).
    ///
    /// - Throws: `MoviesError.ItemNotFound` if the movie isn't in the table.
    ///
    /// - Returns: A `Movie` record with the movie's details.
    func get(title: String, year: Int) async throws -> Movie {
        do {
            guard let client = self.ddbClient else {
                throw MoviesError.UninitializedClient
            }

            let input = GetItemInput(
                key: [
                    "year": .n(String(year)),
                    "title": .s(title)
                ],
                tableName: self.tableName
            )
            let output = try await client.getItem(input: input)
            guard let item = output.item else {
                throw MoviesError.ItemNotFound
            }

            let movie = try Movie(withItem: item)
```

```
            return movie
        } catch {
            print("ERROR: get:", dump(error))
            throw error
        }
    }
```

- Einzelheiten zur API finden Sie GetItemin der API-Referenz zum AWS SDK für Swift.

Eine vollständige Liste der AWS SDK-Entwicklerhandbücher und Codebeispiele finden Sie unterDynamoDB mit einem SDK verwenden AWS. Dieses Thema enthält auch Informationen zu den ersten Schritten und Details zu früheren SDK-Versionen.

Verwendung **`ListTables`** mit einem AWS SDK oder CLI

Die folgenden Code-Beispiele zeigen, wie `ListTables` verwendet wird.

Beispiele für Aktionen sind Codeauszüge aus größeren Programmen und müssen im Kontext ausgeführt werden. Im folgenden Codebeispiel können Sie diese Aktion im Kontext sehen:

- Einrichten der attributbasierten Zugriffskontrolle

.NET

SDK für .NET

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
    private static async Task ListMyTables()
    {
        Console.WriteLine("\n*** Listing tables ***");

        string lastTableNameEvaluated = null;
        do
```

```
        {
            var response = await Client.ListTablesAsync(new ListTablesRequest
            {
                Limit = 2,
                ExclusiveStartTableName = lastTableNameEvaluated
            });

            foreach (var name in response.TableNames)
            {
                Console.WriteLine(name);
            }

            lastTableNameEvaluated = response.LastEvaluatedTableName;
        } while (lastTableNameEvaluated != null);
    }
```

- Einzelheiten zur API finden Sie ListTablesin der AWS SDK für .NET API-Referenz.

Bash

AWS CLI mit Bash-Skript

> **Note**
>
> Es gibt noch mehr dazu. GitHub Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
###############################################################################
# function dynamodb_list_tables
#
# This function lists all the tables in a DynamoDB.
#
# Returns:
#       0 - If successful.
#       1 - If it fails.
###############################################################################
function dynamodb_list_tables() {
  response=$(aws dynamodb list-tables \
    --output text \
    --query "TableNames")
```

```
  local error_code=${?}

  if [[ $error_code -ne 0 ]]; then
    aws_cli_error_log $error_code
    errecho "ERROR: AWS reports batch-write-item operation failed.$response"
    return 1
  fi

  echo "$response" | tr -s "[:space:]" "\n"

  return 0
}
```

Die in diesem Beispiel verwendeten Hilfsfunktionen.

```
########################################################################################
# function errecho
#
# This function outputs everything sent to it to STDERR (standard error output).
########################################################################################
function errecho() {
  printf "%s\n" "$*" 1>&2
}

########################################################################################
# function aws_cli_error_log()
#
# This function is used to log the error messages from the AWS CLI.
#
# See https://docs.aws.amazon.com/cli/latest/topic/return-codes.html#cli-aws-
help-return-codes.
#
# The function expects the following argument:
#         $1 - The error code returned by the AWS CLI.
#
#  Returns:
#          0: - Success.
#
########################################################################################
function aws_cli_error_log() {
  local err_code=$1
```

```
  errecho "Error code : $err_code"
  if [ "$err_code" == 1 ]; then
    errecho "  One or more S3 transfers failed."
  elif [ "$err_code" == 2 ]; then
    errecho "  Command line failed to parse."
  elif [ "$err_code" == 130 ]; then
    errecho "  Process received SIGINT."
  elif [ "$err_code" == 252 ]; then
    errecho "  Command syntax invalid."
  elif [ "$err_code" == 253 ]; then
    errecho "  The system environment or configuration was invalid."
  elif [ "$err_code" == 254 ]; then
    errecho "  The service returned an error."
  elif [ "$err_code" == 255 ]; then
    errecho "  255 is a catch-all error."
  fi

  return 0
}
```

- Einzelheiten zur API finden Sie ListTablesin der AWS CLI Befehlsreferenz.

C++

SDK für C++

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
//! List the Amazon DynamoDB tables for the current AWS account.
/*!
  \sa listTables()
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
 */

bool AwsDoc::DynamoDB::listTables(
        const Aws::Client::ClientConfiguration &clientConfiguration) {
```

```
    Aws::DynamoDB::DynamoDBClient dynamoClient(clientConfiguration);

    Aws::DynamoDB::Model::ListTablesRequest listTablesRequest;
    listTablesRequest.SetLimit(50);
    do {
        const Aws::DynamoDB::Model::ListTablesOutcome &outcome =
 dynamoClient.ListTables(
                listTablesRequest);
        if (!outcome.IsSuccess()) {
            std::cout << "Error: " << outcome.GetError().GetMessage() <<
 std::endl;
            return false;
        }

        for (const auto &tableName: outcome.GetResult().GetTableNames())
            std::cout << tableName << std::endl;
        listTablesRequest.SetExclusiveStartTableName(
                outcome.GetResult().GetLastEvaluatedTableName());

    } while (!listTablesRequest.GetExclusiveStartTableName().empty());

    return true;
}
```

- Einzelheiten zur API finden Sie ListTablesin der AWS SDK für C++ API-Referenz.

CLI

AWS CLI

Beispiel 1: So listen Sie Tabellen auf

Das folgende `list-tables` Beispiel listet alle Tabellen auf, die dem AWS Girokonto und der Region zugeordnet sind.

```
aws dynamodb list-tables
```

Ausgabe:

```
{
    "TableNames": [
```

```
        "Forum",
        "ProductCatalog",
        "Reply",
        "Thread"
    ]
}
```

Weitere Informationen finden Sie unter Auflisten von Tabellennamen im Entwicklerhandbuch zu Amazon DynamoDB.

Beispiel 2: So begrenzen Sie die Seitengröße

Das folgende Beispiel gibt eine Liste aller vorhandenen Tabellen zurück, ruft jedoch bei jedem Aufruf nur ein Element ab und führt gegebenenfalls mehrere Aufrufe durch, um die gesamte Liste abzurufen. Die Begrenzung der Seitengröße ist nützlich, wenn Listenbefehle für eine große Anzahl von Ressourcen ausgeführt werden, was zu einem Timeout-Fehler führen kann, wenn die Standardseitengröße von 1 000 verwendet wird.

```
aws dynamodb list-tables \
    --page-size 1
```

Ausgabe:

```
{
    "TableNames": [
        "Forum",
        "ProductCatalog",
        "Reply",
        "Thread"
    ]
}
```

Weitere Informationen finden Sie unter Auflisten von Tabellennamen im Entwicklerhandbuch zu Amazon DynamoDB.

Beispiel 3: So begrenzen Sie die Anzahl der zurückgegebenen Elemente

Im folgenden Beispiel wird die Anzahl der zurückgegebenen Elemente auf 2 begrenzt. Die Antwort enthält einen `NextToken`-Wert, mit dem die nächste Ergebnisseite abgerufen werden kann.

```
aws dynamodb list-tables \
    --max-items 2
```

Ausgabe:

```
{
    "TableNames": [
        "Forum",
        "ProductCatalog"
    ],
    "NextToken":
 "abCDeFGhiJKlmnOPqrSTuvwxYZ1aBCdEFghijK7LM51nOpqRSTuv3WxY3ZabC5dEFGhI2Jk3LmnoPQ6RST9"
}
```

Weitere Informationen finden Sie unter Auflisten von Tabellennamen im Entwicklerhandbuch zu Amazon DynamoDB.

Beispiel 4: So rufen Sie die nächste Ergebnisseite ab

Der folgende Befehl verwendet den `NextToken`-Wert eines vorherigen Aufrufs des `list-tables`-Befehls, um eine weitere Ergebnisseite abzurufen. Da die Antwort in diesem Fall keinen `NextToken`-Wert enthält, wissen wir, dass wir das Ende der Ergebnisse erreicht haben.

```
aws dynamodb list-tables \
    --starting-
token abCDeFGhiJKlmnOPqrSTuvwxYZ1aBCdEFghijK7LM51nOpqRSTuv3WxY3ZabC5dEFGhI2Jk3LmnoPQ6RST9
```

Ausgabe:

```
{
    "TableNames": [
        "Reply",
        "Thread"
    ]
}
```

Weitere Informationen finden Sie unter Auflisten von Tabellennamen im Entwicklerhandbuch zu Amazon DynamoDB.

- Einzelheiten zur API finden Sie ListTablesunter AWS CLI Befehlsreferenz.

Go

SDK für Go V2

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
import (
 "context"
 "errors"
 "log"
 "time"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/attributevalue"
 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/expression"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb/types"
)

// TableBasics encapsulates the Amazon DynamoDB service actions used in the
 examples.
// It contains a DynamoDB service client that is used to act on the specified
 table.
type TableBasics struct {
 DynamoDbClient *dynamodb.Client
 TableName      string
}



// ListTables lists the DynamoDB table names for the current account.
func (basics TableBasics) ListTables(ctx context.Context) ([]string, error) {
 var tableNames []string
 var output *dynamodb.ListTablesOutput
 var err error
```

```
 tablePaginator := dynamodb.NewListTablesPaginator(basics.DynamoDbClient,
 &dynamodb.ListTablesInput{})
 for tablePaginator.HasMorePages() {
  output, err = tablePaginator.NextPage(ctx)
  if err != nil {
   log.Printf("Couldn't list tables. Here's why: %v\n", err)
   break
  } else {
   tableNames = append(tableNames, output.TableNames...)
  }
 }
 return tableNames, err
}
```

- Einzelheiten zur API finden Sie ListTablesin der AWS SDK für Go API-Referenz.

Java

SDK für Java 2.x

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.model.ListTablesRequest;
import software.amazon.awssdk.services.dynamodb.model.ListTablesResponse;
import java.util.List;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
```

```
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 */
public class ListTables {
    public static void main(String[] args) {
        System.out.println("Listing your Amazon DynamoDB tables:\n");
        Region region = Region.US_EAST_1;
        DynamoDbClient ddb = DynamoDbClient.builder()
                .region(region)
                .build();
        listAllTables(ddb);
        ddb.close();
    }

    public static void listAllTables(DynamoDbClient ddb) {
        boolean moreTables = true;
        String lastName = null;

        while (moreTables) {
            try {
                ListTablesResponse response = null;
                if (lastName == null) {
                    ListTablesRequest request =
 ListTablesRequest.builder().build();
                    response = ddb.listTables(request);
                } else {
                    ListTablesRequest request = ListTablesRequest.builder()
                            .exclusiveStartTableName(lastName).build();
                    response = ddb.listTables(request);
                }

                List<String> tableNames = response.tableNames();
                if (tableNames.size() > 0) {
                    for (String curName : tableNames) {
                        System.out.format("* %s\n", curName);
                    }
                } else {
                    System.out.println("No tables found!");
                    System.exit(0);
                }

                lastName = response.lastEvaluatedTableName();
                if (lastName == null) {
                    moreTables = false;
```

```
                }

            } catch (DynamoDbException e) {
                System.err.println(e.getMessage());
                System.exit(1);
            }
        }
        System.out.println("\nDone!");
    }
}
```

- Einzelheiten zur API finden Sie ListTablesin der AWS SDK for Java 2.x API-Referenz.

JavaScript

SDK für JavaScript (v3)

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
import { ListTablesCommand, DynamoDBClient } from "@aws-sdk/client-dynamodb";

const client = new DynamoDBClient({});

export const main = async () => {
  const command = new ListTablesCommand({});

  const response = await client.send(command);
  console.log(response);
  return response;
};
```

- Weitere Informationen finden Sie im AWS SDK für JavaScript -Entwicklerhandbuch.
- Einzelheiten zur API finden Sie ListTablesin der AWS SDK für JavaScript API-Referenz.

SDK für JavaScript (v2)

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create the DynamoDB service object
var ddb = new AWS.DynamoDB({ apiVersion: "2012-08-10" });

// Call DynamoDB to retrieve the list of tables
ddb.listTables({ Limit: 10 }, function (err, data) {
  if (err) {
    console.log("Error", err.code);
  } else {
    console.log("Table names are ", data.TableNames);
  }
});
```

- Weitere Informationen finden Sie im AWS SDK für JavaScript -Entwicklerhandbuch.
- Einzelheiten zur API finden Sie ListTablesin der AWS SDK für JavaScript API-Referenz.

Kotlin

SDK für Kotlin

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
suspend fun listAllTables() {
```

```
    DynamoDbClient.fromEnvironment { region = "us-east-1" }.use { ddb ->
        val response = ddb.listTables(ListTablesRequest {})
        response.tableNames?.forEach { tableName ->
            println("Table name is $tableName")
        }
    }
}
```

- Einzelheiten zur API finden Sie ListTablesin der API-Referenz zum AWS SDK für Kotlin.

PHP

SDK für PHP

> **Note**
>
> Es gibt noch mehr dazu. GitHub Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
    public function listTables($exclusiveStartTableName = "", $limit = 100)
    {
        $this->dynamoDbClient->listTables([
            'ExclusiveStartTableName' => $exclusiveStartTableName,
            'Limit' => $limit,
        ]);
    }
```

- Einzelheiten zur API finden Sie ListTablesin der AWS SDK für PHP API-Referenz.

PowerShell

Tools für PowerShell V4

Beispiel 1: Gibt Details aller Tabellen zurück und iteriert automatisch, bis der Service anzeigt, dass keine weiteren Tabellen vorhanden sind.

```
Get-DDBTableList
```

- Einzelheiten zur API finden Sie unter ListTables AWS -Tools für PowerShellCmdlet-Referenz (V4).

Tools für V5 PowerShell

Beispiel 1: Gibt Details aller Tabellen zurück und iteriert automatisch, bis der Service anzeigt, dass keine weiteren Tabellen vorhanden sind.

```
Get-DDBTableList
```

- Einzelheiten zur API finden Sie unter ListTables AWS -Tools für PowerShellCmdlet-Referenz (V5).

Python

SDK für Python (Boto3)

> **Note**
>
> Es gibt noch mehr dazu. GitHub Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
class Movies:
    """Encapsulates an Amazon DynamoDB table of movie data.

    Example data structure for a movie record in this table:
        {
            "year": 1999,
            "title": "For Love of the Game",
            "info": {
                "directors": ["Sam Raimi"],
                "release_date": "1999-09-15T00:00:00Z",
                "rating": 6.3,
                "plot": "A washed up pitcher flashes through his career.",
                "rank": 4987,
                "running_time_secs": 8220,
                "actors": [
                    "Kevin Costner",
                    "Kelly Preston",
                    "John C. Reilly"
```

```
                ]
            }
        }
    """

    def __init__(self, dyn_resource):
        """
        :param dyn_resource: A Boto3 DynamoDB resource.
        """
        self.dyn_resource = dyn_resource
        # The table variable is set during the scenario in the call to
        # 'exists' if the table exists. Otherwise, it is set by 'create_table'.
        self.table = None


    def list_tables(self):
        """
        Lists the Amazon DynamoDB tables for the current account.

        :return: The list of tables.
        """
        try:
            tables = []
            for table in self.dyn_resource.tables.all():
                print(table.name)
                tables.append(table)
        except ClientError as err:
            logger.error(
                "Couldn't list tables. Here's why: %s: %s",
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return tables
```

- Einzelheiten zur API finden Sie ListTablesin AWS SDK for Python (Boto3) API Reference.

Ruby

SDK für Ruby

> **Note**
>
> Es gibt noch mehr dazu. GitHub Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

Finden Sie heraus, ob eine Tabelle vorhanden ist.

```
# Encapsulates an Amazon DynamoDB table of movie data.
class Scaffold
  attr_reader :dynamo_resource, :table_name, :table

  def initialize(table_name)
    client = Aws::DynamoDB::Client.new(region: 'us-east-1')
    @dynamo_resource = Aws::DynamoDB::Resource.new(client: client)
    @table_name = table_name
    @table = nil
    @logger = Logger.new($stdout)
    @logger.level = Logger::DEBUG
  end

  # Determines whether a table exists. As a side effect, stores the table in
  # a member variable.
  #
  # @param table_name [String] The name of the table to check.
  # @return [Boolean] True when the table exists; otherwise, False.
  def exists?(table_name)
    @dynamo_resource.client.describe_table(table_name: table_name)
    @logger.debug("Table #{table_name} exists")
  rescue Aws::DynamoDB::Errors::ResourceNotFoundException
    @logger.debug("Table #{table_name} doesn't exist")
    false
  rescue Aws::DynamoDB::Errors::ServiceError => e
    puts("Couldn't check for existence of #{table_name}:\n")
    puts("\t#{e.code}: #{e.message}")
    raise
  end
```

- Einzelheiten zur API finden Sie ListTablesin der AWS SDK für Ruby API-Referenz.

Rust

SDK für Rust

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
pub async fn list_tables(client: &Client) -> Result<Vec<String>, Error> {
    let paginator = client.list_tables().into_paginator().items().send();
    let table_names = paginator.collect::<Result<Vec<_>, _>>().await?;

    println!("Tables:");

    for name in &table_names {
        println!("  {}", name);
    }

    println!("Found {} tables", table_names.len());
    Ok(table_names)
}
```

Finden Sie heraus, ob eine Tabelle vorhanden ist.

```
pub async fn table_exists(client: &Client, table: &str) -> Result<bool, Error> {
    debug!("Checking for table: {table}");
    let table_list = client.list_tables().send().await;

    match table_list {
        Ok(list) => Ok(list.table_names().contains(&table.into())),
        Err(e) => Err(e.into()),
    }
}
```

- Einzelheiten zur API finden Sie ListTablesin der API-Referenz zum AWS SDK für Rust.

SAP ABAP

SDK für SAP ABAP

> **Note**
>
> Es gibt noch mehr dazu. GitHub Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
    TRY.
        oo_result = lo_dyn->listtables( ).
        " You can loop over the oo_result to get table properties like this.
        LOOP AT oo_result->get_tablenames( ) INTO DATA(lo_table_name).
          DATA(lv_tablename) = lo_table_name->get_value( ).
        ENDLOOP.
        DATA(lv_tablecount) = lines( oo_result->get_tablenames( ) ).
        MESSAGE 'Found ' && lv_tablecount && ' tables' TYPE 'I'.
      CATCH /aws1/cx_rt_service_generic INTO DATA(lo_exception).
        DATA(lv_error) = |"{ lo_exception->av_err_code }" - { lo_exception-
>av_err_msg }|.
        MESSAGE lv_error TYPE 'E'.
    ENDTRY.
```

- Einzelheiten zur API finden Sie ListTablesin der API-Referenz zum AWS SDK für SAP ABAP.

Swift

SDK für Swift

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
import AWSDynamoDB
```

```
/// Get a list of the DynamoDB tables available in the specified Region.
///
/// - Returns: An array of strings listing all of the tables available
///   in the Region specified when the session was created.
public func getTableList() async throws -> [String] {
    let input = ListTablesInput(
    )
    return try await session.listTables(input: input)
}
```

- Einzelheiten zur API finden Sie ListTablesin der API-Referenz zum AWS SDK für Swift.

Eine vollständige Liste der AWS SDK-Entwicklerhandbücher und Codebeispiele finden Sie unterDynamoDB mit einem SDK verwenden AWS. Dieses Thema enthält auch Informationen zu den ersten Schritten und Details zu früheren SDK-Versionen.

Verwendung **`PutItem`** mit einem AWS SDK oder CLI

Die folgenden Code-Beispiele zeigen, wie `PutItem` verwendet wird.

Aktionsbeispiele sind Codeauszüge aus größeren Programmen und müssen im Kontext ausgeführt werden. Sie können diese Aktion in den folgenden Codebeispielen im Kontext sehen:

- Kennenlernen der Grundlagen
- Beschleunigen von Lesevorgängen mit DAX
- Erweiterte Szenarien für globale sekundäre Indizes
- Erstellen eines Elements mit einer TTL
- Erstellen und Verwalten globaler Tabellen zur Demonstration von MREC
- Erstellen und Verwalten von globalen MRSC-Tabellen
- Verwenden von bedingten Operationen
- Arbeiten mit globalen Tabellen und letztendlicher multiregionaler Konsistenz (Multi-Region Eventual Consistency, MREC)
- Arbeite mit Streams und Time-to-Live

.NET

SDK für .NET (v4)

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
    /// <summary>
    /// Adds a new item to the table.
    /// </summary>
    /// <param name="newMovie">A Movie object containing informtation for
    /// the movie to add to the table.</param>
    /// <param name="tableName">The name of the table where the item will be
 added.</param>
    /// <returns>A Boolean value that indicates the results of adding the item.</
returns>
    public async Task<bool> PutItemAsync(Movie newMovie, string tableName)
    {
        try
        {
            var item = new Dictionary<string, AttributeValue>
            {
                ["title"] = new AttributeValue { S = newMovie.Title },
                ["year"] = new AttributeValue { N = newMovie.Year.ToString() },
            };

            var request = new PutItemRequest
            {
                TableName = tableName,
                Item = item,
            };

            await _amazonDynamoDB.PutItemAsync(request);
            return true;
        }
        catch (ResourceNotFoundException ex)
        {
            Console.WriteLine($"Table {tableName} was not found. {ex.Message}");
```

```
            return false;
        }
        catch (AmazonDynamoDBException ex)
        {
            Console.WriteLine($"An Amazon DynamoDB error occurred while putting
item. {ex.Message}");
            throw;
        }
        catch (Exception ex)
        {
            Console.WriteLine($"An error occurred while putting item.
{ex.Message}");
            throw;
        }
    }
```

- Einzelheiten zur API finden Sie PutItemin der AWS SDK für .NET API-Referenz.

Bash

AWS CLI mit Bash-Skript

> **Note**
>
> Es gibt noch mehr dazu. GitHub Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
###############################################################################
# function dynamodb_put_item
#
# This function puts an item into a DynamoDB table.
#
# Parameters:
#       -n table_name  -- The name of the table.
#       -i item  -- Path to json file containing the item values.
#
#  Returns:
#       0 - If successful.
#       1 - If it fails.
```

```
###############################################################################
function dynamodb_put_item() {
  local table_name item response
  local option OPTARG # Required to use getopts command in a function.

  ######################################
  # Function usage explanation
  ######################################
  function usage() {
    echo "function dynamodb_put_item"
    echo "Put an item into a DynamoDB table."
    echo " -n table_name  -- The name of the table."
    echo " -i item  -- Path to json file containing the item values."
    echo ""
  }

  while getopts "n:i:h" option; do
    case "${option}" in
      n) table_name="${OPTARG}" ;;
      i) item="${OPTARG}" ;;
      h)
        usage
        return 0
        ;;
      \?)
        echo "Invalid parameter"
        usage
        return 1
        ;;
    esac
  done
  export OPTIND=1

  if [[ -z "$table_name" ]]; then
    errecho "ERROR: You must provide a table name with the -n parameter."
    usage
    return 1
  fi

  if [[ -z "$item" ]]; then
    errecho "ERROR: You must provide an item with the -i parameter."
    usage
    return 1
  fi
```

```
  iecho "Parameters:\n"
  iecho "    table_name:   $table_name"
  iecho "    item:   $item"
  iecho ""
  iecho ""

  response=$(aws dynamodb put-item \
    --table-name "$table_name" \
    --item file://"$item")

  local error_code=${?}

  if [[ $error_code -ne 0 ]]; then
    aws_cli_error_log $error_code
    errecho "ERROR: AWS reports put-item operation failed.$response"
    return 1
  fi

  return 0

}
```

Die in diesem Beispiel verwendeten Hilfsfunktionen.

```
###############################################################################
# function iecho
#
# This function enables the script to display the specified text only if
# the global variable $VERBOSE is set to true.
###############################################################################
function iecho() {
  if [[ $VERBOSE == true ]]; then
    echo "$@"
  fi
}

###############################################################################
# function errecho
#
# This function outputs everything sent to it to STDERR (standard error output).
###############################################################################
```

```
function errecho() {
  printf "%s\n" "$*" 1>&2
}

###############################################################################
# function aws_cli_error_log()
#
# This function is used to log the error messages from the AWS CLI.
#
# See https://docs.aws.amazon.com/cli/latest/topic/return-codes.html#cli-aws-
help-return-codes.
#
# The function expects the following argument:
#         $1 - The error code returned by the AWS CLI.
#
#  Returns:
#          0: - Success.
#
###############################################################################
function aws_cli_error_log() {
  local err_code=$1
  errecho "Error code : $err_code"
  if [ "$err_code" == 1 ]; then
    errecho "  One or more S3 transfers failed."
  elif [ "$err_code" == 2 ]; then
    errecho "  Command line failed to parse."
  elif [ "$err_code" == 130 ]; then
    errecho "  Process received SIGINT."
  elif [ "$err_code" == 252 ]; then
    errecho "  Command syntax invalid."
  elif [ "$err_code" == 253 ]; then
    errecho "  The system environment or configuration was invalid."
  elif [ "$err_code" == 254 ]; then
    errecho "  The service returned an error."
  elif [ "$err_code" == 255 ]; then
    errecho "  255 is a catch-all error."
  fi

  return 0
}
```

- Einzelheiten zur API finden Sie PutItemin der AWS CLI Befehlsreferenz.

C++

SDK für C++

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
//! Put an item in an Amazon DynamoDB table.
/*!
  \sa putItem()
  \param tableName: The table name.
  \param artistKey: The artist key. This is the partition key for the table.
  \param artistValue: The artist value.
  \param albumTitleKey: The album title key.
  \param albumTitleValue: The album title value.
  \param awardsKey: The awards key.
  \param awardsValue: The awards value.
  \param songTitleKey: The song title key.
  \param songTitleValue: The song title value.
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
 */
bool AwsDoc::DynamoDB::putItem(const Aws::String &tableName,
                               const Aws::String &artistKey,
                               const Aws::String &artistValue,
                               const Aws::String &albumTitleKey,
                               const Aws::String &albumTitleValue,
                               const Aws::String &awardsKey,
                               const Aws::String &awardsValue,
                               const Aws::String &songTitleKey,
                               const Aws::String &songTitleValue,
                               const Aws::Client::ClientConfiguration
 &clientConfiguration) {
    Aws::DynamoDB::DynamoDBClient dynamoClient(clientConfiguration);

    Aws::DynamoDB::Model::PutItemRequest putItemRequest;
    putItemRequest.SetTableName(tableName);
```

```
    putItemRequest.AddItem(artistKey,
 Aws::DynamoDB::Model::AttributeValue().SetS(
            artistValue)); // This is the hash key.
    putItemRequest.AddItem(albumTitleKey,
 Aws::DynamoDB::Model::AttributeValue().SetS(
            albumTitleValue));
    putItemRequest.AddItem(awardsKey,

 Aws::DynamoDB::Model::AttributeValue().SetS(awardsValue));
    putItemRequest.AddItem(songTitleKey,

 Aws::DynamoDB::Model::AttributeValue().SetS(songTitleValue));

    const Aws::DynamoDB::Model::PutItemOutcome outcome = dynamoClient.PutItem(
            putItemRequest);
    if (outcome.IsSuccess()) {
        std::cout << "Successfully added Item!" << std::endl;
    }
    else {
        std::cerr << outcome.GetError().GetMessage() << std::endl;
        return false;
    }

    return waitTableActive(tableName, dynamoClient);
}
```

Code, der darauf wartet, dass die Tabelle aktiv wird.

```
//! Query a newly created DynamoDB table until it is active.
/*!
  \sa waitTableActive()
  \param waitTableActive: The DynamoDB table's name.
  \param dynamoClient: A DynamoDB client.
  \return bool: Function succeeded.
*/
bool AwsDoc::DynamoDB::waitTableActive(const Aws::String &tableName,
                                       const Aws::DynamoDB::DynamoDBClient
 &dynamoClient) {

    // Repeatedly call DescribeTable until table is ACTIVE.
    const int MAX_QUERIES = 20;
```

```
    Aws::DynamoDB::Model::DescribeTableRequest request;
    request.SetTableName(tableName);

    int count = 0;
    while (count < MAX_QUERIES) {
        const Aws::DynamoDB::Model::DescribeTableOutcome &result =
 dynamoClient.DescribeTable(
                request);
        if (result.IsSuccess()) {
            Aws::DynamoDB::Model::TableStatus status =
 result.GetResult().GetTable().GetTableStatus();

            if (Aws::DynamoDB::Model::TableStatus::ACTIVE != status) {
                std::this_thread::sleep_for(std::chrono::seconds(1));
            }
            else {
                return true;
            }
        }
        else {
            std::cerr << "Error DynamoDB::waitTableActive "
                      << result.GetError().GetMessage() << std::endl;
            return false;
        }
        count++;
    }
    return false;
}
```

- Einzelheiten zur API finden Sie PutItemin der AWS SDK für C++ API-Referenz.

CLI

AWS CLI

Beispiel 1: So fügen Sie ein Element zu einer Tabelle hinzu

Das folgende `put-item` Beispiel fügt der MusicCollectionTabelle ein neues Element hinzu.

```
aws dynamodb put-item \
    --table-name MusicCollection \
    --item file://item.json \
```

```
    --return-consumed-capacity TOTAL \
    --return-item-collection-metrics SIZE
```

Inhalt von `item.json`:

```
{
    "Artist": {"S": "No One You Know"},
    "SongTitle": {"S": "Call Me Today"},
    "AlbumTitle": {"S": "Greatest Hits"}
}
```

Ausgabe:

```
{
    "ConsumedCapacity": {
        "TableName": "MusicCollection",
        "CapacityUnits": 1.0
    },
    "ItemCollectionMetrics": {
        "ItemCollectionKey": {
            "Artist": {
                "S": "No One You Know"
            }
        },
        "SizeEstimateRangeGB": [
            0.0,
            1.0
        ]
    }
}
```

Weitere Informationen finden Sie unter Schreiben eines Elements im Entwicklerhandbuch zu Amazon DynamoDB.

Beispiel 2: So überschreiben Sie ein Element in einer Tabelle bedingungsabhängig

Im folgenden Beispiel für `put-item` wird ein vorhandenes Element in der `MusicCollection`-Tabelle nur dann überschrieben, wenn dieses vorhandene Element ein `AlbumTitle`-Attribut mit dem Wert `Greatest Hits` hat. Der Befehl gibt den vorherigen Wert des Elements zurück.

```
aws dynamodb put-item \
```

```
    --table-name MusicCollection \
    --item file://item.json \
    --condition-expression "#A = :A" \
    --expression-attribute-names file://names.json \
    --expression-attribute-values file://values.json \
    --return-values ALL_OLD
```

Inhalt von `item.json`:

```
{
    "Artist": {"S": "No One You Know"},
    "SongTitle": {"S": "Call Me Today"},
    "AlbumTitle": {"S": "Somewhat Famous"}
}
```

Inhalt von `names.json`:

```
{
    "#A": "AlbumTitle"
}
```

Inhalt von `values.json`:

```
{
    ":A": {"S": "Greatest Hits"}
}
```

Ausgabe:

```
{
    "Attributes": {
        "AlbumTitle": {
            "S": "Greatest Hits"
        },
        "Artist": {
            "S": "No One You Know"
        },
        "SongTitle": {
            "S": "Call Me Today"
        }
    }
```

```
}
```

Wenn der Schlüssel bereits existiert, sollten Sie die folgende Ausgabe sehen:

```
A client error (ConditionalCheckFailedException) occurred when calling the
 PutItem operation: The conditional request failed.
```

Weitere Informationen finden Sie unter Schreiben eines Elements im Entwicklerhandbuch zu Amazon DynamoDB.

- Einzelheiten zur API finden Sie PutItemunter AWS CLI Befehlsreferenz.

Go

SDK für Go V2

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
import (
 "context"
 "errors"
 "log"
 "time"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/attributevalue"
 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/expression"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb/types"
)

// TableBasics encapsulates the Amazon DynamoDB service actions used in the
 examples.
// It contains a DynamoDB service client that is used to act on the specified
 table.
type TableBasics struct {
```

```
 DynamoDbClient *dynamodb.Client
 TableName      string
}



// AddMovie adds a movie the DynamoDB table.
func (basics TableBasics) AddMovie(ctx context.Context, movie Movie) error {
 item, err := attributevalue.MarshalMap(movie)
 if err != nil {
  panic(err)
 }
 _, err = basics.DynamoDbClient.PutItem(ctx, &dynamodb.PutItemInput{
  TableName: aws.String(basics.TableName), Item: item,
 })
 if err != nil {
  log.Printf("Couldn't add item to table. Here's why: %v\n", err)
 }
 return err
}
```

Definieren Sie eine Movie-Struktur, die in diesem Beispiel verwendet wird.

```
import (
 "archive/zip"
 "bytes"
 "encoding/json"
 "fmt"
 "io"
 "log"
 "net/http"

 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/attributevalue"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb/types"
)

// Movie encapsulates data about a movie. Title and Year are the composite
 primary key
// of the movie in Amazon DynamoDB. Title is the sort key, Year is the partition
 key,
```

```
// and Info is additional data.
type Movie struct {
 Title string                `dynamodbav:"title"`
 Year  int                   `dynamodbav:"year"`
 Info  map[string]interface{} `dynamodbav:"info"`
}

// GetKey returns the composite primary key of the movie in a format that can be
// sent to DynamoDB.
func (movie Movie) GetKey() map[string]types.AttributeValue {
 title, err := attributevalue.Marshal(movie.Title)
 if err != nil {
  panic(err)
 }
 year, err := attributevalue.Marshal(movie.Year)
 if err != nil {
  panic(err)
 }
 return map[string]types.AttributeValue{"title": title, "year": year}
}

// String returns the title, year, rating, and plot of a movie, formatted for the
 example.
func (movie Movie) String() string {
 return fmt.Sprintf("%v\n\tReleased: %v\n\tRating: %v\n\tPlot: %v\n",
  movie.Title, movie.Year, movie.Info["rating"], movie.Info["plot"])
}
```

- Einzelheiten zur API finden Sie PutItemin der AWS SDK für Go API-Referenz.

Java

SDK für Java 2.x

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

Fügt ein Element in eine Tabelle ein mit DynamoDbClient.

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.AttributeValue;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.model.PutItemRequest;
import software.amazon.awssdk.services.dynamodb.model.PutItemResponse;
import software.amazon.awssdk.services.dynamodb.model.ResourceNotFoundException;
import java.util.HashMap;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 *
 * To place items into an Amazon DynamoDB table using the AWS SDK for Java V2,
 * its better practice to use the
 * Enhanced Client. See the EnhancedPutItem example.
 */
public class PutItem {
    public static void main(String[] args) {
        final String usage = """

                Usage:
                    <tableName> <key> <keyVal> <albumtitle> <albumtitleval>
 <awards> <awardsval> <Songtitle> <songtitleval>

                Where:
                    tableName - The Amazon DynamoDB table in which an item is
 placed (for example, Music3).
                    key - The key used in the Amazon DynamoDB table (for example,
 Artist).
                    keyval - The key value that represents the item to get (for
 example, Famous Band).
                    albumTitle - The Album title (for example, AlbumTitle).
                    AlbumTitleValue - The name of the album (for example, Songs
 About Life ).
                    Awards - The awards column (for example, Awards).
                    AwardVal - The value of the awards (for example, 10).
```

```
                    SongTitle - The song title (for example, SongTitle).
                    SongTitleVal - The value of the song title (for example,
Happy Day).
                **Warning** This program will  place an item that you specify
into a table!
                """;

        if (args.length != 9) {
            System.out.println(usage);
            System.exit(1);
        }

        String tableName = args[0];
        String key = args[1];
        String keyVal = args[2];
        String albumTitle = args[3];
        String albumTitleValue = args[4];
        String awards = args[5];
        String awardVal = args[6];
        String songTitle = args[7];
        String songTitleVal = args[8];

        Region region = Region.US_EAST_1;
        DynamoDbClient ddb = DynamoDbClient.builder()
                .region(region)
                .build();

        putItemInTable(ddb, tableName, key, keyVal, albumTitle, albumTitleValue,
awards, awardVal, songTitle,
                songTitleVal);
        System.out.println("Done!");
        ddb.close();
    }

    public static void putItemInTable(DynamoDbClient ddb,
            String tableName,
            String key,
            String keyVal,
            String albumTitle,
            String albumTitleValue,
            String awards,
            String awardVal,
            String songTitle,
            String songTitleVal) {
```

```
        HashMap<String, AttributeValue> itemValues = new HashMap<>();
        itemValues.put(key, AttributeValue.builder().s(keyVal).build());
        itemValues.put(songTitle,
 AttributeValue.builder().s(songTitleVal).build());
        itemValues.put(albumTitle,
 AttributeValue.builder().s(albumTitleValue).build());
        itemValues.put(awards, AttributeValue.builder().s(awardVal).build());

        PutItemRequest request = PutItemRequest.builder()
                .tableName(tableName)
                .item(itemValues)
                .build();

        try {
            PutItemResponse response = ddb.putItem(request);
            System.out.println(tableName + " was successfully updated. The
 request id is "
                    + response.responseMetadata().requestId());

        } catch (ResourceNotFoundException e) {
            System.err.format("Error: The Amazon DynamoDB table \"%s\" can't be
 found.\n", tableName);
            System.err.println("Be sure that it exists and that you've typed its
 name correctly!");
            System.exit(1);
        } catch (DynamoDbException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
    }
}
```

- Einzelheiten zur API finden Sie PutItemunter AWS SDK for Java 2.x API-Referenz.

JavaScript

SDK für JavaScript (v3)

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

In diesem Beispiel wird der Dokument-Client verwendet, um die Arbeit mit Elementen in DynamoDB zu vereinfachen. Einzelheiten zur API finden Sie unter PutCommand.

```
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";
import { PutCommand, DynamoDBDocumentClient } from "@aws-sdk/lib-dynamodb";

const client = new DynamoDBClient({});
const docClient = DynamoDBDocumentClient.from(client);

export const main = async () => {
  const command = new PutCommand({
    TableName: "HappyAnimals",
    Item: {
      CommonName: "Shiba Inu",
    },
  });

  const response = await docClient.send(command);
  console.log(response);
  return response;
};
```

- Einzelheiten zur API finden Sie PutItemin der AWS SDK für JavaScript API-Referenz.

SDK für JavaScript (v2)

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

Fügen Sie ein Element in eine Tabelle ein.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create the DynamoDB service object
var ddb = new AWS.DynamoDB({ apiVersion: "2012-08-10" });

var params = {
  TableName: "CUSTOMER_LIST",
  Item: {
    CUSTOMER_ID: { N: "001" },
    CUSTOMER_NAME: { S: "Richard Roe" },
  },
};

// Call DynamoDB to add the item to the table
ddb.putItem(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data);
  }
});
```

Fügen Sie ein Element mithilfe des DynamoDB-Dokument-Clients in eine Tabelle ein.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create DynamoDB document client
var docClient = new AWS.DynamoDB.DocumentClient({ apiVersion: "2012-08-10" });

var params = {
  TableName: "TABLE",
  Item: {
    HASHKEY: VALUE,
    ATTRIBUTE_1: "STRING_VALUE",
```

```
    ATTRIBUTE_2: VALUE_2,
  },
};

docClient.put(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data);
  }
});
```

- Weitere Informationen finden Sie im AWS SDK für JavaScript -Entwicklerhandbuch.
- Einzelheiten zur API finden Sie PutItemin der AWS SDK für JavaScript API-Referenz.

Kotlin

SDK für Kotlin

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
suspend fun putItemInTable(
    tableNameVal: String,
    key: String,
    keyVal: String,
    albumTitle: String,
    albumTitleValue: String,
    awards: String,
    awardVal: String,
    songTitle: String,
    songTitleVal: String,
) {
    val itemValues = mutableMapOf<String, AttributeValue>()

    // Add all content to the table.
    itemValues[key] = AttributeValue.S(keyVal)
```

```
    itemValues[songTitle] = AttributeValue.S(songTitleVal)
    itemValues[albumTitle] = AttributeValue.S(albumTitleValue)
    itemValues[awards] = AttributeValue.S(awardVal)

    val request =
        PutItemRequest {
            tableName = tableNameVal
            item = itemValues
        }

    DynamoDbClient.fromEnvironment { region = "us-east-1" }.use { ddb ->
        ddb.putItem(request)
        println(" A new item was placed into $tableNameVal.")
    }
}
```

- Einzelheiten zur API finden Sie PutItemin der API-Referenz zum AWS SDK für Kotlin.

PHP

SDK für PHP

> **Note**
>
> Es gibt noch mehr dazu. GitHub Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
        echo "What's the name of the last movie you watched?\n";
        while (empty($movieName)) {
            $movieName = testable_readline("Movie name: ");
        }
        echo "And what year was it released?\n";
        $movieYear = "year";
        while (!is_numeric($movieYear) || intval($movieYear) != $movieYear) {
            $movieYear = testable_readline("Year released: ");
        }

        $service->putItem([
            'Item' => [
                'year' => [
```

```
                    'N' => "$movieYear",
                ],
                'title' => [
                    'S' => $movieName,
                ],
            ],
            'TableName' => $tableName,
        ]);

    public function putItem(array $array)
    {
        $this->dynamoDbClient->putItem($array);
    }
```

- Einzelheiten zur API finden Sie PutItemin der AWS SDK für PHP API-Referenz.

PowerShell

Tools für PowerShell V4

Beispiel 1: Erstellt ein neues Element oder ersetzt ein vorhandenes Element durch ein neues.

```
$item = @{
  SongTitle = 'Somewhere Down The Road'
  Artist = 'No One You Know'
        AlbumTitle = 'Somewhat Famous'
        Price = 1.94
        Genre = 'Country'
        CriticRating = 9.0
} | ConvertTo-DDBItem
Set-DDBItem -TableName 'Music' -Item $item
```

- Einzelheiten zur API finden Sie unter PutItem AWS -Tools für PowerShellCmdlet-Referenz (V4).

Tools für V5 PowerShell

Beispiel 1: Erstellt ein neues Element oder ersetzt ein vorhandenes Element durch ein neues.

```
$item = @{
  SongTitle = 'Somewhere Down The Road'
  Artist = 'No One You Know'
```

```
        AlbumTitle = 'Somewhat Famous'
        Price = 1.94
        Genre = 'Country'
        CriticRating = 9.0
} | ConvertTo-DDBItem
Set-DDBItem -TableName 'Music' -Item $item
```

- Einzelheiten zur API finden Sie unter PutItem AWS -Tools für PowerShellCmdlet-Referenz (V5).

Python

SDK für Python (Boto3)

> **Note**
>
> Es gibt noch mehr dazu. GitHub Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
class Movies:
    """Encapsulates an Amazon DynamoDB table of movie data.

    Example data structure for a movie record in this table:
        {
            "year": 1999,
            "title": "For Love of the Game",
            "info": {
                "directors": ["Sam Raimi"],
                "release_date": "1999-09-15T00:00:00Z",
                "rating": 6.3,
                "plot": "A washed up pitcher flashes through his career.",
                "rank": 4987,
                "running_time_secs": 8220,
                "actors": [
                    "Kevin Costner",
                    "Kelly Preston",
                    "John C. Reilly"
                ]
            }
        }
```

```
    """

    def __init__(self, dyn_resource):
        """
        :param dyn_resource: A Boto3 DynamoDB resource.
        """
        self.dyn_resource = dyn_resource
        # The table variable is set during the scenario in the call to
        # 'exists' if the table exists. Otherwise, it is set by 'create_table'.
        self.table = None


    def add_movie(self, title, year, plot, rating):
        """
        Adds a movie to the table.

        :param title: The title of the movie.
        :param year: The release year of the movie.
        :param plot: The plot summary of the movie.
        :param rating: The quality rating of the movie.
        """
        try:
            self.table.put_item(
                Item={
                    "year": year,
                    "title": title,
                    "info": {"plot": plot, "rating": Decimal(str(rating))},
                }
            )
        except ClientError as err:
            logger.error(
                "Couldn't add movie %s to table %s. Here's why: %s: %s",
                title,
                self.table.name,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
```

- Einzelheiten zur API finden Sie PutItemin AWS SDK for Python (Boto3) API Reference.

Ruby

SDK für Ruby

> **Note**
>
> Es gibt noch mehr dazu. GitHub Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
class DynamoDBBasics
  attr_reader :dynamo_resource, :table

  def initialize(table_name)
    client = Aws::DynamoDB::Client.new(region: 'us-east-1')
    @dynamo_resource = Aws::DynamoDB::Resource.new(client: client)
    @table = @dynamo_resource.table(table_name)
  end

  # Adds a movie to the table.
  #
  # @param movie [Hash] The title, year, plot, and rating of the movie.
  def add_item(movie)
    @table.put_item(
      item: {
        'year' => movie[:year],
        'title' => movie[:title],
        'info' => { 'plot' => movie[:plot], 'rating' => movie[:rating] }
      }
    )
  rescue Aws::DynamoDB::Errors::ServiceError => e
    puts("Couldn't add movie #{title} to table #{@table.name}. Here's why:")
    puts("\t#{e.code}: #{e.message}")
    raise
  end
```

- Einzelheiten zur API finden Sie PutItemin der AWS SDK für Ruby API-Referenz.

Rust

SDK für Rust

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
pub async fn add_item(client: &Client, item: Item, table: &String) ->
 Result<ItemOut, Error> {
    let user_av = AttributeValue::S(item.username);
    let type_av = AttributeValue::S(item.p_type);
    let age_av = AttributeValue::S(item.age);
    let first_av = AttributeValue::S(item.first);
    let last_av = AttributeValue::S(item.last);

    let request = client
        .put_item()
        .table_name(table)
        .item("username", user_av)
        .item("account_type", type_av)
        .item("age", age_av)
        .item("first_name", first_av)
        .item("last_name", last_av);

    println!("Executing request [{request:?}] to add item...");

    let resp = request.send().await?;

    let attributes = resp.attributes().unwrap();

    let username = attributes.get("username").cloned();
    let first_name = attributes.get("first_name").cloned();
    let last_name = attributes.get("last_name").cloned();
    let age = attributes.get("age").cloned();
    let p_type = attributes.get("p_type").cloned();

    println!(
        "Added user {:?}, {:?} {:?}, age {:?} as {:?} user",
        username, first_name, last_name, age, p_type
```

```
    );

    Ok(ItemOut {
        p_type,
        age,
        username,
        first_name,
        last_name,
    })
}
```

- Einzelheiten zur API finden Sie PutItemin der API-Referenz zum AWS SDK für Rust.

SAP ABAP

SDK für SAP ABAP

> **Note**
>
> Es gibt noch mehr dazu. GitHub Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
    TRY.
        DATA(lo_resp) = lo_dyn->putitem(
          iv_tablename = iv_table_name
          it_item      = it_item ).
        MESSAGE '1 row inserted into DynamoDB Table' && iv_table_name TYPE 'I'.
      CATCH /aws1/cx_dyncondalcheckfaile00.
        MESSAGE 'A condition specified in the operation could not be evaluated.'
TYPE 'E'.
      CATCH /aws1/cx_dynresourcenotfoundex.
        MESSAGE 'The table or index does not exist' TYPE 'E'.
      CATCH /aws1/cx_dyntransactconflictex.
        MESSAGE 'Another transaction is using the item' TYPE 'E'.
    ENDTRY.
```

- Einzelheiten zur API finden Sie PutItemin der API-Referenz zum AWS SDK für SAP ABAP.

Swift

SDK für Swift

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
import AWSDynamoDB


    /// Add a movie specified as a `Movie` structure to the Amazon DynamoDB
    /// table.
    ///
    /// - Parameter movie: The `Movie` to add to the table.
    ///
    func add(movie: Movie) async throws {
        do {
            guard let client = self.ddbClient else {
                throw MoviesError.UninitializedClient
            }

            // Get a DynamoDB item containing the movie data.
            let item = try await movie.getAsItem()

            // Send the `PutItem` request to Amazon DynamoDB.

            let input = PutItemInput(
                item: item,
                tableName: self.tableName
            )
            _ = try await client.putItem(input: input)
        } catch {
            print("ERROR: add movie:", dump(error))
            throw error
        }
    }


    ///
```

```
    /// Return an array mapping attribute names to Amazon DynamoDB attribute
    /// values, representing the contents of the `Movie` record as a DynamoDB
    /// item.
    ///
    /// - Returns: The movie item as an array of type
    ///   `[Swift.String:DynamoDBClientTypes.AttributeValue]`.
    ///
    func getAsItem() async throws ->
[Swift.String:DynamoDBClientTypes.AttributeValue]  {
        // Build the item record, starting with the year and title, which are
        // always present.

        var item: [Swift.String:DynamoDBClientTypes.AttributeValue] = [
            "year": .n(String(self.year)),
            "title": .s(self.title)
        ]

        // Add the `info` field with the rating and/or plot if they're
        // available.

        var details: [Swift.String:DynamoDBClientTypes.AttributeValue] = [:]
        if (self.info.rating != nil || self.info.plot != nil) {
            if self.info.rating != nil {
                details["rating"] = .n(String(self.info.rating!))
            }
            if self.info.plot != nil {
                details["plot"] = .s(self.info.plot!)
            }
        }
        item["info"] = .m(details)

        return item
    }
```

- Einzelheiten zur API finden Sie PutItemin der API-Referenz zum AWS SDK für Swift.

Eine vollständige Liste der AWS SDK-Entwicklerhandbücher und Codebeispiele finden Sie unterDynamoDB mit einem SDK verwenden AWS. Dieses Thema enthält auch Informationen zu den ersten Schritten und Details zu früheren SDK-Versionen.

Verwendung **`Query`** mit einem AWS SDK oder CLI

Die folgenden Code-Beispiele zeigen, wie `Query` verwendet wird.

Aktionsbeispiele sind Codeauszüge aus größeren Programmen und müssen im Kontext ausgeführt werden. Sie können diese Aktion in den folgenden Codebeispielen im Kontext sehen:

- Kennenlernen der Grundlagen
- Beschleunigen von Lesevorgängen mit DAX
- Erweiterte Szenarien für globale sekundäre Indizes
- Vergleichen mehrerer Werte mit einem einzigen Attribut
- Verwalten globaler sekundärer Indizes
- Ausführen erweiterter Abfrageoperationen
- Abfragen einer Tabelle mithilfe einer „begins_with“-Bedingung
- Abfragen einer Tabelle mit einem Datumsbereich
- Abfragen einer Tabelle mit einem globalen sekundären Index
- Abfragen einer Tabelle mit einem komplexen Filterausdruck
- Abfragen einer Tabelle mit einem dynamischen Filterausdruck
- Abfragen einer Tabelle mit einem Filterausdruck und einem Limit
- Abfragen einer Tabelle mit verschachtelten Attributen
- Abfragen einer Tabelle mit Paginierung
- Abfragen einer Tabelle mit strikt konsistenten Lesevorgängen
- Abfragen nach TTL-Elementen
- Abfragen von Tabellen mithilfe von Datums- und Uhrzeitmustern
- Verwenden von Ausdrucksattributnamen
- Arbeiten mit lokalen sekundären Indizes

.NET

SDK für .NET (v4)

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
/// <summary>
/// Queries the table for movies released in a particular year and
/// then displays the information for the movies returned.
/// </summary>
/// <param name="tableName">The name of the table to query.</param>
/// <param name="year">The release year for which we want to
/// view movies.</param>
/// <returns>The number of movies that match the query.</returns>
public async Task<int> QueryMoviesAsync(string tableName, int year)
{
    try
    {
        var movieTable = new TableBuilder(_amazonDynamoDB, tableName)
            .AddHashKey("year", DynamoDBEntryType.Numeric)
            .AddRangeKey("title", DynamoDBEntryType.String)
            .Build();

        var filter = new QueryFilter("year", QueryOperator.Equal, year);

        Console.WriteLine("\nFind movies released in: {year}:");

        var config = new QueryOperationConfig()
        {
            Limit = 10, // 10 items per page.
            Select = SelectValues.SpecificAttributes,
            AttributesToGet = new List<string>
            {
                "title",
                "year",
            },
            ConsistentRead = true,
```

```
                Filter = filter,
            };

            // Value used to track how many movies match the
            // supplied criteria.
            var moviesFound = 0;

            var search = movieTable.Query(config);
            do
            {
                var movieList = await search.GetNextSetAsync();
                moviesFound += movieList.Count;

                foreach (var movie in movieList)
                {
                    DisplayDocument(movie);
                }
            }
            while (!search.IsDone);

            return moviesFound;
        }
        catch (ResourceNotFoundException ex)
        {
            Console.WriteLine($"Table {tableName} was not found. {ex.Message}");
            return 0;
        }
        catch (AmazonDynamoDBException ex)
        {
            Console.WriteLine($"An Amazon DynamoDB error occurred while querying
movies. {ex.Message}");
            throw;
        }
        catch (Exception ex)
        {
            Console.WriteLine($"An error occurred while querying movies.
{ex.Message}");
            throw;
        }
    }
```

- Weitere API-Informationen finden Sie unter Abfragen in der AWS SDK für .NET -API-Referenz.

Bash

AWS CLI mit Bash-Skript

> **Note**
>
> Es gibt noch mehr dazu. GitHub Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
###############################################################################
# function dynamodb_query
#
# This function queries a DynamoDB table.
#
# Parameters:
#       -n table_name  -- The name of the table.
#       -k key_condition_expression -- The key condition expression.
#       -a attribute_names -- Path to JSON file containing the attribute names.
#       -v attribute_values -- Path to JSON file containing the attribute values.
#       [-p projection_expression]  -- Optional projection expression.
#
#  Returns:
#       The items as json output.
#  And:
#       0 - If successful.
#       1 - If it fails.
###############################################################################
function dynamodb_query() {
  local table_name key_condition_expression attribute_names attribute_values
 projection_expression response
  local option OPTARG # Required to use getopts command in a function.

  # ######################################
  # Function usage explanation
  ######################################
  function usage() {
    echo "function dynamodb_query"
```

```
   echo "Query a DynamoDB table."
   echo " -n table_name  -- The name of the table."
   echo " -k key_condition_expression -- The key condition expression."
   echo " -a attribute_names -- Path to JSON file containing the attribute
names."
   echo " -v attribute_values -- Path to JSON file containing the attribute
values."
   echo " [-p projection_expression]  -- Optional projection expression."
   echo ""
 }

 while getopts "n:k:a:v:p:h" option; do
   case "${option}" in
     n) table_name="${OPTARG}" ;;
     k) key_condition_expression="${OPTARG}" ;;
     a) attribute_names="${OPTARG}" ;;
     v) attribute_values="${OPTARG}" ;;
     p) projection_expression="${OPTARG}" ;;
     h)
       usage
       return 0
       ;;
     \?)
       echo "Invalid parameter"
       usage
       return 1
       ;;
   esac
 done
 export OPTIND=1

 if [[ -z "$table_name" ]]; then
   errecho "ERROR: You must provide a table name with the -n parameter."
   usage
   return 1
 fi

 if [[ -z "$key_condition_expression" ]]; then
   errecho "ERROR: You must provide a key condition expression with the -k
parameter."
   usage
   return 1
 fi
```

```
  if [[ -z "$attribute_names" ]]; then
    errecho "ERROR: You must provide a attribute names with the -a parameter."
    usage
    return 1
  fi

  if [[ -z "$attribute_values" ]]; then
    errecho "ERROR: You must provide a attribute values with the -v parameter."
    usage
    return 1
  fi

  if [[ -z "$projection_expression" ]]; then
    response=$(aws dynamodb query \
      --table-name "$table_name" \
      --key-condition-expression "$key_condition_expression" \
      --expression-attribute-names file://"$attribute_names" \
      --expression-attribute-values file://"$attribute_values")
  else
    response=$(aws dynamodb query \
      --table-name "$table_name" \
      --key-condition-expression "$key_condition_expression" \
      --expression-attribute-names file://"$attribute_names" \
      --expression-attribute-values file://"$attribute_values" \
      --projection-expression "$projection_expression")
  fi

  local error_code=${?}

  if [[ $error_code -ne 0 ]]; then
    aws_cli_error_log $error_code
    errecho "ERROR: AWS reports query operation failed.$response"
    return 1
  fi

  echo "$response"

  return 0
}
```

Die in diesem Beispiel verwendeten Dienstprogrammfunktionen.

```
#############################################################################
# function errecho
#
# This function outputs everything sent to it to STDERR (standard error output).
#############################################################################
function errecho() {
  printf "%s\n" "$*" 1>&2
}

#############################################################################
# function aws_cli_error_log()
#
# This function is used to log the error messages from the AWS CLI.
#
# See https://docs.aws.amazon.com/cli/latest/topic/return-codes.html#cli-aws-
help-return-codes.
#
# The function expects the following argument:
#         $1 - The error code returned by the AWS CLI.
#
#  Returns:
#          0: - Success.
#
#############################################################################
function aws_cli_error_log() {
  local err_code=$1
  errecho "Error code : $err_code"
  if [ "$err_code" == 1 ]; then
    errecho "  One or more S3 transfers failed."
  elif [ "$err_code" == 2 ]; then
    errecho "  Command line failed to parse."
  elif [ "$err_code" == 130 ]; then
    errecho "  Process received SIGINT."
  elif [ "$err_code" == 252 ]; then
    errecho "  Command syntax invalid."
  elif [ "$err_code" == 253 ]; then
    errecho "  The system environment or configuration was invalid."
  elif [ "$err_code" == 254 ]; then
    errecho "  The service returned an error."
  elif [ "$err_code" == 255 ]; then
    errecho "  255 is a catch-all error."
  fi
```

```
  return 0
}
```

- Weitere API-Informationen finden Sie unter Abfragen in der AWS CLI -Befehlsreferenz.

C++

SDK für C++

> **Note**
>
> Es gibt noch mehr GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
//! Perform a query on an Amazon DynamoDB Table and retrieve items.
/*!
  \sa queryItem()
  \param tableName: The table name.
  \param partitionKey: The partition key.
  \param partitionValue: The value for the partition key.
  \param projectionExpression: The projections expression, which is ignored if
 empty.
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
  */

/*
 * The partition key attribute is searched with the specified value. By default,
 all fields and values
 * contained in the item are returned. If an optional projection expression is
 * specified on the command line, only the specified fields and values are
 * returned.
 */

bool AwsDoc::DynamoDB::queryItems(const Aws::String &tableName,
                                  const Aws::String &partitionKey,
                                  const Aws::String &partitionValue,
                                  const Aws::String &projectionExpression,
                                  const Aws::Client::ClientConfiguration
 &clientConfiguration) {
```

```
   Aws::DynamoDB::DynamoDBClient dynamoClient(clientConfiguration);
   Aws::DynamoDB::Model::QueryRequest request;

   request.SetTableName(tableName);

   if (!projectionExpression.empty()) {
       request.SetProjectionExpression(projectionExpression);
   }

   // Set query key condition expression.
   request.SetKeyConditionExpression(partitionKey + "= :valueToMatch");

   // Set Expression AttributeValues.
   Aws::Map<Aws::String, Aws::DynamoDB::Model::AttributeValue> attributeValues;
   attributeValues.emplace(":valueToMatch", partitionValue);

   request.SetExpressionAttributeValues(attributeValues);

   bool result = true;

   // "exclusiveStartKey" is used for pagination.
   Aws::Map<Aws::String, Aws::DynamoDB::Model::AttributeValue>
exclusiveStartKey;
   do {
       if (!exclusiveStartKey.empty()) {
           request.SetExclusiveStartKey(exclusiveStartKey);
           exclusiveStartKey.clear();
       }
       // Perform Query operation.
       const Aws::DynamoDB::Model::QueryOutcome &outcome =
dynamoClient.Query(request);
       if (outcome.IsSuccess()) {
           // Reference the retrieved items.
           const Aws::Vector<Aws::Map<Aws::String,
Aws::DynamoDB::Model::AttributeValue>> &items = outcome.GetResult().GetItems();
           if (!items.empty()) {
               std::cout << "Number of items retrieved from Query: " <<
items.size()
                         << std::endl;
               // Iterate each item and print.
               for (const auto &item: items) {
                   std::cout
                           <<
"*************************************************"
```

```
                            << std::endl;
                    // Output each retrieved field and its value.
                    for (const auto &i: item)
                        std::cout << i.first << ": " << i.second.GetS() <<
 std::endl;
                }
            }
            else {
                std::cout << "No item found in table: " << tableName <<
 std::endl;
            }

            exclusiveStartKey = outcome.GetResult().GetLastEvaluatedKey();
        }
        else {
            std::cerr << "Failed to Query items: " <<
 outcome.GetError().GetMessage();
            result = false;
            break;
        }
    } while (!exclusiveStartKey.empty());

    return result;
}
```

- Weitere API-Informationen finden Sie unter Abfragen in der AWS SDK für C++ -API-Referenz.

CLI

AWS CLI

Beispiel 1: So fragen Sie eine Tabelle ab

Im folgenden Beispiel für `query` werden Elemente in der Tabelle `MusicCollection` abgefragt. Die Tabelle hat einen hash-and-range Primärschlüssel (`ArtistundSongTitle`), aber diese Abfrage spezifiziert nur den Hashschlüsselwert. Es werden Songtitel des Künstlers mit dem Namen „No One You Know“ zurückgegeben.

```
aws dynamodb query \
    --table-name MusicCollection \
```

```
    --projection-expression "SongTitle" \
    --key-condition-expression "Artist = :v1" \
    --expression-attribute-values file://expression-attributes.json \
    --return-consumed-capacity TOTAL
```

Inhalt von `expression-attributes.json`:

```
{
    ":v1": {"S": "No One You Know"}
}
```

Ausgabe:

```
{
    "Items": [
        {
            "SongTitle": {
                "S": "Call Me Today"
            },
            "SongTitle": {
                "S": "Scared of My Shadow"
            }
        }
    ],
    "Count": 2,
    "ScannedCount": 2,
    "ConsumedCapacity": {
        "TableName": "MusicCollection",
        "CapacityUnits": 0.5
    }
}
```

Weitere Informationen finden Sie unter Arbeiten mit Abfragen in DynamoDB im Entwicklerhandbuch zu Amazon DynamoDB.

Beispiel 2: So fragen Sie eine Tabelle mit strikt konsistenten Lesevorgängen ab und durchlaufen den Index in absteigender Reihenfolge

Im folgenden Beispiel wird dieselbe Abfrage wie im ersten Beispiel ausgeführt, die Ergebnisse werden jedoch in umgekehrter Reihenfolge zurückgegeben und es werden strikt konsistente Lesevorgänge verwendet.

```
aws dynamodb query \
    --table-name MusicCollection \
    --projection-expression "SongTitle" \
    --key-condition-expression "Artist = :v1" \
    --expression-attribute-values file://expression-attributes.json \
    --consistent-read \
    --no-scan-index-forward \
    --return-consumed-capacity TOTAL
```

Inhalt von `expression-attributes.json`:

```
{
    ":v1": {"S": "No One You Know"}
}
```

Ausgabe:

```
{
    "Items": [
        {
            "SongTitle": {
                "S": "Scared of My Shadow"
            }
        },
        {
            "SongTitle": {
                "S": "Call Me Today"
            }
        }
    ],
    "Count": 2,
    "ScannedCount": 2,
    "ConsumedCapacity": {
        "TableName": "MusicCollection",
        "CapacityUnits": 1.0
    }
}
```

Weitere Informationen finden Sie unter Arbeiten mit Abfragen in DynamoDB im Entwicklerhandbuch zu Amazon DynamoDB.

Beispiel 3: So filtern Sie bestimmte Ergebnisse heraus

Im folgende Beispiel wird `MusicCollection` abgefragt, Ergebnisse mit bestimmten Werten im `AlbumTitle`-Attribut werden jedoch ausgeschlossen. Beachten Sie, dass sich dies nicht auf `ScannedCount` oder `ConsumedCapacity` auswirkt, da der Filter angewendet wird, nachdem die Elemente gelesen wurden.

```
aws dynamodb query \
    --table-name MusicCollection \
    --key-condition-expression "#n1 = :v1" \
    --filter-expression "NOT (#n2 IN (:v2, :v3))" \
    --expression-attribute-names file://names.json \
    --expression-attribute-values file://values.json \
    --return-consumed-capacity TOTAL
```

Inhalt von `values.json`:

```
{
    ":v1": {"S": "No One You Know"},
    ":v2": {"S": "Blue Sky Blues"},
    ":v3": {"S": "Greatest Hits"}
}
```

Inhalt von `names.json`:

```
{
    "#n1": "Artist",
    "#n2": "AlbumTitle"
}
```

Ausgabe:

```
{
    "Items": [
        {
            "AlbumTitle": {
                "S": "Somewhat Famous"
            },
            "Artist": {
                "S": "No One You Know"
            },
            "SongTitle": {
                "S": "Call Me Today"
            }
```

```
        }
    ],
    "Count": 1,
    "ScannedCount": 2,
    "ConsumedCapacity": {
        "TableName": "MusicCollection",
        "CapacityUnits": 0.5
    }
}
```

Weitere Informationen finden Sie unter Arbeiten mit Abfragen in DynamoDB im Entwicklerhandbuch zu Amazon DynamoDB.

Beispiel 4: So rufen Sie nur die Elementanzahl ab

Das folgende Beispiel ruft eine Anzahl von Elementen ab, die der Abfrage entsprechen, ruft jedoch keines der Elemente selbst ab.

```
aws dynamodb query \
    --table-name MusicCollection \
    --select COUNT \
    --key-condition-expression "Artist = :v1" \
    --expression-attribute-values file://expression-attributes.json
```

Inhalt von `expression-attributes.json`:

```
{
    ":v1": {"S": "No One You Know"}
}
```

Ausgabe:

```
{
    "Count": 2,
    "ScannedCount": 2,
    "ConsumedCapacity": null
}
```

Weitere Informationen finden Sie unter Arbeiten mit Abfragen in DynamoDB im Entwicklerhandbuch zu Amazon DynamoDB.

Beispiel 5: So rufen Sie einen Index ab

Im folgenden Beispiel wird der lokale sekundäre Index `AlbumTitleIndex` abgefragt. Die Abfrage gibt alle Attribute aus der Basistabelle zurück, die in den lokalen sekundären Index projiziert wurden. Beachten Sie, dass Sie bei der Abfrage eines lokalen sekundären Index oder eines globalen sekundären Index auch den Namen der Basistabelle mit dem `table-name`-Parameter angeben müssen.

```
aws dynamodb query \
    --table-name MusicCollection \
    --index-name AlbumTitleIndex \
    --key-condition-expression "Artist = :v1" \
    --expression-attribute-values file://expression-attributes.json \
    --select ALL_PROJECTED_ATTRIBUTES \
    --return-consumed-capacity INDEXES
```

Inhalt von `expression-attributes.json`:

```
{
    ":v1": {"S": "No One You Know"}
}
```

Ausgabe:

```
{
    "Items": [
        {
            "AlbumTitle": {
                "S": "Blue Sky Blues"
            },
            "Artist": {
                "S": "No One You Know"
            },
            "SongTitle": {
                "S": "Scared of My Shadow"
            }
        },
        {
            "AlbumTitle": {
                "S": "Somewhat Famous"
            },
            "Artist": {
                "S": "No One You Know"
            },
```

```
            "SongTitle": {
                "S": "Call Me Today"
            }
        }
    ],
    "Count": 2,
    "ScannedCount": 2,
    "ConsumedCapacity": {
        "TableName": "MusicCollection",
        "CapacityUnits": 0.5,
        "Table": {
            "CapacityUnits": 0.0
        },
        "LocalSecondaryIndexes": {
            "AlbumTitleIndex": {
                "CapacityUnits": 0.5
            }
        }
    }
}
```

Weitere Informationen finden Sie unter Arbeiten mit Abfragen in DynamoDB im Entwicklerhandbuch zu Amazon DynamoDB.

- Weitere API-Informationen finden Sie unter Abfragen in der AWS CLI -Befehlsreferenz.

Go

SDK für Go V2

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
import (
 "context"
 "errors"
 "log"
 "time"
```

```
 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/attributevalue"
 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/expression"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb/types"
)

// TableBasics encapsulates the Amazon DynamoDB service actions used in the
 examples.
// It contains a DynamoDB service client that is used to act on the specified
 table.
type TableBasics struct {
 DynamoDbClient *dynamodb.Client
 TableName      string
}



// Query gets all movies in the DynamoDB table that were released in the
 specified year.
// The function uses the `expression` package to build the key condition
 expression
// that is used in the query.
func (basics TableBasics) Query(ctx context.Context, releaseYear int) ([]Movie,
 error) {
 var err error
 var response *dynamodb.QueryOutput
 var movies []Movie
 keyEx := expression.Key("year").Equal(expression.Value(releaseYear))
 expr, err := expression.NewBuilder().WithKeyCondition(keyEx).Build()
 if err != nil {
  log.Printf("Couldn't build expression for query. Here's why: %v\n", err)
 } else {
  queryPaginator := dynamodb.NewQueryPaginator(basics.DynamoDbClient,
 &dynamodb.QueryInput{
   TableName:                 aws.String(basics.TableName),
   ExpressionAttributeNames:  expr.Names(),
   ExpressionAttributeValues: expr.Values(),
   KeyConditionExpression:    expr.KeyCondition(),
  })
  for queryPaginator.HasMorePages() {
   response, err = queryPaginator.NextPage(ctx)
   if err != nil {
```

```
    log.Printf("Couldn't query for movies released in %v. Here's why: %v\n",
 releaseYear, err)
    break
   } else {
    var moviePage []Movie
    err = attributevalue.UnmarshalListOfMaps(response.Items, &moviePage)
    if err != nil {
     log.Printf("Couldn't unmarshal query response. Here's why: %v\n", err)
     break
    } else {
     movies = append(movies, moviePage...)
    }
   }
  }
 }
 return movies, err
}
```

Definieren Sie eine Movie-Struktur, die in diesem Beispiel verwendet wird.

```
import (
 "archive/zip"
 "bytes"
 "encoding/json"
 "fmt"
 "io"
 "log"
 "net/http"

 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/attributevalue"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb/types"
)

// Movie encapsulates data about a movie. Title and Year are the composite
 primary key
// of the movie in Amazon DynamoDB. Title is the sort key, Year is the partition
 key,
// and Info is additional data.
type Movie struct {
 Title string                `dynamodbav:"title"`
```

```
 Year  int                    `dynamodbav:"year"`
 Info  map[string]interface{} `dynamodbav:"info"`
}

// GetKey returns the composite primary key of the movie in a format that can be
// sent to DynamoDB.
func (movie Movie) GetKey() map[string]types.AttributeValue {
 title, err := attributevalue.Marshal(movie.Title)
 if err != nil {
  panic(err)
 }
 year, err := attributevalue.Marshal(movie.Year)
 if err != nil {
  panic(err)
 }
 return map[string]types.AttributeValue{"title": title, "year": year}
}

// String returns the title, year, rating, and plot of a movie, formatted for the
 example.
func (movie Movie) String() string {
 return fmt.Sprintf("%v\n\tReleased: %v\n\tRating: %v\n\tPlot: %v\n",
  movie.Title, movie.Year, movie.Info["rating"], movie.Info["plot"])
}
```

- Weitere API-Informationen finden Sie unter Query in der AWS SDK für Go -API-Referenz.

Java

SDK für Java 2.x

> **Note**
>
> Es gibt noch mehr GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

Fragt eine Tabelle ab mithilfe von DynamoDbClient.

```
import software.amazon.awssdk.regions.Region;
```

```
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.AttributeValue;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.model.QueryRequest;
import software.amazon.awssdk.services.dynamodb.model.QueryResponse;
import java.util.HashMap;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 *
 * To query items from an Amazon DynamoDB table using the AWS SDK for Java V2,
 * its better practice to use the
 * Enhanced Client. See the EnhancedQueryRecords example.
 */
public class Query {
    public static void main(String[] args) {
        final String usage = """

                Usage:
                    <tableName> <partitionKeyName> <partitionKeyVal>

                Where:
                    tableName - The Amazon DynamoDB table to put the item in (for
 example, Music3).
                    partitionKeyName - The partition key name of the Amazon
 DynamoDB table (for example, Artist).
                    partitionKeyVal - The value of the partition key that should
 match (for example, Famous Band).
                """;

        if (args.length != 3) {
            System.out.println(usage);
            System.exit(1);
        }

        String tableName = args[0];
        String partitionKeyName = args[1];
        String partitionKeyVal = args[2];
```

```
        // For more information about an alias, see:
        // https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/
Expressions.ExpressionAttributeNames.html
        String partitionAlias = "#a";

        System.out.format("Querying %s", tableName);
        System.out.println("");
        Region region = Region.US_EAST_1;
        DynamoDbClient ddb = DynamoDbClient.builder()
                .region(region)
                .build();

        int count = queryTable(ddb, tableName, partitionKeyName, partitionKeyVal,
 partitionAlias);
        System.out.println("There were " + count + "  record(s) returned");
        ddb.close();
    }

    public static int queryTable(DynamoDbClient ddb, String tableName, String
 partitionKeyName, String partitionKeyVal,
            String partitionAlias) {
        // Set up an alias for the partition key name in case it's a reserved
 word.
        HashMap<String, String> attrNameAlias = new HashMap<String, String>();
        attrNameAlias.put(partitionAlias, partitionKeyName);

        // Set up mapping of the partition name with the value.
        HashMap<String, AttributeValue> attrValues = new HashMap<>();
        attrValues.put(":" + partitionKeyName, AttributeValue.builder()
                .s(partitionKeyVal)
                .build());

        QueryRequest queryReq = QueryRequest.builder()
                .tableName(tableName)
                .keyConditionExpression(partitionAlias + " = :" +
 partitionKeyName)
                .expressionAttributeNames(attrNameAlias)
                .expressionAttributeValues(attrValues)
                .build();

        try {
            QueryResponse response = ddb.query(queryReq);
            return response.count();
```

```
        } catch (DynamoDbException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
        return -1;
    }
}
```

Fragt eine Tabelle mithilfe von `DynamoDbClient` und eines sekundären Index ab.

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.AttributeValue;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.model.QueryRequest;
import software.amazon.awssdk.services.dynamodb.model.QueryResponse;
import java.util.HashMap;
import java.util.Map;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 *
 * Create the Movies table by running the Scenario example and loading the Movie
 * data from the JSON file. Next create a secondary
 * index for the Movies table that uses only the year column. Name the index
 * **year-index**. For more information, see:
 *
 * https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/GSI.html
 */
public class QueryItemsUsingIndex {
    public static void main(String[] args) {
        String tableName = "Movies";
        Region region = Region.US_EAST_1;
        DynamoDbClient ddb = DynamoDbClient.builder()
                .region(region)
```

```
                .build();

        queryIndex(ddb, tableName);
        ddb.close();
    }

    public static void queryIndex(DynamoDbClient ddb, String tableName) {
        try {
            Map<String, String> expressionAttributesNames = new HashMap<>();
            expressionAttributesNames.put("#year", "year");
            Map<String, AttributeValue> expressionAttributeValues = new
 HashMap<>();
            expressionAttributeValues.put(":yearValue",
 AttributeValue.builder().n("2013").build());

            QueryRequest request = QueryRequest.builder()
                    .tableName(tableName)
                    .indexName("year-index")
                    .keyConditionExpression("#year = :yearValue")
                    .expressionAttributeNames(expressionAttributesNames)
                    .expressionAttributeValues(expressionAttributeValues)
                    .build();

            System.out.println("=== Movie Titles ===");
            QueryResponse response = ddb.query(request);
            response.items()
                    .forEach(movie ->
 System.out.println(movie.get("title").s()));

        } catch (DynamoDbException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
    }
}
```

- Weitere API-Informationen finden Sie unter Abfragen in der AWS SDK for Java 2.x -API-Referenz.

JavaScript

SDK für JavaScript (v3)

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

In diesem Beispiel wird der Dokument-Client verwendet, um die Arbeit mit Elementen in DynamoDB zu vereinfachen. Einzelheiten zur API finden Sie unter QueryCommand.

```
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";
import { QueryCommand, DynamoDBDocumentClient } from "@aws-sdk/lib-dynamodb";

const client = new DynamoDBClient({});
const docClient = DynamoDBDocumentClient.from(client);

export const main = async () => {
  const command = new QueryCommand({
    TableName: "CoffeeCrop",
    KeyConditionExpression:
      "OriginCountry = :originCountry AND RoastDate > :roastDate",
    ExpressionAttributeValues: {
      ":originCountry": "Ethiopia",
      ":roastDate": "2023-05-01",
    },
    ConsistentRead: true,
  });

  const response = await docClient.send(command);
  console.log(response);
  return response;
};
```

- Weitere Informationen finden Sie im AWS SDK für JavaScript -Entwicklerhandbuch.
- Weitere API-Informationen finden Sie unter Abfragen in der AWS SDK für JavaScript -API-Referenz.

SDK für JavaScript (v2)

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create DynamoDB document client
var docClient = new AWS.DynamoDB.DocumentClient({ apiVersion: "2012-08-10" });

var params = {
  ExpressionAttributeValues: {
    ":s": 2,
    ":e": 9,
    ":topic": "PHRASE",
  },
  KeyConditionExpression: "Season = :s and Episode > :e",
  FilterExpression: "contains (Subtitle, :topic)",
  TableName: "EPISODES_TABLE",
};

docClient.query(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data.Items);
  }
});
```

- Weitere Informationen finden Sie im AWS SDK für JavaScript -Entwicklerhandbuch.
- Weitere API-Informationen finden Sie unter Abfragen in der AWS SDK für JavaScript -API-Referenz.

Kotlin

SDK für Kotlin

> **Note**
>
> Es gibt noch mehr GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
suspend fun queryDynTable(
    tableNameVal: String,
    partitionKeyName: String,
    partitionKeyVal: String,
    partitionAlias: String,
): Int {
    val attrNameAlias = mutableMapOf<String, String>()
    attrNameAlias[partitionAlias] = partitionKeyName

    // Set up mapping of the partition name with the value.
    val attrValues = mutableMapOf<String, AttributeValue>()
    attrValues[":$partitionKeyName"] = AttributeValue.S(partitionKeyVal)

    val request =
        QueryRequest {
            tableName = tableNameVal
            keyConditionExpression = "$partitionAlias = :$partitionKeyName"
            expressionAttributeNames = attrNameAlias
            this.expressionAttributeValues = attrValues
        }

    DynamoDbClient.fromEnvironment { region = "us-east-1" }.use { ddb ->
        val response = ddb.query(request)
        return response.count
    }
}
```

- Weitere API-Informationen finden Sie unter Abfragen in der API–Referenz zum AWS SDK für Kotlin.

PHP

SDK für PHP

> **Note**
>
> Es gibt noch mehr GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
        $birthKey = [
            'Key' => [
                'year' => [
                    'N' => "$birthYear",
                ],
            ],
        ];
        $result = $service->query($tableName, $birthKey);

    public function query(string $tableName, $key)
    {
        $expressionAttributeValues = [];
        $expressionAttributeNames = [];
        $keyConditionExpression = "";
        $index = 1;
        foreach ($key as $name => $value) {
            $keyConditionExpression .= "#" . array_key_first($value) . " = :v
$index,";
            $expressionAttributeNames["#" . array_key_first($value)] =
 array_key_first($value);
            $hold = array_pop($value);
            $expressionAttributeValues[":v$index"] = [
                array_key_first($hold) => array_pop($hold),
            ];
        }
        $keyConditionExpression = substr($keyConditionExpression, 0, -1);
        $query = [
            'ExpressionAttributeValues' => $expressionAttributeValues,
            'ExpressionAttributeNames' => $expressionAttributeNames,
            'KeyConditionExpression' => $keyConditionExpression,
            'TableName' => $tableName,
        ];
```

```
        return $this->dynamoDbClient->query($query);
    }
```

- Weitere API-Informationen finden Sie unter Abfragen in der AWS SDK für PHP -API-Referenz.

PowerShell

Tools für PowerShell V4

Beispiel 1: Ruft eine Abfrage auf, die DynamoDB-Elemente mit dem angegebenen SongTitle Wert und Artist zurückgibt.

```
$invokeDDBQuery = @{
    TableName = 'Music'
    KeyConditionExpression = ' SongTitle = :SongTitle and Artist = :Artist'
    ExpressionAttributeValues = @{
        ':SongTitle' = 'Somewhere Down The Road'
        ':Artist' = 'No One You Know'
    } | ConvertTo-DDBItem
}
Invoke-DDBQuery @invokeDDBQuery | ConvertFrom-DDBItem
```

Ausgabe:

```
Name                           Value
----                           -----
Genre                          Country
Artist                         No One You Know
Price                          1.94
CriticRating                   9
SongTitle                      Somewhere Down The Road
AlbumTitle                     Somewhat Famous
```

- Weitere API-Informationen finden Sie unter Abfragen in der AWS -Tools für PowerShell -Cmdlet-Referenz (V4).

Tools für V5 PowerShell

Beispiel 1: Ruft eine Abfrage auf, die DynamoDB-Elemente mit dem angegebenen SongTitle Wert und Artist zurückgibt.

```
$invokeDDBQuery = @{
    TableName = 'Music'
    KeyConditionExpression = ' SongTitle = :SongTitle and Artist = :Artist'
    ExpressionAttributeValues = @{
        ':SongTitle' = 'Somewhere Down The Road'
        ':Artist' = 'No One You Know'
    } | ConvertTo-DDBItem
}
Invoke-DDBQuery @invokeDDBQuery | ConvertFrom-DDBItem
```

Ausgabe:

```
Name                           Value
----                           -----
Genre                          Country
Artist                         No One You Know
Price                          1.94
CriticRating                   9
SongTitle                      Somewhere Down The Road
AlbumTitle                     Somewhat Famous
```

- Weitere API-Informationen finden Sie unter Abfragen in der AWS -Tools für PowerShell -Cmdlet-Referenz (V5).

Python

SDK für Python (Boto3)

> **Note**
>
> Es gibt noch mehr dazu. GitHub Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

Fragen Sie Elemente mithilfe eines Schlüsselbedingungsausdrucks ab.

```
class Movies:
    """Encapsulates an Amazon DynamoDB table of movie data.

    Example data structure for a movie record in this table:
        {
```

```
            "year": 1999,
            "title": "For Love of the Game",
            "info": {
                "directors": ["Sam Raimi"],
                "release_date": "1999-09-15T00:00:00Z",
                "rating": 6.3,
                "plot": "A washed up pitcher flashes through his career.",
                "rank": 4987,
                "running_time_secs": 8220,
                "actors": [
                    "Kevin Costner",
                    "Kelly Preston",
                    "John C. Reilly"
                ]
            }
        }
    """

    def __init__(self, dyn_resource):
        """
        :param dyn_resource: A Boto3 DynamoDB resource.
        """
        self.dyn_resource = dyn_resource
        # The table variable is set during the scenario in the call to
        # 'exists' if the table exists. Otherwise, it is set by 'create_table'.
        self.table = None


    def query_movies(self, year):
        """
        Queries for movies that were released in the specified year.

        :param year: The year to query.
        :return: The list of movies that were released in the specified year.
        """
        try:
            response =
self.table.query(KeyConditionExpression=Key("year").eq(year))
        except ClientError as err:
            logger.error(
                "Couldn't query for movies released in %s. Here's why: %s: %s",
                year,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
```

```
            )
            raise
        else:
            return response["Items"]
```

Fragen Sie Elemente ab und projizieren Sie sie, um eine Teilmenge von Daten zurückzugeben.

```
class UpdateQueryWrapper:
    def __init__(self, table):
        self.table = table


    def query_and_project_movies(self, year, title_bounds):
        """
        Query for movies that were released in a specified year and that have
 titles
        that start within a range of letters. A projection expression is used
        to return a subset of data for each movie.

        :param year: The release year to query.
        :param title_bounds: The range of starting letters to query.
        :return: The list of movies.
        """
        try:
            response = self.table.query(
                ProjectionExpression="#yr, title, info.genres, info.actors[0]",
                ExpressionAttributeNames={"#yr": "year"},
                KeyConditionExpression=(
                    Key("year").eq(year)
                    & Key("title").between(
                        title_bounds["first"], title_bounds["second"]
                    )
                ),
            )
        except ClientError as err:
            if err.response["Error"]["Code"] == "ValidationException":
                logger.warning(
                    "There's a validation error. Here's the message: %s: %s",
                    err.response["Error"]["Code"],
                    err.response["Error"]["Message"],
```

```
                )
            else:
                logger.error(
                    "Couldn't query for movies. Here's why: %s: %s",
                    err.response["Error"]["Code"],
                    err.response["Error"]["Message"],
                )
                raise
        else:
            return response["Items"]
```

- Weitere API-Informationen finden Sie unter Abfragen in der API-Referenz zum AWS SDK für Python (Boto3).

Ruby

SDK für Ruby

> **Note**
>
> Es gibt noch mehr GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
class DynamoDBBasics
  attr_reader :dynamo_resource, :table

  def initialize(table_name)
    client = Aws::DynamoDB::Client.new(region: 'us-east-1')
    @dynamo_resource = Aws::DynamoDB::Resource.new(client: client)
    @table = @dynamo_resource.table(table_name)
  end

  # Queries for movies that were released in the specified year.
  #
  # @param year [Integer] The year to query.
  # @return [Array] The list of movies that were released in the specified year.
  def query_items(year)
    response = @table.query(
```

```
      key_condition_expression: '#yr = :year',
      expression_attribute_names: { '#yr' => 'year' },
      expression_attribute_values: { ':year' => year }
    )
  rescue Aws::DynamoDB::Errors::ServiceError => e
    puts("Couldn't query for movies released in #{year}. Here's why:")
    puts("\t#{e.code}: #{e.message}")
    raise
  else
    response.items
  end
```

- Weitere API-Informationen finden Sie unter Query in der AWS SDK für Ruby -API-Referenz.

Rust

SDK für Rust

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

Finden Sie die Filme, die im angegebenen Jahr gedreht wurden.

```
pub async fn movies_in_year(
    client: &Client,
    table_name: &str,
    year: u16,
) -> Result<Vec<Movie>, MovieError> {
    let results = client
        .query()
        .table_name(table_name)
        .key_condition_expression("#yr = :yyyy")
        .expression_attribute_names("#yr", "year")
        .expression_attribute_values(":yyyy",
 AttributeValue::N(year.to_string()))
        .send()
        .await?;
```

```
    if let Some(items) = results.items {
        let movies = items.iter().map(|v| v.into()).collect();
        Ok(movies)
    } else {
        Ok(vec![])
    }
}
```

- Weitere API-Informationen finden Sie unter Abfragen in der API-Referenz zum AWS SDK für Rust.

SAP ABAP

SDK für SAP ABAP

> **Note**
>
> Es gibt noch mehr dazu. GitHub Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
    TRY.
        " Query movies for a given year .
        DATA(lt_attributelist) = VALUE /aws1/
cl_dynattributevalue=>tt_attributevaluelist(
            ( NEW /aws1/cl_dynattributevalue( iv_n = |{ iv_year }| ) ) ).
        DATA(lt_key_conditions) = VALUE /aws1/cl_dyncondition=>tt_keyconditions(
          ( VALUE /aws1/cl_dyncondition=>ts_keyconditions_maprow(
          key = 'year'
          value = NEW /aws1/cl_dyncondition(
          it_attributevaluelist = lt_attributelist
          iv_comparisonoperator = |EQ|
          ) ) ) ).
        oo_result = lo_dyn->query(
          iv_tablename = iv_table_name
          it_keyconditions = lt_key_conditions ).
        DATA(lt_items) = oo_result->get_items( ).
        "You can loop over the results to get item attributes.
        LOOP AT lt_items INTO DATA(lt_item).
```

```
          DATA(lo_title) = lt_item[ key = 'title' ]-value.
          DATA(lo_year) = lt_item[ key = 'year' ]-value.
        ENDLOOP.
        DATA(lv_count) = oo_result->get_count( ).
        MESSAGE 'Item count is: ' && lv_count TYPE 'I'.
      CATCH /aws1/cx_dynresourcenotfoundex.
        MESSAGE 'The table or index does not exist' TYPE 'E'.
    ENDTRY.
```

- Weitere API-Informationen finden Sie unter Abfragen in der API-Referenz für das AWS SDK für SAP ABAP.

Swift

SDK für Swift

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
import AWSDynamoDB


    /// Get all the movies released in the specified year.
    ///
    /// - Parameter year: The release year of the movies to return.
    ///
    /// - Returns: An array of `Movie` objects describing each matching movie.
    ///
    func getMovies(fromYear year: Int) async throws -> [Movie] {
        do {
            guard let client = self.ddbClient else {
                throw MoviesError.UninitializedClient
            }

            let input = QueryInput(
                expressionAttributeNames: [
                    "#y": "year"
```

```
                ],
                expressionAttributeValues: [
                    ":y": .n(String(year))
                ],
                keyConditionExpression: "#y = :y",
                tableName: self.tableName
            )
            // Use "Paginated" to get all the movies.
            // This lets the SDK handle the 'lastEvaluatedKey' property in
"QueryOutput".

            let pages = client.queryPaginated(input: input)

            var movieList: [Movie] = []
            for try await page in pages {
                guard let items = page.items else {
                    print("Error: no items returned.")
                    continue
                }

                // Convert the found movies into `Movie` objects and return an
array
                // of them.

                for item in items {
                    let movie = try Movie(withItem: item)
                    movieList.append(movie)
                }
            }
            return movieList
        } catch {
            print("ERROR: getMovies:", dump(error))
            throw error
        }
    }
```

- Detaillierte API-Informationen finden Sie unter Abfragen in der API-Referenz zum AWS SDK für Swift.

Eine vollständige Liste der AWS SDK-Entwicklerhandbücher und Codebeispiele finden Sie unterDynamoDB mit einem SDK verwenden AWS. Dieses Thema enthält auch Informationen zu den ersten Schritten und Details zu früheren SDK-Versionen.

Verwendung **Scan** mit einem AWS SDK oder CLI

Die folgenden Code-Beispiele zeigen, wie `Scan` verwendet wird.

Aktionsbeispiele sind Codeauszüge aus größeren Programmen und müssen im Kontext ausgeführt werden. Sie können diese Aktion in den folgenden Codebeispielen im Kontext sehen:

- Kennenlernen der Grundlagen
- Beschleunigen von Lesevorgängen mit DAX
- Vergleichen mehrerer Werte mit einem einzigen Attribut

.NET

SDK für .NET (v4)

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
    /// <summary>
    /// Scans the table for movies released between the specified years.
    /// </summary>
    /// <param name="tableName">The name of the table to scan.</param>
    /// <param name="startYear">The starting year for the range.</param>
    /// <param name="endYear">The ending year for the range.</param>
    /// <returns>The number of movies found in the specified year range.</
returns>
    public async Task<int> ScanTableAsync(
        string tableName,
        int startYear,
        int endYear)
    {
        try
        {
```

```
            var request = new ScanRequest
            {
                TableName = tableName,
                ExpressionAttributeNames = new Dictionary<string, string>
                {
                    { "#yr", "year" },
                },
                ExpressionAttributeValues = new Dictionary<string,
AttributeValue>
                {
                    { ":y_a", new AttributeValue { N = startYear.ToString() } },
                    { ":y_z", new AttributeValue { N = endYear.ToString() } },
                },
                FilterExpression = "#yr between :y_a and :y_z",
                ProjectionExpression = "#yr, title, info.actors[0],
info.directors, info.running_time_secs",
                Limit = 10 // Set a limit to demonstrate using the
LastEvaluatedKey.
            };

            // Keep track of how many movies were found.
            int foundCount = 0;

            var response = new ScanResponse();
            do
            {
                response = await _amazonDynamoDB.ScanAsync(request);
                foundCount += response.Items.Count;
                response.Items.ForEach(i => DisplayItem(i));
                request.ExclusiveStartKey = response.LastEvaluatedKey;
            }
            while (response?.LastEvaluatedKey?.Count > 0);
            return foundCount;
        }
        catch (ResourceNotFoundException ex)
        {
            Console.WriteLine($"Table {tableName} was not found. {ex.Message}");
            return 0;
        }
        catch (AmazonDynamoDBException ex)
        {
            Console.WriteLine($"An Amazon DynamoDB error occurred while scanning
table. {ex.Message}");
            throw;
```

```
        }
        catch (Exception ex)
        {
            Console.WriteLine($"An error occurred while scanning table.
 {ex.Message}");
            throw;
        }
    }
```

- Weitere API-Informationen finden Sie unter Scan in der AWS SDK für .NET -API-Referenz.

Bash

AWS CLI mit Bash-Skript

> **Note**
>
> Es gibt noch mehr dazu. GitHub Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
###############################################################################
# function dynamodb_scan
#
# This function scans a DynamoDB table.
#
# Parameters:
#       -n table_name  -- The name of the table.
#       -f filter_expression  -- The filter expression.
#       -a expression_attribute_names -- Path to JSON file containing the
 expression attribute names.
#       -v expression_attribute_values -- Path to JSON file containing the
 expression attribute values.
#       [-p projection_expression]  -- Optional projection expression.
#
#  Returns:
#       The items as json output.
#  And:
#       0 - If successful.
#       1 - If it fails.
```

```
###############################################################################
function dynamodb_scan() {
  local table_name filter_expression expression_attribute_names
 expression_attribute_values projection_expression response
  local option OPTARG # Required to use getopts command in a function.

  # ######################################
  # Function usage explanation
  ######################################
  function usage() {
    echo "function dynamodb_scan"
    echo "Scan a DynamoDB table."
    echo " -n table_name  -- The name of the table."
    echo " -f filter_expression  -- The filter expression."
    echo " -a expression_attribute_names -- Path to JSON file containing the
 expression attribute names."
    echo " -v expression_attribute_values -- Path to JSON file containing the
 expression attribute values."
    echo " [-p projection_expression]  -- Optional projection expression."
    echo ""
  }

  while getopts "n:f:a:v:p:h" option; do
    case "${option}" in
      n) table_name="${OPTARG}" ;;
      f) filter_expression="${OPTARG}" ;;
      a) expression_attribute_names="${OPTARG}" ;;
      v) expression_attribute_values="${OPTARG}" ;;
      p) projection_expression="${OPTARG}" ;;
      h)
        usage
        return 0
        ;;
      \?)
        echo "Invalid parameter"
        usage
        return 1
        ;;
    esac
  done
  export OPTIND=1

  if [[ -z "$table_name" ]]; then
    errecho "ERROR: You must provide a table name with the -n parameter."
```

```
    usage
    return 1
  fi

  if [[ -z "$filter_expression" ]]; then
    errecho "ERROR: You must provide a filter expression with the -f parameter."
    usage
    return 1
  fi

  if [[ -z "$expression_attribute_names" ]]; then
    errecho "ERROR: You must provide expression attribute names with the -a
parameter."
    usage
    return 1
  fi

  if [[ -z "$expression_attribute_values" ]]; then
    errecho "ERROR: You must provide expression attribute values with the -v
parameter."
    usage
    return 1
  fi

  if [[ -z "$projection_expression" ]]; then
    response=$(aws dynamodb scan \
      --table-name "$table_name" \
      --filter-expression "$filter_expression" \
      --expression-attribute-names file://"$expression_attribute_names" \
      --expression-attribute-values file://"$expression_attribute_values")
  else
    response=$(aws dynamodb scan \
      --table-name "$table_name" \
      --filter-expression "$filter_expression" \
      --expression-attribute-names file://"$expression_attribute_names" \
      --expression-attribute-values file://"$expression_attribute_values" \
      --projection-expression "$projection_expression")
  fi

  local error_code=${?}

  if [[ $error_code -ne 0 ]]; then
    aws_cli_error_log $error_code
    errecho "ERROR: AWS reports scan operation failed.$response"
```

```
    return 1
  fi

  echo "$response"

  return 0
}
```

Die in diesem Beispiel verwendeten Dienstprogrammfunktionen.

```
###############################################################################
# function errecho
#
# This function outputs everything sent to it to STDERR (standard error output).
###############################################################################
function errecho() {
  printf "%s\n" "$*" 1>&2
}

###############################################################################
# function aws_cli_error_log()
#
# This function is used to log the error messages from the AWS CLI.
#
# See https://docs.aws.amazon.com/cli/latest/topic/return-codes.html#cli-aws-
help-return-codes.
#
# The function expects the following argument:
#         $1 - The error code returned by the AWS CLI.
#
#  Returns:
#          0: - Success.
#
###############################################################################
function aws_cli_error_log() {
  local err_code=$1
  errecho "Error code : $err_code"
  if [ "$err_code" == 1 ]; then
    errecho "  One or more S3 transfers failed."
  elif [ "$err_code" == 2 ]; then
    errecho "  Command line failed to parse."
  elif [ "$err_code" == 130 ]; then
```

```
    errecho "  Process received SIGINT."
  elif [ "$err_code" == 252 ]; then
    errecho "  Command syntax invalid."
  elif [ "$err_code" == 253 ]; then
    errecho "  The system environment or configuration was invalid."
  elif [ "$err_code" == 254 ]; then
    errecho "  The service returned an error."
  elif [ "$err_code" == 255 ]; then
    errecho "  255 is a catch-all error."
  fi

  return 0
}
```

- Weitere API-Informationen finden Sie unter Scan in der AWS CLI -Befehlsreferenz.

C++

SDK für C++

> **Note**
>
> Es gibt noch mehr GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
//! Scan an Amazon DynamoDB table.
/*!
  \sa scanTable()
  \param tableName: Name for the DynamoDB table.
  \param projectionExpression: An optional projection expression, ignored if
 empty.
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
 */

bool AwsDoc::DynamoDB::scanTable(const Aws::String &tableName,
                                 const Aws::String &projectionExpression,
                                 const Aws::Client::ClientConfiguration
 &clientConfiguration) {
    Aws::DynamoDB::DynamoDBClient dynamoClient(clientConfiguration);
```

```
   Aws::DynamoDB::Model::ScanRequest request;
   request.SetTableName(tableName);

   if (!projectionExpression.empty())
       request.SetProjectionExpression(projectionExpression);

   Aws::Vector<Aws::Map<Aws::String, Aws::DynamoDB::Model::AttributeValue>>
all_items;
   Aws::Map<Aws::String, Aws::DynamoDB::Model::AttributeValue>
last_evaluated_key; // Used for pagination;
   do {
       if (!last_evaluated_key.empty()) {
           request.SetExclusiveStartKey(last_evaluated_key);
       }
       const Aws::DynamoDB::Model::ScanOutcome &outcome =
dynamoClient.Scan(request);
       if (outcome.IsSuccess()) {
           // Reference the retrieved items.
           const Aws::Vector<Aws::Map<Aws::String,
Aws::DynamoDB::Model::AttributeValue>> &items = outcome.GetResult().GetItems();
           all_items.insert(all_items.end(), items.begin(), items.end());

           last_evaluated_key = outcome.GetResult().GetLastEvaluatedKey();
       }
       else {
           std::cerr << "Failed to Scan items: " <<
outcome.GetError().GetMessage()
                     << std::endl;
           return false;
       }

   } while (!last_evaluated_key.empty());

   if (!all_items.empty()) {
       std::cout << "Number of items retrieved from scan: " << all_items.size()
                 << std::endl;
       // Iterate each item and print.
       for (const Aws::Map<Aws::String, Aws::DynamoDB::Model::AttributeValue>
&itemMap: all_items) {
           std::cout << "*************************************************"
                     << std::endl;
           // Output each retrieved field and its value.
           for (const auto &itemEntry: itemMap)
               std::cout << itemEntry.first << ": " << itemEntry.second.GetS()
```

```
                          << std::endl;
        }
    }

    else {
        std::cout << "No items found in table: " << tableName << std::endl;
    }

    return true;
}
```

- Weitere API-Informationen finden Sie unter Scan in der AWS SDK für C++ -API-Referenz.

CLI

AWS CLI

So scannen Sie eine Tabelle

Im folgenden Beispiel für `scan` wird die gesamte `MusicCollection`-Tabelle gescannt und die Ergebnisse werden dann auf Songs des Künstlers „No One You Know“ eingegrenzt. Für jedes Element werden nur der Albumtitel und der Songtitel zurückgegeben.

```
aws dynamodb scan \
    --table-name MusicCollection \
    --filter-expression "Artist = :a" \
    --projection-expression "#ST, #AT" \
    --expression-attribute-names file://expression-attribute-names.json \
    --expression-attribute-values file://expression-attribute-values.json
```

Inhalt von `expression-attribute-names.json`:

```
{
    "#ST": "SongTitle",
    "#AT":"AlbumTitle"
}
```

Inhalt von `expression-attribute-values.json`:

```
{
```

```
    ":a": {"S": "No One You Know"}
}
```

Ausgabe:

```
{
    "Count": 2,
    "Items": [
        {
            "SongTitle": {
                "S": "Call Me Today"
            },
            "AlbumTitle": {
                "S": "Somewhat Famous"
            }
        },
        {
            "SongTitle": {
                "S": "Scared of My Shadow"
            },
            "AlbumTitle": {
                "S": "Blue Sky Blues"
            }
        }
    ],
    "ScannedCount": 3,
    "ConsumedCapacity": null
}
```

Weitere Informationen finden Sie unter Arbeiten mit Scans in DynamoDB im Entwicklerhandbuch zu Amazon DynamoDB.

- Weitere API-Informationen finden Sie unter Scan in der AWS CLI -Befehlsreferenz.

Go

SDK für Go V2

> **Note**
>
> Es gibt noch mehr GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
import (
 "context"
 "errors"
 "log"
 "time"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/attributevalue"
 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/expression"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb/types"
)

// TableBasics encapsulates the Amazon DynamoDB service actions used in the
 examples.
// It contains a DynamoDB service client that is used to act on the specified
 table.
type TableBasics struct {
 DynamoDbClient *dynamodb.Client
 TableName      string
}



// Scan gets all movies in the DynamoDB table that were released in a range of
 years
// and projects them to return a reduced set of fields.
// The function uses the `expression` package to build the filter and projection
// expressions.
func (basics TableBasics) Scan(ctx context.Context, startYear int, endYear int)
 ([]Movie, error) {
```

```
 var movies []Movie
 var err error
 var response *dynamodb.ScanOutput
 filtEx := expression.Name("year").Between(expression.Value(startYear),
 expression.Value(endYear))
 projEx := expression.NamesList(
  expression.Name("year"), expression.Name("title"),
 expression.Name("info.rating"))
 expr, err :=
 expression.NewBuilder().WithFilter(filtEx).WithProjection(projEx).Build()
 if err != nil {
  log.Printf("Couldn't build expressions for scan. Here's why: %v\n", err)
 } else {
  scanPaginator := dynamodb.NewScanPaginator(basics.DynamoDbClient,
 &dynamodb.ScanInput{
   TableName:                 aws.String(basics.TableName),
   ExpressionAttributeNames:  expr.Names(),
   ExpressionAttributeValues: expr.Values(),
   FilterExpression:          expr.Filter(),
   ProjectionExpression:      expr.Projection(),
  })
  for scanPaginator.HasMorePages() {
   response, err = scanPaginator.NextPage(ctx)
   if err != nil {
    log.Printf("Couldn't scan for movies released between %v and %v. Here's why:
 %v\n",
     startYear, endYear, err)
    break
   } else {
    var moviePage []Movie
    err = attributevalue.UnmarshalListOfMaps(response.Items, &moviePage)
    if err != nil {
     log.Printf("Couldn't unmarshal query response. Here's why: %v\n", err)
     break
    } else {
     movies = append(movies, moviePage...)
    }
   }
  }
 }
 return movies, err
}
```

Definieren Sie eine Movie-Struktur, die in diesem Beispiel verwendet wird.

```
import (
 "archive/zip"
 "bytes"
 "encoding/json"
 "fmt"
 "io"
 "log"
 "net/http"

 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/attributevalue"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb/types"
)

// Movie encapsulates data about a movie. Title and Year are the composite
 primary key
// of the movie in Amazon DynamoDB. Title is the sort key, Year is the partition
 key,
// and Info is additional data.
type Movie struct {
 Title string                 `dynamodbav:"title"`
 Year  int                    `dynamodbav:"year"`
 Info  map[string]interface{} `dynamodbav:"info"`
}

// GetKey returns the composite primary key of the movie in a format that can be
// sent to DynamoDB.
func (movie Movie) GetKey() map[string]types.AttributeValue {
 title, err := attributevalue.Marshal(movie.Title)
 if err != nil {
  panic(err)
 }
 year, err := attributevalue.Marshal(movie.Year)
 if err != nil {
  panic(err)
 }
 return map[string]types.AttributeValue{"title": title, "year": year}
}
```

```
// String returns the title, year, rating, and plot of a movie, formatted for the
 example.
func (movie Movie) String() string {
 return fmt.Sprintf("%v\n\tReleased: %v\n\tRating: %v\n\tPlot: %v\n",
  movie.Title, movie.Year, movie.Info["rating"], movie.Info["plot"])
}
```

- Weitere API-Informationen finden Sie unter Scan in der AWS SDK für Go -API-Referenz.

Java

SDK für Java 2.x

> **Note**
>
> Es gibt noch mehr GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

Scannt eine Amazon DynamoDB-Tabelle mit. DynamoDbClient

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.AttributeValue;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.model.ScanRequest;
import software.amazon.awssdk.services.dynamodb.model.ScanResponse;
import java.util.Map;
import java.util.Set;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 *
 * To scan items from an Amazon DynamoDB table using the AWS SDK for Java V2,
```

```
 * its better practice to use the
 * Enhanced Client, See the EnhancedScanRecords example.
 */

public class DynamoDBScanItems {
    public static void main(String[] args) {

        final String usage = """

                Usage:
                    <tableName>

                Where:
                    tableName - The Amazon DynamoDB table to get information from
 (for example, Music3).
                """;

        if (args.length != 1) {
            System.out.println(usage);
            System.exit(1);
        }

        String tableName = args[0];
        Region region = Region.US_EAST_1;
        DynamoDbClient ddb = DynamoDbClient.builder()
                .region(region)
                .build();

        scanItems(ddb, tableName);
        ddb.close();
    }

    public static void scanItems(DynamoDbClient ddb, String tableName) {
        try {
            ScanRequest scanRequest = ScanRequest.builder()
                    .tableName(tableName)
                    .build();

            ScanResponse response = ddb.scan(scanRequest);
            for (Map<String, AttributeValue> item : response.items()) {
                Set<String> keys = item.keySet();
                for (String key : keys) {
                    System.out.println("The key name is " + key + "\n");
                    System.out.println("The value is " + item.get(key).s());
```

```
                }
            }

        } catch (DynamoDbException e) {
            e.printStackTrace();
            System.exit(1);
        }
    }
}
```

- Weitere API-Informationen finden Sie unter Scan in der AWS SDK for Java 2.x -API-Referenz.

JavaScript

SDK für JavaScript (v3)

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

In diesem Beispiel wird der Dokument-Client verwendet, um die Arbeit mit Elementen in DynamoDB zu vereinfachen. Einzelheiten zur API finden Sie unter ScanCommand.

```
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";
import { DynamoDBDocumentClient, ScanCommand } from "@aws-sdk/lib-dynamodb";

const client = new DynamoDBClient({});
const docClient = DynamoDBDocumentClient.from(client);

export const main = async () => {
  const command = new ScanCommand({
    ProjectionExpression: "#Name, Color, AvgLifeSpan",
    ExpressionAttributeNames: { "#Name": "Name" },
    TableName: "Birds",
  });

  const response = await docClient.send(command);
```

```
  for (const bird of response.Items) {
    console.log(`${bird.Name} - (${bird.Color}, ${bird.AvgLifeSpan})`);
  }
  return response;
};
```

- Weitere API-Informationen finden Sie unter Scan in der AWS SDK für JavaScript -API-Referenz.

SDK für JavaScript (v2)

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
// Load the AWS SDK for Node.js.
var AWS = require("aws-sdk");
// Set the AWS Region.
AWS.config.update({ region: "REGION" });

// Create DynamoDB service object.
var ddb = new AWS.DynamoDB({ apiVersion: "2012-08-10" });

const params = {
  // Specify which items in the results are returned.
  FilterExpression: "Subtitle = :topic AND Season = :s AND Episode = :e",
  // Define the expression attribute value, which are substitutes for the values
 you want to compare.
  ExpressionAttributeValues: {
    ":topic": { S: "SubTitle2" },
    ":s": { N: 1 },
    ":e": { N: 2 },
  },
  // Set the projection expression, which are the attributes that you want.
  ProjectionExpression: "Season, Episode, Title, Subtitle",
  TableName: "EPISODES_TABLE",
};

ddb.scan(params, function (err, data) {
  if (err) {
```

```
    console.log("Error", err);
  } else {
    console.log("Success", data);
    data.Items.forEach(function (element, index, array) {
      console.log(
        "printing",
        element.Title.S + " (" + element.Subtitle.S + ")"
      );
    });
  }
});
```

- Weitere Informationen finden Sie im AWS SDK für JavaScript -Entwicklerhandbuch.
- Weitere API-Informationen finden Sie unter Scan in der AWS SDK für JavaScript -API-Referenz.

Kotlin

SDK für Kotlin

> **Note**
>
> Es gibt noch mehr GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
suspend fun scanItems(tableNameVal: String) {
    val request =
        ScanRequest {
            tableName = tableNameVal
        }

    DynamoDbClient.fromEnvironment { region = "us-east-1" }.use { ddb ->
        val response = ddb.scan(request)
        response.items?.forEach { item ->
            item.keys.forEach { key ->
                println("The key name is $key\n")
                println("The value is ${item[key]}")
            }
```

```
        }
    }
}
```

- Weitere API-Informationen finden Sie unter Scan in der API-Referenz zum AWS -SDK für Kotlin.

PHP

SDK für PHP

> **Note**
>
> Es gibt noch mehr GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
        $yearsKey = [
            'Key' => [
                'year' => [
                    'N' => [
                        'minRange' => 1990,
                        'maxRange' => 1999,
                    ],
                ],
            ],
        ];
        $filter = "year between 1990 and 1999";
        echo "\nHere's a list of all the movies released in the 90s:\n";
        $result = $service->scan($tableName, $yearsKey, $filter);
        foreach ($result['Items'] as $movie) {
            $movie = $marshal->unmarshalItem($movie);
            echo $movie['title'] . "\n";
        }

    public function scan(string $tableName, array $key, string $filters)
    {
        $query = [
            'ExpressionAttributeNames' => ['#year' => 'year'],
            'ExpressionAttributeValues' => [
```

```
                ":min" => ['N' => '1990'],
                ":max" => ['N' => '1999'],
            ],
            'FilterExpression' => "#year between :min and :max",
            'TableName' => $tableName,
        ];
        return $this->dynamoDbClient->scan($query);
    }
```

- Weitere API-Informationen finden Sie unter Scan in der AWS SDK für PHP -API-Referenz.

PowerShell

Tools für PowerShell V4

Beispiel 1: Gibt alle Elemente in der Tabelle „Music“ zurück.

```
Invoke-DDBScan -TableName 'Music' | ConvertFrom-DDBItem
```

Ausgabe:

```
Name                           Value
----                           -----
Genre                          Country
Artist                         No One You Know
Price                          1.94
CriticRating                   9
SongTitle                      Somewhere Down The Road
AlbumTitle                     Somewhat Famous
Genre                          Country
Artist                         No One You Know
Price                          1.98
CriticRating                   8.4
SongTitle                      My Dog Spot
AlbumTitle                     Hey Now
```

Beispiel 2: Gibt Elemente in der Tabelle Musik zurück, deren Wert CriticRating größer oder gleich neun ist.

```
$scanFilter = @{
        CriticRating = [Amazon.DynamoDBv2.Model.Condition]@{
```

```
            AttributeValueList = @(@{N = '9'})
            ComparisonOperator = 'GE'
        }
    }
    Invoke-DDBScan -TableName 'Music' -ScanFilter $scanFilter | ConvertFrom-
DDBItem
```

Ausgabe:

```
Name                           Value
----                           -----
Genre                          Country
Artist                         No One You Know
Price                          1.94
CriticRating                   9
SongTitle                      Somewhere Down The Road
AlbumTitle                     Somewhat Famous
```

- Weitere API-Informationen finden Sie unter Scan in der AWS -Tools für PowerShell -Cmdlet-Referenz (V4).

Tools für PowerShell V5

Beispiel 1: Gibt alle Elemente in der Tabelle „Music“ zurück.

```
Invoke-DDBScan -TableName 'Music' | ConvertFrom-DDBItem
```

Ausgabe:

```
Name                           Value
----                           -----
Genre                          Country
Artist                         No One You Know
Price                          1.94
CriticRating                   9
SongTitle                      Somewhere Down The Road
AlbumTitle                     Somewhat Famous
Genre                          Country
Artist                         No One You Know
Price                          1.98
CriticRating                   8.4
SongTitle                      My Dog Spot
AlbumTitle                     Hey Now
```

Beispiel 2: Gibt Elemente in der Tabelle Musik zurück, deren Wert CriticRating größer oder gleich neun ist.

```
$scanFilter = @{
        CriticRating = [Amazon.DynamoDBv2.Model.Condition]@{
            AttributeValueList = @(@{N = '9'})
            ComparisonOperator = 'GE'
        }
    }
    Invoke-DDBScan -TableName 'Music' -ScanFilter $scanFilter | ConvertFrom-
DDBItem
```

Ausgabe:

```
Name                           Value
----                           -----
Genre                          Country
Artist                         No One You Know
Price                          1.94
CriticRating                   9
SongTitle                      Somewhere Down The Road
AlbumTitle                     Somewhat Famous
```

- Weitere API-Informationen finden Sie unter Scan in der AWS -Tools für PowerShell -Cmdlet-Referenz (V5).

Python

SDK für Python (Boto3)

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
class Movies:
    """Encapsulates an Amazon DynamoDB table of movie data.

    Example data structure for a movie record in this table:
```

```
        {
            "year": 1999,
            "title": "For Love of the Game",
            "info": {
                "directors": ["Sam Raimi"],
                "release_date": "1999-09-15T00:00:00Z",
                "rating": 6.3,
                "plot": "A washed up pitcher flashes through his career.",
                "rank": 4987,
                "running_time_secs": 8220,
                "actors": [
                    "Kevin Costner",
                    "Kelly Preston",
                    "John C. Reilly"
                ]
            }
        }
    """

    def __init__(self, dyn_resource):
        """
        :param dyn_resource: A Boto3 DynamoDB resource.
        """
        self.dyn_resource = dyn_resource
        # The table variable is set during the scenario in the call to
        # 'exists' if the table exists. Otherwise, it is set by 'create_table'.
        self.table = None


    def scan_movies(self, year_range):
        """
        Scans for movies that were released in a range of years.
        Uses a projection expression to return a subset of data for each movie.

        :param year_range: The range of years to retrieve.
        :return: The list of movies released in the specified years.
        """
        movies = []
        scan_kwargs = {
            "FilterExpression": Key("year").between(
                year_range["first"], year_range["second"]
            ),
            "ProjectionExpression": "#yr, title, info.rating",
            "ExpressionAttributeNames": {"#yr": "year"},
```

```
        }
        try:
            done = False
            start_key = None
            while not done:
                if start_key:
                    scan_kwargs["ExclusiveStartKey"] = start_key
                response = self.table.scan(**scan_kwargs)
                movies.extend(response.get("Items", []))
                start_key = response.get("LastEvaluatedKey", None)
                done = start_key is None
        except ClientError as err:
            logger.error(
                "Couldn't scan for movies. Here's why: %s: %s",
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise

        return movies
```

- Weitere API-Informationen finden Sie unter Scan in der API-Referenz zum AWS SDK für Python (Boto3).

Ruby

SDK für Ruby

> **Note**
>
> Es gibt noch mehr GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
class DynamoDBBasics
  attr_reader :dynamo_resource, :table

  def initialize(table_name)
    client = Aws::DynamoDB::Client.new(region: 'us-east-1')
```

```
    @dynamo_resource = Aws::DynamoDB::Resource.new(client: client)
    @table = @dynamo_resource.table(table_name)
  end

  # Scans for movies that were released in a range of years.
  # Uses a projection expression to return a subset of data for each movie.
  #
  # @param year_range [Hash] The range of years to retrieve.
  # @return [Array] The list of movies released in the specified years.
  def scan_items(year_range)
    movies = []
    scan_hash = {
      filter_expression: '#yr between :start_yr and :end_yr',
      projection_expression: '#yr, title, info.rating',
      expression_attribute_names: { '#yr' => 'year' },
      expression_attribute_values: {
        ':start_yr' => year_range[:start], ':end_yr' => year_range[:end]
      }
    }
    done = false
    start_key = nil
    until done
      scan_hash[:exclusive_start_key] = start_key unless start_key.nil?
      response = @table.scan(scan_hash)
      movies.concat(response.items) unless response.items.empty?
      start_key = response.last_evaluated_key
      done = start_key.nil?
    end
  rescue Aws::DynamoDB::Errors::ServiceError => e
    puts("Couldn't scan for movies. Here's why:")
    puts("\t#{e.code}: #{e.message}")
    raise
  else
    movies
  end
```

- Weitere API-Informationen finden Sie unter Scan in der AWS SDK für Ruby -API-Referenz.

Rust

SDK für Rust

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
pub async fn list_items(client: &Client, table: &str, page_size: Option<i32>) ->
 Result<(), Error> {
    let page_size = page_size.unwrap_or(10);
    let items: Result<Vec<_>, _> = client
        .scan()
        .table_name(table)
        .limit(page_size)
        .into_paginator()
        .items()
        .send()
        .collect()
        .await;

    println!("Items in table (up to {page_size}):");
    for item in items? {
        println!("   {:?}", item);
    }

    Ok(())
}
```

- Weitere API-Informationen finden Sie unter Scan in der API-Referenz zum AWS -SDK für Rust.

SAP ABAP

SDK für SAP ABAP

> **Note**
>
> Es gibt noch mehr dazu. GitHub Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
    TRY.
        " Scan movies for rating greater than or equal to the rating specified
        DATA(lt_attributelist) = VALUE /aws1/
cl_dynattributevalue=>tt_attributevaluelist(
            ( NEW /aws1/cl_dynattributevalue( iv_n = |{ iv_rating }| ) ) ).
        DATA(lt_filter_conditions) = VALUE /aws1/
cl_dyncondition=>tt_filterconditionmap(
          ( VALUE /aws1/cl_dyncondition=>ts_filterconditionmap_maprow(
          key = 'rating'
          value = NEW /aws1/cl_dyncondition(
          it_attributevaluelist = lt_attributelist
          iv_comparisonoperator = |GE|
          ) ) ) ).
        oo_scan_result = lo_dyn->scan( iv_tablename = iv_table_name
          it_scanfilter = lt_filter_conditions ).
        DATA(lt_items) = oo_scan_result->get_items( ).
        LOOP AT lt_items INTO DATA(lo_item).
          " You can loop over to get individual attributes.
          DATA(lo_title) = lo_item[ key = 'title' ]-value.
          DATA(lo_year) = lo_item[ key = 'year' ]-value.
        ENDLOOP.
        DATA(lv_count) = oo_scan_result->get_count( ).
        MESSAGE 'Found ' && lv_count && ' items' TYPE 'I'.
      CATCH /aws1/cx_dynresourcenotfoundex.
        MESSAGE 'The table or index does not exist' TYPE 'E'.
    ENDTRY.
```

- Weitere API-Informationen finden Sie unter Scan in der API-Referenz für das AWS -SDK für SAP ABAP.

Swift

SDK für Swift

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
import AWSDynamoDB


    /// Return an array of `Movie` objects released in the specified range of
    /// years.
    ///
    /// - Parameters:
    ///   - firstYear: The first year of movies to return.
    ///   - lastYear: The last year of movies to return.
    ///   - startKey: A starting point to resume processing; always use `nil`.
    ///
    /// - Returns: An array of `Movie` objects describing the matching movies.
    ///
    /// > Note: The `startKey` parameter is used by this function when
    ///   recursively calling itself, and should always be `nil` when calling
    ///   directly.
    ///
    func getMovies(firstYear: Int, lastYear: Int,
                   startKey: [Swift.String: DynamoDBClientTypes.AttributeValue]?
 = nil)
        async throws -> [Movie]
    {
        do {
            var movieList: [Movie] = []

            guard let client = self.ddbClient else {
                throw MoviesError.UninitializedClient
            }

            let input = ScanInput(
                consistentRead: true,
                exclusiveStartKey: startKey,
```

```
                expressionAttributeNames: [
                    "#y": "year" // `year` is a reserved word, so use `#y`
instead.
                ],
                expressionAttributeValues: [
                    ":y1": .n(String(firstYear)),
                    ":y2": .n(String(lastYear))
                ],
                filterExpression: "#y BETWEEN :y1 AND :y2",
                tableName: self.tableName
            )

            let pages = client.scanPaginated(input: input)

            for try await page in pages {
                guard let items = page.items else {
                    print("Error: no items returned.")
                    continue
                }

                // Build an array of `Movie` objects for the returned items.

                for item in items {
                    let movie = try Movie(withItem: item)
                    movieList.append(movie)
                }
            }
            return movieList

        } catch {
            print("ERROR: getMovies with scan:", dump(error))
            throw error
        }
    }
```

- Detaillierte API-Informationen finden Sie unter Scan in der API-Referenz zum AWS -SDK für Swift.

Eine vollständige Liste der AWS SDK-Entwicklerhandbücher und Codebeispiele finden Sie unterDynamoDB mit einem SDK verwenden AWS. Dieses Thema enthält auch Informationen zu den ersten Schritten und Details zu früheren SDK-Versionen.

Verwendung **`UpdateItem`** mit einem AWS SDK oder CLI

Die folgenden Code-Beispiele zeigen, wie `UpdateItem` verwendet wird.

Aktionsbeispiele sind Codeauszüge aus größeren Programmen und müssen im Kontext ausgeführt werden. Sie können diese Aktion in den folgenden Codebeispielen im Kontext sehen:

- Kennenlernen der Grundlagen
- Bedingtes Aktualisieren der TTL eines Elements
- Zählen von Ausdrucksoperatoren
- Erstellen und Verwalten von globalen MRSC-Tabellen
- Ausführen von Listenoperationen
- Ausführen von Map-Operationen
- Ausführen von Set-Operationen
- Verstehen der Reihenfolge der Aktualisierungsausdrücke
- Aktualisieren der TTL eines Elements
- Verwenden von Operationen mit unteilbaren Zählern
- Verwenden von bedingten Operationen
- Verwenden von Ausdrucksattributnamen

.NET

SDK für .NET (v4)

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
/// <summary>
```

```
    /// Updates an existing item in the movies table.
    /// </summary>
    /// <param name="newMovie">A Movie object containing information for
    /// the movie to update.</param>
    /// <param name="newInfo">A MovieInfo object that contains the
    /// information that will be changed.</param>
    /// <param name="tableName">The name of the table that contains the movie.</
param>
    /// <returns>A Boolean value that indicates the success of the operation.</
returns>
    public async Task<bool> UpdateItemAsync(
        Movie newMovie,
        MovieInfo newInfo,
        string tableName)
    {
        try
        {
            var key = new Dictionary<string, AttributeValue>
            {
                ["title"] = new AttributeValue { S = newMovie.Title },
                ["year"] = new AttributeValue { N = newMovie.Year.ToString() },
            };
            var updates = new Dictionary<string, AttributeValueUpdate>
            {
                ["info.plot"] = new AttributeValueUpdate
                {
                    Action = AttributeAction.PUT,
                    Value = new AttributeValue { S = newInfo.Plot },
                },

                ["info.rating"] = new AttributeValueUpdate
                {
                    Action = AttributeAction.PUT,
                    Value = new AttributeValue { N = newInfo.Rank.ToString() },
                },
            };

            var request = new UpdateItemRequest
            {
                AttributeUpdates = updates,
                Key = key,
                TableName = tableName,
            };
```

```
            await _amazonDynamoDB.UpdateItemAsync(request);
            return true;
        }
        catch (ResourceNotFoundException ex)
        {
            Console.WriteLine($"Table {tableName} or item was not found.
{ex.Message}");
            return false;
        }
        catch (AmazonDynamoDBException ex)
        {
            Console.WriteLine($"An Amazon DynamoDB error occurred while updating
item. {ex.Message}");
            throw;
        }
        catch (Exception ex)
        {
            Console.WriteLine($"An error occurred while updating item.
{ex.Message}");
            throw;
        }
    }
```

- Einzelheiten zur API finden Sie UpdateItemin der AWS SDK für .NET API-Referenz.

Bash

AWS CLI mit Bash-Skript

> **Note**
>
> Es gibt noch mehr dazu. GitHub Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
###############################################################################
# function dynamodb_update_item
#
# This function updates an item in a DynamoDB table.
#
```

```
#
# Parameters:
#       -n table_name  -- The name of the table.
#       -k keys  -- Path to json file containing the keys that identify the item
 to update.
#       -e update expression  -- An expression that defines one or more
 attributes to be updated.
#       -v values  -- Path to json file containing the update values.
#
#  Returns:
#       0 - If successful.
#       1 - If it fails.
###############################################################################
function dynamodb_update_item() {
  local table_name keys update_expression values response
  local option OPTARG # Required to use getopts command in a function.

  ######################################
  # Function usage explanation
  ######################################
  function usage() {
    echo "function dynamodb_update_item"
    echo "Update an item in a DynamoDB table."
    echo " -n table_name  -- The name of the table."
    echo " -k keys  -- Path to json file containing the keys that identify the
 item to update."
    echo " -e update expression  -- An expression that defines one or more
 attributes to be updated."
    echo " -v values  -- Path to json file containing the update values."
    echo ""
  }

  while getopts "n:k:e:v:h" option; do
    case "${option}" in
      n) table_name="${OPTARG}" ;;
      k) keys="${OPTARG}" ;;
      e) update_expression="${OPTARG}" ;;
      v) values="${OPTARG}" ;;
      h)
        usage
        return 0
        ;;
      \?)
        echo "Invalid parameter"
```

```
      usage
      return 1
      ;;
  esac
done
export OPTIND=1

if [[ -z "$table_name" ]]; then
  errecho "ERROR: You must provide a table name with the -n parameter."
  usage
  return 1
fi

if [[ -z "$keys" ]]; then
  errecho "ERROR: You must provide a keys json file path the -k parameter."
  usage
  return 1
fi
if [[ -z "$update_expression" ]]; then
  errecho "ERROR: You must provide an update expression with the -e parameter."
  usage
  return 1
fi

if [[ -z "$values" ]]; then
  errecho "ERROR: You must provide a values json file path the -v parameter."
  usage
  return 1
fi

iecho "Parameters:\n"
iecho "    table_name:   $table_name"
iecho "    keys:   $keys"
iecho "    update_expression:   $update_expression"
iecho "    values:   $values"

response=$(aws dynamodb update-item \
  --table-name "$table_name" \
  --key file://"$keys" \
  --update-expression "$update_expression" \
  --expression-attribute-values file://"$values")

local error_code=${?}
```

```
  if [[ $error_code -ne 0 ]]; then
    aws_cli_error_log $error_code
    errecho "ERROR: AWS reports update-item operation failed.$response"
    return 1
  fi

  return 0

}
```

Die in diesem Beispiel verwendeten Hilfsfunktionen.

```
###############################################################################
# function iecho
#
# This function enables the script to display the specified text only if
# the global variable $VERBOSE is set to true.
###############################################################################
function iecho() {
  if [[ $VERBOSE == true ]]; then
    echo "$@"
  fi
}

###############################################################################
# function errecho
#
# This function outputs everything sent to it to STDERR (standard error output).
###############################################################################
function errecho() {
  printf "%s\n" "$*" 1>&2
}

###############################################################################
# function aws_cli_error_log()
#
# This function is used to log the error messages from the AWS CLI.
#
# See https://docs.aws.amazon.com/cli/latest/topic/return-codes.html#cli-aws-
help-return-codes.
#
# The function expects the following argument:
```

```
#         $1 - The error code returned by the AWS CLI.
#
#  Returns:
#          0: - Success.
#
###############################################################################
function aws_cli_error_log() {
  local err_code=$1
  errecho "Error code : $err_code"
  if [ "$err_code" == 1 ]; then
    errecho "  One or more S3 transfers failed."
  elif [ "$err_code" == 2 ]; then
    errecho "  Command line failed to parse."
  elif [ "$err_code" == 130 ]; then
    errecho "  Process received SIGINT."
  elif [ "$err_code" == 252 ]; then
    errecho "  Command syntax invalid."
  elif [ "$err_code" == 253 ]; then
    errecho "  The system environment or configuration was invalid."
  elif [ "$err_code" == 254 ]; then
    errecho "  The service returned an error."
  elif [ "$err_code" == 255 ]; then
    errecho "  255 is a catch-all error."
  fi

  return 0
}
```

- Einzelheiten zur API finden Sie UpdateItemin der AWS CLI Befehlsreferenz.

C++

SDK für C++

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
//! Update an Amazon DynamoDB table item.
```

```
/*!
  \sa updateItem()
  \param tableName: The table name.
  \param partitionKey: The partition key.
  \param partitionValue: The value for the partition key.
  \param attributeKey: The key for the attribute to be updated.
  \param attributeValue: The value for the attribute to be updated.
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
  */

/*
 *  The example code only sets/updates an attribute value. It processes
 *  the attribute value as a string, even if the value could be interpreted
 *  as a number. Also, the example code does not remove an existing attribute
 *  from the key value.
 */

bool AwsDoc::DynamoDB::updateItem(const Aws::String &tableName,
                                  const Aws::String &partitionKey,
                                  const Aws::String &partitionValue,
                                  const Aws::String &attributeKey,
                                  const Aws::String &attributeValue,
                                  const Aws::Client::ClientConfiguration
 &clientConfiguration) {
    Aws::DynamoDB::DynamoDBClient dynamoClient(clientConfiguration);

    // *** Define UpdateItem request arguments.
    // Define TableName argument.
    Aws::DynamoDB::Model::UpdateItemRequest request;
    request.SetTableName(tableName);

    // Define KeyName argument.
    Aws::DynamoDB::Model::AttributeValue attribValue;
    attribValue.SetS(partitionValue);
    request.AddKey(partitionKey, attribValue);

    // Construct the SET update expression argument.
    Aws::String update_expression("SET #a = :valueA");
    request.SetUpdateExpression(update_expression);

    // Construct attribute name argument.
    Aws::Map<Aws::String, Aws::String> expressionAttributeNames;
    expressionAttributeNames["#a"] = attributeKey;
```

```
    request.SetExpressionAttributeNames(expressionAttributeNames);

    // Construct attribute value argument.
    Aws::DynamoDB::Model::AttributeValue attributeUpdatedValue;
    attributeUpdatedValue.SetS(attributeValue);
    Aws::Map<Aws::String, Aws::DynamoDB::Model::AttributeValue>
 expressionAttributeValues;
    expressionAttributeValues[":valueA"] = attributeUpdatedValue;
    request.SetExpressionAttributeValues(expressionAttributeValues);

    // Update the item.
    const Aws::DynamoDB::Model::UpdateItemOutcome &outcome =
 dynamoClient.UpdateItem(
            request);
    if (outcome.IsSuccess()) {
        std::cout << "Item was updated" << std::endl;
    } else {
        std::cerr << outcome.GetError().GetMessage() << std::endl;
        return false;
    }

    return waitTableActive(tableName, dynamoClient);
}
```

Code, der darauf wartet, dass die Tabelle aktiv wird.

```
//! Query a newly created DynamoDB table until it is active.
/*!
  \sa waitTableActive()
  \param waitTableActive: The DynamoDB table's name.
  \param dynamoClient: A DynamoDB client.
  \return bool: Function succeeded.
*/
bool AwsDoc::DynamoDB::waitTableActive(const Aws::String &tableName,
                                       const Aws::DynamoDB::DynamoDBClient
 &dynamoClient) {

    // Repeatedly call DescribeTable until table is ACTIVE.
    const int MAX_QUERIES = 20;
    Aws::DynamoDB::Model::DescribeTableRequest request;
    request.SetTableName(tableName);
```

```
    int count = 0;
    while (count < MAX_QUERIES) {
        const Aws::DynamoDB::Model::DescribeTableOutcome &result =
 dynamoClient.DescribeTable(
                request);
        if (result.IsSuccess()) {
            Aws::DynamoDB::Model::TableStatus status =
 result.GetResult().GetTable().GetTableStatus();

            if (Aws::DynamoDB::Model::TableStatus::ACTIVE != status) {
                std::this_thread::sleep_for(std::chrono::seconds(1));
            }
            else {
                return true;
            }
        }
        else {
            std::cerr << "Error DynamoDB::waitTableActive "
                      << result.GetError().GetMessage() << std::endl;
            return false;
        }
        count++;
    }
    return false;
}
```

- Einzelheiten zur API finden Sie UpdateItemin der AWS SDK für C++ API-Referenz.

CLI

AWS CLI

Beispiel 1: So aktualisieren Sie ein Element in einer Tabelle

Im folgenden Beispiel für `update-item` wird ein Element in der Tabelle `MusicCollection` aktualisiert. Es wird ein neues Attribut (`Year`) hinzugefügt und das `AlbumTitle`-Attribut geändert. Alle Attribute im Element, wie sie nach dem Update erscheinen, werden in der Antwort zurückgegeben.

```
aws dynamodb update-item \
    --table-name MusicCollection \
```

```
    --key file://key.json \
    --update-expression "SET #Y = :y, #AT = :t" \
    --expression-attribute-names file://expression-attribute-names.json \
    --expression-attribute-values file://expression-attribute-values.json  \
    --return-values ALL_NEW \
    --return-consumed-capacity TOTAL \
    --return-item-collection-metrics SIZE
```

Inhalt von `key.json`:

```
{
    "Artist": {"S": "Acme Band"},
    "SongTitle": {"S": "Happy Day"}
}
```

Inhalt von `expression-attribute-names.json`:

```
{
    "#Y":"Year", "#AT":"AlbumTitle"
}
```

Inhalt von `expression-attribute-values.json`:

```
{
    ":y":{"N": "2015"},
    ":t":{"S": "Louder Than Ever"}
}
```

Ausgabe:

```
{
    "Attributes": {
        "AlbumTitle": {
            "S": "Louder Than Ever"
        },
        "Awards": {
            "N": "10"
        },
        "Artist": {
            "S": "Acme Band"
        },
        "Year": {
```

```
            "N": "2015"
        },
        "SongTitle": {
            "S": "Happy Day"
        }
    },
    "ConsumedCapacity": {
        "TableName": "MusicCollection",
        "CapacityUnits": 3.0
    },
    "ItemCollectionMetrics": {
        "ItemCollectionKey": {
            "Artist": {
                "S": "Acme Band"
            }
        },
        "SizeEstimateRangeGB": [
            0.0,
            1.0
        ]
    }
}
```

Weitere Informationen finden Sie unter Schreiben eines Elements im Entwicklerhandbuch zu Amazon DynamoDB.

Beispiel 2: So aktualisieren Sie ein Element bedingungsabhängig

Im folgenden Beispiel wird ein Element in der `MusicCollection`-Tabelle aktualisiert, jedoch nur, wenn das vorhandene Element noch kein `Year`-Attribut besitzt.

```
aws dynamodb update-item \
    --table-name MusicCollection \
    --key file://key.json \
    --update-expression "SET #Y = :y, #AT = :t" \
    --expression-attribute-names file://expression-attribute-names.json \
    --expression-attribute-values file://expression-attribute-values.json  \
    --condition-expression "attribute_not_exists(#Y)"
```

Inhalt von `key.json`:

```
{
    "Artist": {"S": "Acme Band"},
```

```
    "SongTitle": {"S": "Happy Day"}
}
```

Inhalt von `expression-attribute-names.json`:

```
{
    "#Y":"Year",
    "#AT":"AlbumTitle"
}
```

Inhalt von `expression-attribute-values.json`:

```
{
    ":y":{"N": "2015"},
    ":t":{"S": "Louder Than Ever"}
}
```

Wenn das Element bereits über ein `Year`-Attribut verfügt, gibt DynamoDB die folgende Ausgabe zurück.

```
An error occurred (ConditionalCheckFailedException) when calling the UpdateItem
 operation: The conditional request failed
```

Weitere Informationen finden Sie unter Schreiben eines Elements im Entwicklerhandbuch zu Amazon DynamoDB.

- Einzelheiten zur API finden Sie UpdateItemin der AWS CLI Befehlsreferenz.

Go

SDK für Go V2

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
import (
```

```
 "context"
 "errors"
 "log"
 "time"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/attributevalue"
 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/expression"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb/types"
)

// TableBasics encapsulates the Amazon DynamoDB service actions used in the
 examples.
// It contains a DynamoDB service client that is used to act on the specified
 table.
type TableBasics struct {
 DynamoDbClient *dynamodb.Client
 TableName      string
}



// UpdateMovie updates the rating and plot of a movie that already exists in the
// DynamoDB table. This function uses the `expression` package to build the
 update
// expression.
func (basics TableBasics) UpdateMovie(ctx context.Context, movie Movie)
 (map[string]map[string]interface{}, error) {
 var err error
 var response *dynamodb.UpdateItemOutput
 var attributeMap map[string]map[string]interface{}
 update := expression.Set(expression.Name("info.rating"),
 expression.Value(movie.Info["rating"]))
 update.Set(expression.Name("info.plot"), expression.Value(movie.Info["plot"]))
 expr, err := expression.NewBuilder().WithUpdate(update).Build()
 if err != nil {
  log.Printf("Couldn't build expression for update. Here's why: %v\n", err)
 } else {
  response, err = basics.DynamoDbClient.UpdateItem(ctx,
 &dynamodb.UpdateItemInput{
   TableName:                 aws.String(basics.TableName),
   Key:                       movie.GetKey(),
   ExpressionAttributeNames:  expr.Names(),
```

```
    ExpressionAttributeValues: expr.Values(),
    UpdateExpression:          expr.Update(),
    ReturnValues:              types.ReturnValueUpdatedNew,
   })
   if err != nil {
    log.Printf("Couldn't update movie %v. Here's why: %v\n", movie.Title, err)
   } else {
    err = attributevalue.UnmarshalMap(response.Attributes, &attributeMap)
    if err != nil {
     log.Printf("Couldn't unmarshall update response. Here's why: %v\n", err)
    }
   }
  }
  return attributeMap, err
}
```

Definieren Sie eine Movie-Struktur, die in diesem Beispiel verwendet wird.

```
import (
 "archive/zip"
 "bytes"
 "encoding/json"
 "fmt"
 "io"
 "log"
 "net/http"

 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/attributevalue"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb/types"
)

// Movie encapsulates data about a movie. Title and Year are the composite
 primary key
// of the movie in Amazon DynamoDB. Title is the sort key, Year is the partition
 key,
// and Info is additional data.
type Movie struct {
 Title string                 `dynamodbav:"title"`
 Year  int                    `dynamodbav:"year"`
 Info  map[string]interface{} `dynamodbav:"info"`
```

```
}

// GetKey returns the composite primary key of the movie in a format that can be
// sent to DynamoDB.
func (movie Movie) GetKey() map[string]types.AttributeValue {
 title, err := attributevalue.Marshal(movie.Title)
 if err != nil {
  panic(err)
 }
 year, err := attributevalue.Marshal(movie.Year)
 if err != nil {
  panic(err)
 }
 return map[string]types.AttributeValue{"title": title, "year": year}
}

// String returns the title, year, rating, and plot of a movie, formatted for the
 example.
func (movie Movie) String() string {
 return fmt.Sprintf("%v\n\tReleased: %v\n\tRating: %v\n\tPlot: %v\n",
  movie.Title, movie.Year, movie.Info["rating"], movie.Info["plot"])
}
```

- Einzelheiten zur API finden Sie UpdateItemin der AWS SDK für Go API-Referenz.

Java

SDK für Java 2.x

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

Aktualisiert ein Element in einer Tabelle mit DynamoDbClient.

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.model.AttributeAction;
```

```
import software.amazon.awssdk.services.dynamodb.model.AttributeValue;
import software.amazon.awssdk.services.dynamodb.model.AttributeValueUpdate;
import software.amazon.awssdk.services.dynamodb.model.UpdateItemRequest;
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import java.util.HashMap;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 *
 * To update an Amazon DynamoDB table using the AWS SDK for Java V2, its better
 * practice to use the
 * Enhanced Client, See the EnhancedModifyItem example.
 */
public class UpdateItem {
    public static void main(String[] args) {
        final String usage = """

                Usage:
                    <tableName> <key> <keyVal> <name> <updateVal>

                Where:
                    tableName - The Amazon DynamoDB table (for example, Music3).
                    key - The name of the key in the table (for example, Artist).
                    keyVal - The value of the key (for example, Famous Band).
                    name - The name of the column where the value is updated (for
 example, Awards).
                    updateVal - The value used to update an item (for example,
 14).
                 Example:
                    UpdateItem Music3 Artist Famous Band Awards 14
                """;

        if (args.length != 5) {
            System.out.println(usage);
            System.exit(1);
        }

        String tableName = args[0];
```

```
        String key = args[1];
        String keyVal = args[2];
        String name = args[3];
        String updateVal = args[4];

        Region region = Region.US_EAST_1;
        DynamoDbClient ddb = DynamoDbClient.builder()
                .region(region)
                .build();
        updateTableItem(ddb, tableName, key, keyVal, name, updateVal);
        ddb.close();
    }

    public static void updateTableItem(DynamoDbClient ddb,
            String tableName,
            String key,
            String keyVal,
            String name,
            String updateVal) {

        HashMap<String, AttributeValue> itemKey = new HashMap<>();
        itemKey.put(key, AttributeValue.builder()
                .s(keyVal)
                .build());

        HashMap<String, AttributeValueUpdate> updatedValues = new HashMap<>();
        updatedValues.put(name, AttributeValueUpdate.builder()
                .value(AttributeValue.builder().s(updateVal).build())
                .action(AttributeAction.PUT)
                .build());

        UpdateItemRequest request = UpdateItemRequest.builder()
                .tableName(tableName)
                .key(itemKey)
                .attributeUpdates(updatedValues)
                .build();

        try {
            ddb.updateItem(request);
        } catch (DynamoDbException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
        System.out.println("The Amazon DynamoDB table was updated!");
```

```
    }
}
```

- Einzelheiten zur API finden Sie UpdateItemunter AWS SDK for Java 2.x API-Referenz.

JavaScript

SDK für JavaScript (v3)

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

In diesem Beispiel wird der Dokument-Client verwendet, um die Arbeit mit Elementen in DynamoDB zu vereinfachen. Einzelheiten zur API finden Sie unter UpdateCommand.

```
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";
import { DynamoDBDocumentClient, UpdateCommand } from "@aws-sdk/lib-dynamodb";

const client = new DynamoDBClient({});
const docClient = DynamoDBDocumentClient.from(client);

export const main = async () => {
  const command = new UpdateCommand({
    TableName: "Dogs",
    Key: {
      Breed: "Labrador",
    },
    UpdateExpression: "set Color = :color",
    ExpressionAttributeValues: {
      ":color": "black",
    },
    ReturnValues: "ALL_NEW",
  });

  const response = await docClient.send(command);
  console.log(response);
  return response;
```

```
};
```

- Einzelheiten zur API finden Sie UpdateItemin der AWS SDK für JavaScript API-Referenz.

Kotlin

SDK für Kotlin

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
suspend fun updateTableItem(
    tableNameVal: String,
    keyName: String,
    keyVal: String,
    name: String,
    updateVal: String,
) {
    val itemKey = mutableMapOf<String, AttributeValue>()
    itemKey[keyName] = AttributeValue.S(keyVal)

    val updatedValues = mutableMapOf<String, AttributeValueUpdate>()
    updatedValues[name] =
        AttributeValueUpdate {
            value = AttributeValue.S(updateVal)
            action = AttributeAction.Put
        }

    val request =
        UpdateItemRequest {
            tableName = tableNameVal
            key = itemKey
            attributeUpdates = updatedValues
        }

    DynamoDbClient.fromEnvironment { region = "us-east-1" }.use { ddb ->
        ddb.updateItem(request)
        println("Item in $tableNameVal was updated")
```

```
    }
}
```

- Einzelheiten zur API finden Sie UpdateItemin der API-Referenz zum AWS SDK für Kotlin.

PHP

SDK für PHP

> **Note**
>
> Es gibt noch mehr dazu. GitHub Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
        echo "What rating would you like to give {$movie['Item']['title']['S']}?
\n";
        $rating = 0;
        while (!is_numeric($rating) || intval($rating) != $rating || $rating < 1
 || $rating > 10) {
            $rating = testable_readline("Rating (1-10): ");
        }
        $service->updateItemAttributeByKey($tableName, $key, 'rating', 'N',
 $rating);

    public function updateItemAttributeByKey(
        string $tableName,
        array $key,
        string $attributeName,
        string $attributeType,
        string $newValue
    ) {
        $this->dynamoDbClient->updateItem([
            'Key' => $key['Item'],
            'TableName' => $tableName,
            'UpdateExpression' => "set #NV=:NV",
            'ExpressionAttributeNames' => [
                '#NV' => $attributeName,
            ],
            'ExpressionAttributeValues' => [
                ':NV' => [
```

```
                    $attributeType => $newValue
                ]
            ],
        ]);
    }
```

- Einzelheiten zur API finden Sie UpdateItemin der AWS SDK für PHP API-Referenz.

PowerShell

Tools für PowerShell V4

Beispiel 1: Setzt das Genre-Attribut auf 'Rap' für das DynamoDB-Element mit dem Partitionsschlüssel SongTitle und dem Sortierschlüssel Artist.

```
$key = @{
    SongTitle = 'Somewhere Down The Road'
    Artist = 'No One You Know'
} | ConvertTo-DDBItem

$updateDdbItem = @{
    TableName = 'Music'
    Key = $key
    UpdateExpression = 'set Genre = :val1'
    ExpressionAttributeValue = (@{
        ':val1' = ([Amazon.DynamoDBv2.Model.AttributeValue]'Rap')
    })
}
Update-DDBItem @updateDdbItem
```

Ausgabe:

```
Name                           Value
----                           -----
Genre                          Rap
```

- Einzelheiten zur API finden Sie unter UpdateItem AWS -Tools für PowerShellCmdlet-Referenz (V4).

Tools für V5 PowerShell

Beispiel 1: Setzt das Genre-Attribut auf 'Rap' für das DynamoDB-Element mit dem Partitionsschlüssel SongTitle und dem Sortierschlüssel Artist.

```
$key = @{
    SongTitle = 'Somewhere Down The Road'
    Artist = 'No One You Know'
} | ConvertTo-DDBItem

$updateDdbItem = @{
    TableName = 'Music'
    Key = $key
    UpdateExpression = 'set Genre = :val1'
    ExpressionAttributeValue = (@{
        ':val1' = ([Amazon.DynamoDBv2.Model.AttributeValue]'Rap')
    })
}
Update-DDBItem @updateDdbItem
```

Ausgabe:

```
Name                           Value
----                           -----
Genre                          Rap
```

- Einzelheiten zur API finden Sie unter UpdateItem AWS -Tools für PowerShellCmdlet-Referenz (V5).

Python

SDK für Python (Boto3)

> **Note**
>
> Es gibt noch mehr dazu. GitHub Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

Aktualisieren Sie ein Element mithilfe eines Aktualisierungsausdrucks.

```
class Movies:
    """Encapsulates an Amazon DynamoDB table of movie data.

    Example data structure for a movie record in this table:
        {
            "year": 1999,
            "title": "For Love of the Game",
            "info": {
                "directors": ["Sam Raimi"],
                "release_date": "1999-09-15T00:00:00Z",
                "rating": 6.3,
                "plot": "A washed up pitcher flashes through his career.",
                "rank": 4987,
                "running_time_secs": 8220,
                "actors": [
                    "Kevin Costner",
                    "Kelly Preston",
                    "John C. Reilly"
                ]
            }
        }
    """

    def __init__(self, dyn_resource):
        """
        :param dyn_resource: A Boto3 DynamoDB resource.
        """
        self.dyn_resource = dyn_resource
        # The table variable is set during the scenario in the call to
        # 'exists' if the table exists. Otherwise, it is set by 'create_table'.
        self.table = None


    def update_movie(self, title, year, rating, plot):
        """
        Updates rating and plot data for a movie in the table.

        :param title: The title of the movie to update.
        :param year: The release year of the movie to update.
        :param rating: The updated rating to the give the movie.
        :param plot: The updated plot summary to give the movie.
        :return: The fields that were updated, with their new values.
        """
```

```
        try:
            response = self.table.update_item(
                Key={"year": year, "title": title},
                UpdateExpression="set info.rating=:r, info.plot=:p",
                ExpressionAttributeValues={":r": Decimal(str(rating)), ":p":
 plot},
                ReturnValues="UPDATED_NEW",
            )
        except ClientError as err:
            logger.error(
                "Couldn't update movie %s in table %s. Here's why: %s: %s",
                title,
                self.table.name,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return response["Attributes"]
```

Aktualisieren Sie ein Element mithilfe eines Aktualisierungsausdrucks, der eine arithmetische Operation enthält.

```
class UpdateQueryWrapper:
    def __init__(self, table):
        self.table = table


    def update_rating(self, title, year, rating_change):
        """
        Updates the quality rating of a movie in the table by using an arithmetic
        operation in the update expression. By specifying an arithmetic
 operation,
        you can adjust a value in a single request, rather than first getting its
        value and then setting its new value.

        :param title: The title of the movie to update.
        :param year: The release year of the movie to update.
        :param rating_change: The amount to add to the current rating for the
 movie.
        :return: The updated rating.
```

```
        """
        try:
            response = self.table.update_item(
                Key={"year": year, "title": title},
                UpdateExpression="set info.rating = info.rating + :val",
                ExpressionAttributeValues={":val": Decimal(str(rating_change))},
                ReturnValues="UPDATED_NEW",
            )
        except ClientError as err:
            logger.error(
                "Couldn't update movie %s in table %s. Here's why: %s: %s",
                title,
                self.table.name,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return response["Attributes"]
```

Aktualisieren Sie ein Element nur, wenn es bestimmte Bedingungen erfüllt.

```
class UpdateQueryWrapper:
    def __init__(self, table):
        self.table = table


    def remove_actors(self, title, year, actor_threshold):
        """
        Removes an actor from a movie, but only when the number of actors is
 greater
        than a specified threshold. If the movie does not list more than the
 threshold,
        no actors are removed.

        :param title: The title of the movie to update.
        :param year: The release year of the movie to update.
        :param actor_threshold: The threshold of actors to check.
        :return: The movie data after the update.
        """
        try:
```

```
            response = self.table.update_item(
                Key={"year": year, "title": title},
                UpdateExpression="remove info.actors[0]",
                ConditionExpression="size(info.actors) > :num",
                ExpressionAttributeValues={":num": actor_threshold},
                ReturnValues="ALL_NEW",
            )
        except ClientError as err:
            if err.response["Error"]["Code"] ==
 "ConditionalCheckFailedException":
                logger.warning(
                    "Didn't update %s because it has fewer than %s actors.",
                    title,
                    actor_threshold + 1,
                )
            else:
                logger.error(
                    "Couldn't update movie %s. Here's why: %s: %s",
                    title,
                    err.response["Error"]["Code"],
                    err.response["Error"]["Message"],
                )
            raise
        else:
            return response["Attributes"]
```

- Einzelheiten zur API finden Sie UpdateItemin AWS SDK for Python (Boto3) API Reference.

Ruby

SDK für Ruby

> **Note**
>
> Es gibt noch mehr dazu. GitHub Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
class DynamoDBBasics
  attr_reader :dynamo_resource, :table
```

```
  def initialize(table_name)
    client = Aws::DynamoDB::Client.new(region: 'us-east-1')
    @dynamo_resource = Aws::DynamoDB::Resource.new(client: client)
    @table = @dynamo_resource.table(table_name)
  end

  # Updates rating and plot data for a movie in the table.
  #
  # @param movie [Hash] The title, year, plot, rating of the movie.
  def update_item(movie)
    response = @table.update_item(
      key: { 'year' => movie[:year], 'title' => movie[:title] },
      update_expression: 'set info.rating=:r',
      expression_attribute_values: { ':r' => movie[:rating] },
      return_values: 'UPDATED_NEW'
    )
  rescue Aws::DynamoDB::Errors::ServiceError => e
    puts("Couldn't update movie #{movie[:title]} (#{movie[:year]}) in table
#{@table.name}\n")
    puts("\t#{e.code}: #{e.message}")
    raise
  else
    response.attributes
  end
```

- Einzelheiten zur API finden Sie UpdateItemin der AWS SDK für Ruby API-Referenz.

SAP ABAP

SDK für SAP ABAP

> **Note**
>
> Es gibt noch mehr dazu. GitHub Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
    TRY.
        oo_output = lo_dyn->updateitem(
          iv_tablename        = iv_table_name
```

```
          it_key              = it_item_key
          it_attributeupdates = it_attribute_updates ).
        MESSAGE '1 item updated in DynamoDB Table' && iv_table_name TYPE 'I'.
      CATCH /aws1/cx_dyncondalcheckfaile00.
        MESSAGE 'A condition specified in the operation could not be evaluated.'
TYPE 'E'.
      CATCH /aws1/cx_dynresourcenotfoundex.
        MESSAGE 'The table or index does not exist' TYPE 'E'.
      CATCH /aws1/cx_dyntransactconflictex.
        MESSAGE 'Another transaction is using the item' TYPE 'E'.
    ENDTRY.
```

- Einzelheiten zur API finden Sie UpdateItemin der API-Referenz zum AWS SDK für SAP ABAP.

Swift

SDK für Swift

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
import AWSDynamoDB


    /// Update the specified movie with new `rating` and `plot` information.
    ///
    /// - Parameters:
    ///   - title: The title of the movie to update.
    ///   - year: The release year of the movie to update.
    ///   - rating: The new rating for the movie.
    ///   - plot: The new plot summary string for the movie.
    ///
    /// - Returns: An array of mappings of attribute names to their new
    ///   listing each item actually changed. Items that didn't need to change
    ///   aren't included in this list. `nil` if no changes were made.
    ///
```

```
   func update(title: String, year: Int, rating: Double? = nil, plot: String? =
nil) async throws
       -> [Swift.String: DynamoDBClientTypes.AttributeValue]?
   {
       do {
           guard let client = self.ddbClient else {
               throw MoviesError.UninitializedClient
           }

           // Build the update expression and the list of expression attribute
           // values. Include only the information that's changed.

           var expressionParts: [String] = []
           var attrValues: [Swift.String: DynamoDBClientTypes.AttributeValue] =
[:]

           if rating != nil {
               expressionParts.append("info.rating=:r")
               attrValues[":r"] = .n(String(rating!))
           }
           if plot != nil {
               expressionParts.append("info.plot=:p")
               attrValues[":p"] = .s(plot!)
           }
           let expression = "set \(expressionParts.joined(separator: ", "))"

           let input = UpdateItemInput(
               // Create substitution tokens for the attribute values, to ensure
               // no conflicts in expression syntax.
               expressionAttributeValues: attrValues,
               // The key identifying the movie to update consists of the
release
               // year and title.
               key: [
                   "year": .n(String(year)),
                   "title": .s(title)
               ],
               returnValues: .updatedNew,
               tableName: self.tableName,
               updateExpression: expression
           )
           let output = try await client.updateItem(input: input)
```

```
            guard let attributes: [Swift.String:
DynamoDBClientTypes.AttributeValue] = output.attributes else {
                throw MoviesError.InvalidAttributes
            }
            return attributes
        } catch {
            print("ERROR: update:", dump(error))
            throw error
        }
    }
```

- Einzelheiten zur API finden Sie UpdateItemin der API-Referenz zum AWS SDK für Swift.

Eine vollständige Liste der AWS SDK-Entwicklerhandbücher und Codebeispiele finden Sie unterDynamoDB mit einem SDK verwenden AWS. Dieses Thema enthält auch Informationen zu den ersten Schritten und Details zu früheren SDK-Versionen.

Verwendung **`UpdateTable`** mit einem AWS SDK oder CLI

Die folgenden Code-Beispiele zeigen, wie `UpdateTable` verwendet wird.

Aktionsbeispiele sind Codeauszüge aus größeren Programmen und müssen im Kontext ausgeführt werden. In den folgenden Codebeispielen können Sie diese Aktion im Kontext sehen:

- Erstellen und Verwalten globaler Tabellen zur Demonstration von MREC
- Erstellen und Verwalten von globalen MRSC-Tabellen
- Verwalten globaler sekundärer Indizes
- Aktualisieren der Einstellung für den Warmdurchsatz einer Tabelle
- Arbeiten mit globalen Tabellen und letztendlicher multiregionaler Konsistenz (Multi-Region Eventual Consistency, MREC)
- Arbeiten mit der Tabellenverschlüsselung

C++

SDK für C++

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
//! Update a DynamoDB table.
/*!
  \sa updateTable()
  \param tableName: Name for the DynamoDB table.
  \param readCapacity: Provisioned read capacity.
  \param writeCapacity: Provisioned write capacity.
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
 */
bool AwsDoc::DynamoDB::updateTable(const Aws::String &tableName,
                                   long long readCapacity, long long
 writeCapacity,
                                   const Aws::Client::ClientConfiguration
 &clientConfiguration) {
    Aws::DynamoDB::DynamoDBClient dynamoClient(clientConfiguration);

    std::cout << "Updating " << tableName << " with new provisioned throughput
 values"
              << std::endl;
    std::cout << "Read capacity : " << readCapacity << std::endl;
    std::cout << "Write capacity: " << writeCapacity << std::endl;

    Aws::DynamoDB::Model::UpdateTableRequest request;
    Aws::DynamoDB::Model::ProvisionedThroughput provisionedThroughput;

 provisionedThroughput.WithReadCapacityUnits(readCapacity).WithWriteCapacityUnits(
           writeCapacity);

 request.WithProvisionedThroughput(provisionedThroughput).WithTableName(tableName);

    const Aws::DynamoDB::Model::UpdateTableOutcome &outcome =
 dynamoClient.UpdateTable(
```

```
            request);
    if (outcome.IsSuccess()) {
        std::cout << "Successfully updated the table." << std::endl;
    } else {
        const Aws::DynamoDB::DynamoDBError &error = outcome.GetError();
        if (error.GetErrorType() == Aws::DynamoDB::DynamoDBErrors::VALIDATION &&
            error.GetMessage().find("The provisioned throughput for the table
 will not change") != std::string::npos) {
            std::cout << "The provisioned throughput for the table will not
 change." << std::endl;
        } else {
            std::cerr << outcome.GetError().GetMessage() << std::endl;
            return false;
        }
    }

    return waitTableActive(tableName, dynamoClient);
}
```

Code, der darauf wartet, dass die Tabelle aktiv wird.

```
//! Query a newly created DynamoDB table until it is active.
/*!
  \sa waitTableActive()
  \param waitTableActive: The DynamoDB table's name.
  \param dynamoClient: A DynamoDB client.
  \return bool: Function succeeded.
*/
bool AwsDoc::DynamoDB::waitTableActive(const Aws::String &tableName,
                                       const Aws::DynamoDB::DynamoDBClient
 &dynamoClient) {

    // Repeatedly call DescribeTable until table is ACTIVE.
    const int MAX_QUERIES = 20;
    Aws::DynamoDB::Model::DescribeTableRequest request;
    request.SetTableName(tableName);

    int count = 0;
    while (count < MAX_QUERIES) {
        const Aws::DynamoDB::Model::DescribeTableOutcome &result =
 dynamoClient.DescribeTable(
                request);
```

```
        if (result.IsSuccess()) {
            Aws::DynamoDB::Model::TableStatus status =
 result.GetResult().GetTable().GetTableStatus();

            if (Aws::DynamoDB::Model::TableStatus::ACTIVE != status) {
                std::this_thread::sleep_for(std::chrono::seconds(1));
            }
            else {
                return true;
            }
        }
        else {
            std::cerr << "Error DynamoDB::waitTableActive "
                      << result.GetError().GetMessage() << std::endl;
            return false;
        }
        count++;
    }
    return false;
}
```

- Einzelheiten zur API finden Sie UpdateTablein der AWS SDK für C++ API-Referenz.

CLI

AWS CLI

Beispiel 1: So ändern Sie den Fakturierungsmodus einer Tabelle

Im folgenden Beispiel für `update-table` wird die bereitgestellte Lese- und Schreibkapazität für die `MusicCollection`-Tabelle erhöht.

```
aws dynamodb update-table \
    --table-name MusicCollection \
    --billing-mode PROVISIONED \
    --provisioned-throughput ReadCapacityUnits=15,WriteCapacityUnits=10
```

Ausgabe:

```
{
    "TableDescription": {
```

```
        "AttributeDefinitions": [
            {
                "AttributeName": "AlbumTitle",
                "AttributeType": "S"
            },
            {
                "AttributeName": "Artist",
                "AttributeType": "S"
            },
            {
                "AttributeName": "SongTitle",
                "AttributeType": "S"
            }
        ],
        "TableName": "MusicCollection",
        "KeySchema": [
            {
                "AttributeName": "Artist",
                "KeyType": "HASH"
            },
            {
                "AttributeName": "SongTitle",
                "KeyType": "RANGE"
            }
        ],
        "TableStatus": "UPDATING",
        "CreationDateTime": "2020-05-26T15:59:49.473000-07:00",
        "ProvisionedThroughput": {
            "LastIncreaseDateTime": "2020-07-28T13:18:18.921000-07:00",
            "NumberOfDecreasesToday": 0,
            "ReadCapacityUnits": 15,
            "WriteCapacityUnits": 10
        },
        "TableSizeBytes": 182,
        "ItemCount": 2,
        "TableArn": "arn:aws:dynamodb:us-west-2:123456789012:table/
MusicCollection",
        "TableId": "abcd0123-01ab-23cd-0123-abcdef123456",
        "BillingModeSummary": {
            "BillingMode": "PROVISIONED",
            "LastUpdateToPayPerRequestDateTime":
 "2020-07-28T13:14:48.366000-07:00"
        }
    }
```

```
}
```

Weitere Informationen finden Sie unter Aktualisieren einer Tabelle im Entwicklerhandbuch zu Amazon DynamoDB.

Beispiel 2: So erstellen Sie einen globalen sekundären Index

Das folgende Beispiel fügt der `MusicCollection`-Tabelle einen globalen sekundären Index hinzu.

```
aws dynamodb update-table \
    --table-name MusicCollection \
    --attribute-definitions AttributeName=AlbumTitle,AttributeType=S \
    --global-secondary-index-updates file://gsi-updates.json
```

Inhalt von `gsi-updates.json`:

```
[
    {
        "Create": {
            "IndexName": "AlbumTitle-index",
            "KeySchema": [
                {
                    "AttributeName": "AlbumTitle",
                    "KeyType": "HASH"
                }
            ],
            "ProvisionedThroughput": {
                "ReadCapacityUnits": 10,
                "WriteCapacityUnits": 10
            },
            "Projection": {
                "ProjectionType": "ALL"
            }
        }
    }
]
```

Ausgabe:

```
{
    "TableDescription": {
```

```
        "AttributeDefinitions": [
            {
                "AttributeName": "AlbumTitle",
                "AttributeType": "S"
            },
            {
                "AttributeName": "Artist",
                "AttributeType": "S"
            },
            {
                "AttributeName": "SongTitle",
                "AttributeType": "S"
            }
        ],
        "TableName": "MusicCollection",
        "KeySchema": [
            {
                "AttributeName": "Artist",
                "KeyType": "HASH"
            },
            {
                "AttributeName": "SongTitle",
                "KeyType": "RANGE"
            }
        ],
        "TableStatus": "UPDATING",
        "CreationDateTime": "2020-05-26T15:59:49.473000-07:00",
        "ProvisionedThroughput": {
            "LastIncreaseDateTime": "2020-07-28T12:59:17.537000-07:00",
            "NumberOfDecreasesToday": 0,
            "ReadCapacityUnits": 15,
            "WriteCapacityUnits": 10
        },
        "TableSizeBytes": 182,
        "ItemCount": 2,
        "TableArn": "arn:aws:dynamodb:us-west-2:123456789012:table/
MusicCollection",
        "TableId": "abcd0123-01ab-23cd-0123-abcdef123456",
        "BillingModeSummary": {
            "BillingMode": "PROVISIONED",
            "LastUpdateToPayPerRequestDateTime":
 "2020-07-28T13:14:48.366000-07:00"
        },
        "GlobalSecondaryIndexes": [
```

```
            {
                "IndexName": "AlbumTitle-index",
                "KeySchema": [
                    {
                        "AttributeName": "AlbumTitle",
                        "KeyType": "HASH"
                    }
                ],
                "Projection": {
                    "ProjectionType": "ALL"
                },
                "IndexStatus": "CREATING",
                "Backfilling": false,
                "ProvisionedThroughput": {
                    "NumberOfDecreasesToday": 0,
                    "ReadCapacityUnits": 10,
                    "WriteCapacityUnits": 10
                },
                "IndexSizeBytes": 0,
                "ItemCount": 0,
                "IndexArn": "arn:aws:dynamodb:us-west-2:123456789012:table/
MusicCollection/index/AlbumTitle-index"
            }
        ]
    }
}
```

Weitere Informationen finden Sie unter Aktualisieren einer Tabelle im Entwicklerhandbuch zu Amazon DynamoDB.

Beispiel 3: So aktivieren Sie DynamoDB Streams für eine Tabelle

Der folgende Befehl aktiviert DynamoDB Streams für die `MusicCollection`-Tabelle.

```
aws dynamodb update-table \
    --table-name MusicCollection \
    --stream-specification StreamEnabled=true,StreamViewType=NEW_IMAGE
```

Ausgabe:

```
{
    "TableDescription": {
        "AttributeDefinitions": [
```

```
            {
                "AttributeName": "AlbumTitle",
                "AttributeType": "S"
            },
            {
                "AttributeName": "Artist",
                "AttributeType": "S"
            },
            {
                "AttributeName": "SongTitle",
                "AttributeType": "S"
            }
        ],
        "TableName": "MusicCollection",
        "KeySchema": [
            {
                "AttributeName": "Artist",
                "KeyType": "HASH"
            },
            {
                "AttributeName": "SongTitle",
                "KeyType": "RANGE"
            }
        ],
        "TableStatus": "UPDATING",
        "CreationDateTime": "2020-05-26T15:59:49.473000-07:00",
        "ProvisionedThroughput": {
            "LastIncreaseDateTime": "2020-07-28T12:59:17.537000-07:00",
            "NumberOfDecreasesToday": 0,
            "ReadCapacityUnits": 15,
            "WriteCapacityUnits": 10
        },
        "TableSizeBytes": 182,
        "ItemCount": 2,
        "TableArn": "arn:aws:dynamodb:us-west-2:123456789012:table/
MusicCollection",
        "TableId": "abcd0123-01ab-23cd-0123-abcdef123456",
        "BillingModeSummary": {
            "BillingMode": "PROVISIONED",
            "LastUpdateToPayPerRequestDateTime":
 "2020-07-28T13:14:48.366000-07:00"
        },
        "LocalSecondaryIndexes": [
            {
```

```
                "IndexName": "AlbumTitleIndex",
                "KeySchema": [
                    {
                        "AttributeName": "Artist",
                        "KeyType": "HASH"
                    },
                    {
                        "AttributeName": "AlbumTitle",
                        "KeyType": "RANGE"
                    }
                ],
                "Projection": {
                    "ProjectionType": "INCLUDE",
                    "NonKeyAttributes": [
                        "Year",
                        "Genre"
                    ]
                },
                "IndexSizeBytes": 139,
                "ItemCount": 2,
                "IndexArn": "arn:aws:dynamodb:us-west-2:123456789012:table/
MusicCollection/index/AlbumTitleIndex"
            }
        ],
        "GlobalSecondaryIndexes": [
            {
                "IndexName": "AlbumTitle-index",
                "KeySchema": [
                    {
                        "AttributeName": "AlbumTitle",
                        "KeyType": "HASH"
                    }
                ],
                "Projection": {
                    "ProjectionType": "ALL"
                },
                "IndexStatus": "ACTIVE",
                "ProvisionedThroughput": {
                    "NumberOfDecreasesToday": 0,
                    "ReadCapacityUnits": 10,
                    "WriteCapacityUnits": 10
                },
                "IndexSizeBytes": 0,
                "ItemCount": 0,
```

```
                "IndexArn": "arn:aws:dynamodb:us-west-2:123456789012:table/
MusicCollection/index/AlbumTitle-index"
            }
        ],
        "StreamSpecification": {
            "StreamEnabled": true,
            "StreamViewType": "NEW_IMAGE"
        },
        "LatestStreamLabel": "2020-07-28T21:53:39.112",
        "LatestStreamArn": "arn:aws:dynamodb:us-west-2:123456789012:table/
MusicCollection/stream/2020-07-28T21:53:39.112"
    }
}
```

Weitere Informationen finden Sie unter Aktualisieren einer Tabelle im Entwicklerhandbuch zu Amazon DynamoDB.

Beispiel 4: So aktivieren Sie die serverseitige Verschlüsselung

Im folgenden Beispiel wird die serverseitige Verschlüsselung für die `MusicCollection`-Tabelle aktiviert.

```
aws dynamodb update-table \
    --table-name MusicCollection \
    --sse-specification Enabled=true,SSEType=KMS
```

Ausgabe:

```
{
    "TableDescription": {
        "AttributeDefinitions": [
            {
                "AttributeName": "AlbumTitle",
                "AttributeType": "S"
            },
            {
                "AttributeName": "Artist",
                "AttributeType": "S"
            },
            {
                "AttributeName": "SongTitle",
                "AttributeType": "S"
            }
```

```
        ],
        "TableName": "MusicCollection",
        "KeySchema": [
            {
                "AttributeName": "Artist",
                "KeyType": "HASH"
            },
            {
                "AttributeName": "SongTitle",
                "KeyType": "RANGE"
            }
        ],
        "TableStatus": "ACTIVE",
        "CreationDateTime": "2020-05-26T15:59:49.473000-07:00",
        "ProvisionedThroughput": {
            "LastIncreaseDateTime": "2020-07-28T12:59:17.537000-07:00",
            "NumberOfDecreasesToday": 0,
            "ReadCapacityUnits": 15,
            "WriteCapacityUnits": 10
        },
        "TableSizeBytes": 182,
        "ItemCount": 2,
        "TableArn": "arn:aws:dynamodb:us-west-2:123456789012:table/
MusicCollection",
        "TableId": "abcd0123-01ab-23cd-0123-abcdef123456",
        "BillingModeSummary": {
            "BillingMode": "PROVISIONED",
            "LastUpdateToPayPerRequestDateTime":
 "2020-07-28T13:14:48.366000-07:00"
        },
        "LocalSecondaryIndexes": [
            {
                "IndexName": "AlbumTitleIndex",
                "KeySchema": [
                    {
                        "AttributeName": "Artist",
                        "KeyType": "HASH"
                    },
                    {
                        "AttributeName": "AlbumTitle",
                        "KeyType": "RANGE"
                    }
                ],
                "Projection": {
```

```
                    "ProjectionType": "INCLUDE",
                    "NonKeyAttributes": [
                        "Year",
                        "Genre"
                    ]
                },
                "IndexSizeBytes": 139,
                "ItemCount": 2,
                "IndexArn": "arn:aws:dynamodb:us-west-2:123456789012:table/
MusicCollection/index/AlbumTitleIndex"
            }
        ],
        "GlobalSecondaryIndexes": [
            {
                "IndexName": "AlbumTitle-index",
                "KeySchema": [
                    {
                        "AttributeName": "AlbumTitle",
                        "KeyType": "HASH"
                    }
                ],
                "Projection": {
                    "ProjectionType": "ALL"
                },
                "IndexStatus": "ACTIVE",
                "ProvisionedThroughput": {
                    "NumberOfDecreasesToday": 0,
                    "ReadCapacityUnits": 10,
                    "WriteCapacityUnits": 10
                },
                "IndexSizeBytes": 0,
                "ItemCount": 0,
                "IndexArn": "arn:aws:dynamodb:us-west-2:123456789012:table/
MusicCollection/index/AlbumTitle-index"
            }
        ],
        "StreamSpecification": {
            "StreamEnabled": true,
            "StreamViewType": "NEW_IMAGE"
        },
        "LatestStreamLabel": "2020-07-28T21:53:39.112",
        "LatestStreamArn": "arn:aws:dynamodb:us-west-2:123456789012:table/
MusicCollection/stream/2020-07-28T21:53:39.112",
        "SSEDescription": {
```

```
            "Status": "UPDATING"
        }
    }
}
```

Weitere Informationen finden Sie unter Aktualisieren einer Tabelle im Entwicklerhandbuch zu Amazon DynamoDB.

- Einzelheiten zur API finden Sie UpdateTablein der AWS CLI Befehlsreferenz.

PowerShell

Tools für PowerShell V4

Beispiel 1: Aktualisiert den bereitgestellten Durchsatz für die angegebene Tabelle.

```
Update-DDBTable -TableName "myTable" -ReadCapacity 10 -WriteCapacity 5
```

- Einzelheiten zur API finden Sie unter UpdateTable AWS -Tools für PowerShellCmdlet-Referenz (V4).

Tools für V5 PowerShell

Beispiel 1: Aktualisiert den bereitgestellten Durchsatz für die angegebene Tabelle.

```
Update-DDBTable -TableName "myTable" -ReadCapacity 10 -WriteCapacity 5
```

- Einzelheiten zur API finden Sie unter UpdateTable AWS -Tools für PowerShellCmdlet-Referenz (V5).

Eine vollständige Liste der AWS SDK-Entwicklerhandbücher und Codebeispiele finden Sie unter. DynamoDB mit einem SDK verwenden AWS Dieses Thema enthält auch Informationen zu den ersten Schritten und Details zu früheren SDK-Versionen.

Verwendung **`UpdateTimeToLive`** mit einem AWS SDK oder CLI

Die folgenden Code-Beispiele zeigen, wie `UpdateTimeToLive` verwendet wird.

Beispiele für Aktionen sind Codeauszüge aus größeren Programmen und müssen im Kontext ausgeführt werden. Im folgenden Codebeispiel können Sie diese Aktion im Kontext sehen:

- Arbeite mit Streams und Time-to-Live

CLI

AWS CLI

So aktualisieren Sie die Time-to-Live-Einstellungen für eine Tabelle

Im folgenden Beispiel für `update-time-to-live` wird Time to Live für die angegebene Tabelle aktiviert.

```
aws dynamodb update-time-to-live \
    --table-name MusicCollection \
    --time-to-live-specification Enabled=true,AttributeName=ttl
```

Ausgabe:

```
{
    "TimeToLiveSpecification": {
        "Enabled": true,
        "AttributeName": "ttl"
    }
}
```

Weitere Informationen finden Sie unter Time to Live im Entwicklerhandbuch zu Amazon DynamoDB.

- Einzelheiten zur API finden Sie UpdateTimeToLivein der AWS CLI Befehlsreferenz.

Java

SDK für Java 2.x

Aktivieren Sie mit AWS SDK for Java 2.x TTL für eine vorhandene DynamoDB-Tabelle.

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.model.ResourceNotFoundException;
import software.amazon.awssdk.services.dynamodb.model.TimeToLiveSpecification;
import software.amazon.awssdk.services.dynamodb.model.UpdateTimeToLiveRequest;
```

```
import software.amazon.awssdk.services.dynamodb.model.UpdateTimeToLiveResponse;

import java.util.logging.Level;
import java.util.logging.Logger;

    public UpdateTimeToLiveResponse enableTTL(final String tableName, final
 String attributeName, final Region region) {
        final TimeToLiveSpecification ttlSpec = TimeToLiveSpecification.builder()
            .attributeName(attributeName)
            .enabled(true)
            .build();

        final UpdateTimeToLiveRequest request = UpdateTimeToLiveRequest.builder()
            .tableName(tableName)
            .timeToLiveSpecification(ttlSpec)
            .build();

        try (DynamoDbClient ddb = dynamoDbClient != null
            ? dynamoDbClient
            : DynamoDbClient.builder().region(region).build()) {
            return ddb.updateTimeToLive(request);
        } catch (ResourceNotFoundException e) {
            System.err.format(TABLE_NOT_FOUND_ERROR, tableName);
            throw e;
        } catch (DynamoDbException e) {
            System.err.println(e.getMessage());
            throw e;
        }
    }
```

Deaktivieren Sie mit AWS SDK for Java 2.x TTL für eine vorhandene DynamoDB-Tabelle.

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.model.ResourceNotFoundException;
import software.amazon.awssdk.services.dynamodb.model.TimeToLiveSpecification;
import software.amazon.awssdk.services.dynamodb.model.UpdateTimeToLiveRequest;
import software.amazon.awssdk.services.dynamodb.model.UpdateTimeToLiveResponse;

import java.util.logging.Level;
import java.util.logging.Logger;
```

```
    public UpdateTimeToLiveResponse disableTTL(
        final String tableName, final String attributeName, final Region region)
 {
        final TimeToLiveSpecification ttlSpec = TimeToLiveSpecification.builder()
            .attributeName(attributeName)
            .enabled(false)
            .build();

        final UpdateTimeToLiveRequest request = UpdateTimeToLiveRequest.builder()
            .tableName(tableName)
            .timeToLiveSpecification(ttlSpec)
            .build();

        try (DynamoDbClient ddb = dynamoDbClient != null
            ? dynamoDbClient
            : DynamoDbClient.builder().region(region).build()) {
            return ddb.updateTimeToLive(request);
        } catch (ResourceNotFoundException e) {
            System.err.format(TABLE_NOT_FOUND_ERROR, tableName);
            throw e;
        } catch (DynamoDbException e) {
            System.err.println(e.getMessage());
            throw e;
        }
    }
```

- Einzelheiten zur API finden Sie UpdateTimeToLivein der AWS SDK for Java 2.x API-Referenz.

JavaScript

SDK für JavaScript (v3)

Aktivieren Sie TTL für eine vorhandene DynamoDB-Tabelle.

```
import { DynamoDBClient, UpdateTimeToLiveCommand } from "@aws-sdk/client-
dynamodb";

export const enableTTL = async (tableName, ttlAttribute, region = 'us-east-1') =>
 {
```

```
    const client = new DynamoDBClient({
        region: region,
        endpoint: `https://dynamodb.${region}.amazonaws.com`
    });

    const params = {
        TableName: tableName,
        TimeToLiveSpecification: {
            Enabled: true,
            AttributeName: ttlAttribute
        }
    };

    try {
        const response = await client.send(new UpdateTimeToLiveCommand(params));
        if (response.$metadata.httpStatusCode === 200) {
            console.log(`TTL enabled successfully for table ${tableName}, using
 attribute name ${ttlAttribute}.`);
        } else {
            console.log(`Failed to enable TTL for table ${tableName}, response
 object: ${response}`);
        }
        return response;
    } catch (e) {
        console.error(`Error enabling TTL: ${e}`);
        throw e;
    }
};

// Example usage (commented out for testing)
// enableTTL('ExampleTable', 'exampleTtlAttribute');
```

Deaktivieren Sie TTL für eine vorhandene DynamoDB-Tabelle.

```
import { DynamoDBClient, UpdateTimeToLiveCommand } from "@aws-sdk/client-
dynamodb";

export const disableTTL = async (tableName, ttlAttribute, region = 'us-east-1')
 => {

    const client = new DynamoDBClient({
        region: region,
        endpoint: `https://dynamodb.${region}.amazonaws.com`
```

```
    });

    const params = {
        TableName: tableName,
        TimeToLiveSpecification: {
            Enabled: false,
            AttributeName: ttlAttribute
        }
    };

    try {
        const response = await client.send(new UpdateTimeToLiveCommand(params));
        if (response.$metadata.httpStatusCode === 200) {
            console.log(`TTL disabled successfully for table ${tableName}, using
 attribute name ${ttlAttribute}.`);
        } else {
            console.log(`Failed to disable TTL for table ${tableName}, response
 object: ${response}`);
        }
        return response;
    } catch (e) {
        console.error(`Error disabling TTL: ${e}`);
        throw e;
    }
};

// Example usage (commented out for testing)
// disableTTL('ExampleTable', 'exampleTtlAttribute');
```

- Einzelheiten zur API finden Sie UpdateTimeToLivein der AWS SDK für JavaScript API-Referenz.

Python

SDK für Python (Boto3)

Aktivieren Sie TTL für eine vorhandene DynamoDB-Tabelle.

```
import boto3


def enable_ttl(table_name, ttl_attribute_name):
    """
```

```
    Enables TTL on DynamoDB table for a given attribute name
        on success, returns a status code of 200
        on error, throws an exception

    :param table_name: Name of the DynamoDB table
    :param ttl_attribute_name: The name of the TTL attribute being provided to
 the table.
    """
    try:
        dynamodb = boto3.client("dynamodb")

        # Enable TTL on an existing DynamoDB table
        response = dynamodb.update_time_to_live(
            TableName=table_name,
            TimeToLiveSpecification={"Enabled": True, "AttributeName":
 ttl_attribute_name},
        )

        # In the returned response, check for a successful status code.
        if response["ResponseMetadata"]["HTTPStatusCode"] == 200:
            print("TTL has been enabled successfully.")
        else:
            print(
                f"Failed to enable TTL, status code {response['ResponseMetadata']
['HTTPStatusCode']}"
            )
        return response
    except Exception as ex:
        print("Couldn't enable TTL in table %s. Here's why: %s" % (table_name,
 ex))
        raise


# your values
enable_ttl("your-table-name", "expireAt")
```

Deaktivieren Sie TTL für eine vorhandene DynamoDB-Tabelle.

```
import boto3


def disable_ttl(table_name, ttl_attribute_name):
```

```
    """
    Disables TTL on DynamoDB table for a given attribute name
        on success, returns a status code of 200
        on error, throws an exception

    :param table_name: Name of the DynamoDB table being modified
    :param ttl_attribute_name: The name of the TTL attribute being provided to
 the table.
    """
    try:
        dynamodb = boto3.client("dynamodb")

        # Enable TTL on an existing DynamoDB table
        response = dynamodb.update_time_to_live(
            TableName=table_name,
            TimeToLiveSpecification={"Enabled": False, "AttributeName":
 ttl_attribute_name},
        )

        # In the returned response, check for a successful status code.
        if response["ResponseMetadata"]["HTTPStatusCode"] == 200:
            print("TTL has been disabled successfully.")
        else:
            print(
                f"Failed to disable TTL, status code
 {response['ResponseMetadata']['HTTPStatusCode']}"
            )
    except Exception as ex:
        print("Couldn't disable TTL in table %s. Here's why: %s" % (table_name,
 ex))
        raise


# your values
disable_ttl("your-table-name", "expireAt")
```

- Einzelheiten zur API finden Sie UpdateTimeToLivein AWS SDK for Python (Boto3) API Reference.

Eine vollständige Liste der AWS SDK-Entwicklerhandbücher und Codebeispiele finden Sie unter. DynamoDB mit einem SDK verwenden AWS Dieses Thema enthält auch Informationen zu den ersten Schritten und Details zu früheren SDK-Versionen.

Szenarien für die Verwendung von DynamoDB AWS SDKs

Die folgenden Codebeispiele zeigen Ihnen, wie Sie gängige Szenarien in DynamoDB mit implementieren. AWS SDKs Diese Szenarien zeigen Ihnen, wie Sie bestimmte Aufgaben durch den Aufruf mehrerer Funktionen innerhalb von DynamoDB oder in Kombination mit anderen AWS-Services erledigen. Jedes Szenario enthält einen Link zum vollständigen Quell-Code, wo Sie Anleitungen zum Einrichten und Ausführen des Codes finden.

Szenarien zielen auf eine mittlere Erfahrungsebene ab, um Ihnen zu helfen, Service-Aktionen im Kontext zu verstehen.

Beispiele

- Beschleunigen Sie DynamoDB-Lesevorgänge mit DAX mithilfe eines SDK AWS
- Arbeiten Sie mit erweiterten DynamoDB Global Secondary Index-Szenarien unter Verwendung von v2 AWS Command Line Interface
- Erstellen Sie eine Anwendung zum Senden von Daten an eine DynamoDB-Tabelle
- Vergleichen Sie mehrere Werte mit einem einzigen Attribut in DynamoDB mit einem SDK AWS
- Bedingtes Aktualisieren eines DynamoDB-Elements mit einer TTL mithilfe eines SDK AWS
- Stellen Sie mithilfe eines SDK eine Connect zu einer lokalen DynamoDB-Instanz her AWS
- Zählen Sie Ausdrucksoperatoren in DynamoDB mit einem SDK AWS
- Erstellen einer API-Gateway-REST-API zur Verfolgung von COVID-19-Daten
- Erstellen einer Messenger-Anwendung mit Step Functions
- Eine Anwendung für Foto-Asset-Management erstellen, mit der Benutzer Fotos mithilfe von Labels verwalten können
- Erstellen Sie mithilfe des SDK eine DynamoDB-Tabelle mit einem globalen sekundären Index AWS
- Erstellen Sie mithilfe eines SDK eine DynamoDB-Tabelle mit Warmdurchsatzeinstellung AWS
- Erstellen einer Webanwendung zur Verfolgung von DynamoDB-Daten
- Erstellen einer Websocket-Chat-Anwendung mit API Gateway
- Erstellen Sie ein DynamoDB-Element mit einer TTL mithilfe eines SDK AWS

- Erstellen und verwalten Sie globale DynamoDB-Tabellen mit Multi-Region Strong Consistency mithilfe eines SDK AWS
- Erstellen und verwalten Sie globale DynamoDB-Tabellen zur Demonstration von MREC mithilfe eines SDK AWS
- Löschen Sie DynamoDB-Daten mithilfe von PartiQL DELETE-Anweisungen mit einem SDK AWS
- Ermitteln Sie persönliche Schutzausrüstung in Bildern mit Amazon Rekognition mithilfe eines SDK AWS
- Fügen Sie DynamoDB-Daten mithilfe von PartiQL INSERT-Anweisungen mit einem SDK ein AWS
- Aufrufen einer Lambda-Funktion von einem Browser aus
- Globale sekundäre DynamoDB-Indizes mit Version 2 verwalten AWS Command Line Interface
- Verwaltung ressourcenbasierter DynamoDB-Richtlinien mit Version 2 AWS Command Line Interface
- Überwachen Sie die Leistung von Amazon DynamoDB mithilfe eines SDK AWS
- Führen Sie erweiterte DynamoDB-Abfrageoperationen mit einem SDK durch AWS
- Führen Sie Listenoperationen in DynamoDB mit einem SDK durch AWS
- Führen Sie Kartenoperationen in DynamoDB mit einem SDK durch AWS
- Führen Sie Set-Operationen in DynamoDB mit einem SDK durch AWS
- Abfragen einer DynamoDB-Tabelle mithilfe von Batches von PartiQL-Anweisungen und einem SDK AWS
- Fragen Sie eine DynamoDB-Tabelle mit PartiQL und einem SDK ab AWS
- Abfragen einer DynamoDB-Tabelle mithilfe eines globalen sekundären Indexes mit einem SDK AWS
- Fragen Sie eine DynamoDB-Tabelle mithilfe einer Begins_with-Bedingung mit einem SDK ab AWS
- Fragen Sie eine DynamoDB-Tabelle mithilfe eines Datumsbereichs im Sortierschlüssel mit einem SDK ab AWS
- Fragen Sie eine DynamoDB-Tabelle mit einem komplexen Filterausdruck mit einem SDK ab AWS
- Fragen Sie eine DynamoDB-Tabelle mit einem dynamischen Filterausdruck mit einem SDK ab AWS
- Fragen Sie eine DynamoDB-Tabelle mit einem Filterausdruck und einem Limit mit einem SDK ab AWS
- Abfragen einer DynamoDB-Tabelle mit verschachtelten Attributen mithilfe eines SDK AWS

- Abfragen einer DynamoDB-Tabelle mit Paginierung mithilfe eines SDK AWS
- Abfragen einer DynamoDB-Tabelle mit stark konsistenten Lesevorgängen mithilfe eines SDK AWS
- Abfragen von DynamoDB-Daten mithilfe von PartiQL SELECT-Anweisungen mit einem SDK AWS
- Abfragen einer DynamoDB-Tabelle nach TTL-Elementen mithilfe eines SDK AWS
- Abfragen von DynamoDB-Tabellen mithilfe von Datums- und Uhrzeitmustern mit einem SDK AWS
- Speichern Sie EXIF und andere Bildinformationen mit einem SDK AWS
- Richten Sie die attributebasierte Zugriffskontrolle für DynamoDB mit v2 ein AWS Command Line Interface
- Die Reihenfolge der Aktualisierungsausdrücke in DynamoDB mit einem SDK verstehen AWS
- Aktualisieren Sie eine DynamoDB-Tabelleneinstellung mit warmem Durchsatz mithilfe eines SDK AWS
- Aktualisieren Sie ein DynamoDB-Element mit einer TTL mithilfe eines SDK AWS
- Aktualisieren Sie DynamoDB-Daten mithilfe von PartiQL UPDATE-Anweisungen mit einem SDK AWS
- Verwenden von API Gateway zum Aufrufen einer Lambda-Funktion
- Verwenden von Step Functions, um Lambda-Funktionen aufzurufen
- Verwenden Sie ein Dokumentmodell für DynamoDB mithilfe eines SDK AWS
- Verwenden Sie ein Objektpersistenzmodell auf hoher Ebene für DynamoDB mithilfe eines SDK AWS
- Verwenden Sie atomare Zähleroperationen in DynamoDB mit einem SDK AWS
- Verwenden Sie bedingte Operationen in DynamoDB mit einem SDK AWS
- Verwenden Sie Ausdrucksattributnamen in DynamoDB mit einem SDK AWS
- Verwendung geplanter Ereignisse zum Aufrufen einer Lambda-Funktion
- Arbeiten Sie mit lokalen sekundären DynamoDB-Indizes unter Verwendung von v2 AWS Command Line Interface
- Arbeiten Sie mit DynamoDB Streams und Time-to-Live verwenden Sie v2 AWS Command Line Interface
- Arbeiten Sie mit globalen DynamoDB-Tabellen und Multi-Region-Replikation mit Eventual Consistency (MREC) unter Verwendung von v2 AWS Command Line Interface
- Arbeiten Sie mit DynamoDB-Ressourcen-Tagging mithilfe von v2 AWS Command Line Interface
- Arbeiten Sie mit DynamoDB-Tabellenverschlüsselung mit Version 2 AWS Command Line Interface

Beschleunigen Sie DynamoDB-Lesevorgänge mit DAX mithilfe eines SDK AWS

Wie das aussehen kann, sehen Sie am nachfolgenden Beispielcode:

- Erstellen und Schreiben von Daten in eine Tabelle mit sowohl den DAX- als auch den SDK-Clients.
- Abrufen, Abfragen und Scannen der Tabelle mit beiden Clients und Vergleichen ihrer Leistung.

Weitere Informationen finden Sie unter Entwickeln mit dem DynamoDB-Accelerator-Client.

Python

SDK für Python (Boto3)

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

Erstellen Sie eine Tabelle mit dem DAX- oder Boto3-Client.

```
import boto3


def create_dax_table(dyn_resource=None):
    """
    Creates a DynamoDB table.

    :param dyn_resource: Either a Boto3 or DAX resource.
    :return: The newly created table.
    """
    if dyn_resource is None:
        dyn_resource = boto3.resource("dynamodb")

    table_name = "TryDaxTable"
    params = {
        "TableName": table_name,
        "KeySchema": [
            {"AttributeName": "partition_key", "KeyType": "HASH"},
```

```
            {"AttributeName": "sort_key", "KeyType": "RANGE"},
        ],
        "AttributeDefinitions": [
            {"AttributeName": "partition_key", "AttributeType": "N"},
            {"AttributeName": "sort_key", "AttributeType": "N"},
        ],
        "BillingMode": "PAY_PER_REQUEST",
    }
    table = dyn_resource.create_table(**params)
    print(f"Creating {table_name}...")
    table.wait_until_exists()
    return table


if __name__ == "__main__":
    dax_table = create_dax_table()
    print(f"Created table.")
```

Schreiben Sie Testdaten in die Tabelle.

```
import boto3


def write_data_to_dax_table(key_count, item_size, dyn_resource=None):
    """
    Writes test data to the demonstration table.

    :param key_count: The number of partition and sort keys to use to populate
 the
                      table. The total number of items is key_count * key_count.
    :param item_size: The size of non-key data for each test item.
    :param dyn_resource: Either a Boto3 or DAX resource.
    """
    if dyn_resource is None:
        dyn_resource = boto3.resource("dynamodb")

    table = dyn_resource.Table("TryDaxTable")
    some_data = "X" * item_size

    for partition_key in range(1, key_count + 1):
        for sort_key in range(1, key_count + 1):
            table.put_item(
```

```
                Item={
                    "partition_key": partition_key,
                    "sort_key": sort_key,
                    "some_data": some_data,
                }
            )
            print(f"Put item ({partition_key}, {sort_key}) succeeded.")


if __name__ == "__main__":
    write_key_count = 10
    write_item_size = 1000
    print(
        f"Writing {write_key_count*write_key_count} items to the table. "
        f"Each item is {write_item_size} characters."
    )
    write_data_to_dax_table(write_key_count, write_item_size)
```

Rufen Sie Elemente für eine Reihe von Iterationen sowohl für den DAX-Client als auch für den Boto3-Client ab und melden Sie die jeweils aufgewendete Zeit.

```
import argparse
import sys
import time
import amazondax
import boto3


def get_item_test(key_count, iterations, dyn_resource=None):
    """
    Gets items from the table a specified number of times. The time before the
    first iteration and the time after the last iteration are both captured
    and reported.

    :param key_count: The number of items to get from the table in each
 iteration.
    :param iterations: The number of iterations to run.
    :param dyn_resource: Either a Boto3 or DAX resource.
    :return: The start and end times of the test.
    """
    if dyn_resource is None:
        dyn_resource = boto3.resource("dynamodb")
```

```
    table = dyn_resource.Table("TryDaxTable")
    start = time.perf_counter()
    for _ in range(iterations):
        for partition_key in range(1, key_count + 1):
            for sort_key in range(1, key_count + 1):
                table.get_item(
                    Key={"partition_key": partition_key, "sort_key": sort_key}
                )
                print(".", end="")
                sys.stdout.flush()
    print()
    end = time.perf_counter()
    return start, end


if __name__ == "__main__":
    # pylint: disable=not-context-manager
    parser = argparse.ArgumentParser()
    parser.add_argument(
        "endpoint_url",
        nargs="?",
        help="When specified, the DAX cluster endpoint. Otherwise, DAX is not
 used.",
    )
    args = parser.parse_args()

    test_key_count = 10
    test_iterations = 50
    if args.endpoint_url:
        print(
            f"Getting each item from the table {test_iterations} times, "
            f"using the DAX client."
        )
        # Use a with statement so the DAX client closes the cluster after
 completion.
        with amazondax.AmazonDaxClient.resource(endpoint_url=args.endpoint_url)
 as dax:
            test_start, test_end = get_item_test(
                test_key_count, test_iterations, dyn_resource=dax
            )
    else:
        print(
            f"Getting each item from the table {test_iterations} times, "
```

```
            f"using the Boto3 client."
        )
        test_start, test_end = get_item_test(test_key_count, test_iterations)
    print(
        f"Total time: {test_end - test_start:.4f} sec. Average time: "
        f"{(test_end - test_start)/ test_iterations}."
    )
```

Fragen Sie die Tabelle im Hinblick auf eine Reihe von Iterationen sowohl für den DAX-Client als auch für den Boto3-Client ab und melden Sie die jeweils aufgewendete Zeit.

```
import argparse
import time
import sys
import amazondax
import boto3
from boto3.dynamodb.conditions import Key


def query_test(partition_key, sort_keys, iterations, dyn_resource=None):
    """
    Queries the table a specified number of times. The time before the
    first iteration and the time after the last iteration are both captured
    and reported.

    :param partition_key: The partition key value to use in the query. The query
                          returns items that have partition keys equal to this
 value.
    :param sort_keys: The range of sort key values for the query. The query
 returns
                      items that have sort key values between these two values.
    :param iterations: The number of iterations to run.
    :param dyn_resource: Either a Boto3 or DAX resource.
    :return: The start and end times of the test.
    """
    if dyn_resource is None:
        dyn_resource = boto3.resource("dynamodb")

    table = dyn_resource.Table("TryDaxTable")
    key_condition_expression = Key("partition_key").eq(partition_key) & Key(
        "sort_key"
    ).between(*sort_keys)
```

```
    start = time.perf_counter()
    for _ in range(iterations):
        table.query(KeyConditionExpression=key_condition_expression)
        print(".", end="")
        sys.stdout.flush()
    print()
    end = time.perf_counter()
    return start, end


if __name__ == "__main__":
    # pylint: disable=not-context-manager
    parser = argparse.ArgumentParser()
    parser.add_argument(
        "endpoint_url",
        nargs="?",
        help="When specified, the DAX cluster endpoint. Otherwise, DAX is not
 used.",
    )
    args = parser.parse_args()

    test_partition_key = 5
    test_sort_keys = (2, 9)
    test_iterations = 100
    if args.endpoint_url:
        print(f"Querying the table {test_iterations} times, using the DAX
 client.")
        # Use a with statement so the DAX client closes the cluster after
 completion.
        with amazondax.AmazonDaxClient.resource(endpoint_url=args.endpoint_url)
 as dax:
            test_start, test_end = query_test(
                test_partition_key, test_sort_keys, test_iterations,
 dyn_resource=dax
            )
    else:
        print(f"Querying the table {test_iterations} times, using the Boto3
 client.")
        test_start, test_end = query_test(
            test_partition_key, test_sort_keys, test_iterations
        )

    print(
```

```
        f"Total time: {test_end - test_start:.4f} sec. Average time: "
        f"{(test_end - test_start)/test_iterations}."
    )
```

Scannen Sie die Tabelle auf eine Reihe von Iterationen sowohl für den DAX-Client als auch für den Boto3-Client und melden Sie die jeweils aufgewendete Zeit.

```
import argparse
import time
import sys
import amazondax
import boto3


def scan_test(iterations, dyn_resource=None):
    """
    Scans the table a specified number of times. The time before the
    first iteration and the time after the last iteration are both captured
    and reported.

    :param iterations: The number of iterations to run.
    :param dyn_resource: Either a Boto3 or DAX resource.
    :return: The start and end times of the test.
    """
    if dyn_resource is None:
        dyn_resource = boto3.resource("dynamodb")

    table = dyn_resource.Table("TryDaxTable")
    start = time.perf_counter()
    for _ in range(iterations):
        table.scan()
        print(".", end="")
        sys.stdout.flush()
    print()
    end = time.perf_counter()
    return start, end


if __name__ == "__main__":
    # pylint: disable=not-context-manager
    parser = argparse.ArgumentParser()
    parser.add_argument(
```

```
        "endpoint_url",
        nargs="?",
        help="When specified, the DAX cluster endpoint. Otherwise, DAX is not
 used.",
    )
    args = parser.parse_args()

    test_iterations = 100
    if args.endpoint_url:
        print(f"Scanning the table {test_iterations} times, using the DAX
 client.")
        # Use a with statement so the DAX client closes the cluster after
 completion.
        with amazondax.AmazonDaxClient.resource(endpoint_url=args.endpoint_url)
 as dax:
            test_start, test_end = scan_test(test_iterations, dyn_resource=dax)
    else:
        print(f"Scanning the table {test_iterations} times, using the Boto3
 client.")
        test_start, test_end = scan_test(test_iterations)
    print(
        f"Total time: {test_end - test_start:.4f} sec. Average time: "
        f"{(test_end - test_start)/test_iterations}."
    )
```

Löschen Sie die Tabelle.

```
import boto3


def delete_dax_table(dyn_resource=None):
    """
    Deletes the demonstration table.

    :param dyn_resource: Either a Boto3 or DAX resource.
    """
    if dyn_resource is None:
        dyn_resource = boto3.resource("dynamodb")

    table = dyn_resource.Table("TryDaxTable")
    table.delete()
```

```
    print(f"Deleting {table.name}...")
    table.wait_until_not_exists()


if __name__ == "__main__":
    delete_dax_table()
    print("Table deleted!")
```

- Weitere API-Informationen finden Sie in den folgenden Themen der API-Referenz zum AWS SDK für Python (Boto3).
  - CreateTable
  - DeleteTable
  - GetItem
  - PutItem
  - Query
  - Scan

Eine vollständige Liste der AWS SDK-Entwicklerhandbücher und Codebeispiele finden Sie unterDynamoDB mit einem SDK verwenden AWS. Dieses Thema enthält auch Informationen zu den ersten Schritten und Details zu früheren SDK-Versionen.

Arbeiten Sie mit erweiterten DynamoDB Global Secondary Index-Szenarien unter Verwendung von v2 AWS Command Line Interface

Das folgende Codebeispiel zeigt, wie Sie mit erweiterten Konfigurationen für globale sekundäre Indizes arbeiten.

- Erstellen Sie eine Tabelle mit mehreren GSIs.
- Erstellen Sie eine Tabelle mit On-Demand-Kapazität und einem GSI.
- Fügt Elemente in eine Tabelle mit mehreren ein GSIs.
- Fragen Sie mehrere GSIs mit unterschiedlichen Bedingungen ab.

Bash

AWS CLI mit Bash-Skript

Erstellen Sie eine Tabelle mit mehreren GSIs.

```
# Create a table with multiple GSIs
aws dynamodb create-table \
    --table-name MusicLibrary \
    --attribute-definitions \
        AttributeName=Artist,AttributeType=S \
        AttributeName=SongTitle,AttributeType=S \
        AttributeName=AlbumTitle,AttributeType=S \
        AttributeName=Genre,AttributeType=S \
        AttributeName=Year,AttributeType=N \
    --key-schema \
        AttributeName=Artist,KeyType=HASH \
        AttributeName=SongTitle,KeyType=RANGE \
    --billing-mode PAY_PER_REQUEST \
    --global-secondary-indexes \
        "[
            {
                \"IndexName\": \"AlbumIndex\",
                \"KeySchema\": [{\"AttributeName\":\"AlbumTitle\",\"KeyType\":
\"HASH\"}],
                \"Projection\": {\"ProjectionType\":\"ALL\"}
            },
            {
                \"IndexName\": \"GenreYearIndex\",
                \"KeySchema\": [
                    {\"AttributeName\":\"Genre\",\"KeyType\":\"HASH\"},
                    {\"AttributeName\":\"Year\",\"KeyType\":\"RANGE\"}
                ],
                \"Projection\": {\"ProjectionType\":\"INCLUDE\",
\"NonKeyAttributes\":[\"Artist\",\"SongTitle\"]}
            }
        ]"
```

Erstellen Sie eine Tabelle mit On-Demand-Kapazität und einem GSI.

```
# Create a table with on-demand capacity and GSI
aws dynamodb create-table \
```

```
    --table-name MusicOnDemand \
    --attribute-definitions \
        AttributeName=Artist,AttributeType=S \
        AttributeName=SongTitle,AttributeType=S \
        AttributeName=Genre,AttributeType=S \
    --key-schema \
        AttributeName=Artist,KeyType=HASH \
        AttributeName=SongTitle,KeyType=RANGE \
    --billing-mode PAY_PER_REQUEST \
    --global-secondary-indexes \
        "[
            {
                \"IndexName\": \"GenreIndex\",
                \"KeySchema\": [{\"AttributeName\":\"Genre\",\"KeyType\":\"HASH
\"}],
                \"Projection\": {\"ProjectionType\":\"ALL\"}
            }
        ]"
```

Fügt Elemente in eine Tabelle mit mehreren ein GSIs.

```
# Add items to MusicLibrary table
aws dynamodb put-item \
    --table-name MusicLibrary \
    --item '{
        "Artist": {"S": "The Beatles"},
        "SongTitle": {"S": "Hey Jude"},
        "AlbumTitle": {"S": "Past Masters"},
        "Genre": {"S": "Rock"},
        "Year": {"N": "1968"}
    }'

aws dynamodb put-item \
    --table-name MusicLibrary \
    --item '{
        "Artist": {"S": "Miles Davis"},
        "SongTitle": {"S": "So What"},
        "AlbumTitle": {"S": "Kind of Blue"},
        "Genre": {"S": "Jazz"},
        "Year": {"N": "1959"}
    }'
```

Fragen Sie Elemente aus einer Tabelle mit mehreren ab GSIs.

```
# Query the AlbumIndex GSI
echo "Querying AlbumIndex GSI:"
aws dynamodb query \
    --table-name MusicLibrary \
    --index-name AlbumIndex \
    --key-condition-expression "AlbumTitle = :album" \
    --expression-attribute-values '{":album":{"S":"Kind of Blue"}}'

# Query the GenreYearIndex GSI with a range condition
echo "Querying GenreYearIndex GSI with range condition:"
aws dynamodb query \
    --table-name MusicLibrary \
    --index-name GenreYearIndex \
    --key-condition-expression "Genre = :genre AND #yr > :year" \
    --expression-attribute-names '{"#yr": "Year"}' \
    --expression-attribute-values '{":genre":{"S":"Rock"},":year":{"N":"1965"}}'
```

- Weitere API-Informationen finden Sie in den folgenden Themen der AWS CLI - Befehlsreferenz.
  - CreateTable
  - PutItem
  - Query

Eine vollständige Liste der AWS SDK-Entwicklerhandbücher und Codebeispiele finden Sie unterDynamoDB mit einem SDK verwenden AWS. Dieses Thema enthält auch Informationen zu den ersten Schritten und Details zu früheren SDK-Versionen.

Erstellen Sie eine Anwendung zum Senden von Daten an eine DynamoDB-Tabelle

Die folgenden Code-Beispiele zeigen, wie man eine Anwendung erstellt, die Daten an eine Amazon-DynamoDB-Tabelle übermittelt und Sie benachrichtigt, wenn ein Benutzer die Tabelle aktualisiert.

Java

SDK für Java 2.x

Zeigt, wie man eine dynamische Webanwendung erstellt, die Daten über die Amazon-DynamoDB-Java-API übermittelt und eine Textnachricht über die Amazon Simple Notification Service Java API sendet.

Den vollständigen Quellcode und Anweisungen zur Einrichtung und Ausführung finden Sie im vollständigen Beispiel unter GitHub.

In diesem Beispiel verwendete Dienste

- DynamoDB
- Amazon SNS

JavaScript

SDK für JavaScript (v3)

Das Beispiel zeigt, wie man eine App erstellt, die es Benutzern ermöglicht, Daten an eine Amazon-DynamoDB-Tabelle zu übermitteln und eine Textnachricht an den Administrator mit Amazon Simple Notification Service (Amazon SNS) zu senden.

Den vollständigen Quellcode und Anweisungen zur Einrichtung und Ausführung finden Sie im vollständigen Beispiel unter GitHub.

Dieses Beispiel ist auch verfügbar im AWS SDK für JavaScript Entwicklerhandbuch für v3.

In diesem Beispiel verwendete Dienste

- DynamoDB
- Amazon SNS

Eine vollständige Liste der AWS SDK-Entwicklerhandbücher und Codebeispiele finden Sie unterDynamoDB mit einem SDK verwenden AWS. Dieses Thema enthält auch Informationen zu den ersten Schritten und Details zu früheren SDK-Versionen.

Vergleichen Sie mehrere Werte mit einem einzigen Attribut in DynamoDB mit einem SDK AWS

In den folgenden Codebeispielen wird gezeigt, wie Sie mehrere Werte mit einem einzigen Attribut in DynamoDB vergleichen.

- Verwenden des IN-Operators, um mehrere Werte mit einem einzigen Attribut zu vergleichen
- Vergleichen des IN-Operators mit mehreren OR-Bedingungen
- Verständnis der Vorteilen von IN hinsichtlich Leistung und Ausdruckskomplexität

Java

SDK für Java 2.x

Vergleichen Sie mehrere Werte mit einem einzelnen Attribut in DynamoDB mit. AWS SDK for Java 2.x

```
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.AttributeValue;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.model.QueryRequest;
import software.amazon.awssdk.services.dynamodb.model.QueryResponse;
import software.amazon.awssdk.services.dynamodb.model.ScanRequest;
import software.amazon.awssdk.services.dynamodb.model.ScanResponse;

import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Locale;
import java.util.Map;

    /**
     * Queries a table using the IN operator to compare multiple values with a
 single attribute.
     *
     * <p>This method demonstrates how to use the IN operator in a filter
 expression
     * to match an attribute against multiple values.
     *
     * @param dynamoDbClient The DynamoDB client
     * @param tableName The name of the DynamoDB table
```

```
     * @param partitionKeyName The name of the partition key attribute
     * @param partitionKeyValue The value of the partition key to query
     * @param attributeName The name of the attribute to compare
     * @param valuesList List of values to compare against
     * @return The query response from DynamoDB
     * @throws DynamoDbException if an error occurs during the operation
     */
    public static QueryResponse compareMultipleValues(
        DynamoDbClient dynamoDbClient,
        String tableName,
        String partitionKeyName,
        AttributeValue partitionKeyValue,
        String attributeName,
        List<AttributeValue> valuesList) {

        // Create expression attribute names
        Map<String, String> expressionAttributeNames = new HashMap<>();
        expressionAttributeNames.put("#pkName", partitionKeyName);
        expressionAttributeNames.put("#attrName", attributeName);

        // Create expression attribute values
        Map<String, AttributeValue> expressionAttributeValues = new HashMap<>();
        expressionAttributeValues.put(":pkValue", partitionKeyValue);

        // Add values for IN operator
        for (int i = 0; i < valuesList.size(); i++) {
            expressionAttributeValues.put(":val" + i, valuesList.get(i));
        }

        // Build the IN clause
        StringBuilder inClause = new StringBuilder();
        for (int i = 0; i < valuesList.size(); i++) {
            if (i > 0) {
                inClause.append(", ");
            }
            inClause.append(":val").append(i);
        }

        // Define the query parameters
        QueryRequest request = QueryRequest.builder()
            .tableName(tableName)
            .keyConditionExpression("#pkName = :pkValue")
            .filterExpression("#attrName IN (" + inClause.toString() + ")")
            .expressionAttributeNames(expressionAttributeNames)
```

```
            .expressionAttributeValues(expressionAttributeValues)
            .build();

        // Perform the query operation
        return dynamoDbClient.query(request);
    }

    /**
     * Queries a table using multiple OR conditions to compare multiple values
with a single attribute.
     *
     * <p>This method demonstrates the alternative approach to using the IN
operator,
     * by using multiple OR conditions.
     *
     * @param dynamoDbClient The DynamoDB client
     * @param tableName The name of the DynamoDB table
     * @param partitionKeyName The name of the partition key attribute
     * @param partitionKeyValue The value of the partition key to query
     * @param attributeName The name of the attribute to compare
     * @param valuesList List of values to compare against
     * @return The query response from DynamoDB
     * @throws DynamoDbException if an error occurs during the operation
     */
    public static QueryResponse compareWithOrConditions(
        DynamoDbClient dynamoDbClient,
        String tableName,
        String partitionKeyName,
        AttributeValue partitionKeyValue,
        String attributeName,
        List<AttributeValue> valuesList) {

        // Create expression attribute names
        Map<String, String> expressionAttributeNames = new HashMap<>();
        expressionAttributeNames.put("#pkName", partitionKeyName);
        expressionAttributeNames.put("#attrName", attributeName);

        // Create expression attribute values
        Map<String, AttributeValue> expressionAttributeValues = new HashMap<>();
        expressionAttributeValues.put(":pkValue", partitionKeyValue);

        // Add values for OR conditions
        for (int i = 0; i < valuesList.size(); i++) {
            expressionAttributeValues.put(":val" + i, valuesList.get(i));
```

```
        }

        // Build the OR conditions
        StringBuilder orConditions = new StringBuilder();
        for (int i = 0; i < valuesList.size(); i++) {
            if (i > 0) {
                orConditions.append(" OR ");
            }
            orConditions.append("#attrName = :val").append(i);
        }

        // Define the query parameters
        QueryRequest request = QueryRequest.builder()
            .tableName(tableName)
            .keyConditionExpression("#pkName = :pkValue")
            .filterExpression(orConditions.toString())
            .expressionAttributeNames(expressionAttributeNames)
            .expressionAttributeValues(expressionAttributeValues)
            .build();

        // Perform the query operation
        return dynamoDbClient.query(request);
    }

    /**
     * Compares the performance of using the IN operator versus multiple OR
conditions.
     *
     * <p>This method demonstrates the performance difference between using the
IN operator
     * and using multiple OR conditions.
     *
     * @param dynamoDbClient The DynamoDB client
     * @param tableName The name of the DynamoDB table
     * @param partitionKeyName The name of the partition key attribute
     * @param partitionKeyValue The value of the partition key to query
     * @param attributeName The name of the attribute to compare
     * @param valuesList List of values to compare against
     * @return Map containing the performance comparison results
     */
    public static Map<String, Object> comparePerformance(
        DynamoDbClient dynamoDbClient,
        String tableName,
        String partitionKeyName,
```

```
        AttributeValue partitionKeyValue,
        String attributeName,
        List<AttributeValue> valuesList) {

        Map<String, Object> results = new HashMap<>();

        try {
            // Measure performance of IN operator
            long inStartTime = System.nanoTime();
            QueryResponse inResponse = compareMultipleValues(
                dynamoDbClient, tableName, partitionKeyName, partitionKeyValue,
attributeName, valuesList);
            long inEndTime = System.nanoTime();
            long inDuration = inEndTime - inStartTime;

            // Measure performance of OR conditions
            long orStartTime = System.nanoTime();
            QueryResponse orResponse = compareWithOrConditions(
                dynamoDbClient, tableName, partitionKeyName, partitionKeyValue,
attributeName, valuesList);
            long orEndTime = System.nanoTime();
            long orDuration = orEndTime - orStartTime;

            // Record results
            results.put("inOperatorDuration", inDuration);
            results.put("orConditionsDuration", orDuration);
            results.put("inOperatorItems", inResponse.count());
            results.put("orConditionsItems", orResponse.count());
            results.put("inOperatorExpression", "IN operator with " +
valuesList.size() + " values");
            results.put("orConditionsExpression", valuesList.size() + " OR
conditions");
            results.put("success", true);

        } catch (DynamoDbException e) {
            results.put("success", false);
            results.put("error", e.getMessage());
        }

        return results;
    }

    /**
     * Scans a table using the IN operator with a large number of values.
```

```
     *
     * <p>This method demonstrates how to use the IN operator with a large number
of values,
     * which can help stay within the 300 operator limit.
     *
     * @param dynamoDbClient The DynamoDB client
     * @param tableName The name of the DynamoDB table
     * @param attributeName The name of the attribute to compare
     * @param valuesList List of values to compare against
     * @return The scan response from DynamoDB
     * @throws DynamoDbException if an error occurs during the operation
     */
    public static ScanResponse scanWithLargeInClause(
        DynamoDbClient dynamoDbClient, String tableName, String attributeName,
List<AttributeValue> valuesList) {

        // Create expression attribute names
        Map<String, String> expressionAttributeNames = new HashMap<>();
        expressionAttributeNames.put("#attrName", attributeName);

        // Create expression attribute values
        Map<String, AttributeValue> expressionAttributeValues = new HashMap<>();

        // Add values for IN operator
        for (int i = 0; i < valuesList.size(); i++) {
            expressionAttributeValues.put(":val" + i, valuesList.get(i));
        }

        // Build the IN clause
        StringBuilder inClause = new StringBuilder();
        for (int i = 0; i < valuesList.size(); i++) {
            if (i > 0) {
                inClause.append(", ");
            }
            inClause.append(":val").append(i);
        }

        // Define the scan parameters
        ScanRequest request = ScanRequest.builder()
            .tableName(tableName)
            .filterExpression("#attrName IN (" + inClause.toString() + ")")
            .expressionAttributeNames(expressionAttributeNames)
            .expressionAttributeValues(expressionAttributeValues)
            .build();
```

```
        // Perform the scan operation
        return dynamoDbClient.scan(request);
    }

    /**
     * Generates a list of sample values for testing.
     *
     * <p>Helper method to generate a list of sample values for testing.
     *
     * @param valueType The type of values to generate (string, number, or
boolean)
     * @param count The number of values to generate
     * @return List of generated attribute values
     */
    public static List<AttributeValue> generateSampleValues(String valueType, int
count) {
        List<AttributeValue> values = new ArrayList<>();

        for (int i = 0; i < count; i++) {
            AttributeValue value;

            switch (valueType.toLowerCase(Locale.ROOT)) {
                case "string":
                    value = AttributeValue.builder().s("Value" + i).build();
                    break;
                case "number":
                    value =
AttributeValue.builder().n(String.valueOf(i)).build();
                    break;
                case "boolean":
                    value = AttributeValue.builder().bool(i % 2 == 0).build();
                    break;
                default:
                    throw new IllegalArgumentException("Unsupported value type: "
+ valueType);
            }

            values.add(value);
        }

        return values;
    }
```

Beispiel für die Verwendung des Vergleichs mehrerer Werte mit. AWS SDK for Java 2.x

```
   public static void exampleUsage(DynamoDbClient dynamoDbClient, String
tableName) {
       System.out.println("Demonstrating how to compare multiple values with a
single attribute in DynamoDB");

       try {
           // Example 1: Using the IN operator
           System.out.println("\nExample 1: Using the IN operator");
           List<AttributeValue> categories = List.of(
               AttributeValue.builder().s("Electronics").build(),
               AttributeValue.builder().s("Computers").build(),
               AttributeValue.builder().s("Accessories").build());

           QueryResponse inResponse = compareMultipleValues(
               dynamoDbClient,
               tableName,
               "Department",
               AttributeValue.builder().s("Retail").build(),
               "Category",
               categories);

           System.out.println("Found " + inResponse.count() + " items using IN
operator");
           System.out.println("Items: " + inResponse.items());

           // Example 2: Using multiple OR conditions
           System.out.println("\nExample 2: Using multiple OR conditions");
           QueryResponse orResponse = compareWithOrConditions(
               dynamoDbClient,
               tableName,
               "Department",
               AttributeValue.builder().s("Retail").build(),
               "Category",
               categories);

           System.out.println("Found " + orResponse.count() + " items using OR
conditions");
           System.out.println("Items: " + orResponse.items());
```

```
            // Example 3: Performance comparison
            System.out.println("\nExample 3: Performance comparison");
            Map<String, Object> perfComparison = comparePerformance(
                dynamoDbClient,
                tableName,
                "Department",
                AttributeValue.builder().s("Retail").build(),
                "Category",
                categories);

            if ((boolean) perfComparison.get("success")) {
                System.out.println("IN operator duration: " +
perfComparison.get("inOperatorDuration") + " ns");
                System.out.println("OR conditions duration: " +
perfComparison.get("orConditionsDuration") + " ns");
                System.out.println("IN operator found " +
perfComparison.get("inOperatorItems") + " items");
                System.out.println("OR conditions found " +
perfComparison.get("orConditionsItems") + " items");
                System.out.println("Expression complexity comparison:");
                System.out.println("  IN operator: " +
perfComparison.get("inOperatorExpression"));
                System.out.println("  OR conditions: " +
perfComparison.get("orConditionsExpression"));
            } else {
                System.out.println("Performance comparison failed: " +
perfComparison.get("error"));
            }

            // Example 4: Using IN with a large number of values
            System.out.println("\nExample 4: Using IN with a large number of
values");
            List<AttributeValue> productIds = generateSampleValues("string", 20);

            ScanResponse largeInResponse = scanWithLargeInClause(dynamoDbClient,
tableName, "ProductId", productIds);

            System.out.println(
                "Found " + largeInResponse.count() + " items using IN with " +
productIds.size() + " values");

            // Explain the benefits of using IN
            System.out.println("\nKey points about using the IN operator in
DynamoDB:");
```

```
            System.out.println("1. The IN operator allows comparing a single
 attribute against multiple values");
            System.out.println("2. IN is more concise than using multiple OR
 conditions");
            System.out.println("3. IN counts as only 1 operator regardless of the
 number of values");
            System.out.println("4. Multiple OR conditions count as 1 operator per
 condition plus 1 per OR");
            System.out.println("5. Using IN helps stay within the 300 operator
 limit for complex expressions");
            System.out.println("6. IN can be used in filter expressions and
 condition expressions");
            System.out.println("7. The IN operator supports up to 100 comparison
 values");

        } catch (DynamoDbException e) {
            System.err.println("Error: " + e.getMessage());
            e.printStackTrace();
        }
    }
```

- Weitere API-Informationen finden Sie in den folgenden Themen der AWS SDK for Java 2.x - API-Referenz.
  - Query
  - Scan

JavaScript

SDK für JavaScript (v3)

Vergleichen Sie mehrere Werte mit einem einzigen Attribut mithilfe von AWS SDK für JavaScript.

```
const { DynamoDBClient } = require("@aws-sdk/client-dynamodb");
const {
  DynamoDBDocumentClient,
  ScanCommand,
  QueryCommand
} = require("@aws-sdk/lib-dynamodb");

/**
```

```
 * Query or scan a DynamoDB table to find items where an attribute matches any
 value from a list.
 *
 * This function demonstrates the use of the IN operator to compare a single
 attribute
 * against multiple possible values, which is more efficient than using multiple
 OR conditions.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {string} attributeName - The name of the attribute to compare against
 the values list
 * @param {Array} valuesList - List of values to compare the attribute against
 * @param {string} [partitionKeyName] - Optional name of the partition key
 attribute for query operations
 * @param {string} [partitionKeyValue] - Optional value of the partition key to
 query
 * @returns {Promise<Object>} - The response from DynamoDB containing the
 matching items
 */
async function compareMultipleValues(
  config,
  tableName,
  attributeName,
  valuesList,
  partitionKeyName,
  partitionKeyValue
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

  // Create the filter expression using the IN operator
  const filterExpression = `${attributeName} IN (${valuesList.map((_, index) =>
 `:val${index}`).join(', ')})`;

  // Create expression attribute values for the values list
  const expressionAttributeValues = valuesList.reduce((acc, val, index) => {
    acc[`:val${index}`] = val;
    return acc;
  }, {});

  // If partition key is provided, perform a query operation
  if (partitionKeyName && partitionKeyValue) {
```

```
    const keyCondition = `${partitionKeyName} = :partitionKey`;
    expressionAttributeValues[':partitionKey'] = partitionKeyValue;

    // Initialize array to collect all items
    let allItems = [];
    let lastEvaluatedKey;

    // Use pagination to get all results
    do {
      const params = {
        TableName: tableName,
        KeyConditionExpression: keyCondition,
        FilterExpression: filterExpression,
        ExpressionAttributeValues: expressionAttributeValues
      };

      // Add ExclusiveStartKey if we have a lastEvaluatedKey from a previous
query
      if (lastEvaluatedKey) {
        params.ExclusiveStartKey = lastEvaluatedKey;
      }

      const response = await docClient.send(new QueryCommand(params));

      // Add the items from this page to our collection
      if (response.Items && response.Items.length > 0) {
        allItems = [...allItems, ...response.Items];
      }

      // Get the key for the next page of results
      lastEvaluatedKey = response.LastEvaluatedKey;
    } while (lastEvaluatedKey);

    // Return the complete result
    return {
      Items: allItems,
      Count: allItems.length
    };
  } else {
    // Otherwise, perform a scan operation
    // Initialize array to collect all items
    let allItems = [];
    let lastEvaluatedKey;
```

```
    // Use pagination to get all results
    do {
      const params = {
        TableName: tableName,
        FilterExpression: filterExpression,
        ExpressionAttributeValues: expressionAttributeValues
      };

      // Add ExclusiveStartKey if we have a lastEvaluatedKey from a previous scan
      if (lastEvaluatedKey) {
        params.ExclusiveStartKey = lastEvaluatedKey;
      }

      const response = await docClient.send(new ScanCommand(params));

      // Add the items from this page to our collection
      if (response.Items && response.Items.length > 0) {
        allItems = [...allItems, ...response.Items];
      }

      // Get the key for the next page of results
      lastEvaluatedKey = response.LastEvaluatedKey;
    } while (lastEvaluatedKey);

    // Return the complete result
    return {
      Items: allItems,
      Count: allItems.length
    };
  }
}

/**
 * Alternative implementation using multiple OR conditions instead of the IN
 operator.
 *
 * This function is provided for comparison to show why using the IN operator is
 preferable.
 * With many values, this approach becomes verbose and less efficient.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {string} attributeName - The name of the attribute to compare against
 the values list
```

```
 * @param {Array} valuesList - List of values to compare the attribute against
 * @param {string} [partitionKeyName] - Optional name of the partition key
 attribute for query operations
 * @param {string} [partitionKeyValue] - Optional value of the partition key to
 query
 * @returns {Promise<Object>} - The response from DynamoDB containing the
 matching items
 */
async function compareWithOrConditions(
  config,
  tableName,
  attributeName,
  valuesList,
  partitionKeyName,
  partitionKeyValue
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

  // If no values provided, return empty result
  if (!valuesList || valuesList.length === 0) {
    return {
      Items: [],
      Count: 0
    };
  }

  // Create the filter expression using multiple OR conditions
  const filterConditions = valuesList.map((_, index) => `${attributeName} = :val
${index}`);
  const filterExpression = filterConditions.join(' OR ');

  // Create expression attribute values for the values list
  const expressionAttributeValues = valuesList.reduce((acc, val, index) => {
    acc[`:val${index}`] = val;
    return acc;
  }, {});

  // If partition key is provided, perform a query operation
  if (partitionKeyName && partitionKeyValue) {
    const keyCondition = `${partitionKeyName} = :partitionKey`;
    expressionAttributeValues[':partitionKey'] = partitionKeyValue;
```

```
    // Initialize array to collect all items
    let allItems = [];
    let lastEvaluatedKey;

    // Use pagination to get all results
    do {
      const params = {
        TableName: tableName,
        KeyConditionExpression: keyCondition,
        FilterExpression: filterExpression,
        ExpressionAttributeValues: expressionAttributeValues
      };

      // Add ExclusiveStartKey if we have a lastEvaluatedKey from a previous
query
      if (lastEvaluatedKey) {
        params.ExclusiveStartKey = lastEvaluatedKey;
      }

      const response = await docClient.send(new QueryCommand(params));

      // Add the items from this page to our collection
      if (response.Items && response.Items.length > 0) {
        allItems = [...allItems, ...response.Items];
      }

      // Get the key for the next page of results
      lastEvaluatedKey = response.LastEvaluatedKey;
    } while (lastEvaluatedKey);

    // Return the complete result
    return {
      Items: allItems,
      Count: allItems.length
    };
  } else {
    // Otherwise, perform a scan operation
    // Initialize array to collect all items
    let allItems = [];
    let lastEvaluatedKey;

    // Use pagination to get all results
    do {
      const params = {
```

```
        TableName: tableName,
        FilterExpression: filterExpression,
        ExpressionAttributeValues: expressionAttributeValues
      };

      // Add ExclusiveStartKey if we have a lastEvaluatedKey from a previous scan
      if (lastEvaluatedKey) {
        params.ExclusiveStartKey = lastEvaluatedKey;
      }

      const response = await docClient.send(new ScanCommand(params));

      // Add the items from this page to our collection
      if (response.Items && response.Items.length > 0) {
        allItems = [...allItems, ...response.Items];
      }

      // Get the key for the next page of results
      lastEvaluatedKey = response.LastEvaluatedKey;
    } while (lastEvaluatedKey);

    // Return the complete result
    return {
      Items: allItems,
      Count: allItems.length
    };
  }
}
```

Beispiel für die Verwendung des Vergleichs mehrerer Werte mit AWS SDK für JavaScript.

```
/**
 * Example of how to use the compareMultipleValues function.
 */
async function exampleUsage() {
  // Example parameters
  const config = { region: "us-west-2" };
  const tableName = "Products";
  const attributeName = "Category";
  const valuesList = ["Electronics", "Computers", "Accessories"];
```

```
 console.log(`Searching for products in any of these categories:
${valuesList.join(', ')}`);

 try {
   // Using the IN operator (recommended approach)
   console.log("\nApproach 1: Using the IN operator");
   const response = await compareMultipleValues(
     config,
     tableName,
     attributeName,
     valuesList
   );

   console.log(`Found ${response.Count} products in the specified categories`);

   // Using multiple OR conditions (alternative approach)
   console.log("\nApproach 2: Using multiple OR conditions");
   const response2 = await compareWithOrConditions(
     config,
     tableName,
     attributeName,
     valuesList
   );

   console.log(`Found ${response2.Count} products in the specified categories`);

   // Example with a query operation
   console.log("\nQuerying a specific manufacturer's products in multiple
categories");
   const partitionKeyName = "Manufacturer";
   const partitionKeyValue = "Acme";

   const response3 = await compareMultipleValues(
     config,
     tableName,
     attributeName,
     valuesList,
     partitionKeyName,
     partitionKeyValue
   );

   console.log(`Found ${response3.Count} Acme products in the specified
categories`);
```

```
    // Explain the benefits of using the IN operator
    console.log("\nBenefits of using the IN operator:");
    console.log("1. More concise expression compared to multiple OR conditions");
    console.log("2. Better readability and maintainability");
    console.log("3. Potentially better performance with large value lists");
    console.log("4. Simpler code that's less prone to errors");
    console.log("5. Easier to modify when adding or removing values");

  } catch (error) {
    console.error("Error:", error);
  }
}
```

- Weitere API-Informationen finden Sie in den folgenden Themen der AWS SDK für JavaScript -API-Referenz.
  - Query
  - Scan

Python

SDK für Python (Boto3)

Vergleichen Sie mehrere Werte mit einem einzigen Attribut mithilfe von AWS SDK für Python (Boto3).

```
import boto3
from boto3.dynamodb.conditions import Attr, Key
from typing import Any, Dict, List, Optional


def compare_multiple_values(
    table_name: str,
    attribute_name: str,
    values_list: List[Any],
    partition_key_name: Optional[str] = None,
    partition_key_value: Optional[str] = None,
) -> Dict[str, Any]:
    """
    Query or scan a DynamoDB table to find items where an attribute matches any
 value from a list.
```

```
    This function demonstrates the use of the IN operator to compare a single
attribute
    against multiple possible values, which is more efficient than using multiple
OR conditions.

    Args:
        table_name (str): The name of the DynamoDB table.
        attribute_name (str): The name of the attribute to compare against the
values list.
        values_list (List[Any]): List of values to compare the attribute against.
        partition_key_name (Optional[str]): The name of the partition key
attribute for query operations.
        partition_key_value (Optional[str]): The value of the partition key to
query.

    Returns:
        Dict[str, Any]: The response from DynamoDB containing the matching items.
    """
    # Initialize the DynamoDB resource
    dynamodb = boto3.resource("dynamodb")
    table = dynamodb.Table(table_name)

    # Create the filter expression using the is_in method
    filter_expression = Attr(attribute_name).is_in(values_list)

    # If partition key is provided, perform a query operation
    if partition_key_name and partition_key_value:
        key_condition = Key(partition_key_name).eq(partition_key_value)
        response = table.query(
            KeyConditionExpression=key_condition,
FilterExpression=filter_expression
        )
    else:
        # Otherwise, perform a scan operation
        response = table.scan(FilterExpression=filter_expression)

    # Handle pagination if there are more results
    items = response.get("Items", [])
    while "LastEvaluatedKey" in response:
        if partition_key_name and partition_key_value:
            response = table.query(
                KeyConditionExpression=key_condition,
                FilterExpression=filter_expression,
                ExclusiveStartKey=response["LastEvaluatedKey"],
```

```
            )
        else:
            response = table.scan(
                FilterExpression=filter_expression,
 ExclusiveStartKey=response["LastEvaluatedKey"]
            )
        items.extend(response.get("Items", []))

    # Return the complete result
    return {"Items": items, "Count": len(items)}


def compare_with_or_conditions(
    table_name: str,
    attribute_name: str,
    values_list: List[Any],
    partition_key_name: Optional[str] = None,
    partition_key_value: Optional[str] = None,
) -> Dict[str, Any]:
    """
    Alternative implementation using multiple OR conditions instead of the IN
 operator.

    This function is provided for comparison to show why using the IN operator is
 preferable.
    With many values, this approach becomes verbose and less efficient.

    Args:
        table_name (str): The name of the DynamoDB table.
        attribute_name (str): The name of the attribute to compare against the
 values list.
        values_list (List[Any]): List of values to compare the attribute against.
        partition_key_name (Optional[str]): The name of the partition key
 attribute for query operations.
        partition_key_value (Optional[str]): The value of the partition key to
 query.

    Returns:
        Dict[str, Any]: The response from DynamoDB containing the matching items.
    """
    # Initialize the DynamoDB resource
    dynamodb = boto3.resource("dynamodb")
    table = dynamodb.Table(table_name)
```

```
    # Create a filter expression with multiple OR conditions
    filter_expression = None
    for value in values_list:
        condition = Attr(attribute_name).eq(value)
        if filter_expression is None:
            filter_expression = condition
        else:
            filter_expression = filter_expression | condition

    # If partition key is provided, perform a query operation
    if partition_key_name and partition_key_value and filter_expression:
        key_condition = Key(partition_key_name).eq(partition_key_value)
        response = table.query(
            KeyConditionExpression=key_condition,
FilterExpression=filter_expression
        )
    elif filter_expression:
        # Otherwise, perform a scan operation
        response = table.scan(FilterExpression=filter_expression)
    else:
        # Return empty response if no values provided
        return {"Items": [], "Count": 0}

    # Handle pagination if there are more results
    items = response.get("Items", [])
    while "LastEvaluatedKey" in response:
        if partition_key_name and partition_key_value:
            response = table.query(
                KeyConditionExpression=key_condition,
                FilterExpression=filter_expression,
                ExclusiveStartKey=response["LastEvaluatedKey"],
            )
        else:
            response = table.scan(
                FilterExpression=filter_expression,
ExclusiveStartKey=response["LastEvaluatedKey"]
            )
        items.extend(response.get("Items", []))

    # Return the complete result
    return {"Items": items, "Count": len(items)}
```

Beispiel für die Verwendung des Vergleichs mehrerer Werte mit AWS SDK für Python (Boto3).

```
def example_usage():
    """Example of how to use the compare_multiple_values function."""
    # Example parameters
    table_name = "Products"
    attribute_name = "Category"
    values_list = ["Electronics", "Computers", "Accessories"]

    print(f"Searching for products in any of these categories: {values_list}")

    # Using the IN operator (recommended approach)
    print("\nApproach 1: Using the IN operator")
    response = compare_multiple_values(
        table_name=table_name, attribute_name=attribute_name,
 values_list=values_list
    )

    print(f"Found {response['Count']} products in the specified categories")

    # Using multiple OR conditions (alternative approach)
    print("\nApproach 2: Using multiple OR conditions")
    response2 = compare_with_or_conditions(
        table_name=table_name, attribute_name=attribute_name,
 values_list=values_list
    )

    print(f"Found {response2['Count']} products in the specified categories")

    # Example with a query operation
    print("\nQuerying a specific manufacturer's products in multiple categories")
    partition_key_name = "Manufacturer"
    partition_key_value = "Acme"

    response3 = compare_multiple_values(
        table_name=table_name,
        attribute_name=attribute_name,
        values_list=values_list,
        partition_key_name=partition_key_name,
        partition_key_value=partition_key_value,
    )
```

```
    print(f"Found {response3['Count']} Acme products in the specified
categories")

    # Explain the benefits of using the IN operator
    print("\nBenefits of using the IN operator:")
    print("1. More concise expression compared to multiple OR conditions")
    print("2. Better readability and maintainability")
    print("3. Potentially better performance with large value lists")
    print("4. Simpler code that's less prone to errors")
    print("5. Easier to modify when adding or removing values")
```

- Weitere API-Informationen finden Sie in den folgenden Themen der API-Referenz zum AWS SDK für Python (Boto3).
  - Query
  - Scan

Eine vollständige Liste der AWS SDK-Entwicklerhandbücher und Codebeispiele finden Sie unterDynamoDB mit einem SDK verwenden AWS. Dieses Thema enthält auch Informationen zu den ersten Schritten und Details zu früheren SDK-Versionen.

Bedingtes Aktualisieren eines DynamoDB-Elements mit einer TTL mithilfe eines SDK AWS

In den folgenden Codebeispielen wird gezeigt, wie die TTL eines Elements bedingungsabhängig aktualisiert wird.

Java

SDK für Java 2.x

Aktualisieren Sie die TTL für ein vorhandenes DynamoDB-Element in einer Tabelle mit einer Bedingung.

```
package com.amazon.samplelib.ttl;
```

```
import com.amazon.samplelib.CodeSampleUtils;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.AttributeValue;
import
 software.amazon.awssdk.services.dynamodb.model.ConditionalCheckFailedException;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.model.ResourceNotFoundException;
import software.amazon.awssdk.services.dynamodb.model.UpdateItemRequest;
import software.amazon.awssdk.services.dynamodb.model.UpdateItemResponse;

import java.util.Map;
import java.util.Optional;

/**
 * Updates an item in a DynamoDB table with TTL attributes using a conditional
 expression.
 * This class demonstrates how to conditionally update TTL expiration timestamps.
 */
public class UpdateTTLConditional {

    private static final String USAGE =
        """
            Usage:
                <tableName> <primaryKey> <sortKey> <region>
            Where:
                tableName - The Amazon DynamoDB table being queried.
                primaryKey - The name of the primary key. Also known as the hash
 or partition key.
                sortKey - The name of the sort key. Also known as the range
 attribute.
                region (optional) - The AWS region that the Amazon DynamoDB table
 is located in. (Default: us-east-1)
            """;
    private static final int DAYS_TO_EXPIRE = 90;
    private static final int SECONDS_PER_DAY = 24 * 60 * 60;
    private static final String PRIMARY_KEY_ATTR = "primaryKey";
    private static final String SORT_KEY_ATTR = "sortKey";
    private static final String UPDATED_AT_ATTR = "updatedAt";
    private static final String EXPIRE_AT_ATTR = "expireAt";
    private static final String UPDATE_EXPRESSION = "SET " + UPDATED_AT_ATTR +
 "=:c, " + EXPIRE_AT_ATTR + "=:e";
    private static final String CONDITION_EXPRESSION = "attribute_exists(" +
 PRIMARY_KEY_ATTR + ")";
```

```
   private static final String SUCCESS_MESSAGE = "%s UpdateItem operation with
TTL successful.";
   private static final String CONDITION_FAILED_MESSAGE = "Condition check
failed. Item does not exist.";
   private static final String TABLE_NOT_FOUND_ERROR = "Error: The Amazon
DynamoDB table \"%s\" can't be found.";

   private final DynamoDbClient dynamoDbClient;

   /**
    * Constructs an UpdateTTLConditional with a default DynamoDB client.
    */
   public UpdateTTLConditional() {
       this.dynamoDbClient = null;
   }

   /**
    * Constructs an UpdateTTLConditional with the specified DynamoDB client.
    *
    * @param dynamoDbClient The DynamoDB client to use
    */
   public UpdateTTLConditional(final DynamoDbClient dynamoDbClient) {
       this.dynamoDbClient = dynamoDbClient;
   }

   /**
    * Main method to demonstrate conditionally updating an item with TTL.
    *
    * @param args Command line arguments
    */
   public static void main(final String[] args) {
       try {
           int result = new UpdateTTLConditional().processArgs(args);
           System.exit(result);
       } catch (Exception e) {
           System.err.println(e.getMessage());
           System.exit(1);
       }
   }

   /**
    * Process command line arguments and conditionally update an item with TTL.
    *
    * @param args Command line arguments
```

```
     * @return 0 if successful, non-zero otherwise
     * @throws ResourceNotFoundException If the table doesn't exist
     * @throws DynamoDbException If an error occurs during the operation
     * @throws IllegalArgumentException If arguments are invalid
     */
    public int processArgs(final String[] args) {
        // Argument validation (remove or replace this line when reusing this
code)
        CodeSampleUtils.validateArgs(args, new int[] {3, 4}, USAGE);

        final String tableName = args[0];
        final String primaryKey = args[1];
        final String sortKey = args[2];
        final Region region = Optional.ofNullable(args.length > 3 ? args[3] :
null)
            .map(Region::of)
            .orElse(Region.US_EAST_1);

        // Get current time in epoch second format
        final long currentTime = System.currentTimeMillis() / 1000;

        // Calculate expiration time 90 days from now in epoch second format
        final long expireDate = currentTime + (DAYS_TO_EXPIRE * SECONDS_PER_DAY);

        // Create the key map for the item to update
        final Map<String, AttributeValue> keyMap = Map.of(
            PRIMARY_KEY_ATTR, AttributeValue.builder().s(primaryKey).build(),
            SORT_KEY_ATTR, AttributeValue.builder().s(sortKey).build());

        // Create the expression attribute values
        final Map<String, AttributeValue> expressionAttributeValues = Map.of(
            ":c",
AttributeValue.builder().n(String.valueOf(currentTime)).build(),
            ":e",
AttributeValue.builder().n(String.valueOf(expireDate)).build());

        final UpdateItemRequest request = UpdateItemRequest.builder()
            .tableName(tableName)
            .key(keyMap)
            .updateExpression(UPDATE_EXPRESSION)
            .conditionExpression(CONDITION_EXPRESSION)
            .expressionAttributeValues(expressionAttributeValues)
            .build();
```

```
        try (DynamoDbClient ddb = dynamoDbClient != null
            ? dynamoDbClient
            : DynamoDbClient.builder().region(region).build()) {
            final UpdateItemResponse response = ddb.updateItem(request);
            System.out.println(String.format(SUCCESS_MESSAGE, tableName));
            return 0;
        } catch (ConditionalCheckFailedException e) {
            System.err.println(CONDITION_FAILED_MESSAGE);
            throw e;
        } catch (ResourceNotFoundException e) {
            System.err.format(TABLE_NOT_FOUND_ERROR, tableName);
            throw e;
        } catch (DynamoDbException e) {
            System.err.println(e.getMessage());
            throw e;
        }
    }
}
```

- Einzelheiten zur API finden Sie UpdateItemin der AWS SDK for Java 2.x API-Referenz.

JavaScript

SDK für JavaScript (v3)

Aktualisieren Sie die TTL für ein vorhandenes DynamoDB-Element in einer Tabelle mit einer Bedingung.

```
import { DynamoDBClient, UpdateItemCommand } from "@aws-sdk/client-dynamodb";
import { marshall, unmarshall } from "@aws-sdk/util-dynamodb";

export const updateItemConditional = async (tableName, partitionKey, sortKey,
 region = 'us-east-1', newAttribute = 'default-value') => {
    const client = new DynamoDBClient({
        region: region,
        endpoint: `https://dynamodb.${region}.amazonaws.com`
    });

    const currentTime = Math.floor(Date.now() / 1000);

    const params = {
        TableName: tableName,
```

```
        Key: marshall({
            artist: partitionKey,
            album: sortKey
        }),
        UpdateExpression: "SET newAttribute = :newAttribute",
        ConditionExpression: "expireAt > :expiration",
        ExpressionAttributeValues: marshall({
            ':newAttribute': newAttribute,
            ':expiration': currentTime
        }),
        ReturnValues: "ALL_NEW"
    };

    try {
        const response = await client.send(new UpdateItemCommand(params));
        const responseData = unmarshall(response.Attributes);
        console.log("Item updated successfully: ", responseData);
        return responseData;
    } catch (error) {
        if (error.name === "ConditionalCheckFailedException") {
            console.log("Condition check failed: Item's 'expireAt' is expired.");
        } else {
            console.error("Error updating item: ", error);
        }
        throw error;
    }
};

// Example usage (commented out for testing)
// updateItemConditional('your-table-name', 'your-partition-key-value', 'your-
sort-key-value');
```

- Einzelheiten zur API finden Sie UpdateItemin der AWS SDK für JavaScript API-Referenz.

Python

SDK für Python (Boto3)

Aktualisieren Sie die TTL für ein vorhandenes DynamoDB-Element in einer Tabelle mit einer Bedingung.

```
from datetime import datetime, timedelta
```

```
import boto3
from botocore.exceptions import ClientError


def update_dynamodb_item_ttl(table_name, region, primary_key, sort_key,
 ttl_attribute):
    """
    Updates an existing record in a DynamoDB table with a new or updated TTL
 attribute.

    :param table_name: Name of the DynamoDB table
    :param region: AWS Region of the table - example `us-east-1`
    :param primary_key: one attribute known as the partition key.
    :param sort_key: Also known as a range attribute.
    :param ttl_attribute: name of the TTL attribute in the target DynamoDB table
    :return:
    """
    try:
        dynamodb = boto3.resource("dynamodb", region_name=region)
        table = dynamodb.Table(table_name)

        # Generate updated TTL in epoch second format
        updated_expiration_time = int((datetime.now() +
 timedelta(days=90)).timestamp())

        # Define the update expression for adding/updating a new attribute
        update_expression = "SET newAttribute = :val1"

        # Define the condition expression for checking if 'expireAt' is not
 expired
        condition_expression = "expireAt > :val2"

        # Define the expression attribute values
        expression_attribute_values = {":val1": ttl_attribute, ":val2":
 updated_expiration_time}

        response = table.update_item(
            Key={"primaryKey": primary_key, "sortKey": sort_key},
            UpdateExpression=update_expression,
            ConditionExpression=condition_expression,
            ExpressionAttributeValues=expression_attribute_values,
        )
```

```
        print("Item updated successfully.")
        return response["ResponseMetadata"]["HTTPStatusCode"]  # Ideally a 200 OK
    except ClientError as e:
        if e.response["Error"]["Code"] == "ConditionalCheckFailedException":
            print("Condition check failed: Item's 'expireAt' is expired.")
        else:
            print(f"Error updating item: {e}")
    except Exception as e:
        print(f"Error updating item: {e}")


# replace with your values
update_dynamodb_item_ttl(
    "your-table-name",
    "us-east-1",
    "your-partition-key-value",
    "your-sort-key-value",
    "your-ttl-attribute-value",
)
```

- Einzelheiten zur API finden Sie UpdateItemin AWS SDK for Python (Boto3) API Reference.

Eine vollständige Liste der AWS SDK-Entwicklerhandbücher und Codebeispiele finden Sie unter. DynamoDB mit einem SDK verwenden AWS Dieses Thema enthält auch Informationen zu den ersten Schritten und Details zu früheren SDK-Versionen.

Stellen Sie mithilfe eines SDK eine Connect zu einer lokalen DynamoDB-Instanz her AWS

Das folgende Codebeispiel zeigt, wie eine Endpunkt-URL überschrieben wird, um eine Verbindung zu einer lokalen Entwicklungsbereitstellung von DynamoDB und einem AWS SDK herzustellen.

Weitere Informationen finden Sie unter Lokale Version von DynamoDB.

Rust

SDK für Rust

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
/// Lists your tables from a local DynamoDB instance by setting the SDK Config's
/// endpoint_url and test_credentials.
#[tokio::main]
async fn main() {
    tracing_subscriber::fmt::init();

    let config = aws_config::defaults(aws_config::BehaviorVersion::latest())
        .test_credentials()
        // DynamoDB run locally uses port 8000 by default.
        .endpoint_url("http://localhost:8000")
        .load()
        .await;
    let dynamodb_local_config =
 aws_sdk_dynamodb::config::Builder::from(&config).build();

    let client = aws_sdk_dynamodb::Client::from_conf(dynamodb_local_config);

    let list_resp = client.list_tables().send().await;
    match list_resp {
        Ok(resp) => {
            println!("Found {} tables", resp.table_names().len());
            for name in resp.table_names() {
                println!("  {}", name);
            }
        }
        Err(err) => eprintln!("Failed to list local dynamodb tables: {err:?}"),
    }
}
```

Eine vollständige Liste der AWS SDK-Entwicklerhandbücher und Codebeispiele finden Sie unterDynamoDB mit einem SDK verwenden AWS. Dieses Thema enthält auch Informationen zu den ersten Schritten und Details zu früheren SDK-Versionen.

Zählen Sie Ausdrucksoperatoren in DynamoDB mit einem SDK AWS

In den folgenden Codebeispielen wird gezeigt, wie Ausdrucksoperatoren in DynamoDB gezählt werden.

- Machen Sie sich mit der Obergrenze von 300 Operatoren in DynamoDB vertraut.
- Zählen Sie Operatoren in komplexen Ausdrücken.
- Optimieren Sie Ausdrücke, um innerhalb der Grenzwerte zu bleiben.

Java

SDK für Java 2.x

Demonstrieren Sie das Zählen von Ausdrucksoperatoren mit AWS SDK for Java 2.x.

```
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.AttributeValue;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.model.UpdateItemRequest;
import software.amazon.awssdk.services.dynamodb.model.UpdateItemResponse;

import java.util.HashMap;
import java.util.Map;
import java.util.regex.Matcher;
import java.util.regex.Pattern;

    /**
     * Creates a complex filter expression with a specified number of conditions.
     *
     * <p>This method demonstrates how to generate a complex expression with
     * a specific number of operators to test the 300 operator limit.
     *
     * @param conditionsCount Number of conditions to include
     * @param useAnd Whether to use AND (true) or OR (false) between conditions
     * @return Map containing the filter expression, attribute values, and
 operator count
     */
```

```
   public static Map<String, Object> createComplexFilterExpression(int
conditionsCount, boolean useAnd) {
       // Initialize the expression parts and attribute values
       StringBuilder filterExpression = new StringBuilder();
       Map<String, AttributeValue> expressionAttributeValues = new HashMap<>();

       // Generate the specified number of conditions
       for (int i = 0; i < conditionsCount; i++) {
           // Add the operator between conditions (except for the first one)
           if (i > 0) {
               filterExpression.append(useAnd ? " AND " : " OR ");
           }

           // Alternate between different comparison operators for variety
           String valueKey = ":val" + i;

           switch (i % 5) {
               case 0:
                   filterExpression.append("attribute").append(i).append(" =
").append(valueKey);
                   expressionAttributeValues.put(
                       valueKey, AttributeValue.builder().s("value" +
i).build());
                   break;
               case 1:
                   filterExpression.append("attribute").append(i).append(" >
").append(valueKey);
                   expressionAttributeValues.put(
                       valueKey,
AttributeValue.builder().n(String.valueOf(i)).build());
                   break;
               case 2:
                   filterExpression.append("attribute").append(i).append(" <
").append(valueKey);
                   expressionAttributeValues.put(
                       valueKey,
                       AttributeValue.builder().n(String.valueOf(i *
10)).build());
                   break;
               case 3:
                   filterExpression
                       .append("contains(attribute")
                       .append(i)
                       .append(", ")
```

```
                        .append(valueKey)
                        .append(")");
                    expressionAttributeValues.put(
                        valueKey, AttributeValue.builder().s("substring" +
i).build());
                    break;
                case 4:
                    filterExpression
                        .append("attribute_exists(attribute")
                        .append(i)
                        .append(")");
                    break;
                default:
                    // This case will never be reached, but added to satisfy
checkstyle
                    break;
            }
        }

        // Calculate the operator count
        // Each condition has 1 operator (=, >, <, contains, attribute_exists)
        // Each AND or OR between conditions is 1 operator
        int operatorCount = conditionsCount + (conditionsCount > 0 ?
conditionsCount - 1 : 0);

        // Create the result map
        Map<String, Object> result = new HashMap<>();
        result.put("filterExpression", filterExpression.toString());
        result.put("expressionAttributeValues", expressionAttributeValues);
        result.put("operatorCount", operatorCount);

        return result;
    }

    /**
     * Creates a complex update expression with a specified number of operations.
     *
     * <p>This method demonstrates how to generate a complex update expression
with
     * a specific number of operators to test the 300 operator limit.
     *
     * @param operationsCount Number of operations to include
     * @return Map containing the update expression, attribute values, and
operator count
```

```
    */
   public static Map<String, Object> createComplexUpdateExpression(int
operationsCount) {
       // Initialize the expression parts and attribute values
       StringBuilder updateExpression = new StringBuilder("SET ");
       Map<String, AttributeValue> expressionAttributeValues = new HashMap<>();

       // Generate the specified number of SET operations
       for (int i = 0; i < operationsCount; i++) {
           // Add comma between operations (except for the first one)
           if (i > 0) {
               updateExpression.append(", ");
           }

           // Alternate between different types of SET operations
           String valueKey = ":val" + i;

           switch (i % 3) {
               case 0:
                   // Simple assignment (1 operator: =)
                   updateExpression.append("attribute").append(i).append(" =
").append(valueKey);
                   expressionAttributeValues.put(
                       valueKey, AttributeValue.builder().s("value" +
i).build());
                   break;
               case 1:
                   // Addition (2 operators: = and +)
                   updateExpression
                       .append("attribute")
                       .append(i)
                       .append(" = attribute")
                       .append(i)
                       .append(" + ")
                       .append(valueKey);
                   expressionAttributeValues.put(
                       valueKey,
AttributeValue.builder().n(String.valueOf(i)).build());
                   break;
               case 2:
                   // Conditional assignment with if_not_exists (2 operators: =
and if_not_exists)
                   updateExpression
                       .append("attribute")
```

```
                        .append(i)
                        .append(" = if_not_exists(attribute")
                        .append(i)
                        .append(", ")
                        .append(valueKey)
                        .append(")");
                    expressionAttributeValues.put(
                        valueKey,
                        AttributeValue.builder().n(String.valueOf(i *
10)).build());
                    break;
                default:
                    // This case will never be reached, but added to satisfy
checkstyle
                    break;
            }
        }

        // Calculate the operator count
        // Each operation has 1-2 operators as noted above
        int operatorCount = 0;
        for (int i = 0; i < operationsCount; i++) {
            operatorCount += (i % 3 == 0) ? 1 : 2;
        }

        // Create the result map
        Map<String, Object> result = new HashMap<>();
        result.put("updateExpression", updateExpression.toString());
        result.put("expressionAttributeValues", expressionAttributeValues);
        result.put("operatorCount", operatorCount);

        return result;
    }

    /**
     * Test the operator limit by attempting an operation with a complex
expression.
     *
     * <p>This method demonstrates what happens when an expression approaches or
     * exceeds the 300 operator limit.
     *
     * @param dynamoDbClient The DynamoDB client
     * @param tableName The name of the DynamoDB table
     * @param key The key of the item to update
```

```
     * @param operatorCount Target number of operators to include
     * @return Map containing the result of the operation attempt
     */
    public static Map<String, Object> testOperatorLimit(
        DynamoDbClient dynamoDbClient, String tableName, Map<String,
AttributeValue> key, int operatorCount) {

        // Create a complex update expression with the specified operator count
        Map<String, Object> expressionData =
            createComplexUpdateExpression((int) Math.ceil(operatorCount /
1.5)); // Adjust to get close to target count

        String updateExpression = (String)
expressionData.get("updateExpression");
        @SuppressWarnings("unchecked")
        Map<String, AttributeValue> expressionAttributeValues =
            (Map<String, AttributeValue>)
expressionData.get("expressionAttributeValues");
        int actualCount = (int) expressionData.get("operatorCount");

        System.out.println("Generated update expression with approximately " +
actualCount + " operators");

        // Define the update parameters
        UpdateItemRequest request = UpdateItemRequest.builder()
            .tableName(tableName)
            .key(key)
            .updateExpression(updateExpression)
            .expressionAttributeValues(expressionAttributeValues)
            .returnValues("UPDATED_NEW")
            .build();

        try {
            // Attempt the update operation
            UpdateItemResponse response = dynamoDbClient.updateItem(request);

            Map<String, Object> result = new HashMap<>();
            result.put("success", true);
            result.put("message", "Operation succeeded with " + actualCount + "
operators");
            result.put("data", response);
            return result;

        } catch (DynamoDbException e) {
```

```
            // Check if the error is due to exceeding the operator limit
            if (e.getMessage().contains("too many operators")) {
                Map<String, Object> result = new HashMap<>();
                result.put("success", false);
                result.put("message", "Operation failed: " + e.getMessage());
                result.put("operatorCount", actualCount);
                return result;
            }

            // Return other errors
            Map<String, Object> result = new HashMap<>();
            result.put("success", false);
            result.put("message", "Operation failed: " + e.getMessage());
            result.put("error", e);
            return result;
        }
    }

    /**
     * Break down a complex expression into multiple simpler operations.
     *
     * <p>This method demonstrates how to handle expressions that would exceed
     * the 300 operator limit by breaking them into multiple operations.
     *
     * @param dynamoDbClient The DynamoDB client
     * @param tableName The name of the DynamoDB table
     * @param key The key of the item to update
     * @param totalOperations Total number of operations to perform
     * @return Map containing the results of the operations
     */
    public static Map<String, Object> breakDownComplexExpression(
        DynamoDbClient dynamoDbClient, String tableName, Map<String,
AttributeValue> key, int totalOperations) {

        // Calculate how many operations we can safely include in each batch
        // Using 150 as a conservative limit (well below 300)
        final int operationsPerBatch = 100;
        final int batchCount = (int) Math.ceil((double) totalOperations /
operationsPerBatch);

        System.out.println("Breaking down " + totalOperations + " operations into
" + batchCount + " batches");

        Map<String, Object> results = new HashMap<>();
```

```
        results.put("totalBatches", batchCount);

        Map<Integer, Map<String, Object>> batchResults = new HashMap<>();

        // Process each batch
        for (int batch = 0; batch < batchCount; batch++) {
            // Calculate the operations for this batch
            int batchStart = batch * operationsPerBatch;
            int batchEnd = Math.min(batchStart + operationsPerBatch,
totalOperations);
            int batchSize = batchEnd - batchStart;

            System.out.println(
                "Processing batch " + (batch + 1) + "/" + batchCount + " with " +
batchSize + " operations");

            // Create an update expression for this batch
            Map<String, Object> expressionData =
createComplexUpdateExpression(batchSize);

            String updateExpression = (String)
expressionData.get("updateExpression");
            @SuppressWarnings("unchecked")
            Map<String, AttributeValue> expressionAttributeValues =
                (Map<String, AttributeValue>)
expressionData.get("expressionAttributeValues");
            int operatorCount = (int) expressionData.get("operatorCount");

            // Define the update parameters
            UpdateItemRequest request = UpdateItemRequest.builder()
                .tableName(tableName)
                .key(key)
                .updateExpression(updateExpression)
                .expressionAttributeValues(expressionAttributeValues)
                .returnValues("UPDATED_NEW")
                .build();

            try {
                // Perform the update operation for this batch
                UpdateItemResponse response = dynamoDbClient.updateItem(request);

                Map<String, Object> batchResult = new HashMap<>();
                batchResult.put("batch", batch + 1);
                batchResult.put("success", true);
```

```
                batchResult.put("operatorCount", operatorCount);
                batchResult.put("attributes", response.attributes());

                batchResults.put(batch, batchResult);

            } catch (DynamoDbException e) {
                Map<String, Object> batchResult = new HashMap<>();
                batchResult.put("batch", batch + 1);
                batchResult.put("success", false);
                batchResult.put("operatorCount", operatorCount);
                batchResult.put("error", e.getMessage());

                batchResults.put(batch, batchResult);

                // Continue with next batch instead of breaking
                continue;
            }
        }

        results.put("results", batchResults);
        return results;
    }

    /**
     * Count operators in a DynamoDB expression based on the rules in the
documentation.
     *
     * <p>This method demonstrates how operators are counted according to the
     * DynamoDB documentation.
     *
     * @param expression The DynamoDB expression to analyze
     * @return Map containing the breakdown of operator counts
     */
    public static Map<String, Integer> countOperatorsInExpression(String
expression) {
        // Initialize counters for different operator types
        Map<String, Integer> counts = new HashMap<>();
        counts.put("comparisonOperators", 0);
        counts.put("logicalOperators", 0);
        counts.put("functions", 0);
        counts.put("arithmeticOperators", 0);
        counts.put("specialOperators", 0);
        counts.put("total", 0);
```

```
        // Count comparison operators (=, <>, <, <=, >, >=)
        // This is a simplified approach and may not catch all cases
        int comparisonCount = 0;
        Pattern comparisonPattern = Pattern.compile("(=|<>|<=|>=|<|>)");
        Matcher comparisonMatcher = comparisonPattern.matcher(expression);
        while (comparisonMatcher.find()) {
            comparisonCount++;
        }
        counts.put("comparisonOperators", comparisonCount);

        // Count logical operators (AND, OR, NOT)
        int andCount = countOccurrences(expression, "\\bAND\\b");
        int orCount = countOccurrences(expression, "\\bOR\\b");
        int notCount = countOccurrences(expression, "\\bNOT\\b");
        counts.put("logicalOperators", andCount + orCount + notCount);

        // Count functions (attribute_exists, attribute_not_exists,
 attribute_type, begins_with, contains, size)
        int functionCount = countOccurrences(
            expression,
            "\\b(attribute_exists|attribute_not_exists|attribute_type|
begins_with|contains|size|if_not_exists)\\(");
        counts.put("functions", functionCount);

        // Count arithmetic operators (+ and -)
        // This is a simplified approach and may not catch all cases
        int arithmeticCount = 0;
        Pattern arithmeticPattern = Pattern.compile("[a-zA-Z0-9_)\\]]\\s*[\\+\
\-]\\s*[a-zA-Z0-9_:(]");
        Matcher arithmeticMatcher = arithmeticPattern.matcher(expression);
        while (arithmeticMatcher.find()) {
            arithmeticCount++;
        }
        counts.put("arithmeticOperators", arithmeticCount);

        // Count special operators (BETWEEN, IN)
        int betweenCount = countOccurrences(expression, "\\bBETWEEN\\b");
        int inCount = countOccurrences(expression, "\\bIN\\b");
        counts.put("specialOperators", betweenCount + inCount);

        // Add extra operators for BETWEEN (each BETWEEN includes an AND)
        int currentLogicalOps = counts.getOrDefault("logicalOperators", 0);
        counts.put("logicalOperators", currentLogicalOps + betweenCount);
```

```
        // Calculate total
        int total = counts.getOrDefault("comparisonOperators", 0)
            + counts.getOrDefault("logicalOperators", 0)
            + counts.getOrDefault("functions", 0)
            + counts.getOrDefault("arithmeticOperators", 0)
            + counts.getOrDefault("specialOperators", 0);
        counts.put("total", total);

        return counts;
    }

    /**
     * Helper method to count occurrences of a pattern in a string.
     *
     * @param text The text to search in
     * @param regex The regular expression pattern to search for
     * @return The number of occurrences
     */
    private static int countOccurrences(String text, String regex) {
        final Pattern pattern = Pattern.compile(regex);
        final Matcher matcher = pattern.matcher(text);
        int count = 0;
        while (matcher.find()) {
            count++;
        }
        return count;
    }
```

Beispiel für die Verwendung des Ausdrucksoperators Counting With AWS SDK for Java 2.x.

```
    public static void exampleUsage(DynamoDbClient dynamoDbClient, String
tableName) {
        // Example key
        Map<String, AttributeValue> key = new HashMap<>();
        key.put("ProductId", AttributeValue.builder().s("P12345").build());

        System.out.println("Demonstrating DynamoDB expression operator counting
and the 300 operator limit");

        try {
            // Example 1: Analyze a simple expression
            System.out.println("\nExample 1: Analyzing a simple expression");
```

```
            String simpleExpression = "Price = :price AND Rating > :rating AND
Category IN (:cat1, :cat2, :cat3)";
            Map<String, Integer> simpleCount =
countOperatorsInExpression(simpleExpression);

            System.out.println("Expression: " + simpleExpression);
            System.out.println("Operator count breakdown:");
            System.out.println("- Comparison operators: " +
simpleCount.get("comparisonOperators"));
            System.out.println("- Logical operators: " +
simpleCount.get("logicalOperators"));
            System.out.println("- Functions: " + simpleCount.get("functions"));
            System.out.println("- Arithmetic operators: " +
simpleCount.get("arithmeticOperators"));
            System.out.println("- Special operators: " +
simpleCount.get("specialOperators"));
            System.out.println("- Total operators: " + simpleCount.get("total"));

            // Example 2: Analyze a complex expression
            System.out.println("\nExample 2: Analyzing a complex expression");
            String complexExpression = "(attribute_exists(Category) AND Size
BETWEEN :min AND :max) OR "
                + "(Price > :price AND contains(Description, :keyword) AND "
                + "(Rating >= :minRating OR Reviews > :minReviews))";
            Map<String, Integer> complexCount =
countOperatorsInExpression(complexExpression);

            System.out.println("Expression: " + complexExpression);
            System.out.println("Operator count breakdown:");
            System.out.println("- Comparison operators: " +
complexCount.get("comparisonOperators"));
            System.out.println("- Logical operators: " +
complexCount.get("logicalOperators"));
            System.out.println("- Functions: " + complexCount.get("functions"));
            System.out.println("- Arithmetic operators: " +
complexCount.get("arithmeticOperators"));
            System.out.println("- Special operators: " +
complexCount.get("specialOperators"));
            System.out.println("- Total operators: " +
complexCount.get("total"));

            // Example 3: Test approaching the operator limit
            System.out.println("\nExample 3: Testing an expression approaching
the operator limit");
```

```
            Map<String, Object> approachingLimit =
testOperatorLimit(dynamoDbClient, tableName, key, 290);
            System.out.println(approachingLimit.get("message"));

            // Example 4: Test exceeding the operator limit
            System.out.println("\nExample 4: Testing an expression exceeding the
operator limit");
            Map<String, Object> exceedingLimit =
testOperatorLimit(dynamoDbClient, tableName, key, 310);
            System.out.println(exceedingLimit.get("message"));

            // Example 5: Breaking down a complex expression
            System.out.println("\nExample 5: Breaking down a complex expression
into multiple operations");
            Map<String, Object> breakdownResult =
breakDownComplexExpression(dynamoDbClient, tableName, key, 500);
            @SuppressWarnings("unchecked")
            Map<Integer, Map<String, Object>> results =
                (Map<Integer, Map<String, Object>>)
breakdownResult.get("results");
            System.out.println(
                "Processed " + results.size() + " of " +
breakdownResult.get("totalBatches") + " batches");

            // Explain the operator counting rules
            System.out.println("\nKey points about DynamoDB expression operator
counting:");
            System.out.println("1. The maximum number of operators in any
expression is 300");
            System.out.println("2. Each comparison operator (=, <>, <, <=, >, >=)
counts as 1 operator");
            System.out.println("3. Each logical operator (AND, OR, NOT) counts as
1 operator");
            System.out.println("4. Each function call (attribute_exists,
contains, etc.) counts as 1 operator");
            System.out.println("5. Each arithmetic operator (+ or -) counts as 1
operator");
            System.out.println("6. BETWEEN counts as 2 operators (BETWEEN itself
and the AND within it)");
            System.out.println("7. IN counts as 1 operator regardless of the
number of values");
            System.out.println("8. Parentheses for grouping and attribute paths
don't count as operators");
```

```
            System.out.println("9. When you exceed the limit, the error always
 reports '301 operators'");
            System.out.println("10. For complex operations, break them into
 multiple smaller operations");

        } catch (Exception e) {
            System.err.println("Error: " + e.getMessage());
            e.printStackTrace();
        }
    }
```

- Einzelheiten zur API finden Sie UpdateItemunter AWS SDK for Java 2.x API-Referenz.

JavaScript

SDK für JavaScript (v3)

Demonstrieren Sie das Zählen von Ausdrucksoperatoren mit AWS SDK für JavaScript.

```
const { DynamoDBClient } = require("@aws-sdk/client-dynamodb");
const {
  DynamoDBDocumentClient,
  UpdateCommand,
  QueryCommand
} = require("@aws-sdk/lib-dynamodb");

/**
 * Create a complex filter expression with a specified number of conditions.
 *
 * This function demonstrates how to generate a complex expression with
 * a specific number of operators to test the 300 operator limit.
 *
 * @param {number} conditionsCount - Number of conditions to include
 * @param {boolean} useAnd - Whether to use AND (true) or OR (false) between
 conditions
 * @returns {Object} - Object containing the filter expression and attribute
 values
 */
function createComplexFilterExpression(conditionsCount, useAnd = true) {
  // Initialize the expression parts and attribute values
  const conditions = [];
  const expressionAttributeValues = {};
```

```
  // Generate the specified number of conditions
  for (let i = 0; i < conditionsCount; i++) {
    // Alternate between different comparison operators for variety
    let condition;
    const valueKey = `:val${i}`;

    switch (i % 5) {
      case 0:
        condition = `attribute${i} = ${valueKey}`;
        expressionAttributeValues[valueKey] = `value${i}`;
        break;
      case 1:
        condition = `attribute${i} > ${valueKey}`;
        expressionAttributeValues[valueKey] = i;
        break;
      case 2:
        condition = `attribute${i} < ${valueKey}`;
        expressionAttributeValues[valueKey] = i * 10;
        break;
      case 3:
        condition = `contains(attribute${i}, ${valueKey})`;
        expressionAttributeValues[valueKey] = `substring${i}`;
        break;
      case 4:
        condition = `attribute_exists(attribute${i})`;
        break;
    }

    conditions.push(condition);
  }

  // Join the conditions with AND or OR
  const operator = useAnd ? " AND " : " OR ";
  const filterExpression = conditions.join(operator);

  // Calculate the operator count
  // Each condition has 1 operator (=, >, <, contains, attribute_exists)
  // Each AND or OR between conditions is 1 operator
  const operatorCount = conditionsCount + (conditionsCount > 0 ? conditionsCount
- 1 : 0);

  return {
    filterExpression,
```

```
    expressionAttributeValues,
    operatorCount
  };
}

/**
 * Create a complex update expression with a specified number of operations.
 *
 * This function demonstrates how to generate a complex update expression with
 * a specific number of operators to test the 300 operator limit.
 *
 * @param {number} operationsCount - Number of operations to include
 * @returns {Object} - Object containing the update expression and attribute
 values
 */
function createComplexUpdateExpression(operationsCount) {
  // Initialize the expression parts and attribute values
  const setOperations = [];
  const expressionAttributeValues = {};

  // Generate the specified number of SET operations
  for (let i = 0; i < operationsCount; i++) {
    // Alternate between different types of SET operations
    let operation;
    const valueKey = `:val${i}`;

    switch (i % 3) {
      case 0:
        // Simple assignment (1 operator: =)
        operation = `attribute${i} = ${valueKey}`;
        expressionAttributeValues[valueKey] = `value${i}`;
        break;
      case 1:
        // Addition (2 operators: = and +)
        operation = `attribute${i} = attribute${i} + ${valueKey}`;
        expressionAttributeValues[valueKey] = i;
        break;
      case 2:
        // Conditional assignment with if_not_exists (2 operators: = and
 if_not_exists)
        operation = `attribute${i} = if_not_exists(attribute${i}, ${valueKey})`;
        expressionAttributeValues[valueKey] = i * 10;
        break;
    }
```

```
    setOperations.push(operation);
  }

  // Create the update expression
  const updateExpression = `SET ${setOperations.join(", ")}`;

  // Calculate the operator count
  // Each operation has 1-2 operators as noted above
  let operatorCount = 0;
  for (let i = 0; i < operationsCount; i++) {
    operatorCount += (i % 3 === 0) ? 1 : 2;
  }

  return {
    updateExpression,
    expressionAttributeValues,
    operatorCount
  };
}

/**
 * Test the operator limit by attempting an operation with a complex expression.
 *
 * This function demonstrates what happens when an expression approaches or
 * exceeds the 300 operator limit.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The key of the item to update
 * @param {number} operatorCount - Target number of operators to include
 * @returns {Promise<Object>} - Result of the operation attempt
 */
async function testOperatorLimit(
  config,
  tableName,
  key,
  operatorCount
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

  // Create a complex update expression with the specified operator count
```

```
  const { updateExpression, expressionAttributeValues, operatorCount:
 actualCount } =
    createComplexUpdateExpression(Math.ceil(operatorCount / 1.5)); // Adjust to
 get close to target count

  console.log(`Generated update expression with approximately ${actualCount}
 operators`);

  // Define the update parameters
  const params = {
    TableName: tableName,
    Key: key,
    UpdateExpression: updateExpression,
    ExpressionAttributeValues: expressionAttributeValues,
    ReturnValues: "UPDATED_NEW"
  };

  try {
    // Attempt the update operation
    const response = await docClient.send(new UpdateCommand(params));
    return {
      success: true,
      message: `Operation succeeded with ${actualCount} operators`,
      data: response
    };
  } catch (error) {
    // Check if the error is due to exceeding the operator limit
    if (error.name === "ValidationException" &&
        error.message.includes("too many operators")) {
      return {
        success: false,
        message: `Operation failed: ${error.message}`,
        operatorCount: actualCount
      };
    }

    // Return other errors
    return {
      success: false,
      message: `Operation failed: ${error.message}`,
      error
    };
  }
}
```

```
/**
 * Break down a complex expression into multiple simpler operations.
 *
 * This function demonstrates how to handle expressions that would exceed
 * the 300 operator limit by breaking them into multiple operations.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The key of the item to update
 * @param {number} totalOperations - Total number of operations to perform
 * @returns {Promise<Object>} - Result of the operations
 */
async function breakDownComplexExpression(
  config,
  tableName,
  key,
  totalOperations
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

  // Calculate how many operations we can safely include in each batch
  // Using 150 as a conservative limit (well below 300)
  const operationsPerBatch = 100;
  const batchCount = Math.ceil(totalOperations / operationsPerBatch);

  console.log(`Breaking down ${totalOperations} operations into ${batchCount}
 batches`);

  const results = [];

  // Process each batch
  for (let batch = 0; batch < batchCount; batch++) {
    // Calculate the operations for this batch
    const batchStart = batch * operationsPerBatch;
    const batchEnd = Math.min(batchStart + operationsPerBatch, totalOperations);
    const batchSize = batchEnd - batchStart;

    console.log(`Processing batch ${batch + 1}/${batchCount} with ${batchSize}
 operations`);

    // Create an update expression for this batch
```

```
    const { updateExpression, expressionAttributeValues, operatorCount } =
      createComplexUpdateExpression(batchSize);

    // Define the update parameters
    const params = {
      TableName: tableName,
      Key: key,
      UpdateExpression: updateExpression,
      ExpressionAttributeValues: expressionAttributeValues,
      ReturnValues: "UPDATED_NEW"
    };

    try {
      // Perform the update operation for this batch
      const response = await docClient.send(new UpdateCommand(params));

      results.push({
        batch: batch + 1,
        success: true,
        operatorCount,
        attributes: response.Attributes
      });
    } catch (error) {
      results.push({
        batch: batch + 1,
        success: false,
        operatorCount,
        error: error.message
      });

      // Stop processing if an error occurs
      break;
    }
  }

  return {
    totalBatches: batchCount,
    results
  };
}

/**
 * Count operators in a DynamoDB expression based on the rules in the
 documentation.
```

```
 *
 * This function demonstrates how operators are counted according to the
 * DynamoDB documentation.
 *
 * @param {string} expression - The DynamoDB expression to analyze
 * @returns {Object} - Breakdown of operator counts
 */
function countOperatorsInExpression(expression) {
  // Initialize counters for different operator types
  const counts = {
    comparisonOperators: 0,
    logicalOperators: 0,
    functions: 0,
    arithmeticOperators: 0,
    specialOperators: 0,
    total: 0
  };

  // Count comparison operators (=, <>, <, <=, >, >=)
  const comparisonRegex = /[^<>]=[^=]|<>|<=|>=|[^<]>[^=]|[^>]<[^=]/g;
  const comparisonMatches = expression.match(comparisonRegex) || [];
  counts.comparisonOperators = comparisonMatches.length;

  // Count logical operators (AND, OR, NOT)
  const andMatches = expression.match(/\bAND\b/g) || [];
  const orMatches = expression.match(/\bOR\b/g) || [];
  const notMatches = expression.match(/\bNOT\b/g) || [];
  counts.logicalOperators = andMatches.length + orMatches.length +
 notMatches.length;

  // Count functions (attribute_exists, attribute_not_exists, attribute_type,
 begins_with, contains, size)
  const functionRegex = /\b(attribute_exists|attribute_not_exists|attribute_type|
begins_with|contains|size|if_not_exists)\(/g;
  const functionMatches = expression.match(functionRegex) || [];
  counts.functions = functionMatches.length;

  // Count arithmetic operators (+ and -)
  const arithmeticMatches = expression.match(/[a-zA-Z0-9_)\]]\s*[\+\-]\s*[a-zA-
Z0-9_(:]/g) || [];
  counts.arithmeticOperators = arithmeticMatches.length;

  // Count special operators (BETWEEN, IN)
  const betweenMatches = expression.match(/\bBETWEEN\b/g) || [];
```

```
  const inMatches = expression.match(/\bIN\b/g) || [];
  counts.specialOperators = betweenMatches.length + inMatches.length;

  // Add extra operators for BETWEEN (each BETWEEN includes an AND)
  counts.logicalOperators += betweenMatches.length;

  // Calculate total
  counts.total = counts.comparisonOperators +
                 counts.logicalOperators +
                 counts.functions +
                 counts.arithmeticOperators +
                 counts.specialOperators;

  return counts;
}
```

Beispiel für die Verwendung des Ausdrucksoperators Counting with AWS SDK für JavaScript.

```
/**
 * Example of how to work with expression operator counting.
 */
async function exampleUsage() {
  // Example parameters
  const config = { region: "us-west-2" };
  const tableName = "Products";
  const key = { ProductId: "P12345" };

  console.log("Demonstrating DynamoDB expression operator counting and the 300
 operator limit");

  try {
    // Example 1: Analyze a simple expression
    console.log("\nExample 1: Analyzing a simple expression");
    const simpleExpression = "Price = :price AND Rating > :rating AND Category IN
 (:cat1, :cat2, :cat3)";
    const simpleCount = countOperatorsInExpression(simpleExpression);

    console.log(`Expression: ${simpleExpression}`);
    console.log("Operator count breakdown:");
    console.log(`- Comparison operators: ${simpleCount.comparisonOperators}`);
    console.log(`- Logical operators: ${simpleCount.logicalOperators}`);
    console.log(`- Functions: ${simpleCount.functions}`);
```

```
   console.log(`- Arithmetic operators: ${simpleCount.arithmeticOperators}`);
   console.log(`- Special operators: ${simpleCount.specialOperators}`);
   console.log(`- Total operators: ${simpleCount.total}`);

   // Example 2: Analyze a complex expression
   console.log("\nExample 2: Analyzing a complex expression");
   const complexExpression =
     "(attribute_exists(Category) AND Size BETWEEN :min AND :max) OR " +
     "(Price > :price AND contains(Description, :keyword) AND " +
     "(Rating >= :minRating OR Reviews > :minReviews))";
   const complexCount = countOperatorsInExpression(complexExpression);

   console.log(`Expression: ${complexExpression}`);
   console.log("Operator count breakdown:");
   console.log(`- Comparison operators: ${complexCount.comparisonOperators}`);
   console.log(`- Logical operators: ${complexCount.logicalOperators}`);
   console.log(`- Functions: ${complexCount.functions}`);
   console.log(`- Arithmetic operators: ${complexCount.arithmeticOperators}`);
   console.log(`- Special operators: ${complexCount.specialOperators}`);
   console.log(`- Total operators: ${complexCount.total}`);

   // Example 3: Test approaching the operator limit
   console.log("\nExample 3: Testing an expression approaching the operator
limit");
   const approachingLimit = await testOperatorLimit(config, tableName, key,
290);
   console.log(approachingLimit.message);

   // Example 4: Test exceeding the operator limit
   console.log("\nExample 4: Testing an expression exceeding the operator
limit");
   const exceedingLimit = await testOperatorLimit(config, tableName, key, 310);
   console.log(exceedingLimit.message);

   // Example 5: Breaking down a complex expression
   console.log("\nExample 5: Breaking down a complex expression into multiple
operations");
   const breakdownResult = await breakDownComplexExpression(config, tableName,
key, 500);
   console.log(`Processed ${breakdownResult.results.length} of
${breakdownResult.totalBatches} batches`);

   // Explain the operator counting rules
   console.log("\nKey points about DynamoDB expression operator counting:");
```

```
    console.log("1. The maximum number of operators in any expression is 300");
    console.log("2. Each comparison operator (=, <>, <, <=, >, >=) counts as 1
 operator");
    console.log("3. Each logical operator (AND, OR, NOT) counts as 1 operator");
    console.log("4. Each function call (attribute_exists, contains, etc.) counts
 as 1 operator");
    console.log("5. Each arithmetic operator (+ or -) counts as 1 operator");
    console.log("6. BETWEEN counts as 2 operators (BETWEEN itself and the AND
 within it)");
    console.log("7. IN counts as 1 operator regardless of the number of values");
    console.log("8. Parentheses for grouping and attribute paths don't count as
 operators");
    console.log("9. When you exceed the limit, the error always reports '301
 operators'");
    console.log("10. For complex operations, break them into multiple smaller
 operations");

  } catch (error) {
    console.error("Error:", error);
  }
}
```

- Einzelheiten zur API finden Sie UpdateItemunter AWS SDK für JavaScript API-Referenz.

Python

SDK für Python (Boto3)

Demonstrieren Sie das Zählen von Ausdrucksoperatoren mit AWS SDK für Python (Boto3).

```
import boto3
from botocore.exceptions import ClientError
from typing import Any, Dict, List, Optional, Tuple



def create_complex_filter_expression(
    attribute_name: str, values: List[Any], use_or: bool = True
) -> Tuple[str, Dict[str, Any], Dict[str, str], int]:
    """
    Create a complex filter expression with multiple conditions.
```

```
    This function demonstrates how to build a complex filter expression
    and count the number of operators used.

    Args:
        attribute_name (str): The name of the attribute to filter on.
        values (List[Any]): List of values to compare against.
        use_or (bool, optional): Whether to use OR between conditions. Defaults
to True.

    Returns:
        Tuple[str, Dict[str, Any], Dict[str, str], int]: A tuple containing:
            - The filter expression string
            - Expression attribute values
            - Expression attribute names
            - The number of operators used
    """
    if not values:
        return "", {}, {}, 0

    # Initialize expression components
    filter_expression = ""
    expression_attribute_values = {}
    expression_attribute_names = {"#attr": attribute_name}
    operator_count = 0

    # Build the filter expression
    for i, value in enumerate(values):
        value_placeholder = f":val{i}"
        expression_attribute_values[value_placeholder] = value

        if i > 0:
            # Add OR or AND operator between conditions
            filter_expression += " OR " if use_or else " AND "
            operator_count += 1  # Count the OR/AND operator

        # Add the condition
        filter_expression += f"#attr = {value_placeholder}"
        operator_count += 1  # Count the = operator

    return (
        filter_expression,
        expression_attribute_values,
        expression_attribute_names,
        operator_count,
```

```
    )


def create_nested_filter_expression(
    depth: int, conditions_per_level: int
) -> Tuple[str, Dict[str, Any], Dict[str, str], int]:
    """
    Create a deeply nested filter expression with multiple conditions.

    This function demonstrates how to build a complex nested filter expression
    and count the number of operators used.

    Args:
        depth (int): The depth of nesting.
        conditions_per_level (int): Number of conditions at each level.

    Returns:
        Tuple[str, Dict[str, Any], Dict[str, str], int]: A tuple containing:
            - The filter expression string
            - Expression attribute values
            - Expression attribute names
            - The number of operators used
    """
    if depth <= 0 or conditions_per_level <= 0:
        return "", {}, {}, 0

    # Initialize expression components
    expression_attribute_values = {}
    expression_attribute_names = {}
    operator_count = 0

    def build_nested_expression(current_depth: int, prefix: str) -> str:
        nonlocal operator_count

        if current_depth <= 0:
            return ""

        # Build conditions at this level
        conditions = []
        for i in range(conditions_per_level):
            attr_name = f"attr{prefix}_{i}"
            attr_placeholder = f"#attr{prefix}_{i}"
            val_placeholder = f":val{prefix}_{i}"
```

```
            expression_attribute_names[attr_placeholder] = attr_name
            expression_attribute_values[val_placeholder] = i

            conditions.append(f"{attr_placeholder} = {val_placeholder}")
            operator_count += 1  # Count the = operator

        # Join conditions with AND
        level_expression = " AND ".join(conditions)
        operator_count += max(0, len(conditions) - 1)  # Count the AND operators

        # If not at the deepest level, add nested expressions
        if current_depth > 1:
            nested_expr = build_nested_expression(current_depth - 1,
 f"{prefix}_{current_depth}")
            if nested_expr:
                level_expression = f"({level_expression}) OR ({nested_expr})"
                operator_count += 1  # Count the OR operator

        return level_expression

    # Build the expression starting from the top level
    filter_expression = build_nested_expression(depth, "1")

    return (
        filter_expression,
        expression_attribute_values,
        expression_attribute_names,
        operator_count,
    )


def count_operators_in_update_expression(update_expression: str) -> int:
    """
    Count the number of operators in an update expression.

    This function demonstrates how to count operators in an update expression
    based on DynamoDB's rules.

    Args:
        update_expression (str): The update expression to analyze.

    Returns:
        int: The number of operators in the expression.
    """
```

```
    operator_count = 0

    # Count SET operations
    if "SET" in update_expression:
        set_section = (
            update_expression.split("SET")[1].split("REMOVE")[0].split("ADD")
[0].split("DELETE")[0]
        )

        # Count assignment operators (=)
        operator_count += set_section.count("=")

        # Count arithmetic operators (+, -)
        operator_count += set_section.count("+")
        operator_count += set_section.count("-")

        # Count list_append function calls (each counts as 1 operator)
        operator_count += set_section.lower().count("list_append")

        # Count if_not_exists function calls (each counts as 1 operator)
        operator_count += set_section.lower().count("if_not_exists")

    # Count REMOVE operations (no additional operators)

    # Count ADD operations (each ADD counts as 1 operator)
    if "ADD" in update_expression:
        add_section = (
            update_expression.split("ADD")[1].split("DELETE")[0].split("SET")
[0].split("REMOVE")[0]
        )
        operator_count += add_section.count(",") + 1

    # Count DELETE operations (each DELETE counts as 1 operator)
    if "DELETE" in update_expression:
        delete_section = (
            update_expression.split("DELETE")[1].split("SET")[0].split("ADD")
[0].split("REMOVE")[0]
        )
        operator_count += delete_section.count(",") + 1

    return operator_count


def count_operators_in_condition_expression(condition_expression: str) -> int:
```

```
    """
    Count the number of operators in a condition expression.

    This function demonstrates how to count operators in a condition expression
    based on DynamoDB's rules.

    Args:
        condition_expression (str): The condition expression to analyze.

    Returns:
        int: The number of operators in the expression.
    """
    operator_count = 0

    # Count comparison operators
    comparison_operators = ["=", "<>", "<", "<=", ">", ">="]
    for op in comparison_operators:
        operator_count += condition_expression.count(op)

    # Count logical operators
    operator_count += condition_expression.upper().count(" AND ")
    operator_count += condition_expression.upper().count(" OR ")
    operator_count += condition_expression.upper().count("NOT ")

    # Count BETWEEN operator (counts as 2: BETWEEN + AND)
    between_count = condition_expression.upper().count(" BETWEEN ")
    operator_count += between_count * 2

    # Count IN operator (counts as 1 regardless of number of values)
    operator_count += condition_expression.upper().count(" IN ")

    # Count functions (each counts as 1 operator)
    functions = [
        "attribute_exists",
        "attribute_not_exists",
        "attribute_type",
        "begins_with",
        "contains",
        "size",
    ]
    for func in functions:
        operator_count += condition_expression.lower().count(func)

    return operator_count
```

```
# Note: This function is for demonstration purposes only and should be called
 from example_usage()
# It's not meant to be used directly as a test function
def _test_expression_limit(
    table_name: str, key: Dict[str, Any], operator_count: int, attribute_name:
 str = "TestAttribute"
) -> Tuple[bool, Optional[str]]:
    """
    Test if an expression with a specific number of operators exceeds the limit.

    This function demonstrates how to test the 300 operator limit by creating
    an expression with a specified number of operators.

    Args:
        table_name (str): The name of the DynamoDB table.
        key (Dict[str, Any]): The primary key of the item to update.
        operator_count (int): The number of operators to include in the
 expression.
        attribute_name (str, optional): The name of the attribute to update.
 Defaults to "TestAttribute".

    Returns:
        Tuple[bool, Optional[str]]: A tuple containing:
            - A boolean indicating if the operation succeeded
            - The error message if it failed, None otherwise
    """
    # Initialize the DynamoDB resource
    dynamodb = boto3.resource("dynamodb")
    table = dynamodb.Table(table_name)

    # Create an update expression with the specified number of operators
    update_expression = f"SET #{attribute_name} = :val0"
    expression_attribute_names = {f"#{attribute_name}": attribute_name}
    expression_attribute_values = {":val0": 0}

    # Add additional SET operations to reach the desired operator count
    # Each assignment adds 1 operator
    for i in range(1, operator_count):
        attr_name = f"{attribute_name}{i}"
        attr_placeholder = f"#attr{i}"
        val_placeholder = f":val{i}"
```

```
        update_expression += f", {attr_placeholder} = {val_placeholder}"
        expression_attribute_names[attr_placeholder] = attr_name
        expression_attribute_values[val_placeholder] = i

    try:
        # Attempt the update operation
        table.update_item(
            Key=key,
            UpdateExpression=update_expression,
            ExpressionAttributeNames=expression_attribute_names,
            ExpressionAttributeValues=expression_attribute_values,
        )
        return True, None
    except ClientError as e:
        error_message = e.response["Error"]["Message"]

        if "expression contains too many operators" in error_message.lower():
            return False, error_message
        else:
            # Other error occurred
            raise
```

Beispiel für die Verwendung des Ausdrucksoperators Counting with AWS SDK für Python (Boto3).

```
def example_usage():
    """Example of how to use the expression operator counting functions."""

    print("Example 1: Creating a complex filter expression with multiple
 conditions")
    attribute_name = "Status"
    values = ["Active", "Pending", "Processing", "Shipped", "Delivered"]

    filter_expr, expr_attr_vals, expr_attr_names, op_count =
 create_complex_filter_expression(
        attribute_name=attribute_name, values=values, use_or=True
    )

    print(f"Filter Expression: {filter_expr}")
    print(f"Expression Attribute Values: {expr_attr_vals}")
```

```
    print(f"Expression Attribute Names: {expr_attr_names}")
    print(f"Operator Count: {op_count}")

    print("\nExample 2: Creating a nested filter expression")
    nested_expr, nested_vals, nested_names, nested_count =
create_nested_filter_expression(
        depth=3, conditions_per_level=2
    )

    print(f"Nested Filter Expression: {nested_expr}")
    print(f"Operator Count: {nested_count}")

    print("\nExample 3: Counting operators in an update expression")
    update_expression = "SET #name = :name, #age = :age + :increment,
#address.#city = :city, #status = if_not_exists(#status, :default_status) REMOVE
#old_field ADD #counter :value DELETE #set_attr :set_val"
    update_op_count = count_operators_in_update_expression(update_expression)

    print(f"Update Expression: {update_expression}")
    print(f"Operator Count: {update_op_count}")

    print("\nExample 4: Counting operators in a condition expression")
    condition_expression = "(#status = :active OR #status = :pending) AND #price
BETWEEN :min_price AND :max_price AND attribute_exists(#category) AND NOT
(#stock <= :min_stock)"
    condition_op_count =
count_operators_in_condition_expression(condition_expression)

    print(f"Condition Expression: {condition_expression}")
    print(f"Operator Count: {condition_op_count}")

    print("\nExample 5: Testing the 300 operator limit")

    # This is just for demonstration - in a real application, you would use your
actual table
    # Note: This function is renamed to _test_expression_limit to avoid pytest
trying to run it
    print("In a real application, you would test with _test_expression_limit
function")
    print("Expression with 250 operators would be under the limit")
    print("Expression with 350 operators would exceed the 300 operator limit")

    print("\nOperator Counting Rules in DynamoDB:")
    print("1. Comparison Operators (=, <>, <, <=, >, >=): 1 operator each")
```

```
    print("2. Logical Operators (AND, OR, NOT): 1 operator each")
    print("3. BETWEEN: 2 operators (BETWEEN + AND)")
    print("4. IN: 1 operator (regardless of number of values)")
    print("5. Functions (attribute_exists, begins_with, etc.): 1 operator each")
    print("6. Arithmetic Operators (+, -): 1 operator each")
    print("7. SET assignments (=): 1 operator each")
    print("8. ADD and DELETE operations: 1 operator each")

    print("\nStrategies for Working Within the 300 Operator Limit:")
    print("1. Break operations into multiple requests")
    print("2. Use DynamoDB Transactions for complex operations")
    print("3. Optimize data model to reduce query complexity")
    print("4. Use application-side filtering for less critical filters")
    print("5. Consider using IN operator instead of multiple OR conditions")
```

- Einzelheiten zur API finden Sie UpdateItemin AWS SDK for Python (Boto3) API Reference.

Eine vollständige Liste der AWS SDK-Entwicklerhandbücher und Codebeispiele finden Sie unter. DynamoDB mit einem SDK verwenden AWS Dieses Thema enthält auch Informationen zu den ersten Schritten und Details zu früheren SDK-Versionen.

Erstellen einer API-Gateway-REST-API zur Verfolgung von COVID-19-Daten

Das folgende Codebeispiel zeigt, wie eine REST-API erstellt wird, die ein System zur Verfolgung der täglichen COVID-19-Fälle in den Vereinigten Staaten unter Verwendung fiktiver Daten simuliert.

Python

SDK für Python (Boto3)

Zeigt, wie AWS Chalice mit dem verwendet wird AWS SDK für Python (Boto3) , um eine serverlose REST-API zu erstellen, die Amazon API Gateway und Amazon AWS Lambda DynamoDB verwendet. Die REST-API simuliert ein System, das die täglichen COVID-19-Fälle in den Vereinigten Staaten unter Verwendung fiktiver Daten simuliert. Lernen Sie Folgendes:

- Verwenden Sie AWS Chalice, um Routen in Lambda-Funktionen zu definieren, die aufgerufen werden, um REST-Anfragen zu bearbeiten, die über API Gateway eingehen.

- Verwenden Sie Lambda-Funktionen zum Abrufen und Speichern von Daten in einer DynamoDB-Tabelle, um REST-Anforderungen zu bearbeiten.
- Definieren Sie die Tabellenstruktur und die Ressourcen für Sicherheitsrollen in einer AWS CloudFormation Vorlage.
- Verwenden Sie AWS Chalice und CloudFormation , um alle erforderlichen Ressourcen zu verpacken und bereitzustellen.
- Wird verwendet CloudFormation , um alle erstellten Ressourcen zu bereinigen.

Den vollständigen Quellcode und Anweisungen zur Einrichtung und Ausführung finden Sie im vollständigen Beispiel unter GitHub.

In diesem Beispiel verwendete Dienste

- API Gateway
- CloudFormation
- DynamoDB
- Lambda

Eine vollständige Liste der AWS SDK-Entwicklerhandbücher und Codebeispiele finden Sie unterDynamoDB mit einem SDK verwenden AWS. Dieses Thema enthält auch Informationen zu den ersten Schritten und Details zu früheren SDK-Versionen.

Erstellen einer Messenger-Anwendung mit Step Functions

Das folgende Codebeispiel zeigt, wie eine AWS Step Functions Messenger-Anwendung erstellt wird, die Nachrichtendatensätze aus einer Datenbanktabelle abruft.

Python

SDK für Python (Boto3)

Zeigt, wie AWS SDK für Python (Boto3) mit AWS Step Functions dem with eine Messenger-Anwendung erstellt wird, die Nachrichtendatensätze aus einer Amazon DynamoDB-Tabelle abruft und sie mit Amazon Simple Queue Service (Amazon SQS) sendet. Die Zustandsmaschine ist mit einer AWS Lambda Funktion integriert, mit der die Datenbank nach nicht gesendeten Nachrichten durchsucht werden kann.

- Erstellen Sie einen Zustandsautomaten, der Nachrichtendatensätze aus einer Amazon-DynamoDB-Tabelle abruft und aktualisiert.

- Aktualisieren Sie die Definition des Zustandsautomaten, um auch Nachrichten an Amazon Simple Queue Service (Amazon SQS) zu senden.
- Starten und stoppen Sie Ausführungen des Zustandsautomaten.
- Stellen Sie vom Zustandsautomaten aus über Serviceintegrationen eine Verbindung zu Lambda, DynamoDB und Amazon SQS her.

Den vollständigen Quellcode und Anweisungen zur Einrichtung und Ausführung finden Sie im vollständigen Beispiel unter GitHub.

In diesem Beispiel verwendete Dienste

- DynamoDB
- Lambda
- Amazon SQS
- Step Functions

Eine vollständige Liste der AWS SDK-Entwicklerhandbücher und Codebeispiele finden Sie unterDynamoDB mit einem SDK verwenden AWS. Dieses Thema enthält auch Informationen zu den ersten Schritten und Details zu früheren SDK-Versionen.

Eine Anwendung für Foto-Asset-Management erstellen, mit der Benutzer Fotos mithilfe von Labels verwalten können

Die folgenden Codebeispiele zeigen, wie eine Serverless-Anwendung erstellt wird, mit der Benutzer Fotos mithilfe von Labels verwalten können.

.NET

SDK für .NET

Zeigt, wie eine Anwendung zur Verwaltung von Fotobeständen entwickelt wird, die mithilfe von Amazon Rekognition Labels in Bildern erkennt und sie für einen späteren Abruf speichert.

Den vollständigen Quellcode und Anweisungen zur Einrichtung und Ausführung finden Sie im vollständigen Beispiel unter GitHub.

Einen tiefen Einblick in den Ursprung dieses Beispiels finden Sie im Beitrag in der AWS - Community.

In diesem Beispiel verwendete Dienste

- API Gateway
- DynamoDB
- Lambda
- Amazon Rekognition
- Amazon S3
- Amazon SNS

C++

SDK für C++

Zeigt, wie eine Anwendung zur Verwaltung von Fotobeständen entwickelt wird, die mithilfe von Amazon Rekognition Labels in Bildern erkennt und sie für einen späteren Abruf speichert.

Den vollständigen Quellcode und Anweisungen zur Einrichtung und Ausführung finden Sie im vollständigen Beispiel unter GitHub.

Einen tiefen Einblick in den Ursprung dieses Beispiels finden Sie im Beitrag in der AWS - Community.

In diesem Beispiel verwendete Dienste

- API Gateway
- DynamoDB
- Lambda
- Amazon Rekognition
- Amazon S3
- Amazon SNS

Java

SDK für Java 2.x

Zeigt, wie eine Anwendung zur Verwaltung von Fotobeständen entwickelt wird, die mithilfe von Amazon Rekognition Labels in Bildern erkennt und sie für einen späteren Abruf speichert.

Den vollständigen Quellcode und Anweisungen zur Einrichtung und Ausführung finden Sie im vollständigen Beispiel unter GitHub.

Einen tiefen Einblick in den Ursprung dieses Beispiels finden Sie im Beitrag in der AWS - Community.

In diesem Beispiel verwendete Dienste

- API Gateway
- DynamoDB
- Lambda
- Amazon Rekognition
- Amazon S3
- Amazon SNS

JavaScript

SDK für JavaScript (v3)

Zeigt, wie eine Anwendung zur Verwaltung von Fotobeständen entwickelt wird, die mithilfe von Amazon Rekognition Labels in Bildern erkennt und sie für einen späteren Abruf speichert.

Den vollständigen Quellcode und Anweisungen zur Einrichtung und Ausführung finden Sie im vollständigen Beispiel unter GitHub.

Einen tiefen Einblick in den Ursprung dieses Beispiels finden Sie im Beitrag in der AWS - Community.

In diesem Beispiel verwendete Dienste

- API Gateway
- DynamoDB
- Lambda
- Amazon Rekognition
- Amazon S3
- Amazon SNS

Kotlin

SDK für Kotlin

Zeigt, wie eine Anwendung zur Verwaltung von Fotobeständen entwickelt wird, die mithilfe von Amazon Rekognition Labels in Bildern erkennt und sie für einen späteren Abruf speichert.

Den vollständigen Quellcode und Anweisungen zur Einrichtung und Ausführung finden Sie im vollständigen Beispiel unter GitHub.

Einen tiefen Einblick in den Ursprung dieses Beispiels finden Sie im Beitrag in der AWS - Community.

In diesem Beispiel verwendete Dienste

- API Gateway
- DynamoDB
- Lambda
- Amazon Rekognition
- Amazon S3
- Amazon SNS

PHP

SDK für PHP

Zeigt, wie eine Anwendung zur Verwaltung von Fotobeständen entwickelt wird, die mithilfe von Amazon Rekognition Labels in Bildern erkennt und sie für einen späteren Abruf speichert.

Den vollständigen Quellcode und Anweisungen zur Einrichtung und Ausführung finden Sie im vollständigen Beispiel unter GitHub.

Einen tiefen Einblick in den Ursprung dieses Beispiels finden Sie im Beitrag in der AWS - Community.

In diesem Beispiel verwendete Dienste

- API Gateway
- DynamoDB

- Lambda
- Amazon Rekognition
- Amazon S3
- Amazon SNS

Rust

SDK für Rust

Zeigt, wie eine Anwendung zur Verwaltung von Fotobeständen entwickelt wird, die mithilfe von Amazon Rekognition Labels in Bildern erkennt und sie für einen späteren Abruf speichert.

Den vollständigen Quellcode und Anweisungen zur Einrichtung und Ausführung finden Sie im vollständigen Beispiel unter GitHub.

Einen tiefen Einblick in den Ursprung dieses Beispiels finden Sie im Beitrag in der AWS - Community.

In diesem Beispiel verwendete Dienste

- API Gateway
- DynamoDB
- Lambda
- Amazon Rekognition
- Amazon S3
- Amazon SNS

Eine vollständige Liste der AWS SDK-Entwicklerhandbücher und Codebeispiele finden Sie unterDynamoDB mit einem SDK verwenden AWS. Dieses Thema enthält auch Informationen zu den ersten Schritten und Details zu früheren SDK-Versionen.

Erstellen Sie mithilfe des SDK eine DynamoDB-Tabelle mit einem globalen sekundären Index AWS

Das folgende Codebeispiel zeigt, wie Sie eine Tabelle mit einem globalen sekundären Index erstellen.

Java

SDK für Java 2.x

Erstellen Sie eine DynamoDB-Tabelle mit Global Secondary Index mithilfe von. AWS SDK for Java 2.x

```
import software.amazon.awssdk.auth.credentials.DefaultCredentialsProvider;
import software.amazon.awssdk.core.waiters.WaiterResponse;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.AttributeDefinition;
import software.amazon.awssdk.services.dynamodb.model.AttributeValue;
import software.amazon.awssdk.services.dynamodb.model.CreateTableRequest;
import software.amazon.awssdk.services.dynamodb.model.DeleteTableRequest;
import software.amazon.awssdk.services.dynamodb.model.DescribeTableRequest;
import software.amazon.awssdk.services.dynamodb.model.DescribeTableResponse;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.model.GlobalSecondaryIndex;
import software.amazon.awssdk.services.dynamodb.model.KeySchemaElement;
import software.amazon.awssdk.services.dynamodb.model.KeyType;
import software.amazon.awssdk.services.dynamodb.model.Projection;
import software.amazon.awssdk.services.dynamodb.model.ProjectionType;
import software.amazon.awssdk.services.dynamodb.model.ProvisionedThroughput;
import software.amazon.awssdk.services.dynamodb.model.PutItemRequest;
import software.amazon.awssdk.services.dynamodb.model.QueryRequest;
import software.amazon.awssdk.services.dynamodb.model.QueryResponse;
import software.amazon.awssdk.services.dynamodb.model.ScalarAttributeType;
import software.amazon.awssdk.services.dynamodb.waiters.DynamoDbWaiter;

import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;

    public void createTable() {
        try {
            // Attribute definitions
            final List<AttributeDefinition> attributeDefinitions = new
 ArrayList<>();
            attributeDefinitions.add(AttributeDefinition.builder()
                .attributeName(ISSUE_ID_ATTR)
                .attributeType(ScalarAttributeType.S)
                .build());
```

```
            attributeDefinitions.add(AttributeDefinition.builder()
                .attributeName(TITLE_ATTR)
                .attributeType(ScalarAttributeType.S)
                .build());
            attributeDefinitions.add(AttributeDefinition.builder()
                .attributeName(CREATE_DATE_ATTR)
                .attributeType(ScalarAttributeType.S)
                .build());
            attributeDefinitions.add(AttributeDefinition.builder()
                .attributeName(DUE_DATE_ATTR)
                .attributeType(ScalarAttributeType.S)
                .build());

            // Key schema for table
            final List<KeySchemaElement> tableKeySchema = new ArrayList<>();
            tableKeySchema.add(KeySchemaElement.builder()
                .attributeName(ISSUE_ID_ATTR)
                .keyType(KeyType.HASH)
                .build()); // Partition key
            tableKeySchema.add(KeySchemaElement.builder()
                .attributeName(TITLE_ATTR)
                .keyType(KeyType.RANGE)
                .build()); // Sort key

            // Initial provisioned throughput settings for the indexes
            final ProvisionedThroughput ptIndex = ProvisionedThroughput.builder()
                .readCapacityUnits(1L)
                .writeCapacityUnits(1L)
                .build();

            // CreateDateIndex
            final List<KeySchemaElement> createDateKeySchema = new ArrayList<>();
            createDateKeySchema.add(KeySchemaElement.builder()
                .attributeName(CREATE_DATE_ATTR)
                .keyType(KeyType.HASH)
                .build());
            createDateKeySchema.add(KeySchemaElement.builder()
                .attributeName(ISSUE_ID_ATTR)
                .keyType(KeyType.RANGE)
                .build());

            final Projection createDateProjection = Projection.builder()
                .projectionType(ProjectionType.INCLUDE)
                .nonKeyAttributes(DESCRIPTION_ATTR, STATUS_ATTR)
```

```
                .build();

            final GlobalSecondaryIndex createDateIndex =
GlobalSecondaryIndex.builder()
                .indexName(CREATE_DATE_INDEX)
                .keySchema(createDateKeySchema)
                .projection(createDateProjection)
                .provisionedThroughput(ptIndex)
                .build();

            // TitleIndex
            final List<KeySchemaElement> titleKeySchema = new ArrayList<>();
            titleKeySchema.add(KeySchemaElement.builder()
                .attributeName(TITLE_ATTR)
                .keyType(KeyType.HASH)
                .build());
            titleKeySchema.add(KeySchemaElement.builder()
                .attributeName(ISSUE_ID_ATTR)
                .keyType(KeyType.RANGE)
                .build());

            final Projection titleProjection =

Projection.builder().projectionType(ProjectionType.KEYS_ONLY).build();

            final GlobalSecondaryIndex titleIndex =
GlobalSecondaryIndex.builder()
                .indexName(TITLE_INDEX)
                .keySchema(titleKeySchema)
                .projection(titleProjection)
                .provisionedThroughput(ptIndex)
                .build();

            // DueDateIndex
            final List<KeySchemaElement> dueDateKeySchema = new ArrayList<>();
            dueDateKeySchema.add(KeySchemaElement.builder()
                .attributeName(DUE_DATE_ATTR)
                .keyType(KeyType.HASH)
                .build());

            final Projection dueDateProjection =
                Projection.builder().projectionType(ProjectionType.ALL).build();
```

```
            final GlobalSecondaryIndex dueDateIndex =
GlobalSecondaryIndex.builder()
                .indexName(DUE_DATE_INDEX)
                .keySchema(dueDateKeySchema)
                .projection(dueDateProjection)
                .provisionedThroughput(ptIndex)
                .build();

            final CreateTableRequest createTableRequest =
CreateTableRequest.builder()
                .tableName(TABLE_NAME)
                .keySchema(tableKeySchema)
                .attributeDefinitions(attributeDefinitions)
                .globalSecondaryIndexes(createDateIndex, titleIndex,
dueDateIndex)
                .provisionedThroughput(ProvisionedThroughput.builder()
                    .readCapacityUnits(1L)
                    .writeCapacityUnits(1L)
                    .build())
                .build();

            System.out.println("Creating table " + TABLE_NAME + "...");
            dynamoDbClient.createTable(createTableRequest);

            // Wait for table to become active
            System.out.println("Waiting for " + TABLE_NAME + " to become
ACTIVE...");
            final DynamoDbWaiter waiter = dynamoDbClient.waiter();
            final DescribeTableRequest describeTableRequest =
                DescribeTableRequest.builder().tableName(TABLE_NAME).build();

            final WaiterResponse<DescribeTableResponse> waiterResponse =
                waiter.waitUntilTableExists(describeTableRequest);
            waiterResponse.matched().response().ifPresent(response ->
System.out.println("Table is now ready for use"));

        } catch (DynamoDbException e) {
            System.err.println("Error creating table: " + e.getMessage());
            e.printStackTrace();
        }
    }
```

- Einzelheiten zur API finden Sie unter CreateTable AWS SDK for Java 2.xAPI-Referenz.

Eine vollständige Liste der AWS SDK-Entwicklerhandbücher und Codebeispiele finden Sie unterDynamoDB mit einem SDK verwenden AWS. Dieses Thema enthält auch Informationen zu den ersten Schritten und Details zu früheren SDK-Versionen.

Erstellen Sie mithilfe eines SDK eine DynamoDB-Tabelle mit Warmdurchsatzeinstellung AWS

In den folgenden Codebeispielen wird gezeigt, wie eine Tabelle mit aktiviertem Warmdurchsatz erstellt wird.

Java

SDK für Java 2.x

Erstellen Sie mithilfe von AWS SDK for Java 2.x eine DynamoDB-Tabelle mit Warmdurchsatz-Einstellung.

```
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.AttributeDefinition;
import software.amazon.awssdk.services.dynamodb.model.CreateTableRequest;
import software.amazon.awssdk.services.dynamodb.model.CreateTableResponse;
import software.amazon.awssdk.services.dynamodb.model.GlobalSecondaryIndex;
import software.amazon.awssdk.services.dynamodb.model.KeySchemaElement;
import software.amazon.awssdk.services.dynamodb.model.KeyType;
import software.amazon.awssdk.services.dynamodb.model.Projection;
import software.amazon.awssdk.services.dynamodb.model.ProvisionedThroughput;
import software.amazon.awssdk.services.dynamodb.model.ScalarAttributeType;
import software.amazon.awssdk.services.dynamodb.model.WarmThroughput;

    public static WarmThroughput buildWarmThroughput(final Long
 readUnitsPerSecond, final Long writeUnitsPerSecond) {
        return WarmThroughput.builder()
            .readUnitsPerSecond(readUnitsPerSecond)
            .writeUnitsPerSecond(writeUnitsPerSecond)
            .build();
    }

    /**
     * Builds a ProvisionedThroughput object with the specified read and write
 capacity units.
     *
     * @param readCapacityUnits The read capacity units
```

```
     * @param writeCapacityUnits The write capacity units
     * @return A configured ProvisionedThroughput object
     */
    public static ProvisionedThroughput buildProvisionedThroughput(
        final Long readCapacityUnits, final Long writeCapacityUnits) {
        return ProvisionedThroughput.builder()
            .readCapacityUnits(readCapacityUnits)
            .writeCapacityUnits(writeCapacityUnits)
            .build();
    }

    /**
     * Builds an AttributeDefinition with the specified name and type.
     *
     * @param attributeName The attribute name
     * @param scalarAttributeType The attribute type
     * @return A configured AttributeDefinition
     */
    private static AttributeDefinition buildAttributeDefinition(
        final String attributeName, final ScalarAttributeType
scalarAttributeType) {
        return AttributeDefinition.builder()
            .attributeName(attributeName)
            .attributeType(scalarAttributeType)
            .build();
    }

    /**
     * Builds a KeySchemaElement with the specified name and key type.
     *
     * @param attributeName The attribute name
     * @param keyType The key type (HASH or RANGE)
     * @return A configured KeySchemaElement
     */
    private static KeySchemaElement buildKeySchemaElement(final String
attributeName, final KeyType keyType) {
        return KeySchemaElement.builder()
            .attributeName(attributeName)
            .keyType(keyType)
            .build();
    }

    /**
```

```
    * Creates a DynamoDB table with the specified configuration including warm
throughput settings.
    *
    * @param ddb The DynamoDB client
    * @param tableName The name of the table to create
    * @param partitionKey The partition key attribute name
    * @param sortKey The sort key attribute name
    * @param miscellaneousKeyAttribute Additional key attribute name for GSI
    * @param nonKeyAttribute Non-key attribute to include in GSI projection
    * @param tableReadCapacityUnits Read capacity units for the table
    * @param tableWriteCapacityUnits Write capacity units for the table
    * @param tableWarmReadUnitsPerSecond Warm read units per second for the
table
    * @param tableWarmWriteUnitsPerSecond Warm write units per second for the
table
    * @param globalSecondaryIndexName The name of the GSI to create
    * @param globalSecondaryIndexReadCapacityUnits Read capacity units for the
GSI
    * @param globalSecondaryIndexWriteCapacityUnits Write capacity units for the
GSI
    * @param globalSecondaryIndexWarmReadUnitsPerSecond Warm read units per
second for the GSI
    * @param globalSecondaryIndexWarmWriteUnitsPerSecond Warm write units per
second for the GSI
    */
   public static void createDynamoDBTable(
       final DynamoDbClient ddb,
       final String tableName,
       final String partitionKey,
       final String sortKey,
       final String miscellaneousKeyAttribute,
       final String nonKeyAttribute,
       final Long tableReadCapacityUnits,
       final Long tableWriteCapacityUnits,
       final Long tableWarmReadUnitsPerSecond,
       final Long tableWarmWriteUnitsPerSecond,
       final String globalSecondaryIndexName,
       final Long globalSecondaryIndexReadCapacityUnits,
       final Long globalSecondaryIndexWriteCapacityUnits,
       final Long globalSecondaryIndexWarmReadUnitsPerSecond,
       final Long globalSecondaryIndexWarmWriteUnitsPerSecond) {

       // Define the table attributes
```

```
        final AttributeDefinition partitionKeyAttribute =
buildAttributeDefinition(partitionKey, ScalarAttributeType.S);
        final AttributeDefinition sortKeyAttribute =
buildAttributeDefinition(sortKey, ScalarAttributeType.S);
        final AttributeDefinition miscellaneousKeyAttributeDefinition =
            buildAttributeDefinition(miscellaneousKeyAttribute,
ScalarAttributeType.N);
        final AttributeDefinition[] attributeDefinitions = {
            partitionKeyAttribute, sortKeyAttribute,
miscellaneousKeyAttributeDefinition
        };

        // Define the table key schema
        final KeySchemaElement partitionKeyElement =
buildKeySchemaElement(partitionKey, KeyType.HASH);
        final KeySchemaElement sortKeyElement = buildKeySchemaElement(sortKey,
KeyType.RANGE);
        final KeySchemaElement[] keySchema = {partitionKeyElement,
sortKeyElement};

        // Define the provisioned throughput for the table
        final ProvisionedThroughput provisionedThroughput =
            buildProvisionedThroughput(tableReadCapacityUnits,
tableWriteCapacityUnits);

        // Define the Global Secondary Index (GSI)
        final KeySchemaElement globalSecondaryIndexPartitionKeyElement =
buildKeySchemaElement(sortKey, KeyType.HASH);
        final KeySchemaElement globalSecondaryIndexSortKeyElement =
            buildKeySchemaElement(miscellaneousKeyAttribute, KeyType.RANGE);
        final KeySchemaElement[] gsiKeySchema = {
            globalSecondaryIndexPartitionKeyElement,
globalSecondaryIndexSortKeyElement
        };

        final Projection gsiProjection = Projection.builder()
            .projectionType(PROJECTION_TYPE_INCLUDE)
            .nonKeyAttributes(nonKeyAttribute)
            .build();

        final ProvisionedThroughput gsiProvisionedThroughput =
            buildProvisionedThroughput(globalSecondaryIndexReadCapacityUnits,
globalSecondaryIndexWriteCapacityUnits);
```

```
        // Define the warm throughput for the Global Secondary Index (GSI)
        final WarmThroughput gsiWarmThroughput = buildWarmThroughput(
            globalSecondaryIndexWarmReadUnitsPerSecond,
globalSecondaryIndexWarmWriteUnitsPerSecond);

        final GlobalSecondaryIndex globalSecondaryIndex =
GlobalSecondaryIndex.builder()
            .indexName(globalSecondaryIndexName)
            .keySchema(gsiKeySchema)
            .projection(gsiProjection)
            .provisionedThroughput(gsiProvisionedThroughput)
            .warmThroughput(gsiWarmThroughput)
            .build();

        // Define the warm throughput for the table
        final WarmThroughput tableWarmThroughput =
            buildWarmThroughput(tableWarmReadUnitsPerSecond,
tableWarmWriteUnitsPerSecond);

        final CreateTableRequest request = CreateTableRequest.builder()
            .tableName(tableName)
            .attributeDefinitions(attributeDefinitions)
            .keySchema(keySchema)
            .provisionedThroughput(provisionedThroughput)
            .globalSecondaryIndexes(globalSecondaryIndex)
            .warmThroughput(tableWarmThroughput)
            .build();

        final CreateTableResponse response = ddb.createTable(request);
        System.out.println(response);
    }
```

- Einzelheiten zur API finden Sie CreateTablein der AWS SDK for Java 2.x API-Referenz.

JavaScript

SDK für JavaScript (v3)

Erstellen Sie eine DynamoDB-Tabelle mit Warmdurchsatz-Einstellung mithilfe von AWS SDK für JavaScript.

```
import { DynamoDBClient, CreateTableCommand } from "@aws-sdk/client-dynamodb";
```

```
export async function createDynamoDBTableWithWarmThroughput(
  tableName,
  partitionKey,
  sortKey,
  miscKeyAttr,
  nonKeyAttr,
  tableProvisionedReadUnits,
  tableProvisionedWriteUnits,
  tableWarmReads,
  tableWarmWrites,
  indexName,
  indexProvisionedReadUnits,
  indexProvisionedWriteUnits,
  indexWarmReads,
  indexWarmWrites,
  region = "us-east-1"
) {
  try {
    const ddbClient = new DynamoDBClient({ region: region });
    const command = new CreateTableCommand({
      TableName: tableName,
      AttributeDefinitions: [
          { AttributeName: partitionKey, AttributeType: "S" },
          { AttributeName: sortKey, AttributeType: "S" },
          { AttributeName: miscKeyAttr, AttributeType: "N" },
      ],
      KeySchema: [
          { AttributeName: partitionKey, KeyType: "HASH" },
          { AttributeName: sortKey, KeyType: "RANGE" },
      ],
      ProvisionedThroughput: {
          ReadCapacityUnits: tableProvisionedReadUnits,
          WriteCapacityUnits: tableProvisionedWriteUnits,
      },
      WarmThroughput: {
          ReadUnitsPerSecond: tableWarmReads,
          WriteUnitsPerSecond: tableWarmWrites,
      },
      GlobalSecondaryIndexes: [
          {
            IndexName: indexName,
            KeySchema: [
                { AttributeName: sortKey, KeyType: "HASH" },
```

```
                { AttributeName: miscKeyAttr, KeyType: "RANGE" },
            ],
            Projection: {
                ProjectionType: "INCLUDE",
                NonKeyAttributes: [nonKeyAttr],
            },
            ProvisionedThroughput: {
                ReadCapacityUnits: indexProvisionedReadUnits,
                WriteCapacityUnits: indexProvisionedWriteUnits,
            },
            WarmThroughput: {
                ReadUnitsPerSecond: indexWarmReads,
                WriteUnitsPerSecond: indexWarmWrites,
            },
          },
      ],
    });
    const response = await ddbClient.send(command);
    console.log(response);
    return response;
  } catch (error) {
    console.error(`Error creating table: ${error}`);
    throw error;
  }
}

// Example usage (commented out for testing)
/*
createDynamoDBTableWithWarmThroughput(
  'example-table',
  'pk',
  'sk',
  'gsiKey',
  'data',
  10, 10, 5, 5,
  'example-index',
  5, 5, 2, 2
);
*/
```

- Einzelheiten zur API finden Sie CreateTablein der AWS SDK für JavaScript API-Referenz.

Python

SDK für Python (Boto3)

Erstellen Sie eine DynamoDB-Tabelle mit Warmdurchsatz-Einstellung mithilfe von AWS SDK für Python (Boto3).

```
from boto3 import client
from botocore.exceptions import ClientError


def create_dynamodb_table_warm_throughput(
    table_name,
    partition_key,
    sort_key,
    misc_key_attr,
    non_key_attr,
    table_provisioned_read_units,
    table_provisioned_write_units,
    table_warm_reads,
    table_warm_writes,
    gsi_name,
    gsi_provisioned_read_units,
    gsi_provisioned_write_units,
    gsi_warm_reads,
    gsi_warm_writes,
    region_name="us-east-1",
):
    """
    Creates a DynamoDB table with a warm throughput setting configured.

    :param table_name: The name of the table to be created.
    :param partition_key: The partition key for the table being created.
    :param sort_key: The sort key for the table being created.
    :param misc_key_attr: A miscellaneous key attribute for the table being
 created.
    :param non_key_attr: A non-key attribute for the table being created.
    :param table_provisioned_read_units: The newly created table's provisioned
 read capacity units.
    :param table_provisioned_write_units: The newly created table's provisioned
 write capacity units.
    :param table_warm_reads: The read units per second setting for the table's
 warm throughput.
```

```
   :param table_warm_writes: The write units per second setting for the table's
warm throughput.
   :param gsi_name: The name of the Global Secondary Index (GSI) to be created
on the table.
   :param gsi_provisioned_read_units: The configured Global Secondary Index
(GSI) provisioned read capacity units.
   :param gsi_provisioned_write_units: The configured Global Secondary Index
(GSI) provisioned write capacity units.
   :param gsi_warm_reads: The read units per second setting for the Global
Secondary Index (GSI)'s warm throughput.
   :param gsi_warm_writes: The write units per second setting for the Global
Secondary Index (GSI)'s warm throughput.
   :param region_name: The AWS Region name to target. defaults to us-east-1
   """
   try:
       ddb = client("dynamodb", region_name=region_name)

       # Define the table attributes
       attribute_definitions = [
           {"AttributeName": partition_key, "AttributeType": "S"},
           {"AttributeName": sort_key, "AttributeType": "S"},
           {"AttributeName": misc_key_attr, "AttributeType": "N"},
       ]

       # Define the table key schema
       key_schema = [
           {"AttributeName": partition_key, "KeyType": "HASH"},
           {"AttributeName": sort_key, "KeyType": "RANGE"},
       ]

       # Define the provisioned throughput for the table
       provisioned_throughput = {
           "ReadCapacityUnits": table_provisioned_read_units,
           "WriteCapacityUnits": table_provisioned_write_units,
       }

       # Define the global secondary index
       gsi_key_schema = [
           {"AttributeName": sort_key, "KeyType": "HASH"},
           {"AttributeName": misc_key_attr, "KeyType": "RANGE"},
       ]
       gsi_projection = {"ProjectionType": "INCLUDE", "NonKeyAttributes":
[non_key_attr]}
       gsi_provisioned_throughput = {
```

```
            "ReadCapacityUnits": gsi_provisioned_read_units,
            "WriteCapacityUnits": gsi_provisioned_write_units,
        }
        gsi_warm_throughput = {
            "ReadUnitsPerSecond": gsi_warm_reads,
            "WriteUnitsPerSecond": gsi_warm_writes,
        }
        global_secondary_indexes = [
            {
                "IndexName": gsi_name,
                "KeySchema": gsi_key_schema,
                "Projection": gsi_projection,
                "ProvisionedThroughput": gsi_provisioned_throughput,
                "WarmThroughput": gsi_warm_throughput,
            }
        ]

        # Define the warm throughput for the table
        warm_throughput = {
            "ReadUnitsPerSecond": table_warm_reads,
            "WriteUnitsPerSecond": table_warm_writes,
        }

        # Create the DynamoDB client and create the table
        response = ddb.create_table(
            TableName=table_name,
            AttributeDefinitions=attribute_definitions,
            KeySchema=key_schema,
            ProvisionedThroughput=provisioned_throughput,
            GlobalSecondaryIndexes=global_secondary_indexes,
            WarmThroughput=warm_throughput,
        )

        print(response)
        return response
    except ClientError as e:
        print(f"Error creating table: {e}")
        raise e
```

- Einzelheiten zur API finden Sie CreateTablein AWS SDK for Python (Boto3) API Reference.

Eine vollständige Liste der AWS SDK-Entwicklerhandbücher und Codebeispiele finden Sie unter. DynamoDB mit einem SDK verwenden AWS Dieses Thema enthält auch Informationen zu den ersten Schritten und Details zu früheren SDK-Versionen.

Erstellen einer Webanwendung zur Verfolgung von DynamoDB-Daten

Die folgenden Code-Beispiele zeigen, wie eine Webanwendung erstellt wird, die Arbeitselemente in einer Amazon DynamoDB-Tabelle verfolgt und mithilfe von Amazon Simple Email Service (Amazon SES) Berichte sendet.

.NET

SDK für .NET

Zeigt, wie man die Amazon-DynamoDB-.NET-API verwendet, um eine dynamische Webanwendung zu erstellen, die DynamoDB-Arbeitsdaten verfolgt.

Den vollständigen Quellcode und Anweisungen zur Einrichtung und Ausführung finden Sie im vollständigen Beispiel unter GitHub.

In diesem Beispiel verwendete Dienste

- DynamoDB
- Amazon SES

Java

SDK für Java 2.x

Zeigt, wie man die Amazon-DynamoDB-API verwendet, um eine dynamische Webanwendung zu erstellen, die DynamoDB-Arbeitsdaten verfolgt.

Den vollständigen Quellcode und Anweisungen zur Einrichtung und Ausführung finden Sie im vollständigen Beispiel unter GitHub.

In diesem Beispiel verwendete Dienste

- DynamoDB
- Amazon SES

Kotlin

SDK für Kotlin

Zeigt, wie man die Amazon-DynamoDB-API verwendet, um eine dynamische Webanwendung zu erstellen, die DynamoDB-Arbeitsdaten verfolgt.

Den vollständigen Quellcode und Anweisungen zur Einrichtung und Ausführung finden Sie im vollständigen Beispiel unter GitHub.

In diesem Beispiel verwendete Dienste

- DynamoDB
- Amazon SES

Python

SDK für Python (Boto3)

Zeigt, wie Sie mithilfe von Amazon Simple Email Service (Amazon SES) einen REST-Service erstellen, der Arbeitselemente in Amazon DynamoDB verfolgt und Berichte per E-Mail versendet. AWS SDK für Python (Boto3) In diesem Beispiel wird das Flask-Web-Framework für das HTTP-Routing verwendet und in eine React-Webseite integriert, um eine voll funktionsfähige Webanwendung zu präsentieren.

- Erstellen Sie einen Flask-REST-Service, der sich in integrieren lässt. AWS-Services
- Lesen, schreiben und aktualisieren Sie Arbeitsaufgaben, die in einer DynamoDB-Tabelle gespeichert sind.
- Verwenden Sie Amazon SES, um E-Mail-Berichte über Arbeitsaufgaben zu senden.

Den vollständigen Quellcode und Anweisungen zur Einrichtung und Ausführung finden Sie im vollständigen Beispiel im AWS Code Examples Repository unter GitHub.

In diesem Beispiel verwendete Dienste

- DynamoDB
- Amazon SES

Eine vollständige Liste der AWS SDK-Entwicklerhandbücher und Codebeispiele finden Sie unterDynamoDB mit einem SDK verwenden AWS. Dieses Thema enthält auch Informationen zu den ersten Schritten und Details zu früheren SDK-Versionen.

Erstellen einer Websocket-Chat-Anwendung mit API Gateway

Das folgende Codebeispiel zeigt, wie eine Chat-Anwendung erstellt wird, die von einer auf Amazon API Gateway basierenden Websocket-API bereitgestellt wird.

Python

SDK für Python (Boto3)

Zeigt, wie Sie AWS SDK für Python (Boto3) mit Amazon API Gateway V2 eine Websocket-API erstellen, die in Amazon DynamoDB integriert AWS Lambda werden kann.

- Erstellen Sie eine WebSocket-API, die von API Gateway bereitgestellt wird.
- Definieren Sie einen Lambda-Handler, der Verbindungen in DynamoDB speichert und Nachrichten an andere Chat-Teilnehmer sendet.
- Stellen Sie eine Verbindung zur Websocket-Chat-Anwendung her und senden Sie Nachrichten mit dem Websockets-Paket.

Den vollständigen Quellcode und Anweisungen zur Einrichtung und Ausführung finden Sie im vollständigen Beispiel unter. GitHub

In diesem Beispiel verwendete Dienste

- API Gateway
- DynamoDB
- Lambda

Eine vollständige Liste der AWS SDK-Entwicklerhandbücher und Codebeispiele finden Sie unterDynamoDB mit einem SDK verwenden AWS. Dieses Thema enthält auch Informationen zu den ersten Schritten und Details zu früheren SDK-Versionen.

Erstellen Sie ein DynamoDB-Element mit einer TTL mithilfe eines SDK AWS

Die folgenden Codebeispiele veranschaulichen, wie Sie ein Element mit TTL erstellen.

Java

SDK für Java 2.x

```
package com.amazon.samplelib.ttl;

import com.amazon.samplelib.CodeSampleUtils;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.AttributeValue;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.model.PutItemRequest;
import software.amazon.awssdk.services.dynamodb.model.PutItemResponse;
import software.amazon.awssdk.services.dynamodb.model.ResourceNotFoundException;

import java.util.HashMap;
import java.util.Map;
import java.util.Optional;

/**
 * Creates an item in a DynamoDB table with TTL attributes.
 * This class demonstrates how to add TTL expiration timestamps to DynamoDB
 items.
 */
public class CreateTTL {

    private static final String USAGE =
        """
            Usage:
                <tableName> <primaryKey> <sortKey> <region>
            Where:
                tableName - The Amazon DynamoDB table being queried.
                primaryKey - The name of the primary key. Also known as the hash
 or partition key.
                sortKey - The name of the sort key. Also known as the range
 attribute.
                region (optional) - The AWS region that the Amazon DynamoDB table
 is located in. (Default: us-east-1)
            """;
    private static final int DAYS_TO_EXPIRE = 90;
    private static final int SECONDS_PER_DAY = 24 * 60 * 60;
    private static final String PRIMARY_KEY_ATTR = "primaryKey";
    private static final String SORT_KEY_ATTR = "sortKey";
```

```
    private static final String CREATION_DATE_ATTR = "creationDate";
    private static final String EXPIRE_AT_ATTR = "expireAt";
    private static final String SUCCESS_MESSAGE = "%s PutItem operation with TTL
successful.";
    private static final String TABLE_NOT_FOUND_ERROR = "Error: The Amazon
DynamoDB table \"%s\" can't be found.";

    private final DynamoDbClient dynamoDbClient;

    /**
     * Constructs a CreateTTL instance with the specified DynamoDB client.
     *
     * @param dynamoDbClient The DynamoDB client to use
     */
    public CreateTTL(final DynamoDbClient dynamoDbClient) {
        this.dynamoDbClient = dynamoDbClient;
    }

    /**
     * Constructs a CreateTTL with a default DynamoDB client.
     */
    public CreateTTL() {
        this.dynamoDbClient = null;
    }

    /**
     * Main method to demonstrate creating an item with TTL.
     *
     * @param args Command line arguments
     */
    public static void main(final String[] args) {
        try {
            int result = new CreateTTL().processArgs(args);
            System.exit(result);
        } catch (Exception e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
    }

    /**
     * Process command line arguments and create an item with TTL.
     *
     * @param args Command line arguments
```

```
     * @return 0 if successful, non-zero otherwise
     * @throws ResourceNotFoundException If the table doesn't exist
     * @throws DynamoDbException If an error occurs during the operation
     * @throws IllegalArgumentException If arguments are invalid
     */
    public int processArgs(final String[] args) {
        // Argument validation (remove or replace this line when reusing this
code)
        CodeSampleUtils.validateArgs(args, new int[] {3, 4}, USAGE);

        final String tableName = args[0];
        final String primaryKey = args[1];
        final String sortKey = args[2];
        final Region region = Optional.ofNullable(args.length > 3 ? args[3] :
null)
            .map(Region::of)
            .orElse(Region.US_EAST_1);

        try (DynamoDbClient ddb = dynamoDbClient != null
            ? dynamoDbClient
            : DynamoDbClient.builder().region(region).build()) {
            final CreateTTL createTTL = new CreateTTL(ddb);
            createTTL.createItemWithTTL(tableName, primaryKey, sortKey);
            return 0;
        } catch (Exception e) {
            throw e;
        }
    }

    /**
     * Creates an item in the specified table with TTL attributes.
     *
     * @param tableName The name of the table
     * @param primaryKeyValue The value for the primary key
     * @param sortKeyValue The value for the sort key
     * @return The response from the PutItem operation
     * @throws ResourceNotFoundException If the table doesn't exist
     * @throws DynamoDbException If an error occurs during the operation
     */
    public PutItemResponse createItemWithTTL(
        final String tableName, final String primaryKeyValue, final String
sortKeyValue) {
        // Get current time in epoch second format
        final long createDate = System.currentTimeMillis() / 1000;
```

```
        // Calculate expiration time 90 days from now in epoch second format
        final long expireDate = createDate + (DAYS_TO_EXPIRE * SECONDS_PER_DAY);

        final Map<String, AttributeValue> itemMap = new HashMap<>();
        itemMap.put(
            PRIMARY_KEY_ATTR,
 AttributeValue.builder().s(primaryKeyValue).build());
        itemMap.put(SORT_KEY_ATTR,
 AttributeValue.builder().s(sortKeyValue).build());
        itemMap.put(
            CREATION_DATE_ATTR,
            AttributeValue.builder().n(String.valueOf(createDate)).build());
        itemMap.put(
            EXPIRE_AT_ATTR,
            AttributeValue.builder().n(String.valueOf(expireDate)).build());

        final PutItemRequest request =
            PutItemRequest.builder().tableName(tableName).item(itemMap).build();

        try {
            final PutItemResponse response = dynamoDbClient.putItem(request);
            System.out.println(String.format(SUCCESS_MESSAGE, tableName));
            return response;
        } catch (ResourceNotFoundException e) {
            System.err.format(TABLE_NOT_FOUND_ERROR, tableName);
            throw e;
        } catch (DynamoDbException e) {
            System.err.println(e.getMessage());
            throw e;
        }
    }
}
```

- Einzelheiten zur API finden Sie PutItemin der AWS SDK for Java 2.x API-Referenz.

JavaScript

SDK für JavaScript (v3)

```
import { DynamoDBClient, PutItemCommand } from "@aws-sdk/client-dynamodb";
```

```
export function createDynamoDBItem(table_name, region, partition_key, sort_key) {
    const client = new DynamoDBClient({
        region: region,
        endpoint: `https://dynamodb.${region}.amazonaws.com`
    });

    // Get the current time in epoch second format
    const current_time = Math.floor(new Date().getTime() / 1000);

    // Calculate the expireAt time (90 days from now) in epoch second format
    const expire_at = Math.floor((new Date().getTime() + 90 * 24 * 60 * 60 *
 1000) / 1000);

    // Create DynamoDB item
    const item = {
        'partitionKey': {'S': partition_key},
        'sortKey': {'S': sort_key},
        'createdAt': {'N': current_time.toString()},
        'expireAt': {'N': expire_at.toString()}
    };

    const putItemCommand = new PutItemCommand({
        TableName: table_name,
        Item: item,
        ProvisionedThroughput: {
            ReadCapacityUnits: 1,
            WriteCapacityUnits: 1,
        },
    });

    client.send(putItemCommand, function(err, data) {
        if (err) {
            console.log("Exception encountered when creating item %s, here's what
 happened: ", data, err);
            throw err;
        } else {
            console.log("Item created successfully: %s.", data);
            return data;
        }
    });
}

// Example usage (commented out for testing)
```

```
// createDynamoDBItem('your-table-name', 'us-east-1', 'your-partition-key-value',
 'your-sort-key-value');
```

- Einzelheiten zur API finden Sie PutItemin der AWS SDK für JavaScript API-Referenz.

Python

SDK für Python (Boto3)

```
from datetime import datetime, timedelta

import boto3


def create_dynamodb_item(table_name, region, primary_key, sort_key):
    """
    Creates a DynamoDB item with an attached expiry attribute.

    :param table_name: Table name for the boto3 resource to target when creating
 an item
    :param region: string representing the AWS region. Example: `us-east-1`
    :param primary_key: one attribute known as the partition key.
    :param sort_key: Also known as a range attribute.
    :return: Void (nothing)
    """
    try:
        dynamodb = boto3.resource("dynamodb", region_name=region)
        table = dynamodb.Table(table_name)

        # Get the current time in epoch second format
        current_time = int(datetime.now().timestamp())

        # Calculate the expiration time (90 days from now) in epoch second format
        expiration_time = int((datetime.now() + timedelta(days=90)).timestamp())

        item = {
            "primaryKey": primary_key,
            "sortKey": sort_key,
            "creationDate": current_time,
            "expireAt": expiration_time,
        }
        response = table.put_item(Item=item)
```

```
        print("Item created successfully.")
        return response
    except Exception as e:
        print(f"Error creating item: {e}")
        raise e


# Use your own values
create_dynamodb_item(
    "your-table-name", "us-west-2", "your-partition-key-value", "your-sort-key-
value"
)
```

- Einzelheiten zur API finden Sie PutItemin AWS SDK for Python (Boto3) API Reference.

Eine vollständige Liste der AWS SDK-Entwicklerhandbücher und Codebeispiele finden Sie unter. DynamoDB mit einem SDK verwenden AWS Dieses Thema enthält auch Informationen zu den ersten Schritten und Details zu früheren SDK-Versionen.

Erstellen und verwalten Sie globale DynamoDB-Tabellen mit Multi-Region Strong Consistency mithilfe eines SDK AWS

In den folgenden Codebeispielen wird gezeigt, wie globale DynamoDB-Tabellen mit Multi-Region Strong Consistency (MRSC) erstellt und verwaltet werden.

- Erstellen einer Tabelle mit multiregionaler starker Konsistenz
- Überprüfen Sie die MRSC-Konfiguration und den Replikatstatus.
- Testen Sie die starke Konsistenz in allen Regionen mit sofortigen Lesevorgängen.
- Führen Sie bedingte Schreibvorgänge mit MRSC-Garantien durch.
- Bereinigen Sie die globalen MRSC-Tabellenressourcen.

Bash

AWS CLI mit Bash-Skript

Erstellen Sie eine Tabelle mit multiregionaler starker Konsistenz (Multiregion Strong Consistency, MRSC).

```
# Step 1: Create a new table in us-east-2 (primary region for MRSC)
# Note: Table must be empty when enabling MRSC
aws dynamodb create-table \
    --table-name MusicTable \
    --attribute-definitions \
        AttributeName=Artist,AttributeType=S \
        AttributeName=SongTitle,AttributeType=S \
    --key-schema \
        AttributeName=Artist,KeyType=HASH \
        AttributeName=SongTitle,KeyType=RANGE \
    --billing-mode PAY_PER_REQUEST \
    --region us-east-2

# Wait for table to become active
aws dynamodb wait table-exists --table-name MusicTable --region us-east-2

# Step 2: Add replica and witness with Multi-Region Strong Consistency
# MRSC requires exactly three replicas in supported regions
aws dynamodb update-table \
    --table-name MusicTable \
    --replica-updates '[{"Create": {"RegionName": "us-east-1"}}]' \
    --global-table-witness-updates '[{"Create": {"RegionName": "us-west-2"}}]' \
    --multi-region-consistency STRONG \
    --region us-east-2
```

Überprüfen Sie die MRSC-Konfiguration und den Replikatstatus.

```
# Verify the global table configuration and MRSC setting
aws dynamodb describe-table \
    --table-name MusicTable \
    --region us-east-2 \
    --query 'Table.
{TableName:TableName,TableStatus:TableStatus,MultiRegionConsistency:MultiRegionConsistenc
{Region:RegionName,Status:ReplicaStatus}}'
```

Testen Sie die starke Konsistenz mit sofortigen Lesevorgängen in allen Regionen.

```
# Write an item to the primary region
aws dynamodb put-item \
    --table-name MusicTable \
```

```
    --item '{"Artist": {"S":"The Beatles"},"SongTitle": {"S":"Hey Jude"},"Album":
 {"S":"The Beatles 1967-1970"},"Year": {"N":"1968"}}' \
    --region us-east-2

# Read the item from replica region to verify strong consistency (cannot read or
 write to witness)
# No wait time needed - MRSC provides immediate consistency
echo "Reading from us-east-1 (immediate consistency):"
aws dynamodb get-item \
    --table-name MusicTable \
    --key '{"Artist": {"S":"The Beatles"},"SongTitle": {"S":"Hey Jude"}}' \
    --consistent-read \
    --region us-east-1
```

Führen Sie bedingte Schreibvorgänge mit MRSC-Garantien durch.

```
# Perform a conditional update from a different region
# This demonstrates that conditions work consistently across all regions
aws dynamodb update-item \
    --table-name MusicTable \
    --key '{"Artist": {"S":"The Beatles"},"SongTitle": {"S":"Hey Jude"}}' \
    --update-expression "SET #rating = :rating" \
    --condition-expression "attribute_exists(Artist)" \
    --expression-attribute-names '{"#rating": "Rating"}' \
    --expression-attribute-values '{":rating": {"N":"5"}}' \
    --region us-east-1
```

Bereinigen Sie die globalen MRSC-Tabellenressourcen.

```
# Remove replica tables (must be done before deleting the primary table)
aws dynamodb update-table \
    --table-name MusicTable \
    --replica-updates '[{"Delete": {"RegionName": "us-east-1"}}]' \
    --global-table-witness-updates '[{"Delete": {"RegionName": "us-west-2"}}]' \
    --region us-east-2

# Wait for replicas to be deleted
echo "Waiting for replicas to be deleted..."
sleep 30

# Delete the primary table
```

```
aws dynamodb delete-table \
    --table-name MusicTable \
    --region us-east-2
```

- Weitere API-Informationen finden Sie in den folgenden Themen der AWS CLI - Befehlsreferenz.
  - CreateTable
  - DeleteTable
  - DescribeTable
  - GetItem
  - PutItem
  - UpdateItem
  - UpdateTable

Java

SDK für Java 2.x

Erstellen Sie eine regionale Tabelle, die für die MRSC-Konvertierung bereit ist, mit. AWS SDK for Java 2.x

```
   public static CreateTableResponse createRegionalTable(final DynamoDbClient
dynamoDbClient, final String tableName) {

       if (dynamoDbClient == null) {
           throw new IllegalArgumentException("DynamoDB client cannot be null");
       }
       if (tableName == null || tableName.trim().isEmpty()) {
           throw new IllegalArgumentException("Table name cannot be null or
empty");
       }

       try {
           LOGGER.info("Creating regional table: " + tableName + " (must be
empty for MRSC)");

           CreateTableRequest createTableRequest = CreateTableRequest.builder()
               .tableName(tableName)
```

```
                .attributeDefinitions(
                    AttributeDefinition.builder()
                        .attributeName("Artist")
                        .attributeType(ScalarAttributeType.S)
                        .build(),
                    AttributeDefinition.builder()
                        .attributeName("SongTitle")
                        .attributeType(ScalarAttributeType.S)
                        .build())
                .keySchema(
                    KeySchemaElement.builder()
                        .attributeName("Artist")
                        .keyType(KeyType.HASH)
                        .build(),
                    KeySchemaElement.builder()
                        .attributeName("SongTitle")
                        .keyType(KeyType.RANGE)
                        .build())
                .billingMode(BillingMode.PAY_PER_REQUEST)
                .build();

            CreateTableResponse response =
dynamoDbClient.createTable(createTableRequest);
            LOGGER.info("Regional table creation initiated. Status: "
                + response.tableDescription().tableStatus());

            return response;

        } catch (DynamoDbException e) {
            LOGGER.severe("Failed to create regional table: " + tableName + " - "
+ e.getMessage());
            throw DynamoDbException.builder()
                .message("Failed to create regional table: " + tableName)
                .cause(e)
                .build();
        }
    }
```

Konvertiert eine regionale Tabelle in eine MRSC-Tabelle mit Replikaten und Zeugen unter Verwendung von. AWS SDK for Java 2.x

```
    public static UpdateTableResponse convertToMRSCWithWitness(
```

```
        final DynamoDbClient dynamoDbClient,
        final String tableName,
        final Region replicaRegion,
        final Region witnessRegion) {

        if (dynamoDbClient == null) {
            throw new IllegalArgumentException("DynamoDB client cannot be null");
        }
        if (tableName == null || tableName.trim().isEmpty()) {
            throw new IllegalArgumentException("Table name cannot be null or
empty");
        }
        if (replicaRegion == null) {
            throw new IllegalArgumentException("Replica region cannot be null");
        }
        if (witnessRegion == null) {
            throw new IllegalArgumentException("Witness region cannot be null");
        }

        try {
            LOGGER.info("Converting table to MRSC with replica in " +
replicaRegion.id() + " and witness in "
                + witnessRegion.id());

            // Create replica update using ReplicationGroupUpdate
            ReplicationGroupUpdate replicaUpdate =
ReplicationGroupUpdate.builder()
                .create(CreateReplicationGroupMemberAction.builder()
                    .regionName(replicaRegion.id())
                    .build())
                .build();

            // Create witness update
            GlobalTableWitnessGroupUpdate witnessUpdate =
GlobalTableWitnessGroupUpdate.builder()
                .create(CreateGlobalTableWitnessGroupMemberAction.builder()
                    .regionName(witnessRegion.id())
                    .build())
                .build();

            UpdateTableRequest updateTableRequest = UpdateTableRequest.builder()
                .tableName(tableName)
                .replicaUpdates(List.of(replicaUpdate))
                .globalTableWitnessUpdates(List.of(witnessUpdate))
```

```
                .multiRegionConsistency(MultiRegionConsistency.STRONG)
                .build();

            UpdateTableResponse response =
dynamoDbClient.updateTable(updateTableRequest);
            LOGGER.info("MRSC conversion initiated. Status: "
                + response.tableDescription().tableStatus());
            LOGGER.info("UpdateTableResponse full object: " + response);
            return response;

        } catch (DynamoDbException e) {
            LOGGER.severe("Failed to convert table to MRSC: " + tableName + " - "
+ e.getMessage());
            throw DynamoDbException.builder()
                .message("Failed to convert table to MRSC: " + tableName)
                .cause(e)
                .build();
        }
    }
```

Beschreiben Sie eine globale MRSC-Tabellenkonfiguration mit. AWS SDK for Java 2.x

```
    public static DescribeTableResponse describeMRSCTable(final DynamoDbClient
dynamoDbClient, final String tableName) {

        if (dynamoDbClient == null) {
            throw new IllegalArgumentException("DynamoDB client cannot be null");
        }
        if (tableName == null || tableName.trim().isEmpty()) {
            throw new IllegalArgumentException("Table name cannot be null or
empty");
        }

        try {
            LOGGER.info("Describing MRSC global table: " + tableName);

            DescribeTableRequest request =
                DescribeTableRequest.builder().tableName(tableName).build();

            DescribeTableResponse response =
dynamoDbClient.describeTable(request);
```

```
            LOGGER.info("Table status: " + response.table().tableStatus());
            LOGGER.info("Multi-region consistency: " +
response.table().multiRegionConsistency());

            if (response.table().replicas() != null
                && !response.table().replicas().isEmpty()) {
                LOGGER.info("Number of replicas: " +
response.table().replicas().size());
                response.table()
                    .replicas()
                    .forEach(replica -> LOGGER.info(
                        "Replica region: " + replica.regionName() + ", Status: "
+ replica.replicaStatus()));
            }

            if (response.table().globalTableWitnesses() != null
                && !response.table().globalTableWitnesses().isEmpty()) {
                LOGGER.info("Number of witnesses: "
                    + response.table().globalTableWitnesses().size());
                response.table()
                    .globalTableWitnesses()
                    .forEach(witness -> LOGGER.info(
                        "Witness region: " + witness.regionName() + ", Status: "
+ witness.witnessStatus()));
            }

            return response;

        } catch (ResourceNotFoundException e) {
            LOGGER.severe("Table not found: " + tableName + " - " +
e.getMessage());
            throw DynamoDbException.builder()
                .message("Table not found: " + tableName)
                .cause(e)
                .build();
        } catch (DynamoDbException e) {
            LOGGER.severe("Failed to describe table: " + tableName + " - " +
e.getMessage());
            throw DynamoDbException.builder()
                .message("Failed to describe table: " + tableName)
                .cause(e)
                .build();
        }
    }
```

Fügen Sie mit AWS SDK for Java 2.x Testelemente hinzu, um die starke Konsistenz von MRSC zu überprüfen.

```
    public static PutItemResponse putTestItem(
        final DynamoDbClient dynamoDbClient,
        final String tableName,
        final String artist,
        final String songTitle,
        final String album,
        final String year) {

        if (dynamoDbClient == null) {
            throw new IllegalArgumentException("DynamoDB client cannot be null");
        }
        if (tableName == null || tableName.trim().isEmpty()) {
            throw new IllegalArgumentException("Table name cannot be null or
empty");
        }
        if (artist == null || artist.trim().isEmpty()) {
            throw new IllegalArgumentException("Artist cannot be null or empty");
        }
        if (songTitle == null || songTitle.trim().isEmpty()) {
            throw new IllegalArgumentException("Song title cannot be null or
empty");
        }

        try {
            LOGGER.info("Adding test item to MRSC global table: " + tableName);

            Map<String, AttributeValue> item = new HashMap<>();
            item.put("Artist", AttributeValue.builder().s(artist).build());
            item.put("SongTitle", AttributeValue.builder().s(songTitle).build());

            if (album != null && !album.trim().isEmpty()) {
                item.put("Album", AttributeValue.builder().s(album).build());
            }
            if (year != null && !year.trim().isEmpty()) {
                item.put("Year", AttributeValue.builder().n(year).build());
            }

            PutItemRequest putItemRequest =
```

```
                PutItemRequest.builder().tableName(tableName).item(item).build();

            PutItemResponse response = dynamoDbClient.putItem(putItemRequest);
            LOGGER.info("Test item added successfully with strong consistency");

            return response;

        } catch (DynamoDbException e) {
            LOGGER.severe("Failed to add test item to table: " + tableName + " -
" + e.getMessage());
            throw DynamoDbException.builder()
                .message("Failed to add test item to table: " + tableName)
                .cause(e)
                .build();
        }
    }
```

Lesen Sie Elemente mit konsistenten Lesevorgängen aus MRSC-Replikaten unter Verwendung von. AWS SDK for Java 2.x

```
    public static GetItemResponse getItemWithConsistentRead(
        final DynamoDbClient dynamoDbClient, final String tableName, final String
artist, final String songTitle) {

        if (dynamoDbClient == null) {
            throw new IllegalArgumentException("DynamoDB client cannot be null");
        }
        if (tableName == null || tableName.trim().isEmpty()) {
            throw new IllegalArgumentException("Table name cannot be null or
empty");
        }
        if (artist == null || artist.trim().isEmpty()) {
            throw new IllegalArgumentException("Artist cannot be null or empty");
        }
        if (songTitle == null || songTitle.trim().isEmpty()) {
            throw new IllegalArgumentException("Song title cannot be null or
empty");
        }

        try {
            LOGGER.info("Reading item from MRSC global table with consistent
read: " + tableName);
```

```
            Map<String, AttributeValue> key = new HashMap<>();
            key.put("Artist", AttributeValue.builder().s(artist).build());
            key.put("SongTitle", AttributeValue.builder().s(songTitle).build());

            GetItemRequest getItemRequest = GetItemRequest.builder()
                .tableName(tableName)
                .key(key)
                .consistentRead(true)
                .build();

            GetItemResponse response = dynamoDbClient.getItem(getItemRequest);

            if (response.hasItem()) {
                LOGGER.info("Item found with strong consistency - no wait time
needed");
            } else {
                LOGGER.info("Item not found");
            }

            return response;

        } catch (DynamoDbException e) {
            LOGGER.severe("Failed to read item from table: " + tableName + " - "
+ e.getMessage());
            throw DynamoDbException.builder()
                .message("Failed to read item from table: " + tableName)
                .cause(e)
                .build();
        }
    }
```

Führen Sie bedingte Aktualisierungen mit MRSC-Garantien durch, indem Sie. AWS SDK for Java 2.x

```
    public static UpdateItemResponse performConditionalUpdate(
        final DynamoDbClient dynamoDbClient,
        final String tableName,
        final String artist,
        final String songTitle,
        final String rating) {
```

```
        if (dynamoDbClient == null) {
            throw new IllegalArgumentException("DynamoDB client cannot be null");
        }
        if (tableName == null || tableName.trim().isEmpty()) {
            throw new IllegalArgumentException("Table name cannot be null or
empty");
        }
        if (artist == null || artist.trim().isEmpty()) {
            throw new IllegalArgumentException("Artist cannot be null or empty");
        }
        if (songTitle == null || songTitle.trim().isEmpty()) {
            throw new IllegalArgumentException("Song title cannot be null or
empty");
        }
        if (rating == null || rating.trim().isEmpty()) {
            throw new IllegalArgumentException("Rating cannot be null or empty");
        }

        try {
            LOGGER.info("Performing conditional update on MRSC global table: " +
tableName);

            Map<String, AttributeValue> key = new HashMap<>();
            key.put("Artist", AttributeValue.builder().s(artist).build());
            key.put("SongTitle", AttributeValue.builder().s(songTitle).build());

            Map<String, String> expressionAttributeNames = new HashMap<>();
            expressionAttributeNames.put("#rating", "Rating");

            Map<String, AttributeValue> expressionAttributeValues = new
HashMap<>();
            expressionAttributeValues.put(
                ":rating", AttributeValue.builder().n(rating).build());

            UpdateItemRequest updateItemRequest = UpdateItemRequest.builder()
                .tableName(tableName)
                .key(key)
                .updateExpression("SET #rating = :rating")
                .conditionExpression("attribute_exists(Artist)")
                .expressionAttributeNames(expressionAttributeNames)
                .expressionAttributeValues(expressionAttributeValues)
                .build();
```

```
            UpdateItemResponse response =
dynamoDbClient.updateItem(updateItemRequest);
            LOGGER.info("Conditional update successful - demonstrates strong
consistency");

            return response;

        } catch (ConditionalCheckFailedException e) {
            LOGGER.warning("Conditional check failed: " + e.getMessage());
            throw e;
        } catch (DynamoDbException e) {
            LOGGER.severe("Failed to perform conditional update: " + tableName +
" - " + e.getMessage());
            throw DynamoDbException.builder()
                .message("Failed to perform conditional update: " + tableName)
                .cause(e)
                .build();
        }
    }
```

Warten Sie, bis MRSC-Replikate und Zeugen die Nutzung aktiviert haben. AWS SDK for Java 2.x

```
    public static void waitForMRSCReplicasActive(
        final DynamoDbClient dynamoDbClient, final String tableName, final int
maxWaitTimeSeconds)
        throws InterruptedException {

        if (dynamoDbClient == null) {
            throw new IllegalArgumentException("DynamoDB client cannot be null");
        }
        if (tableName == null || tableName.trim().isEmpty()) {
            throw new IllegalArgumentException("Table name cannot be null or
empty");
        }
        if (maxWaitTimeSeconds <= 0) {
            throw new IllegalArgumentException("Max wait time must be positive");
        }

        try {
            LOGGER.info("Waiting for MRSC replicas and witnesses to become
active: " + tableName);
```

```
            final long startTime = System.currentTimeMillis();
            final long maxWaitTimeMillis = maxWaitTimeSeconds * 1000L;
            int backoffSeconds = 5; // Start with 5 second intervals
            final int maxBackoffSeconds = 30; // Cap at 30 seconds

            while (System.currentTimeMillis() - startTime < maxWaitTimeMillis) {
                DescribeTableResponse response =
describeMRSCTable(dynamoDbClient, tableName);

                boolean allActive = true;
                StringBuilder statusReport = new StringBuilder();

                if (response.table().multiRegionConsistency() == null
                    || !MultiRegionConsistency.STRONG
                        .toString()

.equals(response.table().multiRegionConsistency().toString())) {
                    allActive = false;
                    statusReport
                        .append("MultiRegionConsistency: ")
                        .append(response.table().multiRegionConsistency())
                        .append(" ");
                }
                if (response.table().replicas() == null
                    || response.table().replicas().isEmpty()) {
                    allActive = false;
                    statusReport.append("No replicas found. ");
                }
                if (response.table().globalTableWitnesses() == null
                    || response.table().globalTableWitnesses().isEmpty()) {
                    allActive = false;
                    statusReport.append("No witnesses found. ");
                }

                // Check table status
                if (!"ACTIVE".equals(response.table().tableStatus().toString()))
{
                    allActive = false;
                    statusReport
                        .append("Table: ")
                        .append(response.table().tableStatus())
                        .append(" ");
                }
```

```
                // Check replica status
                if (response.table().replicas() != null) {
                    for (var replica : response.table().replicas()) {
                        if (!"ACTIVE".equals(replica.replicaStatus().toString()))
{
                            allActive = false;
                            statusReport
                                .append("Replica(")
                                .append(replica.regionName())
                                .append("): ")
                                .append(replica.replicaStatus())
                                .append(" ");
                        }
                    }
                }

                // Check witness status
                if (response.table().globalTableWitnesses() != null) {
                    for (var witness : response.table().globalTableWitnesses()) {
                        if (!"ACTIVE".equals(witness.witnessStatus().toString()))
{
                            allActive = false;
                            statusReport
                                .append("Witness(")
                                .append(witness.regionName())
                                .append("): ")
                                .append(witness.witnessStatus())
                                .append(" ");
                        }
                    }
                }

                if (allActive) {
                    LOGGER.info("All MRSC replicas and witnesses are now active:
" + tableName);
                    return;
                }

                LOGGER.info("Waiting for MRSC components to become active.
Status: " + statusReport.toString());
                LOGGER.info("Next check in " + backoffSeconds + " seconds...");

                tempWait(backoffSeconds);
```

```
                // Exponential backoff with cap
                backoffSeconds = Math.min(backoffSeconds * 2, maxBackoffSeconds);
            }

            throw DynamoDbException.builder()
                .message("Timeout waiting for MRSC replicas to become active
after " + maxWaitTimeSeconds + " seconds")
                .build();

        } catch (DynamoDbException | InterruptedException e) {
            LOGGER.severe("Failed to wait for MRSC replicas to become active: " +
tableName + " - " + e.getMessage());
            throw e;
        }
    }
```

Bereinigen Sie MRSC-Replikate und Zeugen mithilfe von. AWS SDK for Java 2.x

```
    public static UpdateTableResponse cleanupMRSCReplicas(
        final DynamoDbClient dynamoDbClient,
        final String tableName,
        final Region replicaRegion,
        final Region witnessRegion) {

        if (dynamoDbClient == null) {
            throw new IllegalArgumentException("DynamoDB client cannot be null");
        }
        if (tableName == null || tableName.trim().isEmpty()) {
            throw new IllegalArgumentException("Table name cannot be null or
empty");
        }
        if (replicaRegion == null) {
            throw new IllegalArgumentException("Replica region cannot be null");
        }
        if (witnessRegion == null) {
            throw new IllegalArgumentException("Witness region cannot be null");
        }

        try {
            LOGGER.info("Cleaning up MRSC replicas and witnesses for table: " +
tableName);
```

```
            // Remove replica using ReplicationGroupUpdate
            ReplicationGroupUpdate replicaUpdate =
ReplicationGroupUpdate.builder()
                .delete(DeleteReplicationGroupMemberAction.builder()
                    .regionName(replicaRegion.id())
                    .build())
                .build();

            // Remove witness
            GlobalTableWitnessGroupUpdate witnessUpdate =
GlobalTableWitnessGroupUpdate.builder()
                .delete(DeleteGlobalTableWitnessGroupMemberAction.builder()
                    .regionName(witnessRegion.id())
                    .build())
                .build();

            UpdateTableRequest updateTableRequest = UpdateTableRequest.builder()
                .tableName(tableName)
                .replicaUpdates(List.of(replicaUpdate))
                .globalTableWitnessUpdates(List.of(witnessUpdate))
                .build();

            UpdateTableResponse response =
dynamoDbClient.updateTable(updateTableRequest);
            LOGGER.info("MRSC cleanup initiated - removing replica and witness.
Response: " + response);

            return response;

        } catch (DynamoDbException e) {
            LOGGER.severe("Failed to cleanup MRSC replicas: " + tableName + " - "
+ e.getMessage());
            throw DynamoDbException.builder()
                .message("Failed to cleanup MRSC replicas: " + tableName)
                .cause(e)
                .build();
        }
    }
```

Vollständige Demonstration des MRSC-Workflows mithilfe von. AWS SDK for Java 2.x

```
   public static void demonstrateCompleteMRSCWorkflow(
       final DynamoDbClient primaryClient,
       final DynamoDbClient replicaClient,
       final String tableName,
       final Region replicaRegion,
       final Region witnessRegion)
       throws InterruptedException {

       if (primaryClient == null) {
           throw new IllegalArgumentException("Primary DynamoDB client cannot be
null");
       }
       if (replicaClient == null) {
           throw new IllegalArgumentException("Replica DynamoDB client cannot be
null");
       }
       if (tableName == null || tableName.trim().isEmpty()) {
           throw new IllegalArgumentException("Table name cannot be null or
empty");
       }
       if (replicaRegion == null) {
           throw new IllegalArgumentException("Replica region cannot be null");
       }
       if (witnessRegion == null) {
           throw new IllegalArgumentException("Witness region cannot be null");
       }

       try {
           LOGGER.info("=== Starting Complete MRSC Workflow Demonstration ===");

           // Step 1: Create an empty single-Region table
           LOGGER.info("Step 1: Creating empty single-Region table");
           createRegionalTable(primaryClient, tableName);

           // Use the existing GlobalTableOperations method for basic table
waiting
           LOGGER.info("Intermediate step: Waiting for table [" + tableName + "]
to become active before continuing");
           GlobalTableOperations.waitForTableActive(primaryClient, tableName);

           // Step 2: Convert to MRSC with replica and witness
           LOGGER.info("Step 2: Converting to MRSC with replica and witness");
```

```
            convertToMRSCWithWitness(primaryClient, tableName, replicaRegion,
witnessRegion);

            // Wait for MRSC conversion to complete using MRSC-specific waiter
            LOGGER.info("Waiting for MRSC conversion to complete...");
            waitForMRSCReplicasActive(primaryClient, tableName);

            LOGGER.info("Intermediate step: Waiting for table [" + tableName + "]
to become active before continuing");
            GlobalTableOperations.waitForTableActive(primaryClient, tableName);

            // Step 3: Verify MRSC configuration
            LOGGER.info("Step 3: Verifying MRSC configuration");
            describeMRSCTable(primaryClient, tableName);

            // Step 4: Test strong consistency with data operations
            LOGGER.info("Step 4: Testing strong consistency with data
operations");

            // Add test item to primary region
            putTestItem(primaryClient, tableName, "The Beatles", "Hey Jude", "The
Beatles 1967-1970", "1968");

            // Immediately read from replica region (no wait needed with MRSC)
            LOGGER.info("Reading from replica region immediately (strong
consistency):");
            GetItemResponse getResponse =
                getItemWithConsistentRead(replicaClient, tableName, "The
Beatles", "Hey Jude");

            if (getResponse.hasItem()) {
                LOGGER.info("# Strong consistency verified - item immediately
available in replica region");
            } else {
                LOGGER.warning("# Item not found in replica region");
            }

            // Test conditional update from replica region
            LOGGER.info("Testing conditional update from replica region:");
            performConditionalUpdate(replicaClient, tableName, "The Beatles",
"Hey Jude", "5");
            LOGGER.info("# Conditional update successful - demonstrates strong
consistency");
```

```
            // Step 5: Cleanup
            LOGGER.info("Step 5: Cleaning up resources");
            cleanupMRSCReplicas(primaryClient, tableName, replicaRegion,
witnessRegion);

            // Wait for cleanup to complete using basic table waiter
            LOGGER.info("Waiting for replica cleanup to complete...");
            GlobalTableOperations.waitForTableActive(primaryClient, tableName);

            // "Halt" until replica/witness cleanup is complete
            DescribeTableResponse cleanupVerification =
describeMRSCTable(primaryClient, tableName);
            int backoffSeconds = 5; // Start with 5 second intervals
            while (cleanupVerification.table().multiRegionConsistency() != null)
{
                LOGGER.info("Waiting additional time (" + backoffSeconds + "
seconds) for MRSC cleanup to complete...");
                tempWait(backoffSeconds);

                // Exponential backoff with cap
                backoffSeconds = Math.min(backoffSeconds * 2, 30);
                cleanupVerification = describeMRSCTable(primaryClient,
tableName);
            }

            // Delete the primary table
            deleteTable(primaryClient, tableName);

            LOGGER.info("=== MRSC Workflow Demonstration Complete ===");
            LOGGER.info("");
            LOGGER.info("Key benefits of Multi-Region Strong Consistency
(MRSC):");
            LOGGER.info("- Immediate consistency across all regions (no eventual
consistency delays)");
            LOGGER.info("- Simplified application logic (no need to handle
eventual consistency)");
            LOGGER.info("- Support for conditional writes and transactions across
regions");
            LOGGER.info("- Consistent read operations from any region without
waiting");

        } catch (DynamoDbException | InterruptedException e) {
            LOGGER.severe("MRSC workflow failed: " + e.getMessage());
            throw e;
```

```
        }
    }
```

- Weitere API-Informationen finden Sie in den folgenden Themen der AWS SDK for Java 2.x - API-Referenz.
  - CreateTable
  - DeleteTable
  - DescribeTable
  - GetItem
  - PutItem
  - UpdateItem
  - UpdateTable

Eine vollständige Liste der AWS SDK-Entwicklerhandbücher und Codebeispiele finden Sie unterDynamoDB mit einem SDK verwenden AWS. Dieses Thema enthält auch Informationen zu den ersten Schritten und Details zu früheren SDK-Versionen.

Erstellen und verwalten Sie globale DynamoDB-Tabellen zur Demonstration von MREC mithilfe eines SDK AWS

Im folgenden Codebeispiel wird gezeigt, wie globale DynamoDB-Tabellen mit Replikaten in mehreren Regionen erstellt und verwaltet werden.

- Erstellen Sie eine Tabelle mit einem globalen sekundären Index und DynamoDB Streams.
- Fügen Sie Replikate in verschiedenen Regionen hinzu, um eine globale Tabelle zu erstellen.
- Entfernen Sie Replikate aus einer globalen Tabelle.
- Fügen Sie Testelemente hinzu, um die regionsübergreifende Replikation zu überprüfen.
- Beschreiben Sie die globale Tabellenkonfiguration und den Replikatstatus.

Java

SDK für Java 2.x

Erstellen Sie eine Tabelle mit Global Secondary Index und DynamoDB Streams mithilfe von. AWS SDK for Java 2.x

```
   public static CreateTableResponse createTableWithGSI(
       final DynamoDbClient dynamoDbClient, final String tableName, final String
indexName) {

       if (dynamoDbClient == null) {
           throw new IllegalArgumentException("DynamoDB client cannot be null");
       }
       if (tableName == null || tableName.trim().isEmpty()) {
           throw new IllegalArgumentException("Table name cannot be null or
empty");
       }
       if (indexName == null || indexName.trim().isEmpty()) {
           throw new IllegalArgumentException("Index name cannot be null or
empty");
       }

       try {
           LOGGER.info("Creating table: " + tableName + " with GSI: " +
indexName);

           CreateTableRequest createTableRequest = CreateTableRequest.builder()
               .tableName(tableName)
               .attributeDefinitions(
                   AttributeDefinition.builder()
                       .attributeName("Artist")
                       .attributeType(ScalarAttributeType.S)
                       .build(),
                   AttributeDefinition.builder()
                       .attributeName("SongTitle")
                       .attributeType(ScalarAttributeType.S)
                       .build())
               .keySchema(
                   KeySchemaElement.builder()
                       .attributeName("Artist")
                       .keyType(KeyType.HASH)
                       .build(),
                   KeySchemaElement.builder()
                       .attributeName("SongTitle")
                       .keyType(KeyType.RANGE)
                       .build())
               .billingMode(BillingMode.PAY_PER_REQUEST)
               .globalSecondaryIndexes(GlobalSecondaryIndex.builder()
                   .indexName(indexName)
```

```
                    .keySchema(KeySchemaElement.builder()
                        .attributeName("SongTitle")
                        .keyType(KeyType.HASH)
                        .build())
                    .projection(

Projection.builder().projectionType(ProjectionType.ALL).build())
                    .build())
                .streamSpecification(StreamSpecification.builder()
                    .streamEnabled(true)
                    .streamViewType(StreamViewType.NEW_AND_OLD_IMAGES)
                    .build())
                .build();

            CreateTableResponse response =
dynamoDbClient.createTable(createTableRequest);
            LOGGER.info("Table creation initiated. Status: "
                + response.tableDescription().tableStatus());

            return response;

        } catch (DynamoDbException e) {
            LOGGER.severe("Failed to create table: " + tableName + " - " +
e.getMessage());
            throw e;
        }
    }
```

Warten Sie, bis eine Tabelle aktiv wird mit. AWS SDK for Java 2.x

```
    public static void waitForTableActive(final DynamoDbClient dynamoDbClient,
final String tableName) {

        if (dynamoDbClient == null) {
            throw new IllegalArgumentException("DynamoDB client cannot be null");
        }
        if (tableName == null || tableName.trim().isEmpty()) {
            throw new IllegalArgumentException("Table name cannot be null or
empty");
        }

        try {
```

```
            LOGGER.info("Waiting for table to become active: " + tableName);

            try (DynamoDbWaiter waiter =
                DynamoDbWaiter.builder().client(dynamoDbClient).build()) {
                DescribeTableRequest request =
                    DescribeTableRequest.builder().tableName(tableName).build();

                waiter.waitUntilTableExists(request);
                LOGGER.info("Table is now active: " + tableName);
            }

        } catch (DynamoDbException e) {
            LOGGER.severe("Failed to wait for table to become active: " +
tableName + " - " + e.getMessage());
            throw e;
        }
    }
```

Fügen Sie ein Replikat hinzu, um eine globale Tabelle zu erstellen oder zu erweitern mit AWS SDK for Java 2.x.

```
    public static UpdateTableResponse addReplica(
        final DynamoDbClient dynamoDbClient,
        final String tableName,
        final Region replicaRegion,
        final String indexName,
        final Long readCapacity) {

        if (dynamoDbClient == null) {
            throw new IllegalArgumentException("DynamoDB client cannot be null");
        }
        if (tableName == null || tableName.trim().isEmpty()) {
            throw new IllegalArgumentException("Table name cannot be null or
empty");
        }
        if (replicaRegion == null) {
            throw new IllegalArgumentException("Replica region cannot be null");
        }
        if (indexName == null || indexName.trim().isEmpty()) {
            throw new IllegalArgumentException("Index name cannot be null or
empty");
        }
```

```
        if (readCapacity == null || readCapacity <= 0) {
            throw new IllegalArgumentException("Read capacity must be a positive
number");
        }

        try {
            LOGGER.info("Adding replica in region: " + replicaRegion.id() + " for
table: " + tableName);

            // Create a ReplicationGroupUpdate for adding a replica
            ReplicationGroupUpdate replicationGroupUpdate =
ReplicationGroupUpdate.builder()
                .create(builder -> builder.regionName(replicaRegion.id())
                    .globalSecondaryIndexes(ReplicaGlobalSecondaryIndex.builder()
                        .indexName(indexName)

.provisionedThroughputOverride(ProvisionedThroughputOverride.builder()
                            .readCapacityUnits(readCapacity)
                            .build())
                        .build())
                    .build())
                .build();

            UpdateTableRequest updateTableRequest = UpdateTableRequest.builder()
                .tableName(tableName)
                .replicaUpdates(replicationGroupUpdate)
                .build();

            UpdateTableResponse response =
dynamoDbClient.updateTable(updateTableRequest);
            LOGGER.info("Replica addition initiated in region: " +
replicaRegion.id());

            return response;

        } catch (DynamoDbException e) {
            LOGGER.severe("Failed to add replica in region: " +
replicaRegion.id() + " - " + e.getMessage());
            throw e;
        }
    }
```

Entfernen Sie ein Replikat aus einer globalen Tabelle mit. AWS SDK for Java 2.x

```
   public static UpdateTableResponse removeReplica(
       final DynamoDbClient dynamoDbClient, final String tableName, final Region
replicaRegion) {

       if (dynamoDbClient == null) {
           throw new IllegalArgumentException("DynamoDB client cannot be null");
       }
       if (tableName == null || tableName.trim().isEmpty()) {
           throw new IllegalArgumentException("Table name cannot be null or
empty");
       }
       if (replicaRegion == null) {
           throw new IllegalArgumentException("Replica region cannot be null");
       }

       try {
           LOGGER.info("Removing replica in region: " + replicaRegion.id() + "
for table: " + tableName);

           // Create a ReplicationGroupUpdate for removing a replica
           ReplicationGroupUpdate replicationGroupUpdate =
ReplicationGroupUpdate.builder()
               .delete(builder ->
builder.regionName(replicaRegion.id()).build())
               .build();

           UpdateTableRequest updateTableRequest = UpdateTableRequest.builder()
               .tableName(tableName)
               .replicaUpdates(replicationGroupUpdate)
               .build();

           UpdateTableResponse response =
dynamoDbClient.updateTable(updateTableRequest);
           LOGGER.info("Replica removal initiated in region: " +
replicaRegion.id());

           return response;

       } catch (DynamoDbException e) {
           LOGGER.severe("Failed to remove replica in region: " +
replicaRegion.id() + " - " + e.getMessage());
           throw e;
```

```
        }
    }
```

Fügen Sie Testelemente hinzu, um die Replikation mithilfe von AWS SDK for Java 2.x zu überprüfen.

```
    public static PutItemResponse putTestItem(
        final DynamoDbClient dynamoDbClient, final String tableName, final String
artist, final String songTitle) {

        if (dynamoDbClient == null) {
            throw new IllegalArgumentException("DynamoDB client cannot be null");
        }
        if (tableName == null || tableName.trim().isEmpty()) {
            throw new IllegalArgumentException("Table name cannot be null or
empty");
        }
        if (artist == null || artist.trim().isEmpty()) {
            throw new IllegalArgumentException("Artist cannot be null or empty");
        }
        if (songTitle == null || songTitle.trim().isEmpty()) {
            throw new IllegalArgumentException("Song title cannot be null or
empty");
        }

        try {
            LOGGER.info("Adding test item to table: " + tableName);

            Map<String,
software.amazon.awssdk.services.dynamodb.model.AttributeValue> item = new
HashMap<>();
            item.put(
                "Artist",

software.amazon.awssdk.services.dynamodb.model.AttributeValue.builder()
                    .s(artist)
                    .build());
            item.put(
                "SongTitle",

software.amazon.awssdk.services.dynamodb.model.AttributeValue.builder()
                    .s(songTitle)
```

```
                    .build());

            PutItemRequest putItemRequest =
                PutItemRequest.builder().tableName(tableName).item(item).build();

            PutItemResponse response = dynamoDbClient.putItem(putItemRequest);
            LOGGER.info("Test item added successfully");

            return response;

        } catch (DynamoDbException e) {
            LOGGER.severe("Failed to add test item to table: " + tableName + " -
" + e.getMessage());
            throw e;
        }
    }
```

Beschreiben Sie die globale Tabellenkonfiguration und die Verwendung von AWS SDK for Java 2.x Replikaten.

```
    public static DescribeTableResponse describeTable(final DynamoDbClient
dynamoDbClient, final String tableName) {

        if (dynamoDbClient == null) {
            throw new IllegalArgumentException("DynamoDB client cannot be null");
        }
        if (tableName == null || tableName.trim().isEmpty()) {
            throw new IllegalArgumentException("Table name cannot be null or
empty");
        }

        try {
            LOGGER.info("Describing table: " + tableName);

            DescribeTableRequest request =
                DescribeTableRequest.builder().tableName(tableName).build();

            DescribeTableResponse response =
dynamoDbClient.describeTable(request);

            LOGGER.info("Table status: " + response.table().tableStatus());
            if (response.table().replicas() != null
```

```
                && !response.table().replicas().isEmpty()) {
                LOGGER.info("Number of replicas: " +
response.table().replicas().size());
                response.table()
                    .replicas()
                    .forEach(replica -> LOGGER.info(
                        "Replica region: " + replica.regionName() + ", Status: "
+ replica.replicaStatus()));
            }

            return response;

        } catch (ResourceNotFoundException e) {
            LOGGER.severe("Table not found: " + tableName + " - " +
e.getMessage());
            throw e;
        } catch (DynamoDbException e) {
            LOGGER.severe("Failed to describe table: " + tableName + " - " +
e.getMessage());
            throw e;
        }
    }
```

Vollständiges Beispiel für globale Tabellenoperationen unter Verwendung von AWS SDK for Java 2.x.

```
    public static void exampleUsage(final Region sourceRegion, final Region
replicaRegion) {

        String tableName = "Music";
        String indexName = "SongTitleIndex";
        Long readCapacity = 15L;

        // Create DynamoDB client for the source region
        try (DynamoDbClient dynamoDbClient =
            DynamoDbClient.builder().region(sourceRegion).build()) {

            try {
                // Step 1: Create the initial table with GSI and streams
                LOGGER.info("Step 1: Creating table in source region: " +
sourceRegion.id());
                createTableWithGSI(dynamoDbClient, tableName, indexName);
```

```
                // Step 2: Wait for table to become active
                LOGGER.info("Step 2: Waiting for table to become active");
                waitForTableActive(dynamoDbClient, tableName);

                // Step 3: Add replica in destination region
                LOGGER.info("Step 3: Adding replica in region: " +
replicaRegion.id());
                addReplica(dynamoDbClient, tableName, replicaRegion, indexName,
readCapacity);

                // Step 4: Wait a moment for replica creation to start
                Thread.sleep(5000);

                // Step 5: Describe table to view replica information
                LOGGER.info("Step 5: Describing table to view replicas");
                describeTable(dynamoDbClient, tableName);

                // Step 6: Add a test item to verify replication
                LOGGER.info("Step 6: Adding test item to verify replication");
                putTestItem(dynamoDbClient, tableName, "TestArtist", "TestSong");

                LOGGER.info("Global table setup completed successfully!");
                LOGGER.info("You can verify replication by checking the item in
region: " + replicaRegion.id());

                // Step 7: Remove replica and clean up table
                LOGGER.info("Step 7: Removing replica from region: " +
replicaRegion.id());
                removeReplica(dynamoDbClient, tableName, replicaRegion);
                DeleteTableResponse deleteTableResponse =
dynamoDbClient.deleteTable(
                    DeleteTableRequest.builder().tableName(tableName).build());
                LOGGER.info("MREC global table demonstration completed
successfully!");

            } catch (InterruptedException e) {
                Thread.currentThread().interrupt();
                throw new RuntimeException("Thread was interrupted", e);
            } catch (DynamoDbException e) {
                LOGGER.severe("DynamoDB operation failed: " + e.getMessage());
                throw e;
            }
        }
```

```
    }
```

- Weitere API-Informationen finden Sie in den folgenden Themen der AWS SDK for Java 2.x - API-Referenz.
  - CreateTable
  - DescribeTable
  - PutItem
  - UpdateTable

Eine vollständige Liste der AWS SDK-Entwicklerhandbücher und Codebeispiele finden Sie unterDynamoDB mit einem SDK verwenden AWS. Dieses Thema enthält auch Informationen zu den ersten Schritten und Details zu früheren SDK-Versionen.

Löschen Sie DynamoDB-Daten mithilfe von PartiQL DELETE-Anweisungen mit einem SDK AWS

Im folgenden Codebeispiel wird gezeigt, wie Sie Daten mit PartiQL-DELETE-Anweisungen löschen.

JavaScript

SDK für JavaScript (v3)

Löschen Sie Elemente aus einer DynamoDB-Tabelle mithilfe von PartiQL DELETE-Anweisungen mit. AWS SDK für JavaScript

```
/**
 * This example demonstrates how to delete items from a DynamoDB table using
 PartiQL.
 * It shows different ways to delete documents with various index types.
 */
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";
import {
  DynamoDBDocumentClient,
  ExecuteStatementCommand,
  BatchExecuteStatementCommand,
} from "@aws-sdk/lib-dynamodb";

/**
```

```
 * Delete a single item by its partition key using PartiQL.
 *
 * @param tableName - The name of the DynamoDB table
 * @param partitionKeyName - The name of the partition key attribute
 * @param partitionKeyValue - The value of the partition key
 * @returns The response from the ExecuteStatementCommand
 */
export const deleteItemByPartitionKey = async (
  tableName: string,
  partitionKeyName: string,
  partitionKeyValue: string | number
) => {
  const client = new DynamoDBClient({});
  const docClient = DynamoDBDocumentClient.from(client);

  const params = {
    Statement: `DELETE FROM "${tableName}" WHERE ${partitionKeyName} = ?`,
    Parameters: [partitionKeyValue],
  };

  try {
    const data = await docClient.send(new ExecuteStatementCommand(params));
    console.log("Item deleted successfully");
    return data;
  } catch (err) {
    console.error("Error deleting item:", err);
    throw err;
  }
};

/**
 * Delete an item by its composite key (partition key + sort key) using PartiQL.
 *
 * @param tableName - The name of the DynamoDB table
 * @param partitionKeyName - The name of the partition key attribute
 * @param partitionKeyValue - The value of the partition key
 * @param sortKeyName - The name of the sort key attribute
 * @param sortKeyValue - The value of the sort key
 * @returns The response from the ExecuteStatementCommand
 */
export const deleteItemByCompositeKey = async (
  tableName: string,
  partitionKeyName: string,
  partitionKeyValue: string | number,
```

```
  sortKeyName: string,
  sortKeyValue: string | number
) => {
  const client = new DynamoDBClient({});
  const docClient = DynamoDBDocumentClient.from(client);

  const params = {
    Statement: `DELETE FROM "${tableName}" WHERE ${partitionKeyName} = ? AND
 ${sortKeyName} = ?`,
    Parameters: [partitionKeyValue, sortKeyValue],
  };

  try {
    const data = await docClient.send(new ExecuteStatementCommand(params));
    console.log("Item deleted successfully");
    return data;
  } catch (err) {
    console.error("Error deleting item:", err);
    throw err;
  }
};

/**
 * Delete an item with a condition to ensure the delete only happens if a
 condition is met.
 *
 * @param tableName - The name of the DynamoDB table
 * @param partitionKeyName - The name of the partition key attribute
 * @param partitionKeyValue - The value of the partition key
 * @param conditionAttribute - The attribute to check in the condition
 * @param conditionValue - The value to compare against in the condition
 * @returns The response from the ExecuteStatementCommand
 */
export const deleteItemWithCondition = async (
  tableName: string,
  partitionKeyName: string,
  partitionKeyValue: string | number,
  conditionAttribute: string,
  conditionValue: any
) => {
  const client = new DynamoDBClient({});
  const docClient = DynamoDBDocumentClient.from(client);

  const params = {
```

```
    Statement: `DELETE FROM "${tableName}" WHERE ${partitionKeyName} = ? AND
 ${conditionAttribute} = ?`,
    Parameters: [partitionKeyValue, conditionValue],
  };

  try {
    const data = await docClient.send(new ExecuteStatementCommand(params));
    console.log("Item deleted with condition successfully");
    return data;
  } catch (err) {
    console.error("Error deleting item with condition:", err);
    throw err;
  }
};

/**
 * Batch delete multiple items using PartiQL.
 *
 * @param tableName - The name of the DynamoDB table
 * @param keys - Array of objects containing key information
 * @returns The response from the BatchExecuteStatementCommand
 */
export const batchDeleteItems = async (
  tableName: string,
  keys: Array<{
    partitionKeyName: string;
    partitionKeyValue: string | number;
    sortKeyName?: string;
    sortKeyValue?: string | number;
  }>
) => {
  const client = new DynamoDBClient({});
  const docClient = DynamoDBDocumentClient.from(client);

  // Create statements for each delete
  const statements = keys.map((key) => {
    if (key.sortKeyName && key.sortKeyValue !== undefined) {
      return {
        Statement: `DELETE FROM "${tableName}" WHERE ${key.partitionKeyName} = ?
 AND ${key.sortKeyName} = ?`,
        Parameters: [key.partitionKeyValue, key.sortKeyValue],
      };
    } else {
      return {
```

```
        Statement: `DELETE FROM "${tableName}" WHERE ${key.partitionKeyName} = ?
`,
        Parameters: [key.partitionKeyValue],
      };
    }
  });

  const params = {
    Statements: statements,
  };

  try {
    const data = await docClient.send(new BatchExecuteStatementCommand(params));
    console.log("Items batch deleted successfully");
    return data;
  } catch (err) {
    console.error("Error batch deleting items:", err);
    throw err;
  }
};

/**
 * Delete multiple items that match a filter condition.
 * Note: This performs a scan operation which can be expensive on large tables.
 *
 * @param tableName - The name of the DynamoDB table
 * @param filterAttribute - The attribute to filter on
 * @param filterValue - The value to filter by
 * @returns The response from the ExecuteStatementCommand
 */
export const deleteItemsByFilter = async (
  tableName: string,
  filterAttribute: string,
  filterValue: any
) => {
  const client = new DynamoDBClient({});
  const docClient = DynamoDBDocumentClient.from(client);

  const params = {
    Statement: `DELETE FROM "${tableName}" WHERE ${filterAttribute} = ?`,
    Parameters: [filterValue],
  };

  try {
```

```
    const data = await docClient.send(new ExecuteStatementCommand(params));
    console.log("Items deleted by filter successfully");
    return data;
  } catch (err) {
    console.error("Error deleting items by filter:", err);
    throw err;
  }
};

/**
 * Example usage showing how to delete items with different index types
 */
export const deleteExamples = async () => {
  // Delete an item by partition key (simple primary key)
  await deleteItemByPartitionKey("UsersTable", "userId", "user123");

  // Delete an item by composite key (partition key + sort key)
  await deleteItemByCompositeKey(
    "OrdersTable",
    "orderId",
    "order456",
    "productId",
    "prod789"
  );

  // Delete with a condition
  await deleteItemWithCondition(
    "UsersTable",
    "userId",
    "user789",
    "userStatus",
    "inactive"
  );

  // Batch delete multiple items
  await batchDeleteItems("UsersTable", [
    { partitionKeyName: "userId", partitionKeyValue: "user234" },
    { partitionKeyName: "userId", partitionKeyValue: "user345" },
  ]);

  // Batch delete items with composite keys
  await batchDeleteItems("OrdersTable", [
    {
      partitionKeyName: "orderId",
```

```
      partitionKeyValue: "order567",
      sortKeyName: "productId",
      sortKeyValue: "prod123",
    },
    {
      partitionKeyName: "orderId",
      partitionKeyValue: "order678",
      sortKeyName: "productId",
      sortKeyValue: "prod456",
    },
  ]);

  // Delete items by filter (use with caution)
  await deleteItemsByFilter("UsersTable", "userStatus", "deleted");
};
```

- Weitere API-Informationen finden Sie in den folgenden Themen der AWS SDK für JavaScript -API-Referenz.
  - BatchExecuteStatement
  - ExecuteStatement

Eine vollständige Liste der AWS SDK-Entwicklerhandbücher und Codebeispiele finden Sie unter. DynamoDB mit einem SDK verwenden AWS Dieses Thema enthält auch Informationen zu den ersten Schritten und Details zu früheren SDK-Versionen.

Ermitteln Sie persönliche Schutzausrüstung in Bildern mit Amazon Rekognition mithilfe eines SDK AWS

Das folgende Codebeispiel zeigt, wie eine App erstellt wird, die Amazon Rekognition verwendet, um persönliche Schutzausrüstung (PSA) auf Bildern zu erkennen.

Java

SDK für Java 2.x

Zeigt, wie eine AWS Lambda Funktion erstellt wird, die Bilder mit persönlicher Schutzausrüstung erkennt.

Den vollständigen Quellcode und Anweisungen zur Einrichtung und Ausführung finden Sie im vollständigen Beispiel unter GitHub.

In diesem Beispiel verwendete Dienste

- DynamoDB
- Amazon Rekognition
- Amazon S3
- Amazon SES

Eine vollständige Liste der AWS SDK-Entwicklerhandbücher und Codebeispiele finden Sie unterDynamoDB mit einem SDK verwenden AWS. Dieses Thema enthält auch Informationen zu den ersten Schritten und Details zu früheren SDK-Versionen.

Fügen Sie DynamoDB-Daten mithilfe von PartiQL INSERT-Anweisungen mit einem SDK ein AWS

Im folgenden Codebeispiel wird gezeigt, wie Sie Daten mit PartiQL-INSERT-Anweisungen einfügen.

JavaScript

SDK für JavaScript (v3)

Fügen Sie Elemente mithilfe von PartiQL INSERT-Anweisungen mit in eine DynamoDB-Tabelle ein. AWS SDK für JavaScript

```
/**
 * This example demonstrates how to insert items into a DynamoDB table using
 PartiQL.
 * It shows different ways to insert documents with various index types.
 */
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";
import {
  DynamoDBDocumentClient,
  ExecuteStatementCommand,
  BatchExecuteStatementCommand,
} from "@aws-sdk/lib-dynamodb";

/**
 * Insert a single item into a DynamoDB table using PartiQL.
```

```
 *
 * @param tableName - The name of the DynamoDB table
 * @param item - The item to insert
 * @returns The response from the ExecuteStatementCommand
 */
export const insertItem = async (tableName: string, item: Record<string, any>) =>
 {
  const client = new DynamoDBClient({});
  const docClient = DynamoDBDocumentClient.from(client);

  // Convert the item to a string representation for PartiQL
  const itemString = JSON.stringify(item).replace(/"([^"]+)":/g, '$1:');

  const params = {
    Statement: `INSERT INTO "${tableName}" VALUE ${itemString}`,
  };

  try {
    const data = await docClient.send(new ExecuteStatementCommand(params));
    console.log("Item inserted successfully");
    return data;
  } catch (err) {
    console.error("Error inserting item:", err);
    throw err;
  }
};

/**
 * Insert multiple items into a DynamoDB table using PartiQL batch operation.
 * This is more efficient than inserting items one by one.
 *
 * @param tableName - The name of the DynamoDB table
 * @param items - Array of items to insert
 * @returns The response from the BatchExecuteStatementCommand
 */
export const batchInsertItems = async (tableName: string, items: Record<string,
 any>[]) => {
  const client = new DynamoDBClient({});
  const docClient = DynamoDBDocumentClient.from(client);

  // Create statements for each item
  const statements = items.map((item) => {
    const itemString = JSON.stringify(item).replace(/"([^"]+)":/g, '$1:');
    return {
```

```
      Statement: `INSERT INTO "${tableName}" VALUE ${itemString}`,
    };
  });

  const params = {
    Statements: statements,
  };

  try {
    const data = await docClient.send(new BatchExecuteStatementCommand(params));
    console.log("Items inserted successfully");
    return data;
  } catch (err) {
    console.error("Error batch inserting items:", err);
    throw err;
  }
};

/**
 * Insert an item with a condition to prevent overwriting existing items.
 * This is useful for ensuring you don't accidentally overwrite data.
 *
 * @param tableName - The name of the DynamoDB table
 * @param item - The item to insert
 * @param partitionKeyName - The name of the partition key attribute
 * @returns The response from the ExecuteStatementCommand
 */
export const insertItemWithCondition = async (
  tableName: string,
  item: Record<string, any>,
  partitionKeyName: string
) => {
  const client = new DynamoDBClient({});
  const docClient = DynamoDBDocumentClient.from(client);

  const itemString = JSON.stringify(item).replace(/"([^"]+)":/g, '$1:');
  const partitionKeyValue = JSON.stringify(item[partitionKeyName]);

  const params = {
    Statement: `INSERT INTO "${tableName}" VALUE ${itemString} WHERE
 attribute_not_exists(${partitionKeyName})`,
    Parameters: [{ S: partitionKeyValue }],
  };
```

```
  try {
    const data = await docClient.send(new ExecuteStatementCommand(params));
    console.log("Item inserted with condition successfully");
    return data;
  } catch (err) {
    console.error("Error inserting item with condition:", err);
    throw err;
  }
};

/**
 * Example usage showing how to insert items with different index types
 */
export const insertExamples = async () => {
  // Example table with a simple primary key (just partition key)
  const simpleKeyItem = {
    userId: "user123",
    name: "John Doe",
    email: "john@example.com",
  };
  await insertItem("UsersTable", simpleKeyItem);

  // Example table with composite key (partition key + sort key)
  const compositeKeyItem = {
    orderId: "order456",
    productId: "prod789",
    quantity: 2,
    price: 29.99,
  };
  await insertItem("OrdersTable", compositeKeyItem);

  // Example with Global Secondary Index (GSI)
  // The GSI might be on the email attribute
  const gsiItem = {
    userId: "user789",
    email: "jane@example.com",
    name: "Jane Smith",
    userType: "premium",  // This could be part of a GSI
  };
  await insertItem("UsersTable", gsiItem);

  // Example with Local Secondary Index (LSI)
  // LSI uses the same partition key but different sort key
  const lsiItem = {
```

```
    orderId: "order567",  // Partition key
    productId: "prod123", // Sort key for the table
    orderDate: "2023-11-15", // Potential sort key for an LSI
    quantity: 1,
    price: 19.99,
  };
  await insertItem("OrdersTable", lsiItem);

  // Batch insert example with multiple items
  const batchItems = [
    {
      userId: "user234",
      name: "Alice Johnson",
      email: "alice@example.com",
    },
    {
      userId: "user345",
      name: "Bob Williams",
      email: "bob@example.com",
    },
  ];
  await batchInsertItems("UsersTable", batchItems);
};
```

- Weitere API-Informationen finden Sie in den folgenden Themen der AWS SDK für JavaScript -API-Referenz.
  - BatchExecuteStatement
  - ExecuteStatement

Eine vollständige Liste der AWS SDK-Entwicklerhandbücher und Codebeispiele finden Sie unter. DynamoDB mit einem SDK verwenden AWS Dieses Thema enthält auch Informationen zu den ersten Schritten und Details zu früheren SDK-Versionen.

Aufrufen einer Lambda-Funktion von einem Browser aus

Das folgende Codebeispiel zeigt, wie eine AWS Lambda Funktion von einem Browser aus aufgerufen wird.

JavaScript

SDK für JavaScript (v2)

Sie können eine browserbasierte Anwendung erstellen, die eine AWS Lambda Funktion verwendet, um eine Amazon DynamoDB-Tabelle mit Benutzerauswahlen zu aktualisieren.

Den vollständigen Quellcode und Anweisungen zur Einrichtung und Ausführung finden Sie im vollständigen Beispiel unter. GitHub

In diesem Beispiel verwendete Dienste

- DynamoDB
- Lambda

SDK für JavaScript (v3)

Sie können eine browserbasierte Anwendung erstellen, die eine AWS Lambda Funktion verwendet, um eine Amazon DynamoDB-Tabelle mit Benutzerauswahlen zu aktualisieren. Diese App verwendet v3. AWS SDK für JavaScript

Den vollständigen Quellcode und Anweisungen zur Einrichtung und Ausführung finden Sie im vollständigen Beispiel unter GitHub.

In diesem Beispiel verwendete Dienste

- DynamoDB
- Lambda

Eine vollständige Liste der AWS SDK-Entwicklerhandbücher und Codebeispiele finden Sie unterDynamoDB mit einem SDK verwenden AWS. Dieses Thema enthält auch Informationen zu den ersten Schritten und Details zu früheren SDK-Versionen.

Globale sekundäre DynamoDB-Indizes mit Version 2 verwalten AWS Command Line Interface

Im folgenden Codebeispiel wird gezeigt, wie Sie den gesamten Lebenszyklus globaler sekundärer Indizes verwalten.

- Erstellen Sie eine Tabelle mit einem globalen sekundären Index.

- Fügen Sie einer neuen GSI zu einer vorhandenen Tabelle.
- Aktualisieren (erhöhen) Sie den GSI-Warmdurchsatz.
- Daten abfragen mit GSIs.
- Löschen Sie einen GSI.

Bash

AWS CLI mit Bash-Skript

Erstellen Sie eine Tabelle mit einem globalen sekundären Index.

```
# Create a table with a GSI
aws dynamodb create-table \
    --table-name MusicCollection \
    --attribute-definitions \
        AttributeName=Artist,AttributeType=S \
        AttributeName=SongTitle,AttributeType=S \
        AttributeName=AlbumTitle,AttributeType=S \
    --key-schema \
        AttributeName=Artist,KeyType=HASH \
        AttributeName=SongTitle,KeyType=RANGE \
    --billing-mode PAY_PER_REQUEST \
    --global-secondary-indexes \
        "IndexName=AlbumIndex,\
        KeySchema=[{AttributeName=AlbumTitle,KeyType=HASH}],\
        Projection={ProjectionType=ALL}"
```

Fügen Sie einer vorhandenen Tabelle einen neuen (On-Demand-)GSI hinzu.

```
# Add a new GSI to an existing table
aws dynamodb update-table \
    --table-name MusicCollection \
    --attribute-definitions \
        AttributeName=Genre,AttributeType=S \
    --global-secondary-index-updates \
        "[{\"Create\":{\"IndexName\":\"GenreIndex\",\
        \"KeySchema\":[{\"AttributeName\":\"Genre\",\"KeyType\":\"HASH\"}],\
        \"Projection\":{\"ProjectionType\":\"ALL\"}}}]"
```

Aktualisieren (erhöhen) Sie den GSI-Warmdurchsatz.

```
# Increase the warm throughput of a GSI (default values are 12k reads, 4k writes)
aws dynamodb update-table \
    --table-name MusicCollection \
    --global-secondary-index-updates \
        "[{\"Update\":{\"IndexName\":\"AlbumIndex\",\
        \"WarmThroughput\":{\"ReadUnitsPerSecond\":15000,\"WriteUnitsPerSecond
\":6000}}}]"
```

Daten abfragen mit GSIs.

```
# Query the AlbumIndex GSI
aws dynamodb query \
    --table-name MusicCollection \
    --index-name AlbumIndex \
    --key-condition-expression "AlbumTitle = :album" \
    --expression-attribute-values '{":album":{"S":"Let It Be"}}'

# Query the GenreIndex GSI
aws dynamodb query \
    --table-name MusicCollection \
    --index-name GenreIndex \
    --key-condition-expression "Genre = :genre" \
    --expression-attribute-values '{":genre":{"S":"Jazz"}}'
```

Löschen Sie einen GSI.

```
# Delete a GSI from a table
aws dynamodb update-table \
    --table-name MusicCollection \
    --global-secondary-index-updates \
        "[{\"Delete\":{\"IndexName\":\"GenreIndex\"}}]"
```

- Weitere API-Informationen finden Sie in den folgenden Themen der AWS CLI - Befehlsreferenz.
  - CreateTable
  - DeleteTable

- Query
- UpdateTable

Eine vollständige Liste der AWS SDK-Entwicklerhandbücher und Codebeispiele finden Sie unterDynamoDB mit einem SDK verwenden AWS. Dieses Thema enthält auch Informationen zu den ersten Schritten und Details zu früheren SDK-Versionen.

Verwaltung ressourcenbasierter DynamoDB-Richtlinien mit Version 2 AWS Command Line Interface

Im folgenden Codebeispiel wird gezeigt, wie Sie den gesamten Lebenszyklus ressourcenbasierter Richtlinien für DynamoDB-Tabellen verwalten.

- Erstellen einer Tabelle mit einer Ressourcenrichtlinie
- Rufen Sie eine Ressourcenrichtlinie ab.
- Aktualisieren Sie eine Ressourcenrichtlinie.
- Löschen Sie eine Ressourcenrichtlinie.

Bash

AWS CLI mit Bash-Skript

Erstellen Sie eine Tabelle mit einer Ressourcenrichtlinie.

```
# Step 1: Create a DynamoDB table
aws dynamodb create-table \
    --table-name MusicCollection \
    --attribute-definitions \
        AttributeName=Artist,AttributeType=S \
        AttributeName=SongTitle,AttributeType=S \
    --key-schema \
        AttributeName=Artist,KeyType=HASH \
        AttributeName=SongTitle,KeyType=RANGE \
    --billing-mode PAY_PER_REQUEST

# Step 2: Create a resource-based policy document
cat > policy.json << 'EOF'
{
  "Version":"2012-10-17",
```

```
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "AWS": "arn:aws:iam::123456789012:role/DynamoDBReadOnly"
      },
      "Action": [
        "dynamodb:GetItem",
        "dynamodb:BatchGetItem",
        "dynamodb:Query",
        "dynamodb:Scan"
      ],
      "Resource": "arn:aws:dynamodb:us-west-2:123456789012:table/MusicCollection"
    }
  ]
}
EOF

# Step 3: Attach the resource-based policy to the table
aws dynamodb put-resource-policy \
    --resource-arn arn:aws:dynamodb:us-west-2:123456789012:table/MusicCollection
 \
    --policy file://policy.json
```

Rufen Sie eine Ressourcenrichtlinie ab.

```
# Get the resource-based policy attached to a table
aws dynamodb get-resource-policy \
    --resource-arn arn:aws:dynamodb:us-west-2:123456789012:table/MusicCollection
```

Aktualisieren Sie eine Ressourcenrichtlinie.

```
# Step 1: Create an updated policy document
cat > updated-policy.json << 'EOF'
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "AWS": [
```

```
          "arn:aws:iam::123456789012:role/DynamoDBReadOnly",
          "arn:aws:iam::123456789012:role/DynamoDBAnalytics"
        ]
      },
      "Action": [
        "dynamodb:GetItem",
        "dynamodb:BatchGetItem",
        "dynamodb:Query",
        "dynamodb:Scan"
      ],
      "Resource": "arn:aws:dynamodb:us-west-2:123456789012:table/MusicCollection"
    }
  ]
}
EOF

# Step 2: Update the resource-based policy on the table
aws dynamodb put-resource-policy \
    --resource-arn arn:aws:dynamodb:us-west-2:123456789012:table/MusicCollection
 \
    --policy file://updated-policy.json
```

Löschen Sie eine Ressourcenrichtlinie.

```
# Delete the resource-based policy from a table
aws dynamodb delete-resource-policy \
    --resource-arn arn:aws:dynamodb:us-west-2:123456789012:table/MusicCollection
```

- Weitere API-Informationen finden Sie in den folgenden Themen der AWS CLI - Befehlsreferenz.
  - CreateTable
  - DeleteResourcePolicy
  - GetResourcePolicy
  - PutResourcePolicy

Eine vollständige Liste der AWS SDK-Entwicklerhandbücher und Codebeispiele finden Sie unterDynamoDB mit einem SDK verwenden AWS. Dieses Thema enthält auch Informationen zu den ersten Schritten und Details zu früheren SDK-Versionen.

Überwachen Sie die Leistung von Amazon DynamoDB mithilfe eines SDK AWS

Im folgenden Codebeispiel wird gezeigt, wie die Verwendung von DynamoDB durch eine Anwendung zur Leistungsüberwachung konfiguriert wird.

Java

SDK für Java 2.x

Im folgenden Codebeispiel wird gezeigt, wie eine Java-Anwendung zur Leistungsüberwachung von DynamoDB konfiguriert wird. Die Anwendung sendet Metrikdaten, CloudWatch an die Sie die Leistung überwachen können.

Den vollständigen Quellcode und Anweisungen zur Einrichtung und Ausführung finden Sie im vollständigen Beispiel unter GitHub.

In diesem Beispiel verwendete Dienste

- CloudWatch
- DynamoDB

Eine vollständige Liste der AWS SDK-Entwicklerhandbücher und Codebeispiele finden Sie unterDynamoDB mit einem SDK verwenden AWS. Dieses Thema enthält auch Informationen zu den ersten Schritten und Details zu früheren SDK-Versionen.

Führen Sie erweiterte DynamoDB-Abfrageoperationen mit einem SDK durch AWS

In den folgenden Codebeispielen wird gezeigt, wie Sie erweiterte Abfrageoperationen in DynamoDB ausführen.

- Abfragen von Tabellen mithilfe verschiedener Filter- und Bedingungstechniken
- Implementieren der Paginierung für große Ergebnismengen
- Verwenden globaler sekundärer Indizes für alternative Zugriffsmuster
- Anwenden von Konsistenzkontrollen auf Grundlage der Anwendungsanforderungen

Java

SDK für Java 2.x

Abfrage mit stark konsistenten Lesevorgängen mithilfe von AWS SDK for Java 2.x.

```
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.AttributeValue;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.model.QueryRequest;
import software.amazon.awssdk.services.dynamodb.model.QueryResponse;
import software.amazon.awssdk.services.dynamodb.model.ResourceNotFoundException;

import java.util.HashMap;
import java.util.Map;
import java.util.logging.Level;
import java.util.logging.Logger;

    public QueryResponse queryWithConsistentReads(
        final String tableName,
        final String partitionKeyName,
        final String partitionKeyValue,
        final boolean useConsistentRead) {

        CodeSampleUtils.validateTableParameters(tableName, partitionKeyName,
 partitionKeyValue);

        // Create expression attribute names for the column names
        final Map<String, String> expressionAttributeNames = new HashMap<>();
        expressionAttributeNames.put(EXPRESSION_ATTRIBUTE_NAME_PK,
 partitionKeyName);

        // Create expression attribute values for the column values
        final Map<String, AttributeValue> expressionAttributeValues = new
 HashMap<>();
        expressionAttributeValues.put(
            EXPRESSION_ATTRIBUTE_VALUE_PK,
            AttributeValue.builder().s(partitionKeyValue).build());

        // Create the query request
        final QueryRequest queryRequest = QueryRequest.builder()
            .tableName(tableName)
            .keyConditionExpression(KEY_CONDITION_EXPRESSION)
            .expressionAttributeNames(expressionAttributeNames)
```

```
            .expressionAttributeValues(expressionAttributeValues)
            .consistentRead(useConsistentRead)
            .build();

        try {
            final QueryResponse response = dynamoDbClient.query(queryRequest);
            LOGGER.log(Level.INFO, "Query successful. Found {0} items",
 response.count());
            return response;
        } catch (ResourceNotFoundException e) {
            LOGGER.log(Level.SEVERE, "Table not found: {0}", tableName);
            throw e;
        } catch (DynamoDbException e) {
            LOGGER.log(Level.SEVERE, "Error querying with consistent reads", e);
            throw e;
        }
    }
```

Abfrage unter Verwendung eines globalen sekundären Indexes mit AWS SDK for Java 2.x.

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.AttributeValue;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.model.QueryRequest;
import software.amazon.awssdk.services.dynamodb.model.QueryResponse;
import software.amazon.awssdk.services.dynamodb.model.ResourceNotFoundException;

import java.util.HashMap;
import java.util.Map;

    public QueryResponse queryTable(
        final String tableName, final String partitionKeyName, final String
 partitionKeyValue) {

        CodeSampleUtils.validateTableParameters(tableName, partitionKeyName,
 partitionKeyValue);

        // Create expression attribute names for the column names
        final Map<String, String> expressionAttributeNames = new HashMap<>();
        expressionAttributeNames.put(EXPRESSION_ATTRIBUTE_NAME_PK,
 partitionKeyName);
```

```
        // Create expression attribute values for the column values
        final Map<String, AttributeValue> expressionAttributeValues = new
HashMap<>();
        expressionAttributeValues.put(
            EXPRESSION_ATTRIBUTE_VALUE_PK,
            AttributeValue.builder().s(partitionKeyValue).build());

        // Create the query request
        final QueryRequest queryRequest = QueryRequest.builder()
            .tableName(tableName)
            .keyConditionExpression(KEY_CONDITION_EXPRESSION)
            .expressionAttributeNames(expressionAttributeNames)
            .expressionAttributeValues(expressionAttributeValues)
            .build();

        try {
            final QueryResponse response = dynamoDbClient.query(queryRequest);
            System.out.println("Query on base table successful. Found " +
response.count() + " items");
            return response;
        } catch (ResourceNotFoundException e) {
            System.err.format("Error: The Amazon DynamoDB table \"%s\" can't be
found.\n", tableName);
            throw new DynamoDbQueryException("Table not found: " + tableName, e);
        } catch (DynamoDbException e) {
            System.err.println("Error querying base table: " + e.getMessage());
            throw new DynamoDbQueryException("Failed to execute query on base
table", e);
        }
    }

    /**
     * Queries a DynamoDB Global Secondary Index (GSI) by partition key.
     *
     * @param tableName         The name of the DynamoDB table
     * @param indexName         The name of the GSI
     * @param partitionKeyName  The name of the GSI partition key attribute
     * @param partitionKeyValue The value of the GSI partition key to query
     * @return The query response from DynamoDB
     * @throws ResourceNotFoundException if the table or index doesn't exist
     * @throws DynamoDbException if the query fails
     */
    public QueryResponse queryGlobalSecondaryIndex(
```

```
        final String tableName, final String indexName, final String
partitionKeyName, final String partitionKeyValue) {

        CodeSampleUtils.validateTableParameters(tableName, partitionKeyName,
partitionKeyValue);
        CodeSampleUtils.validateStringParameter("Index name", indexName);

        // Create expression attribute names for the column names
        final Map<String, String> expressionAttributeNames = new HashMap<>();
        expressionAttributeNames.put(EXPRESSION_ATTRIBUTE_NAME_IK,
partitionKeyName);

        // Create expression attribute values for the column values
        final Map<String, AttributeValue> expressionAttributeValues = new
HashMap<>();
        expressionAttributeValues.put(
            EXPRESSION_ATTRIBUTE_VALUE_IK,
            AttributeValue.builder().s(partitionKeyValue).build());

        // Create the query request
        final QueryRequest queryRequest = QueryRequest.builder()
            .tableName(tableName)
            .indexName(indexName)
            .keyConditionExpression(GSI_KEY_CONDITION_EXPRESSION)
            .expressionAttributeNames(expressionAttributeNames)
            .expressionAttributeValues(expressionAttributeValues)
            .build();

        try {
            final QueryResponse response = dynamoDbClient.query(queryRequest);
            System.out.println("Query on GSI successful. Found " +
response.count() + " items");
            return response;
        } catch (ResourceNotFoundException e) {
            System.err.format(
                "Error: The Amazon DynamoDB table \"%s\" or index \"%s\" can't be
found.\n", tableName, indexName);
            throw new DynamoDbQueryException("Table or index not found: " +
tableName + "/" + indexName, e);
        } catch (DynamoDbException e) {
            System.err.println("Error querying GSI: " + e.getMessage());
            throw new DynamoDbQueryException("Failed to execute query on GSI",
e);
        }
```

```
    }
```

Abfrage mit Paginierung unter Verwendung von AWS SDK for Java 2.x.

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.AttributeValue;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.model.QueryRequest;
import software.amazon.awssdk.services.dynamodb.model.QueryResponse;
import software.amazon.awssdk.services.dynamodb.model.ResourceNotFoundException;

import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;

    public List<Map<String, AttributeValue>> queryWithPagination(
        final String tableName, final String partitionKeyName, final String
 partitionKeyValue, final int pageSize) {

        CodeSampleUtils.validateTableParameters(tableName, partitionKeyName,
 partitionKeyValue);
        CodeSampleUtils.validatePositiveInteger("Page size", pageSize);

        // Create expression attribute names for the column names
        final Map<String, String> expressionAttributeNames = new HashMap<>();
        expressionAttributeNames.put(EXPRESSION_ATTRIBUTE_NAME_PK,
 partitionKeyName);

        // Create expression attribute values for the column values
        final Map<String, AttributeValue> expressionAttributeValues = new
 HashMap<>();
        expressionAttributeValues.put(
            EXPRESSION_ATTRIBUTE_VALUE_PK,
            AttributeValue.builder().s(partitionKeyValue).build());

        // Create the query request
        QueryRequest.Builder queryRequestBuilder = QueryRequest.builder()
            .tableName(tableName)
            .keyConditionExpression(KEY_CONDITION_EXPRESSION)
            .expressionAttributeNames(expressionAttributeNames)
```

```
            .expressionAttributeValues(expressionAttributeValues)
            .limit(pageSize);

        // List to store all items from all pages
        final List<Map<String, AttributeValue>> allItems = new ArrayList<>();

        // Map to store the last evaluated key for pagination
        Map<String, AttributeValue> lastEvaluatedKey = null;
        int pageNumber = 1;

        try {
            do {
                // If we have a last evaluated key, use it for the next page
                if (lastEvaluatedKey != null) {
                    queryRequestBuilder.exclusiveStartKey(lastEvaluatedKey);
                }

                // Execute the query
                final QueryResponse response =
dynamoDbClient.query(queryRequestBuilder.build());

                // Process the current page of results
                final List<Map<String, AttributeValue>> pageItems =
response.items();
                allItems.addAll(pageItems);

                // Get the last evaluated key for the next page
                lastEvaluatedKey = response.lastEvaluatedKey();
                if (lastEvaluatedKey != null && lastEvaluatedKey.isEmpty()) {
                    lastEvaluatedKey = null;
                }

                System.out.println("Page " + pageNumber + ": Retrieved " +
pageItems.size() + " items (Running total: "
                    + allItems.size() + ")");

                pageNumber++;

            } while (lastEvaluatedKey != null);

            System.out.println("Query with pagination complete. Retrieved a total
of " + allItems.size()
                + " items across " + (pageNumber - 1) + " pages");
```

```
            return allItems;
        } catch (ResourceNotFoundException e) {
            System.err.format("Error: The Amazon DynamoDB table \"%s\" can't be
 found.\n", tableName);
            throw e;
        } catch (DynamoDbException e) {
            System.err.println("Error querying with pagination: " +
 e.getMessage());
            throw e;
        }
    }
```

Abfrage mit komplexen Filtern unter Verwendung von AWS SDK for Java 2.x.

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.AttributeValue;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.model.QueryRequest;
import software.amazon.awssdk.services.dynamodb.model.QueryResponse;
import software.amazon.awssdk.services.dynamodb.model.ResourceNotFoundException;

import java.util.HashMap;
import java.util.Map;
import java.util.logging.Level;
import java.util.logging.Logger;

    public QueryResponse queryWithComplexFilter(
        final String tableName,
        final String partitionKeyName,
        final String partitionKeyValue,
        final String statusAttrName,
        final String activeStatus,
        final String pendingStatus,
        final String priceAttrName,
        final double minPrice,
        final double maxPrice,
        final String categoryAttrName) {

        // Validate parameters
        CodeSampleUtils.validateTableParameters(tableName, partitionKeyName,
 partitionKeyValue);
```

```
        CodeSampleUtils.validateStringParameter("Status attribute name",
statusAttrName);
        CodeSampleUtils.validateStringParameter("Active status", activeStatus);
        CodeSampleUtils.validateStringParameter("Pending status", pendingStatus);
        CodeSampleUtils.validateStringParameter("Price attribute name",
priceAttrName);
        CodeSampleUtils.validateStringParameter("Category attribute name",
categoryAttrName);
        CodeSampleUtils.validateNumericRange("Minimum price", minPrice, 0.0,
Double.MAX_VALUE);
        CodeSampleUtils.validateNumericRange("Maximum price", maxPrice, minPrice,
Double.MAX_VALUE);

        // Create expression attribute names for the column names
        final Map<String, String> expressionAttributeNames = new HashMap<>();
        expressionAttributeNames.put("#pk", partitionKeyName);
        expressionAttributeNames.put(EXPRESSION_ATTRIBUTE_NAME_STATUS,
statusAttrName);
        expressionAttributeNames.put(EXPRESSION_ATTRIBUTE_NAME_PRICE,
priceAttrName);
        expressionAttributeNames.put(EXPRESSION_ATTRIBUTE_NAME_CATEGORY,
categoryAttrName);

        // Create expression attribute values for the column values
        final Map<String, AttributeValue> expressionAttributeValues = new
HashMap<>();
        expressionAttributeValues.put(
            ":pkValue", AttributeValue.builder().s(partitionKeyValue).build());
        expressionAttributeValues.put(
            EXPRESSION_ATTRIBUTE_VALUE_ACTIVE,
            AttributeValue.builder().s(activeStatus).build());
        expressionAttributeValues.put(
            EXPRESSION_ATTRIBUTE_VALUE_PENDING,
            AttributeValue.builder().s(pendingStatus).build());
        expressionAttributeValues.put(
            EXPRESSION_ATTRIBUTE_VALUE_MIN_PRICE,
            AttributeValue.builder().n(String.valueOf(minPrice)).build());
        expressionAttributeValues.put(
            EXPRESSION_ATTRIBUTE_VALUE_MAX_PRICE,
            AttributeValue.builder().n(String.valueOf(maxPrice)).build());

        // Create the query request
        final QueryRequest queryRequest = QueryRequest.builder()
            .tableName(tableName)
```

```
            .keyConditionExpression(KEY_CONDITION_EXPRESSION)
            .filterExpression(FILTER_EXPRESSION)
            .expressionAttributeNames(expressionAttributeNames)
            .expressionAttributeValues(expressionAttributeValues)
            .build();

        return dynamoDbClient.query(queryRequest);
    }
```

Abfrage mit einem dynamisch erstellten Filterausdruck unter Verwendung von AWS SDK for Java 2.x.

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.AttributeValue;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.model.QueryRequest;
import software.amazon.awssdk.services.dynamodb.model.QueryResponse;
import software.amazon.awssdk.services.dynamodb.model.ResourceNotFoundException;

import java.util.HashMap;
import java.util.Map;

    public static QueryResponse queryWithDynamicFilter(
        final String tableName,
        final String partitionKeyName,
        final String partitionKeyValue,
        final Map<String, Object> filterCriteria,
        final Region region,
        final DynamoDbClient dynamoDbClient) {

        validateParameters(tableName, partitionKeyName, partitionKeyValue,
 filterCriteria);

        DynamoDbClient ddbClient = dynamoDbClient;
        boolean shouldClose = false;

        try {
            if (ddbClient == null) {
                ddbClient = createClient(region);
                shouldClose = true;
            }
```

```
            final QueryWithDynamicFilter queryHelper = new
QueryWithDynamicFilter(ddbClient);
            return queryHelper.queryWithDynamicFilter(tableName,
partitionKeyName, partitionKeyValue, filterCriteria);
        } catch (ResourceNotFoundException e) {
            System.err.println("Table not found: " + tableName);
            throw e;
        } catch (DynamoDbException e) {
            System.err.println("Failed to execute dynamic filter query: " +
e.getMessage());
            throw e;
        } catch (Exception e) {
            System.err.println("Unexpected error during query: " +
e.getMessage());
            throw e;
        } finally {
            if (shouldClose && ddbClient != null) {
                ddbClient.close();
            }
        }
    }

    public static void main(String[] args) {
        final String usage =
            """
                Usage:
                    <tableName> <partitionKeyName> <partitionKeyValue>
<filterAttrName> <filterAttrValue> [region]
                Where:
                    tableName - The Amazon DynamoDB table to query.
                    partitionKeyName - The name of the partition key attribute.
                    partitionKeyValue - The value of the partition key to query.
                    filterAttrName - The name of the attribute to filter on.
                    filterAttrValue - The value to filter by.
                    region (optional) - The AWS region where the table exists.
(Default: us-east-1)
                """;

        if (args.length < 5) {
            System.out.println(usage);
            System.exit(1);
        }
```

```
        final String tableName = args[0];
        final String partitionKeyName = args[1];
        final String partitionKeyValue = args[2];
        final String filterAttrName = args[3];
        final String filterAttrValue = args[4];
        final Region region = args.length > 5 ? Region.of(args[5]) :
Region.US_EAST_1;

        System.out.println("Querying items with dynamic filter: " +
filterAttrName + " = " + filterAttrValue);

        try {
            // Using the builder pattern to create and execute the query
            final QueryResponse response = new DynamicFilterQueryBuilder()
                .withTableName(tableName)
                .withPartitionKeyName(partitionKeyName)
                .withPartitionKeyValue(partitionKeyValue)
                .withFilterCriterion(filterAttrName, filterAttrValue)
                .withRegion(region)
                .execute();

            // Process the results
            System.out.println("Found " + response.count() + " items:");
            response.items().forEach(item -> System.out.println(item));

            // Demonstrate multiple filter criteria
            System.out.println("\nNow querying with multiple filter criteria:");

            Map<String, Object> multipleFilters = new HashMap<>();
            multipleFilters.put(filterAttrName, filterAttrValue);
            multipleFilters.put("status", "active");

            final QueryResponse multiFilterResponse = new
DynamicFilterQueryBuilder()
                .withTableName(tableName)
                .withPartitionKeyName(partitionKeyName)
                .withPartitionKeyValue(partitionKeyValue)
                .withFilterCriteria(multipleFilters)
                .withRegion(region)
                .execute();

            System.out.println("Found " + multiFilterResponse.count() + " items
with multiple filters:");
```

```
            multiFilterResponse.items().forEach(item ->
 System.out.println(item));

        } catch (IllegalArgumentException e) {
            System.err.println("Invalid input: " + e.getMessage());
            System.exit(1);
        } catch (ResourceNotFoundException e) {
            System.err.println("Table not found: " + tableName);
            System.exit(1);
        } catch (DynamoDbException e) {
            System.err.println("DynamoDB error: " + e.getMessage());
            System.exit(1);
        } catch (Exception e) {
            System.err.println("Unexpected error: " + e.getMessage());
            System.exit(1);
        }
    }
```

Abfrage mit einem Filterausdruck und Grenzwert unter Verwendung von AWS SDK for Java 2.x.

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.AttributeValue;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.model.QueryRequest;
import software.amazon.awssdk.services.dynamodb.model.QueryResponse;
import software.amazon.awssdk.services.dynamodb.model.ResourceNotFoundException;

import java.util.HashMap;
import java.util.Map;
import java.util.logging.Level;
import java.util.logging.Logger;

    public QueryResponse queryWithFilterAndLimit(
        final String tableName,
        final String partitionKeyName,
        final String partitionKeyValue,
        final String filterAttrName,
        final String filterAttrValue,
        final int limit) {
```

```
        CodeSampleUtils.validateTableParameters(tableName, partitionKeyName,
partitionKeyValue);
        CodeSampleUtils.validateStringParameter("Filter attribute name",
filterAttrName);
        CodeSampleUtils.validateStringParameter("Filter attribute value",
filterAttrValue);
        CodeSampleUtils.validatePositiveInteger("Limit", limit);

        // Create expression attribute names for the column names
        final Map<String, String> expressionAttributeNames = new HashMap<>();
        expressionAttributeNames.put(EXPRESSION_ATTRIBUTE_NAME_PK,
partitionKeyName);
        expressionAttributeNames.put(EXPRESSION_ATTRIBUTE_NAME_FILTER,
filterAttrName);

        // Create expression attribute values for the column values
        final Map<String, AttributeValue> expressionAttributeValues = new
HashMap<>();
        expressionAttributeValues.put(
            EXPRESSION_ATTRIBUTE_VALUE_PK,
            AttributeValue.builder().s(partitionKeyValue).build());
        expressionAttributeValues.put(
            EXPRESSION_ATTRIBUTE_VALUE_FILTER,
            AttributeValue.builder().s(filterAttrValue).build());

        // Create the filter expression
        final String filterExpression = "#filterAttr = :filterValue";

        // Create the query request
        final QueryRequest queryRequest = QueryRequest.builder()
            .tableName(tableName)
            .keyConditionExpression(KEY_CONDITION_EXPRESSION)
            .filterExpression(filterExpression)
            .expressionAttributeNames(expressionAttributeNames)
            .expressionAttributeValues(expressionAttributeValues)
            .limit(limit)
            .build();

        try {
            final QueryResponse response = dynamoDbClient.query(queryRequest);
            LOGGER.log(Level.INFO, "Query with filter and limit successful. Found
{0} items", response.count());
            LOGGER.log(
```

```
                Level.INFO, "ScannedCount: {0} (total items evaluated before
 filtering)", response.scannedCount());
            return response;
        } catch (ResourceNotFoundException e) {
            LOGGER.log(Level.SEVERE, "Table not found: {0}", tableName);
            throw e;
        } catch (DynamoDbException e) {
            LOGGER.log(Level.SEVERE, "Error querying with filter and limit: {0}",
 e.getMessage());
            throw e;
        }
    }
```

- Weitere API-Informationen finden Sie unter Abfragen in der AWS SDK for Java 2.x -API-Referenz.

JavaScript

SDK für JavaScript (v3)

Abfrage mit stark konsistenten Lesevorgängen unter Verwendung von AWS SDK für JavaScript.

```
const { DynamoDBClient, QueryCommand } = require("@aws-sdk/client-dynamodb");

/**
 * Queries a DynamoDB table with configurable read consistency
 *
 * @param {Object} config - AWS SDK configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {string} partitionKeyName - The name of the partition key
 * @param {string} partitionKeyValue - The value of the partition key
 * @param {boolean} useConsistentRead - Whether to use strongly consistent reads
 * @returns {Promise<Object>} - The query response
 */
async function queryWithConsistentRead(
  config,
  tableName,
  partitionKeyName,
  partitionKeyValue,
  useConsistentRead = false
```

```
) {
  try {
    // Create DynamoDB client
    const client = new DynamoDBClient(config);

    // Construct the query input
    const input = {
      TableName: tableName,
      KeyConditionExpression: "#pk = :pkValue",
      ExpressionAttributeNames: {
        "#pk": partitionKeyName
      },
      ExpressionAttributeValues: {
        ":pkValue": { S: partitionKeyValue }
      },
      ConsistentRead: useConsistentRead
    };

    // Execute the query
    const command = new QueryCommand(input);
    return await client.send(command);
  } catch (error) {
    console.error(`Error querying with consistent read: ${error}`);
    throw error;
  }
}
```

Abfrage unter Verwendung eines globalen sekundären Indexes mit AWS SDK für JavaScript.

```
const { DynamoDBClient, QueryCommand } = require("@aws-sdk/client-dynamodb");

/**
 * Queries a DynamoDB table using the primary key
 *
 * @param {Object} config - AWS SDK configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {string} userId - The user ID to query by (partition key)
 * @returns {Promise<Object>} - The query response
 */
async function queryTable(
  config,
  tableName,
```

```
  userId
) {
  try {
    // Create DynamoDB client
    const client = new DynamoDBClient(config);

    // Construct the query input for the base table
    const input = {
      TableName: tableName,
      KeyConditionExpression: "user_id = :userId",
      ExpressionAttributeValues: {
        ":userId": { S: userId }
      }
    };

    // Execute the query
    const command = new QueryCommand(input);
    return await client.send(command);
  } catch (error) {
    console.error(`Error querying table: ${error}`);
    throw error;
  }
}

/**
 * Queries a DynamoDB Global Secondary Index (GSI)
 *
 * @param {Object} config - AWS SDK configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {string} indexName - The name of the GSI to query
 * @param {string} gameId - The game ID to query by (GSI partition key)
 * @returns {Promise<Object>} - The query response
 */
async function queryGSI(
  config,
  tableName,
  indexName,
  gameId
) {
  try {
    // Create DynamoDB client
    const client = new DynamoDBClient(config);

    // Construct the query input for the GSI
```

```
    const input = {
      TableName: tableName,
      IndexName: indexName,
      KeyConditionExpression: "game_id = :gameId",
      ExpressionAttributeValues: {
        ":gameId": { S: gameId }
      }
    };

    // Execute the query
    const command = new QueryCommand(input);
    return await client.send(command);
  } catch (error) {
    console.error(`Error querying GSI: ${error}`);
    throw error;
  }
}
```

Abfrage mit Paginierung unter Verwendung von AWS SDK für JavaScript.

```
/**
 * Example demonstrating how to handle large query result sets in DynamoDB using
 pagination
 *
 * This example shows:
 * - How to use pagination to handle large result sets
 * - How to use LastEvaluatedKey to retrieve the next page of results
 * - How to construct subsequent query requests using ExclusiveStartKey
 */
const { DynamoDBClient, QueryCommand } = require("@aws-sdk/client-dynamodb");

/**
 * Queries a DynamoDB table with pagination to handle large result sets
 *
 * @param {Object} config - AWS SDK configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {string} partitionKeyName - The name of the partition key
 * @param {string} partitionKeyValue - The value of the partition key
 * @param {number} pageSize - Number of items per page
 * @returns {Promise<Array>} - All items from the query
 */
```

```
async function queryWithPagination(
  config,
  tableName,
  partitionKeyName,
  partitionKeyValue,
  pageSize = 25
) {
  try {
    // Create DynamoDB client
    const client = new DynamoDBClient(config);

    // Initialize variables for pagination
    let lastEvaluatedKey = undefined;
    const allItems = [];
    let pageCount = 0;

    // Loop until all pages are retrieved
    do {
      // Construct the query input
      const input = {
        TableName: tableName,
        KeyConditionExpression: "#pk = :pkValue",
        Limit: pageSize,
        ExpressionAttributeNames: {
          "#pk": partitionKeyName
        },
        ExpressionAttributeValues: {
          ":pkValue": { S: partitionKeyValue }
        }
      };

      // Add ExclusiveStartKey if we have a LastEvaluatedKey from a previous
 query
      if (lastEvaluatedKey) {
        input.ExclusiveStartKey = lastEvaluatedKey;
      }

      // Execute the query
      const command = new QueryCommand(input);
      const response = await client.send(command);

      // Process the current page of results
      pageCount++;
```

```
      console.log(`Processing page ${pageCount} with ${response.Items.length}
 items`);

      // Add the items from this page to our collection
      if (response.Items && response.Items.length > 0) {
        allItems.push(...response.Items);
      }

      // Get the LastEvaluatedKey for the next page
      lastEvaluatedKey = response.LastEvaluatedKey;

    } while (lastEvaluatedKey); // Continue until there are no more pages

    console.log(`Query complete. Retrieved ${allItems.length} items in
 ${pageCount} pages.`);
    return allItems;
  } catch (error) {
    console.error(`Error querying with pagination: ${error}`);
    throw error;
  }
}

/**
 * Example usage:
 *
 * // Query all items in the "AWS DynamoDB" forum with pagination
 * const allItems = await queryWithPagination(
 *   { region: "us-west-2" },
 *   "ForumThreads",
 *   "ForumName",
 *   "AWS DynamoDB",
 *   25 // 25 items per page
 * );
 *
 * console.log(`Total items retrieved: ${allItems.length}`);
 *
 * // Notes on pagination:
 * // - LastEvaluatedKey contains the primary key of the last evaluated item
 * // - When LastEvaluatedKey is undefined/null, there are no more items to
 retrieve
 * // - ExclusiveStartKey tells DynamoDB where to start the next page
 * // - Pagination helps manage memory usage for large result sets
 * // - Each page requires a separate network request to DynamoDB
 */
```

```
module.exports = { queryWithPagination };
```

Abfrage mit komplexen Filtern unter Verwendung von AWS SDK für JavaScript.

```
const { DynamoDBClient, QueryCommand } = require("@aws-sdk/client-dynamodb");

/**
 * Queries a DynamoDB table with a complex filter expression
 *
 * @param {Object} config - AWS SDK configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {string} partitionKeyName - The name of the partition key
 * @param {string} partitionKeyValue - The value of the partition key
 * @param {number|string} minViews - Minimum number of views for filtering
 * @param {number|string} minReplies - Minimum number of replies for filtering
 * @param {string} requiredTag - Tag that must be present in the item's tags set
 * @returns {Promise<Object>} - The query response
 */
async function queryWithComplexFilter(
  config,
  tableName,
  partitionKeyName,
  partitionKeyValue,
  minViews,
  minReplies,
  requiredTag
) {
  try {
    // Create DynamoDB client
    const client = new DynamoDBClient(config);

    // Construct the query input
    const input = {
      TableName: tableName,
      KeyConditionExpression: "#pk = :pkValue",
      FilterExpression: "views >= :minViews AND replies >= :minReplies AND
 contains(tags, :tag)",
      ExpressionAttributeNames: {
        "#pk": partitionKeyName
      },
      ExpressionAttributeValues: {
```

```
        ":pkValue": { S: partitionKeyValue },
        ":minViews": { N: minViews.toString() },
        ":minReplies": { N: minReplies.toString() },
        ":tag": { S: requiredTag }
      }
    };

    // Execute the query
    const command = new QueryCommand(input);
    return await client.send(command);
  } catch (error) {
    console.error(`Error querying with complex filter: ${error}`);
    throw error;
  }
}
```

Abfrage mit einem dynamisch erstellten Filterausdruck unter Verwendung von AWS SDK für JavaScript.

```
const { DynamoDBClient, QueryCommand } = require("@aws-sdk/client-dynamodb");

async function queryWithDynamicFilter(
  config,
  tableName,
  partitionKeyName,
  partitionKeyValue,
  sortKeyName,
  sortKeyValue,
  filterParams = {}
) {
  try {
    // Create DynamoDB client
    const client = new DynamoDBClient(config);

    // Initialize filter expression components
    let filterExpressions = [];
    const expressionAttributeValues = {
      ":pkValue": { S: partitionKeyValue },
      ":skValue": { S: sortKeyValue }
    };
    const expressionAttributeNames = {
      "#pk": partitionKeyName,
```

```
      "#sk": sortKeyName
    };

    // Add status filter if provided
    if (filterParams.status) {
      filterExpressions.push("status = :status");
      expressionAttributeValues[":status"] = { S: filterParams.status };
    }

    // Add minimum views filter if provided
    if (filterParams.minViews !== undefined) {
      filterExpressions.push("views >= :minViews");
      expressionAttributeValues[":minViews"] = { N:
filterParams.minViews.toString() };
    }

    // Add author filter if provided
    if (filterParams.author) {
      filterExpressions.push("author = :author");
      expressionAttributeValues[":author"] = { S: filterParams.author };
    }

    // Construct the query input
    const input = {
      TableName: tableName,
      KeyConditionExpression: "#pk = :pkValue AND #sk = :skValue"
    };

    // Add filter expression if any filters were provided
    if (filterExpressions.length > 0) {
      input.FilterExpression = filterExpressions.join(" AND ");
    }

    // Add expression attribute names and values
    input.ExpressionAttributeNames = expressionAttributeNames;
    input.ExpressionAttributeValues = expressionAttributeValues;

    // Execute the query
    const command = new QueryCommand(input);
    return await client.send(command);
  } catch (error) {
    console.error(`Error querying with dynamic filter: ${error}`);
    throw error;
  }
```

```
}
```

- Weitere API-Informationen finden Sie unter Abfragen in der AWS SDK für JavaScript -API-Referenz.

Python

SDK für Python (Boto3)

Abfrage mit stark konsistenten Lesevorgängen unter Verwendung von AWS SDK für Python (Boto3).

```
import time

import boto3
from boto3.dynamodb.conditions import Key


def query_with_consistent_read(
    table_name,
    partition_key_name,
    partition_key_value,
    sort_key_name=None,
    sort_key_value=None,
    consistent_read=True,
):
    """
    Query a DynamoDB table with the option for strongly consistent reads.

    Args:
        table_name (str): The name of the DynamoDB table.
        partition_key_name (str): The name of the partition key attribute.
        partition_key_value (str): The value of the partition key to query.
        sort_key_name (str, optional): The name of the sort key attribute.
        sort_key_value (str, optional): The value of the sort key to query.
        consistent_read (bool, optional): Whether to use strongly consistent
 reads. Defaults to True.

    Returns:
        dict: The response from DynamoDB containing the query results.
    """
```

```
    # Initialize the DynamoDB resource
    dynamodb = boto3.resource("dynamodb")
    table = dynamodb.Table(table_name)

    # Build the key condition expression
    key_condition = Key(partition_key_name).eq(partition_key_value)

    if sort_key_name and sort_key_value:
        key_condition = key_condition & Key(sort_key_name).eq(sort_key_value)

    # Perform the query with the consistent read option
    response = table.query(KeyConditionExpression=key_condition,
 ConsistentRead=consistent_read)

    return response
```

Abfrage unter Verwendung eines globalen sekundären Indexes mit AWS SDK für Python (Boto3).

```
import boto3
from boto3.dynamodb.conditions import Key


def query_table(table_name, partition_key_name, partition_key_value):
    """
    Query a DynamoDB table using its primary key.

    Args:
        table_name (str): The name of the DynamoDB table.
        partition_key_name (str): The name of the partition key attribute.
        partition_key_value (str): The value of the partition key to query.

    Returns:
        dict: The response from DynamoDB containing the query results.
    """
    # Initialize the DynamoDB resource
    dynamodb = boto3.resource("dynamodb")
    table = dynamodb.Table(table_name)

    # Perform the query on the table's primary key
```

```
    response =
 table.query(KeyConditionExpression=Key(partition_key_name).eq(partition_key_value))

    return response


def query_gsi(table_name, index_name, partition_key_name, partition_key_value):
    """
    Query a Global Secondary Index (GSI) on a DynamoDB table.

    Args:
        table_name (str): The name of the DynamoDB table.
        index_name (str): The name of the Global Secondary Index.
        partition_key_name (str): The name of the GSI's partition key attribute.
        partition_key_value (str): The value of the GSI's partition key to query.

    Returns:
        dict: The response from DynamoDB containing the query results.
    """
    # Initialize the DynamoDB resource
    dynamodb = boto3.resource("dynamodb")
    table = dynamodb.Table(table_name)

    # Perform the query on the GSI
    response = table.query(
        IndexName=index_name,
 KeyConditionExpression=Key(partition_key_name).eq(partition_key_value)
    )

    return response
```

Abfrage mit Paginierung unter Verwendung von AWS SDK für Python (Boto3).

```
import boto3
from boto3.dynamodb.conditions import Key


def query_with_pagination(
    table_name, partition_key_name, partition_key_value, page_size=25,
 max_pages=None
```

```
):
    """
    Query a DynamoDB table with pagination to handle large result sets.

    Args:
        table_name (str): The name of the DynamoDB table.
        partition_key_name (str): The name of the partition key attribute.
        partition_key_value (str): The value of the partition key to query.
        page_size (int, optional): The number of items to return per page.
 Defaults to 25.
        max_pages (int, optional): The maximum number of pages to retrieve. If
 None, retrieves all pages.

    Returns:
        list: All items retrieved from the query across all pages.
    """
    # Initialize the DynamoDB resource
    dynamodb = boto3.resource("dynamodb")
    table = dynamodb.Table(table_name)

    # Initialize variables for pagination
    last_evaluated_key = None
    page_count = 0
    all_items = []

    # Paginate through the results
    while True:
        # Check if we've reached the maximum number of pages
        if max_pages is not None and page_count >= max_pages:
            break

        # Prepare the query parameters
        query_params = {
            "KeyConditionExpression":
Key(partition_key_name).eq(partition_key_value),
            "Limit": page_size,
        }

        # Add the ExclusiveStartKey if we have a LastEvaluatedKey from a previous
query
        if last_evaluated_key:
            query_params["ExclusiveStartKey"] = last_evaluated_key

        # Execute the query
```

```
        response = table.query(**query_params)

        # Process the current page of results
        items = response.get("Items", [])
        all_items.extend(items)

        # Update pagination tracking
        page_count += 1

        # Get the LastEvaluatedKey for the next page, if any
        last_evaluated_key = response.get("LastEvaluatedKey")

        # If there's no LastEvaluatedKey, we've reached the end of the results
        if not last_evaluated_key:
            break

    return all_items


def query_with_pagination_generator(
    table_name, partition_key_name, partition_key_value, page_size=25
):
    """
    Query a DynamoDB table with pagination using a generator to handle large
 result sets.
    This approach is memory-efficient as it yields one page at a time.

    Args:
        table_name (str): The name of the DynamoDB table.
        partition_key_name (str): The name of the partition key attribute.
        partition_key_value (str): The value of the partition key to query.
        page_size (int, optional): The number of items to return per page.
 Defaults to 25.

    Yields:
        tuple: A tuple containing (items, page_number, last_page) where:
            - items is a list of items for the current page
            - page_number is the current page number (starting from 1)
            - last_page is a boolean indicating if this is the last page
    """
    # Initialize the DynamoDB resource
    dynamodb = boto3.resource("dynamodb")
    table = dynamodb.Table(table_name)
```

```
    # Initialize variables for pagination
    last_evaluated_key = None
    page_number = 0

    # Paginate through the results
    while True:
        # Prepare the query parameters
        query_params = {
            "KeyConditionExpression":
Key(partition_key_name).eq(partition_key_value),
            "Limit": page_size,
        }

        # Add the ExclusiveStartKey if we have a LastEvaluatedKey from a previous
query
        if last_evaluated_key:
            query_params["ExclusiveStartKey"] = last_evaluated_key

        # Execute the query
        response = table.query(**query_params)

        # Get the current page of results
        items = response.get("Items", [])
        page_number += 1

        # Get the LastEvaluatedKey for the next page, if any
        last_evaluated_key = response.get("LastEvaluatedKey")

        # Determine if this is the last page
        is_last_page = last_evaluated_key is None

        # Yield the current page of results
        yield (items, page_number, is_last_page)

        # If there's no LastEvaluatedKey, we've reached the end of the results
        if is_last_page:
            break
```

Abfrage mit komplexen Filtern unter Verwendung von AWS SDK für Python (Boto3).

```
import boto3
from boto3.dynamodb.conditions import Attr, Key


def query_with_complex_filter(
    table_name,
    partition_key_name,
    partition_key_value,
    min_rating=None,
    status_list=None,
    max_price=None,
):
    """
    Query a DynamoDB table with a complex filter expression.

    Args:
        table_name (str): The name of the DynamoDB table.
        partition_key_name (str): The name of the partition key attribute.
        partition_key_value (str): The value of the partition key to query.
        min_rating (float, optional): Minimum rating value for filtering.
        status_list (list, optional): List of status values to include.
        max_price (float, optional): Maximum price value for filtering.

    Returns:
        dict: The response from DynamoDB containing the query results.
    """
    # Initialize the DynamoDB resource
    dynamodb = boto3.resource("dynamodb")
    table = dynamodb.Table(table_name)

    # Start with the key condition expression
    key_condition = Key(partition_key_name).eq(partition_key_value)

    # Initialize the filter expression and expression attribute values
    filter_expression = None
    expression_attribute_values = {}

    # Build the filter expression based on provided parameters
    if min_rating is not None:
        filter_expression = Attr("rating").gte(min_rating)
        expression_attribute_values[":min_rating"] = min_rating

    if status_list and len(status_list) > 0:
```

```
        status_condition = None
        for i, status in enumerate(status_list):
            status_value_name = f":status{i}"
            expression_attribute_values[status_value_name] = status

            if status_condition is None:
                status_condition = Attr("status").eq(status)
            else:
                status_condition = status_condition | Attr("status").eq(status)

        if filter_expression is None:
            filter_expression = status_condition
        else:
            filter_expression = filter_expression & status_condition

    if max_price is not None:
        price_condition = Attr("price").lte(max_price)
        expression_attribute_values[":max_price"] = max_price

        if filter_expression is None:
            filter_expression = price_condition
        else:
            filter_expression = filter_expression & price_condition

    # Prepare the query parameters
    query_params = {"KeyConditionExpression": key_condition}

    if filter_expression:
        query_params["FilterExpression"] = filter_expression
        if expression_attribute_values:
            query_params["ExpressionAttributeValues"] =
 expression_attribute_values

    # Execute the query
    response = table.query(**query_params)
    return response


def query_with_complex_filter_and_or(
    table_name,
    partition_key_name,
    partition_key_value,
    category=None,
    min_rating=None,
```

```
    max_price=None,
):
    """
    Query a DynamoDB table with a complex filter expression using AND and OR
 operators.

    Args:
        table_name (str): The name of the DynamoDB table.
        partition_key_name (str): The name of the partition key attribute.
        partition_key_value (str): The value of the partition key to query.
        category (str, optional): Category value for filtering.
        min_rating (float, optional): Minimum rating value for filtering.
        max_price (float, optional): Maximum price value for filtering.

    Returns:
        dict: The response from DynamoDB containing the query results.
    """
    # Initialize the DynamoDB resource
    dynamodb = boto3.resource("dynamodb")
    table = dynamodb.Table(table_name)

    # Start with the key condition expression
    key_condition = Key(partition_key_name).eq(partition_key_value)

    # Build a complex filter expression with AND and OR operators
    filter_expression = None
    expression_attribute_values = {}

    # Build the category condition
    if category:
        filter_expression = Attr("category").eq(category)
        expression_attribute_values[":category"] = category

    # Build the rating and price condition (rating >= min_rating OR price <=
 max_price)
    rating_price_condition = None

    if min_rating is not None:
        rating_price_condition = Attr("rating").gte(min_rating)
        expression_attribute_values[":min_rating"] = min_rating

    if max_price is not None:
        price_condition = Attr("price").lte(max_price)
        expression_attribute_values[":max_price"] = max_price
```

```
        if rating_price_condition is None:
            rating_price_condition = price_condition
        else:
            rating_price_condition = rating_price_condition | price_condition

    # Combine the conditions
    if rating_price_condition:
        if filter_expression is None:
            filter_expression = rating_price_condition
        else:
            filter_expression = filter_expression & rating_price_condition

    # Prepare the query parameters
    query_params = {"KeyConditionExpression": key_condition}

    if filter_expression:
        query_params["FilterExpression"] = filter_expression
        if expression_attribute_values:
            query_params["ExpressionAttributeValues"] =
 expression_attribute_values

    # Execute the query
    response = table.query(**query_params)
    return response
```

Abfrage mit einem dynamisch erstellten Filterausdruck unter Verwendung von AWS SDK für Python (Boto3).

```
import boto3
from boto3.dynamodb.conditions import Attr, Key


def query_with_dynamic_filter(
    table_name, partition_key_name, partition_key_value, filter_conditions=None
):
    """
    Query a DynamoDB table with a dynamically constructed filter expression.

    Args:
```

```
        table_name (str): The name of the DynamoDB table.
        partition_key_name (str): The name of the partition key attribute.
        partition_key_value (str): The value of the partition key to query.
        filter_conditions (dict, optional): A dictionary of filter conditions
where
            keys are attribute names and values are dictionaries with 'operator'
and 'value'.
            Example: {'rating': {'operator': '>=', 'value': 4}, 'status':
{'operator': '=', 'value': 'active'}}

    Returns:
        dict: The response from DynamoDB containing the query results.
    """
    # Initialize the DynamoDB resource
    dynamodb = boto3.resource("dynamodb")
    table = dynamodb.Table(table_name)

    # Start with the key condition expression
    key_condition = Key(partition_key_name).eq(partition_key_value)

    # Initialize variables for the filter expression and attribute values
    filter_expression = None
    expression_attribute_values = {":pk_val": partition_key_value}

    # Dynamically build the filter expression if filter conditions are provided
    if filter_conditions:
        for attr_name, condition in filter_conditions.items():
            operator = condition.get("operator")
            value = condition.get("value")
            attr_value_name = f":{attr_name}"
            expression_attribute_values[attr_value_name] = value

            # Create the appropriate filter expression based on the operator
            current_condition = None
            if operator == "=":
                current_condition = Attr(attr_name).eq(value)
            elif operator == "!=":
                current_condition = Attr(attr_name).ne(value)
            elif operator == ">":
                current_condition = Attr(attr_name).gt(value)
            elif operator == ">=":
                current_condition = Attr(attr_name).gte(value)
            elif operator == "<":
                current_condition = Attr(attr_name).lt(value)
```

```
            elif operator == "<=":
                current_condition = Attr(attr_name).lte(value)
            elif operator == "contains":
                current_condition = Attr(attr_name).contains(value)
            elif operator == "begins_with":
                current_condition = Attr(attr_name).begins_with(value)

            # Combine with existing filter expression using AND
            if current_condition:
                if filter_expression is None:
                    filter_expression = current_condition
                else:
                    filter_expression = filter_expression & current_condition

    # Perform the query with the dynamically built filter expression
    query_params = {"KeyConditionExpression": key_condition}

    if filter_expression:
        query_params["FilterExpression"] = filter_expression

    response = table.query(**query_params)
    return response
```

Abfrage mit einem Filterausdruck und Grenzwert unter Verwendung von AWS SDK für Python (Boto3).

```
import boto3
from boto3.dynamodb.conditions import Attr, Key


def query_with_filter_and_limit(
    table_name,
    partition_key_name,
    partition_key_value,
    filter_attribute=None,
    filter_value=None,
    limit=10,
):
    """
```

```
    Query a DynamoDB table with a filter expression and limit the number of
results.

    Args:
        table_name (str): The name of the DynamoDB table.
        partition_key_name (str): The name of the partition key attribute.
        partition_key_value (str): The value of the partition key to query.
        filter_attribute (str, optional): The attribute name to filter on.
        filter_value (any, optional): The value to compare against in the filter.
        limit (int, optional): The maximum number of items to evaluate. Defaults
to 10.

    Returns:
        dict: The response from DynamoDB containing the query results.
    """
    # Initialize the DynamoDB resource
    dynamodb = boto3.resource("dynamodb")
    table = dynamodb.Table(table_name)

    # Build the key condition expression
    key_condition = Key(partition_key_name).eq(partition_key_value)

    # Prepare the query parameters
    query_params = {"KeyConditionExpression": key_condition, "Limit": limit}

    # Add the filter expression if filter attributes are provided
    if filter_attribute and filter_value is not None:
        query_params["FilterExpression"] =
Attr(filter_attribute).gt(filter_value)
        query_params["ExpressionAttributeValues"] = {":filter_value":
filter_value}

    # Execute the query
    response = table.query(**query_params)
    return response
```

- Weitere API-Informationen finden Sie unter Abfragen in der API-Referenz zum AWS SDK für Python (Boto3).

Eine vollständige Liste der AWS SDK-Entwicklerhandbücher und Codebeispiele finden Sie unterDynamoDB mit einem SDK verwenden AWS. Dieses Thema enthält auch Informationen zu den ersten Schritten und Details zu früheren SDK-Versionen.

Führen Sie Listenoperationen in DynamoDB mit einem SDK durch AWS

Das folgende Codebeispiel veranschaulicht, wie Listenoperationen für DynamoDB-Tabellen ausgeführt werden.

- Hinzufügen von Elementen zu einem Listenattribut
- Entfernen von Elementen aus einem Listenattribut
- Aktualisieren von bestimmten Elementen in einer Liste nach Index
- Verwenden von Funktionen zum Anfügen und zum Auflisten von Indizes

Java

SDK für Java 2.x

Demonstrieren Sie Listenoperationen mit AWS SDK for Java 2.x.

```
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.AttributeValue;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.model.GetItemRequest;
import software.amazon.awssdk.services.dynamodb.model.GetItemResponse;
import software.amazon.awssdk.services.dynamodb.model.UpdateItemRequest;
import software.amazon.awssdk.services.dynamodb.model.UpdateItemResponse;

import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;

    /**
     * Appends items to a list attribute.
     *
     * <p>This method demonstrates how to use the list_append function to add
     * items to the end of a list attribute.
     *
     * @param dynamoDbClient The DynamoDB client
     * @param tableName The name of the DynamoDB table
```

```
     * @param key The key of the item to update
     * @param listAttributeName The name of the list attribute
     * @param itemsToAppend The items to append to the list
     * @return The response from DynamoDB
     * @throws DynamoDbException if an error occurs during the operation
     */
    public static UpdateItemResponse appendToList(
        DynamoDbClient dynamoDbClient,
        String tableName,
        Map<String, AttributeValue> key,
        String listAttributeName,
        List<AttributeValue> itemsToAppend) {

        // Create a list value from the items to append
        AttributeValue listValue =
AttributeValue.builder().l(itemsToAppend).build();

        // Define the update parameters
        UpdateItemRequest request = UpdateItemRequest.builder()
            .tableName(tableName)
            .key(key)
            .updateExpression("SET #attrName =
list_append(if_not_exists(#attrName, :emptyList), :newItems)")
            .expressionAttributeNames(Map.of("#attrName", listAttributeName))
            .expressionAttributeValues(Map.of(
                ":newItems",
                listValue,
                ":emptyList",
                AttributeValue.builder().l(new
ArrayList<AttributeValue>()).build()))
            .returnValues("UPDATED_NEW")
            .build();

        // Perform the update operation
        return dynamoDbClient.updateItem(request);
    }

    /**
     * Prepends items to a list attribute.
     *
     * <p>This method demonstrates how to use the list_append function to add
     * items to the beginning of a list attribute.
     *
     * @param dynamoDbClient The DynamoDB client
```

```
     * @param tableName The name of the DynamoDB table
     * @param key The key of the item to update
     * @param listAttributeName The name of the list attribute
     * @param itemsToPrepend The items to prepend to the list
     * @return The response from DynamoDB
     * @throws DynamoDbException if an error occurs during the operation
     */
    public static UpdateItemResponse prependToList(
        DynamoDbClient dynamoDbClient,
        String tableName,
        Map<String, AttributeValue> key,
        String listAttributeName,
        List<AttributeValue> itemsToPrepend) {

        // Create a list value from the items to prepend
        AttributeValue listValue =
AttributeValue.builder().l(itemsToPrepend).build();

        // Define the update parameters
        // Note: To prepend, we put the new items first in the list_append
function
        UpdateItemRequest request = UpdateItemRequest.builder()
            .tableName(tableName)
            .key(key)
            .updateExpression("SET #attrName = list_append(:newItems,
if_not_exists(#attrName, :emptyList))")
            .expressionAttributeNames(Map.of("#attrName", listAttributeName))
            .expressionAttributeValues(Map.of(
                ":newItems",
                listValue,
                ":emptyList",
                AttributeValue.builder().l(new
ArrayList<AttributeValue>()).build()))
            .returnValues("UPDATED_NEW")
            .build();

        // Perform the update operation
        return dynamoDbClient.updateItem(request);
    }

    /**
     * Updates a specific element in a list attribute.
     *
     * <p>This method demonstrates how to update a specific element in a list
```

```
     * by its index.
     *
     * @param dynamoDbClient The DynamoDB client
     * @param tableName The name of the DynamoDB table
     * @param key The key of the item to update
     * @param listAttributeName The name of the list attribute
     * @param index The index of the element to update
     * @param newValue The new value for the element
     * @return The response from DynamoDB
     * @throws DynamoDbException if an error occurs during the operation
     */
    public static UpdateItemResponse updateListElement(
        DynamoDbClient dynamoDbClient,
        String tableName,
        Map<String, AttributeValue> key,
        String listAttributeName,
        int index,
        AttributeValue newValue) {

        // Define the update parameters
        UpdateItemRequest request = UpdateItemRequest.builder()
            .tableName(tableName)
            .key(key)
            .updateExpression("SET #attrName[" + index + "] = :newValue")
            .expressionAttributeNames(Map.of("#attrName", listAttributeName))
            .expressionAttributeValues(Map.of(":newValue", newValue))
            .returnValues("UPDATED_NEW")
            .build();

        // Perform the update operation
        return dynamoDbClient.updateItem(request);
    }

    /**
     * Removes a specific element from a list attribute.
     *
     * <p>This method demonstrates how to remove a specific element from a list
     * by its index.
     *
     * @param dynamoDbClient The DynamoDB client
     * @param tableName The name of the DynamoDB table
     * @param key The key of the item to update
     * @param listAttributeName The name of the list attribute
     * @param index The index of the element to remove
```

```
     * @return The response from DynamoDB
     * @throws DynamoDbException if an error occurs during the operation
     */
    public static UpdateItemResponse removeListElement(
        DynamoDbClient dynamoDbClient,
        String tableName,
        Map<String, AttributeValue> key,
        String listAttributeName,
        int index) {

        // Define the update parameters
        UpdateItemRequest request = UpdateItemRequest.builder()
            .tableName(tableName)
            .key(key)
            .updateExpression("REMOVE #attrName[" + index + "]")
            .expressionAttributeNames(Map.of("#attrName", listAttributeName))
            .returnValues("UPDATED_NEW")
            .build();

        // Perform the update operation
        return dynamoDbClient.updateItem(request);
    }

    /**
     * Gets the current value of a list attribute.
     *
     * <p>Helper method to retrieve the current value of a list attribute.
     *
     * @param dynamoDbClient The DynamoDB client
     * @param tableName The name of the DynamoDB table
     * @param key The key of the item to get
     * @param listAttributeName The name of the list attribute
     * @return The list attribute value or null if not found
     * @throws DynamoDbException if an error occurs during the operation
     */
    public static List<AttributeValue> getListAttribute(
        DynamoDbClient dynamoDbClient, String tableName, Map<String,
AttributeValue> key, String listAttributeName) {

        // Define the get parameters
        GetItemRequest request = GetItemRequest.builder()
            .tableName(tableName)
            .key(key)
            .projectionExpression(listAttributeName)
```

```
            .build();

        try {
            // Perform the get operation
            GetItemResponse response = dynamoDbClient.getItem(request);

            // Return the list attribute if it exists, otherwise null
            if (response.item() != null &&
response.item().containsKey(listAttributeName)) {
                return response.item().get(listAttributeName).l();
            }

            return null;
        } catch (DynamoDbException e) {
            throw DynamoDbException.builder()
                .message("Failed to get list attribute: " + e.getMessage())
                .cause(e)
                .build();
        }
    }
```

Beispiel für die Verwendung von Listenoperationen mit AWS SDK for Java 2.x.

```
    public static void exampleUsage(DynamoDbClient dynamoDbClient, String
tableName) {
        // Example key
        Map<String, AttributeValue> key = new HashMap<>();
        key.put("ProductId", AttributeValue.builder().s("P12345").build());

        System.out.println("Demonstrating list operations in DynamoDB");

        try {
            // Example 1: Append items to a list
            System.out.println("\nExample 1: Appending items to a list");
            List<AttributeValue> tagsToAppend = List.of(
                AttributeValue.builder().s("Electronics").build(),
                AttributeValue.builder().s("Gadget").build());

            UpdateItemResponse appendResponse = appendToList(dynamoDbClient,
tableName, key, "Tags", tagsToAppend);
```

```
            System.out.println("Updated list attribute: " +
appendResponse.attributes());

            // Example 2: Prepend items to a list
            System.out.println("\nExample 2: Prepending items to a list");
            List<AttributeValue> tagsToPrepend = List.of(
                AttributeValue.builder().s("Featured").build(),
                AttributeValue.builder().s("New").build());

            UpdateItemResponse prependResponse = prependToList(dynamoDbClient,
tableName, key, "Tags", tagsToPrepend);

            System.out.println("Updated list attribute: " +
prependResponse.attributes());

            // Example 3: Update a specific element in a list
            System.out.println("\nExample 3: Updating a specific element in a
list");
            UpdateItemResponse updateResponse = updateListElement(
                dynamoDbClient,
                tableName,
                key,
                "Tags",
                0,
                AttributeValue.builder().s("BestSeller").build());

            System.out.println("Updated list attribute: " +
updateResponse.attributes());

            // Example 4: Remove a specific element from a list
            System.out.println("\nExample 4: Removing a specific element from a
list");
            UpdateItemResponse removeResponse = removeListElement(dynamoDbClient,
tableName, key, "Tags", 1);

            System.out.println("Updated list attribute: " +
removeResponse.attributes());

            // Example 5: Get the current value of a list attribute
            System.out.println("\nExample 5: Getting the current value of a list
attribute");
            List<AttributeValue> currentList = getListAttribute(dynamoDbClient,
tableName, key, "Tags");
```

```
            if (currentList != null) {
                System.out.println("Current list attribute:");
                for (int i = 0; i < currentList.size(); i++) {
                    System.out.println("  [" + i + "]: " +
currentList.get(i).s());
                }
            } else {
                System.out.println("List attribute not found");
            }

            // Explain list operations
            System.out.println("\nKey points about DynamoDB list operations:");
            System.out.println("1. Lists are ordered collections of attributes");
            System.out.println("2. Use list_append to add items to a list");
            System.out.println("3. To append items, use list_append(existingList,
newItems)");
            System.out.println("4. To prepend items, use list_append(newItems,
existingList)");
            System.out.println("5. Use index notation (list[0]) to access or
update specific elements");
            System.out.println("6. Use REMOVE to delete elements from a list");
            System.out.println("7. List indices are zero-based");
            System.out.println("8. Use if_not_exists to handle the case where the
list doesn't exist yet");

        } catch (DynamoDbException e) {
            System.err.println("Error: " + e.getMessage());
            e.printStackTrace();
        }
    }
```

- Einzelheiten zur API finden Sie UpdateItemin der AWS SDK for Java 2.x API-Referenz.

JavaScript

SDK für JavaScript (v3)

Demonstrieren Sie Listenoperationen mit AWS SDK für JavaScript.

```
const { DynamoDBClient } = require("@aws-sdk/client-dynamodb");
const {
  DynamoDBDocumentClient,
```

```
  UpdateCommand,
  GetCommand,
  PutCommand
} = require("@aws-sdk/lib-dynamodb");

/**
 * Append elements to a list attribute.
 *
 * This function demonstrates how to use the list_append function to add elements
 * to the end of a list.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The key of the item to update
 * @param {string} listName - The name of the list attribute
 * @param {Array} values - The values to append to the list
 * @returns {Promise<Object>} - The response from DynamoDB
 */
async function appendToList(
  config,
  tableName,
  key,
  listName,
  values
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

  // Define the update parameters using list_append
  const params = {
    TableName: tableName,
    Key: key,
    UpdateExpression: `SET ${listName} =
 list_append(if_not_exists(${listName}, :empty_list), :values)`,
    ExpressionAttributeValues: {
      ":empty_list": [],
      ":values": values
    },
    ReturnValues: "UPDATED_NEW"
  };

  // Perform the update operation
  const response = await docClient.send(new UpdateCommand(params));
```

```
  return response;
}

/**
 * Prepend elements to a list attribute.
 *
 * This function demonstrates how to use the list_append function to add elements
 * to the beginning of a list.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The key of the item to update
 * @param {string} listName - The name of the list attribute
 * @param {Array} values - The values to prepend to the list
 * @returns {Promise<Object>} - The response from DynamoDB
 */
async function prependToList(
  config,
  tableName,
  key,
  listName,
  values
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

  // Define the update parameters using list_append
  // Note: To prepend, we put the new values first in the list_append function
  const params = {
    TableName: tableName,
    Key: key,
    UpdateExpression: `SET ${listName} = list_append(:values,
 if_not_exists(${listName}, :empty_list))`,
    ExpressionAttributeValues: {
      ":empty_list": [],
      ":values": values
    },
    ReturnValues: "UPDATED_NEW"
  };

  // Perform the update operation
  const response = await docClient.send(new UpdateCommand(params));
```

```
  return response;
}

/**
 * Update a specific element in a list by index.
 *
 * This function demonstrates how to update a specific element in a list
 * using the index notation.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The key of the item to update
 * @param {string} listName - The name of the list attribute
 * @param {number} index - The index of the element to update
 * @param {any} value - The new value for the element
 * @returns {Promise<Object>} - The response from DynamoDB
 */
async function updateListElement(
  config,
  tableName,
  key,
  listName,
  index,
  value
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

  // Define the update parameters using index notation
  const params = {
    TableName: tableName,
    Key: key,
    UpdateExpression: `SET ${listName}[${index}] = :value`,
    ExpressionAttributeValues: {
      ":value": value
    },
    ReturnValues: "UPDATED_NEW"
  };

  // Perform the update operation
  const response = await docClient.send(new UpdateCommand(params));
```

```
  return response;
}

/**
 * Remove an element from a list by index.
 *
 * This function demonstrates how to remove a specific element from a list
 * using the REMOVE action with index notation.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The key of the item to update
 * @param {string} listName - The name of the list attribute
 * @param {number} index - The index of the element to remove
 * @returns {Promise<Object>} - The response from DynamoDB
 */
async function removeListElement(
  config,
  tableName,
  key,
  listName,
  index
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

  // Define the update parameters using REMOVE with index notation
  const params = {
    TableName: tableName,
    Key: key,
    UpdateExpression: `REMOVE ${listName}[${index}]`,
    ReturnValues: "UPDATED_NEW"
  };

  // Perform the update operation
  const response = await docClient.send(new UpdateCommand(params));

  return response;
}

/**
 * Concatenate two lists.
 *
```

```
 * This function demonstrates how to concatenate two lists using the list_append
 function.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The key of the item to update
 * @param {string} listName1 - The name of the first list attribute
 * @param {string} listName2 - The name of the second list attribute
 * @param {string} resultListName - The name of the attribute to store the
 concatenated list
 * @returns {Promise<Object>} - The response from DynamoDB
 */
async function concatenateLists(
  config,
  tableName,
  key,
  listName1,
  listName2,
  resultListName
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

  // Define the update parameters using list_append
  const params = {
    TableName: tableName,
    Key: key,
    UpdateExpression: `SET ${resultListName} =
 list_append(if_not_exists(${listName1}, :empty_list),
 if_not_exists(${listName2}, :empty_list))`,
    ExpressionAttributeValues: {
      ":empty_list": []
    },
    ReturnValues: "UPDATED_NEW"
  };

  // Perform the update operation
  const response = await docClient.send(new UpdateCommand(params));

  return response;
}

/**
```

```
 * Create a nested list structure.
 *
 * This function demonstrates how to create and work with nested lists.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The key of the item to update
 * @param {string} listName - The name of the list attribute
 * @param {Array} nestedLists - An array of arrays to create a nested list
 structure
 * @returns {Promise<Object>} - The response from DynamoDB
 */
async function createNestedList(
  config,
  tableName,
  key,
  listName,
  nestedLists
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

  // Define the update parameters to create a nested list
  const params = {
    TableName: tableName,
    Key: key,
    UpdateExpression: `SET ${listName} = :nested_lists`,
    ExpressionAttributeValues: {
      ":nested_lists": nestedLists
    },
    ReturnValues: "UPDATED_NEW"
  };

  // Perform the update operation
  const response = await docClient.send(new UpdateCommand(params));

  return response;
}

/**
 * Update an element in a nested list.
 *
 * This function demonstrates how to update an element in a nested list
```

```
 * using multiple index notations.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The key of the item to update
 * @param {string} listName - The name of the list attribute
 * @param {number} outerIndex - The index in the outer list
 * @param {number} innerIndex - The index in the inner list
 * @param {any} value - The new value for the element
 * @returns {Promise<Object>} - The response from DynamoDB
 */
async function updateNestedListElement(
  config,
  tableName,
  key,
  listName,
  outerIndex,
  innerIndex,
  value
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

  // Define the update parameters using multiple index notations
  const params = {
    TableName: tableName,
    Key: key,
    UpdateExpression: `SET ${listName}[${outerIndex}][${innerIndex}] = :value`,
    ExpressionAttributeValues: {
      ":value": value
    },
    ReturnValues: "UPDATED_NEW"
  };

  // Perform the update operation
  const response = await docClient.send(new UpdateCommand(params));

  return response;
}

/**
 * Get the current value of an item.
 *
```

```
 * Helper function to retrieve the current value of an item.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The key of the item to get
 * @returns {Promise<Object|null>} - The item or null if not found
 */
async function getItem(
  config,
  tableName,
  key
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

  // Define the get parameters
  const params = {
    TableName: tableName,
    Key: key
  };

  // Perform the get operation
  const response = await docClient.send(new GetCommand(params));

  // Return the item if it exists, otherwise null
  return response.Item || null;
}
```

Beispiel für die Verwendung von Listenoperationen mit AWS SDK für JavaScript.

```
/**
 * Example of how to work with lists in DynamoDB.
 */
async function exampleUsage() {
  // Example parameters
  const config = { region: "us-west-2" };
  const tableName = "UserProfiles";
  const key = { UserId: "U12345" };

  console.log("Demonstrating list operations in DynamoDB");
```

```
 try {
   // Example 1: Append elements to a list
   console.log("\nExample 1: Appending elements to a list");
   const response1 = await appendToList(
     config,
     tableName,
     key,
     "RecentSearches",
     ["laptop", "headphones", "monitor"]
   );

   console.log("Appended to list:", response1.Attributes);

   // Example 2: Prepend elements to a list
   console.log("\nExample 2: Prepending elements to a list");
   const response2 = await prependToList(
     config,
     tableName,
     key,
     "RecentSearches",
     ["keyboard", "mouse"]
   );

   console.log("Prepended to list:", response2.Attributes);

   // Get the current state of the item
   let currentItem = await getItem(config, tableName, key);
   console.log("\nCurrent state of RecentSearches:",
currentItem?.RecentSearches);

   // Example 3: Update a specific element in a list
   console.log("\nExample 3: Updating a specific element in a list");
   const response3 = await updateListElement(
     config,
     tableName,
     key,
     "RecentSearches",
     0, // Update the first element
     "mechanical keyboard" // New value
   );

   console.log("Updated list element:", response3.Attributes);

   // Example 4: Remove an element from a list
```

```
    console.log("\nExample 4: Removing an element from a list");
    const response4 = await removeListElement(
      config,
      tableName,
      key,
      "RecentSearches",
      2 // Remove the third element
    );

    console.log("List after removing element:", response4.Attributes);

    // Example 5: Create and concatenate lists
    console.log("\nExample 5: Creating and concatenating lists");

    // First, create two separate lists
    await updateWithMultipleActions(
      config,
      tableName,
      key,
      "SET WishList = :wishlist, SavedItems = :saveditems",
      null,
      {
        ":wishlist": ["gaming laptop", "wireless earbuds"],
        ":saveditems": ["smartphone", "tablet"]
      }
    );

    // Then, concatenate them
    const response5 = await concatenateLists(
      config,
      tableName,
      key,
      "WishList",
      "SavedItems",
      "AllItems"
    );

    console.log("Concatenated lists:", response5.Attributes);

    // Example 6: Create a nested list structure
    console.log("\nExample 6: Creating a nested list structure");
    const response6 = await createNestedList(
      config,
      tableName,
```

```
      key,
      "Categories",
      [
        ["Electronics", "Computers", "Accessories"],
        ["Books", "Magazines", "E-books"],
        ["Clothing", "Shoes", "Watches"]
      ]
    );

    console.log("Created nested list:", response6.Attributes);

    // Example 7: Update an element in a nested list
    console.log("\nExample 7: Updating an element in a nested list");
    const response7 = await updateNestedListElement(
      config,
      tableName,
      key,
      "Categories",
      0, // First inner list
      1, // Second element in that list
      "Laptops" // New value
    );

    console.log("Updated nested list element:", response7.Attributes);

    // Get the final state of the item
    currentItem = await getItem(config, tableName, key);
    console.log("\nFinal state of the item:", JSON.stringify(currentItem, null,
2));

    // Explain list operations
    console.log("\nKey points about list operations in DynamoDB:");
    console.log("1. Use list_append to add elements to a list");
    console.log("2. To append elements, use list_append(existingList,
newElements)");
    console.log("3. To prepend elements, use list_append(newElements,
existingList)");
    console.log("4. Use if_not_exists to handle cases where the list might not
exist yet");
    console.log("5. Use index notation (list[0]) to access or update specific
elements");
    console.log("6. Use REMOVE with index notation to remove elements from a
list");
    console.log("7. Lists can contain elements of different types");
```

```
    console.log("8. Lists can be nested (lists of lists)");
    console.log("9. Use multiple index notations (list[0][1]) to access nested
 list elements");

  } catch (error) {
    console.error("Error:", error);
  }
}

/**
 * Helper function for the examples.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The key of the item to update
 * @param {string} updateExpression - The update expression
 * @param {Object} expressionAttributeNames - Expression attribute name
 placeholders
 * @param {Object} expressionAttributeValues - Expression attribute value
 placeholders
 * @returns {Promise<Object>} - The response from DynamoDB
 */
async function updateWithMultipleActions(
  config,
  tableName,
  key,
  updateExpression,
  expressionAttributeNames,
  expressionAttributeValues
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

  // Prepare the update parameters
  const updateParams = {
    TableName: tableName,
    Key: key,
    UpdateExpression: updateExpression,
    ReturnValues: "UPDATED_NEW"
  };

  // Add expression attribute names if provided
  if (expressionAttributeNames) {
```

```
    updateParams.ExpressionAttributeNames = expressionAttributeNames;
  }

  // Add expression attribute values if provided
  if (expressionAttributeValues) {
    updateParams.ExpressionAttributeValues = expressionAttributeValues;
  }

  // Execute the update
  const response = await docClient.send(new UpdateCommand(updateParams));

  return response;
}
```

- Einzelheiten zur API finden Sie UpdateItemin der AWS SDK für JavaScript API-Referenz.

Python

SDK für Python (Boto3)

Demonstrieren Sie Listenoperationen mit AWS SDK für Python (Boto3).

```
import boto3
import json
from typing import Any, Dict, List, Optional, Union


def create_list_attribute(
    table_name: str, key: Dict[str, Any], list_name: str, list_values: List[Any]
) -> Dict[str, Any]:
    """
    Create a new list attribute or replace an existing one.

    This function demonstrates how to create a new list attribute or replace
    an existing list with new values.

    Args:
        table_name (str): The name of the DynamoDB table.
        key (Dict[str, Any]): The primary key of the item to update.
        list_name (str): The name of the list attribute.
        list_values (List[Any]): The values to set in the list.
```

```
    Returns:
        Dict[str, Any]: The response from DynamoDB containing the updated
 attribute values.
    """
    # Initialize the DynamoDB resource
    dynamodb = boto3.resource("dynamodb")
    table = dynamodb.Table(table_name)

    # Use the SET operation to create or replace the list
    response = table.update_item(
        Key=key,
        UpdateExpression=f"SET {list_name} = :list_values",
        ExpressionAttributeValues={":list_values": list_values},
        ReturnValues="UPDATED_NEW",
    )

    return response


def append_to_list(
    table_name: str, key: Dict[str, Any], list_name: str, values_to_append:
 List[Any]
) -> Dict[str, Any]:
    """
    Append values to the end of a list attribute.

    This function demonstrates how to use the list_append function to add
 elements
    to the end of a list attribute.

    Args:
        table_name (str): The name of the DynamoDB table.
        key (Dict[str, Any]): The primary key of the item to update.
        list_name (str): The name of the list attribute.
        values_to_append (List[Any]): The values to append to the list.

    Returns:
        Dict[str, Any]: The response from DynamoDB containing the updated
 attribute values.
    """
    # Initialize the DynamoDB resource
    dynamodb = boto3.resource("dynamodb")
    table = dynamodb.Table(table_name)
```

```
    # Use list_append to add values to the end of the list
    response = table.update_item(
        Key=key,
        UpdateExpression=f"SET {list_name} = list_append({list_name}, :values)",
        ExpressionAttributeValues={":values": values_to_append},
        ReturnValues="UPDATED_NEW",
    )

    return response


def prepend_to_list(
    table_name: str, key: Dict[str, Any], list_name: str, values_to_prepend:
 List[Any]
) -> Dict[str, Any]:
    """
    Prepend values to the beginning of a list attribute.

    This function demonstrates how to use the list_append function to add
 elements
    to the beginning of a list attribute.

    Args:
        table_name (str): The name of the DynamoDB table.
        key (Dict[str, Any]): The primary key of the item to update.
        list_name (str): The name of the list attribute.
        values_to_prepend (List[Any]): The values to prepend to the list.

    Returns:
        Dict[str, Any]: The response from DynamoDB containing the updated
 attribute values.
    """
    # Initialize the DynamoDB resource
    dynamodb = boto3.resource("dynamodb")
    table = dynamodb.Table(table_name)

    # Use list_append with reversed order to add values to the beginning of the
 list
    response = table.update_item(
        Key=key,
        UpdateExpression=f"SET {list_name} = list_append(:values, {list_name})",
        ExpressionAttributeValues={":values": values_to_prepend},
        ReturnValues="UPDATED_NEW",
    )
```

```
    return response


def update_list_element(
    table_name: str, key: Dict[str, Any], list_name: str, index: int, new_value:
 Any
) -> Dict[str, Any]:
    """
    Update a specific element in a list attribute.

    This function demonstrates how to update a specific element in a list
 attribute
    using the index notation.

    Args:
        table_name (str): The name of the DynamoDB table.
        key (Dict[str, Any]): The primary key of the item to update.
        list_name (str): The name of the list attribute.
        index (int): The zero-based index of the element to update.
        new_value (Any): The new value for the element.

    Returns:
        Dict[str, Any]: The response from DynamoDB containing the updated
 attribute values.
    """
    # Initialize the DynamoDB resource
    dynamodb = boto3.resource("dynamodb")
    table = dynamodb.Table(table_name)

    # Use the index notation to update a specific element
    response = table.update_item(
        Key=key,
        UpdateExpression=f"SET {list_name}[{index}] = :value",
        ExpressionAttributeValues={":value": new_value},
        ReturnValues="UPDATED_NEW",
    )

    return response


def remove_list_element(
    table_name: str, key: Dict[str, Any], list_name: str, index: int
) -> Dict[str, Any]:
```

```
    """
    Remove a specific element from a list attribute.

    This function demonstrates how to remove a specific element from a list
 attribute
    using the REMOVE action with index notation.

    Args:
        table_name (str): The name of the DynamoDB table.
        key (Dict[str, Any]): The primary key of the item to update.
        list_name (str): The name of the list attribute.
        index (int): The zero-based index of the element to remove.

    Returns:
        Dict[str, Any]: The response from DynamoDB containing the updated
 attribute values.
    """
    # Initialize the DynamoDB resource
    dynamodb = boto3.resource("dynamodb")
    table = dynamodb.Table(table_name)

    # Use the REMOVE action with index notation to remove a specific element
    response = table.update_item(
        Key=key, UpdateExpression=f"REMOVE {list_name}[{index}]",
 ReturnValues="UPDATED_NEW"
    )

    return response


def update_nested_list_element(
    table_name: str, key: Dict[str, Any], path: str, new_value: Any
) -> Dict[str, Any]:
    """
    Update an element in a nested list structure.

    This function demonstrates how to update an element in a nested list
 structure
    using expression attribute names for the path components.

    Args:
        table_name (str): The name of the DynamoDB table.
        key (Dict[str, Any]): The primary key of the item to update.
        path (str): The path to the nested element (e.g., "parent[0].child[1]").
```

```
        new_value (Any): The new value for the element.

    Returns:
        Dict[str, Any]: The response from DynamoDB containing the updated
attribute values.
    """
    # Define a type for path parts
    path_part = Dict[str, Union[str, int]]
    # Initialize the DynamoDB resource
    dynamodb = boto3.resource("dynamodb")
    table = dynamodb.Table(table_name)

    # Parse the path to extract attribute names and indices
    path_parts: List[path_part] = []
    current_part = ""
    in_bracket = False

    for char in path:
        if char == "[":
            if current_part:
                path_parts.append({"type": "attribute", "value": current_part})
                current_part = ""
            in_bracket = True
        elif char == "]":
            if current_part:
                # Fix for mypy: Use a properly typed dictionary with Union type
                path_parts.append({"type": "index", "value": int(current_part)})
                current_part = ""
            in_bracket = False
        elif char == "." and not in_bracket:
            if current_part:
                path_parts.append({"type": "attribute", "value": current_part})
                current_part = ""
        else:
            current_part += char

    if current_part:
        path_parts.append({"type": "attribute", "value": current_part})

    # Build the update expression and attribute names
    update_expression = "SET "
    expression_attribute_names = {}

    # Build the path expression
```

```
    path_expression = ""
    for i, part in enumerate(path_parts):
        if part["type"] == "attribute":
            name_placeholder = f"#attr{i}"
            expression_attribute_names[name_placeholder] = part["value"]

            if path_expression:
                path_expression += "."
            path_expression += name_placeholder
        elif part["type"] == "index":
            path_expression += f"[{part['value']}]"

    # Complete the update expression
    update_expression += f"{path_expression} = :value"

    # Execute the update
    response = table.update_item(
        Key=key,
        UpdateExpression=update_expression,
        ExpressionAttributeNames=expression_attribute_names,
        ExpressionAttributeValues={":value": new_value},
        ReturnValues="UPDATED_NEW",
    )

    return response


def create_list_if_not_exists(
    table_name: str, key: Dict[str, Any], list_name: str, default_values:
 List[Any]
) -> Dict[str, Any]:
    """
    Create a list attribute if it doesn't exist.

    This function demonstrates how to use if_not_exists to create a list
 attribute
    with default values if it doesn't already exist.

    Args:
        table_name (str): The name of the DynamoDB table.
        key (Dict[str, Any]): The primary key of the item to update.
        list_name (str): The name of the list attribute.
        default_values (List[Any]): The default values for the list.
```

```
    Returns:
        Dict[str, Any]: The response from DynamoDB containing the updated
 attribute values.
    """
    # Initialize the DynamoDB resource
    dynamodb = boto3.resource("dynamodb")
    table = dynamodb.Table(table_name)

    # Use if_not_exists to create the list if it doesn't exist
    response = table.update_item(
        Key=key,
        UpdateExpression=f"SET {list_name} =
 if_not_exists({list_name}, :default)",
        ExpressionAttributeValues={":default": default_values},
        ReturnValues="UPDATED_NEW",
    )

    return response


def append_to_list_safely(
    table_name: str,
    key: Dict[str, Any],
    list_name: str,
    values_to_append: List[Any],
    default_values: Optional[List[Any]] = None,
) -> Dict[str, Any]:
    """
    Append values to a list, creating it if it doesn't exist.

    This function demonstrates how to safely append values to a list attribute,
    creating the list with default values if it doesn't exist.

    Args:
        table_name (str): The name of the DynamoDB table.
        key (Dict[str, Any]): The primary key of the item to update.
        list_name (str): The name of the list attribute.
        values_to_append (List[Any]): The values to append to the list.
        default_values (Optional[List[Any]]): The default values if the list
 doesn't exist.
            If not provided, values_to_append will be used as the default.

    Returns:
```

```
        Dict[str, Any]: The response from DynamoDB containing the updated
 attribute values.
    """
    # Initialize the DynamoDB resource
    dynamodb = boto3.resource("dynamodb")
    table = dynamodb.Table(table_name)

    # If default_values is not provided, use values_to_append
    if default_values is None:
        default_values = values_to_append

    # Use if_not_exists with list_append to safely append to the list
    response = table.update_item(
        Key=key,
        UpdateExpression=f"SET {list_name} =
 list_append(if_not_exists({list_name}, :default), :values)",
        ExpressionAttributeValues={
            ":default": default_values if default_values else [],
            ":values": values_to_append,
        },
        ReturnValues="UPDATED_NEW",
    )

    return response
```

Beispiel für die Verwendung von Listenoperationen mit AWS SDK für Python (Boto3).

```
def example_usage():
    """Example of how to use list operations in DynamoDB."""
    # Example parameters
    table_name = "UserData"
    key = {"UserId": "user123"}

    print("Example 1: Creating a list attribute")
    try:
        response = create_list_attribute(
            table_name=table_name,
            key=key,
            list_name="Interests",
            list_values=["Reading", "Hiking", "Photography"],
```

```
        )
        print(
            f"List attribute created successfully:
{json.dumps(response.get('Attributes', {}), default=str)}"
        )
    except Exception as e:
        print(f"Error creating list attribute: {e}")

    print("\nExample 2: Appending values to a list")
    try:
        response = append_to_list(
            table_name=table_name,
            key=key,
            list_name="Interests",
            values_to_append=["Cooking", "Gardening"],
        )
        print(
            f"Values appended to list successfully:
{json.dumps(response.get('Attributes', {}), default=str)}"
        )
    except Exception as e:
        print(f"Error appending to list: {e}")

    print("\nExample 3: Prepending values to a list")
    try:
        response = prepend_to_list(
            table_name=table_name,
            key=key,
            list_name="Interests",
            values_to_prepend=["Travel", "Music"],
        )
        print(
            f"Values prepended to list successfully:
{json.dumps(response.get('Attributes', {}), default=str)}"
        )
    except Exception as e:
        print(f"Error prepending to list: {e}")

    print("\nExample 4: Updating a specific list element")
    try:
        response = update_list_element(
            table_name=table_name,
            key=key,
            list_name="Interests",
```

```
            index=2,
            new_value="Mountain Hiking",
        )
        print(
            f"List element updated successfully:
{json.dumps(response.get('Attributes', {}), default=str)}"
        )
    except Exception as e:
        print(f"Error updating list element: {e}")

    print("\nExample 5: Removing a list element")
    try:
        response = remove_list_element(
            table_name=table_name, key=key, list_name="Interests", index=0
        )
        print(
            f"List element removed successfully:
{json.dumps(response.get('Attributes', {}), default=str)}"
        )
    except Exception as e:
        print(f"Error removing list element: {e}")

    print("\nExample 6: Working with nested lists")
    try:
        # First, create an item with a nested structure
        dynamodb = boto3.resource("dynamodb")
        table = dynamodb.Table(table_name)

        table.update_item(
            Key={"UserId": "user456"},
            UpdateExpression="SET #skills = :skills",
            ExpressionAttributeNames={"#skills": "Skills"},
            ExpressionAttributeValues={
                ":skills": [
                    {"Category": "Programming", "Languages": ["Python", "Java",
"JavaScript"]},
                    {"Category": "Database", "Systems": ["DynamoDB", "MongoDB",
"PostgreSQL"]},
                ]
            },
        )

        # Now update a nested element
        response = update_nested_list_element(
```

```
            table_name=table_name,
            key={"UserId": "user456"},
            path="Skills[0].Languages[1]",
            new_value="TypeScript",
        )
        print(
            f"Nested list element updated successfully:
{json.dumps(response.get('Attributes', {}), default=str)}"
        )
    except Exception as e:
        print(f"Error working with nested lists: {e}")

    print("\nExample 7: Creating a list if it doesn't exist")
    try:
        response = create_list_if_not_exists(
            table_name=table_name,
            key={"UserId": "user789"},
            list_name="Preferences",
            default_values=["Default1", "Default2", "Default3"],
        )
        print(
            f"List created with default values:
{json.dumps(response.get('Attributes', {}), default=str)}"
        )
    except Exception as e:
        print(f"Error creating list with default values: {e}")

    print("\nExample 8: Safely appending to a list")
    try:
        response = append_to_list_safely(
            table_name=table_name,
            key={"UserId": "user789"},
            list_name="Notifications",
            values_to_append=["New message received"],
            default_values=[],
        )
        print(f"Safely appended to list: {json.dumps(response.get('Attributes',
{}), default=str)}")
    except Exception as e:
        print(f"Error safely appending to list: {e}")

    print("\nKey Points About Working with Lists in DynamoDB:")
    print("1. Lists are ordered collections of elements that can be of different
types")
```

```
    print("2. Use the SET operation with direct assignment to create or replace a
list")
    print("3. Use list_append() to add elements to a list without replacing the
entire list")
    print("4. To append to the end: list_append(list_name, :values)")
    print("5. To prepend to the beginning: list_append(:values, list_name)")
    print("6. Use index notation list_name[index] to access or update specific
elements")
    print("7. Use the REMOVE action with index notation to remove specific
elements")
    print("8. Lists can contain nested structures like maps and other lists")
    print("9. Use if_not_exists() to create a list with default values if it
doesn't exist")
    print("10. List indices are zero-based (the first element is at index 0)")
    print("11. Attempting to access an index beyond the list bounds will result
in an error")
```

- Einzelheiten zur API finden Sie UpdateItemin AWS SDK for Python (Boto3) API Reference.

Eine vollständige Liste der AWS SDK-Entwicklerhandbücher und Codebeispiele finden Sie unter. DynamoDB mit einem SDK verwenden AWS Dieses Thema enthält auch Informationen zu den ersten Schritten und Details zu früheren SDK-Versionen.

Führen Sie Kartenoperationen in DynamoDB mit einem SDK durch AWS

In den folgenden Codebeispielen wird gezeigt, wie Map-Operationen in DynamoDB ausgeführt werden.

- Fügen Sie verschachtelte Attribute in Map-Strukturen hinzu und aktualisieren Sie sie.
- Entfernen bestimmter Felder aus Maps
- Arbeiten mit tief verschachtelten Map-Attributen

Java

SDK für Java 2.x

Demonstrieren Sie Kartenoperationen mit AWS SDK for Java 2.x.

```
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.AttributeValue;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.model.GetItemRequest;
import software.amazon.awssdk.services.dynamodb.model.GetItemResponse;
import software.amazon.awssdk.services.dynamodb.model.UpdateItemRequest;
import software.amazon.awssdk.services.dynamodb.model.UpdateItemResponse;

import java.util.HashMap;
import java.util.List;
import java.util.Map;

    /**
     * Updates a map attribute that may not exist.
     *
     * <p>This method demonstrates how to safely update a map attribute
     * by using if_not_exists to handle the case where the map doesn't exist yet.
     *
     * @param dynamoDbClient The DynamoDB client
     * @param tableName The name of the DynamoDB table
     * @param key The key of the item to update
     * @param mapName The name of the map attribute
     * @param mapKey The key within the map to update
     * @param value The value to set
     * @return The response from DynamoDB
     * @throws DynamoDbException if an error occurs during the operation
     */
    public static UpdateItemResponse updateMapAttributeSafe(
        DynamoDbClient dynamoDbClient,
        String tableName,
        Map<String, AttributeValue> key,
        String mapName,
        String mapKey,
        AttributeValue value) {

        // Create an empty map to use if the map doesn't exist
        Map<String, AttributeValue> emptyMap = new HashMap<>();
        AttributeValue emptyMapValue =
 AttributeValue.builder().m(emptyMap).build();

        // Define the update parameters
        UpdateItemRequest request = UpdateItemRequest.builder()
            .tableName(tableName)
```

```
            .key(key)
            .updateExpression("SET #mapName = if_not_exists(#mapName, :emptyMap),
#mapName.#mapKey = :value")
            .expressionAttributeNames(Map.of(
                "#mapName", mapName,
                "#mapKey", mapKey))
            .expressionAttributeValues(Map.of(
                ":value",
                value,
                ":emptyMap",
                AttributeValue.builder().m(new HashMap<>()).build()))
            .returnValues("UPDATED_NEW")
            .build();

        // Perform the update operation
        return dynamoDbClient.updateItem(request);
    }

    /**
     * Adds an attribute to a nested map.
     *
     * <p>This method demonstrates how to update a nested attribute without
     * overwriting the entire map.
     *
     * @param dynamoDbClient The DynamoDB client
     * @param tableName The name of the DynamoDB table
     * @param key The key of the item to update
     * @param path The path to the nested attribute as a list
     * @param value The value to set
     * @return The response from DynamoDB
     * @throws DynamoDbException if an error occurs during the operation
     */
    public static UpdateItemResponse addToNestedMap(
        DynamoDbClient dynamoDbClient,
        String tableName,
        Map<String, AttributeValue> key,
        List<String> path,
        AttributeValue value) {

        // Create expression attribute names for each part of the path
        Map<String, String> expressionAttributeNames = new HashMap<>();
        for (int i = 0; i < path.size(); i++) {
            expressionAttributeNames.put("#attr" + i, path.get(i));
        }
```

```
        // Build the attribute path using the expression attribute names
        StringBuilder attributePathExpression = new StringBuilder();
        for (int i = 0; i < path.size(); i++) {
            if (i > 0) {
                attributePathExpression.append(".");
            }
            attributePathExpression.append("#attr").append(i);
        }

        // Define the update parameters
        UpdateItemRequest request = UpdateItemRequest.builder()
            .tableName(tableName)
            .key(key)
            .updateExpression("SET " + attributePathExpression.toString() + "
= :value")
            .expressionAttributeNames(expressionAttributeNames)
            .expressionAttributeValues(Map.of(":value", value))
            .returnValues("UPDATED_NEW")
            .build();

        // Perform the update operation
        return dynamoDbClient.updateItem(request);
    }

    /**
     * Removes an attribute from a map.
     *
     * <p>This method demonstrates how to remove a specific attribute from a map.
     *
     * @param dynamoDbClient The DynamoDB client
     * @param tableName The name of the DynamoDB table
     * @param key The key of the item to update
     * @param mapName The name of the map attribute
     * @param mapKey The key within the map to remove
     * @return The response from DynamoDB
     * @throws DynamoDbException if an error occurs during the operation
     */
    public static UpdateItemResponse removeMapAttribute(
        DynamoDbClient dynamoDbClient,
        String tableName,
        Map<String, AttributeValue> key,
        String mapName,
        String mapKey) {
```

```
        // Define the update parameters
        UpdateItemRequest request = UpdateItemRequest.builder()
            .tableName(tableName)
            .key(key)
            .updateExpression("REMOVE #mapName.#mapKey")
            .expressionAttributeNames(Map.of(
                "#mapName", mapName,
                "#mapKey", mapKey))
            .returnValues("UPDATED_NEW")
            .build();

        // Perform the update operation
        return dynamoDbClient.updateItem(request);
    }

    /**
     * Creates a map with multiple attributes in a single operation.
     *
     * <p>This method demonstrates how to create a map with multiple attributes
     * in a single update operation.
     *
     * @param dynamoDbClient The DynamoDB client
     * @param tableName The name of the DynamoDB table
     * @param key The key of the item to update
     * @param mapName The name of the map attribute
     * @param attributes The attributes to set in the map
     * @return The response from DynamoDB
     * @throws DynamoDbException if an error occurs during the operation
     */
    public static UpdateItemResponse createMapWithAttributes(
        DynamoDbClient dynamoDbClient,
        String tableName,
        Map<String, AttributeValue> key,
        String mapName,
        Map<String, AttributeValue> attributes) {

        // Create a map value from the attributes
        AttributeValue mapValue = AttributeValue.builder().m(attributes).build();

        // Define the update parameters
        UpdateItemRequest request = UpdateItemRequest.builder()
            .tableName(tableName)
            .key(key)
```

```
            .updateExpression("SET #mapName = :mapValue")
            .expressionAttributeNames(Map.of("#mapName", mapName))
            .expressionAttributeValues(Map.of(":mapValue", mapValue))
            .returnValues("UPDATED_NEW")
            .build();

        // Perform the update operation
        return dynamoDbClient.updateItem(request);
    }

    /**
     * Gets the current value of a map attribute.
     *
     * <p>Helper method to retrieve the current value of a map attribute.
     *
     * @param dynamoDbClient The DynamoDB client
     * @param tableName The name of the DynamoDB table
     * @param key The key of the item to get
     * @param mapName The name of the map attribute
     * @return The map attribute value or null if not found
     * @throws DynamoDbException if an error occurs during the operation
     */
    public static Map<String, AttributeValue> getMapAttribute(
        DynamoDbClient dynamoDbClient, String tableName, Map<String,
AttributeValue> key, String mapName) {

        // Define the get parameters
        GetItemRequest request = GetItemRequest.builder()
            .tableName(tableName)
            .key(key)
            .projectionExpression(mapName)
            .build();

        try {
            // Perform the get operation
            GetItemResponse response = dynamoDbClient.getItem(request);

            // Return the map attribute if it exists, otherwise null
            if (response.item() != null && response.item().containsKey(mapName))
{
                return response.item().get(mapName).m();
            }

            return null;
```

```
        } catch (DynamoDbException e) {
            throw DynamoDbException.builder()
                .message("Failed to get map attribute: " + e.getMessage())
                .cause(e)
                .build();
        }
    }
```

Beispiel für die Verwendung von Kartenoperationen mit AWS SDK for Java 2.x.

```
   public static void exampleUsage(DynamoDbClient dynamoDbClient, String
tableName) {
        // Example key
        Map<String, AttributeValue> key = new HashMap<>();
        key.put("ProductId", AttributeValue.builder().s("P12345").build());

        System.out.println("Demonstrating map operations in DynamoDB");

        try {
            // Example 1: Create a map with multiple attributes
            System.out.println("\nExample 1: Creating a map with multiple
attributes");
            Map<String, AttributeValue> productDetails = new HashMap<>();
            productDetails.put("Color",
AttributeValue.builder().s("Red").build());
            productDetails.put("Weight",
AttributeValue.builder().n("2.5").build());
            productDetails.put(
                "Dimensions", AttributeValue.builder().s("10x20x5").build());

            UpdateItemResponse createResponse =
                createMapWithAttributes(dynamoDbClient, tableName, key,
"Details", productDetails);

            System.out.println("Created map attribute: " +
createResponse.attributes());

            // Example 2: Update a specific attribute in a map
            System.out.println("\nExample 2: Updating a specific attribute in a
map");
            UpdateItemResponse updateResponse = updateMapAttributeSafe(
                dynamoDbClient,
```

```
                tableName,
                key,
                "Details",
                "Color",
                AttributeValue.builder().s("Blue").build());

            System.out.println("Updated map attribute: " +
updateResponse.attributes());

            // Example 3: Add an attribute to a nested map
            System.out.println("\nExample 3: Adding an attribute to a nested
map");
            UpdateItemResponse nestedResponse = addToNestedMap(
                dynamoDbClient,
                tableName,
                key,
                List.of("Specifications", "Technical", "Resolution"),
                AttributeValue.builder().s("1920x1080").build());

            System.out.println("Added to nested map: " +
nestedResponse.attributes());

            // Example 4: Remove an attribute from a map
            System.out.println("\nExample 4: Removing an attribute from a map");
            UpdateItemResponse removeResponse =
                removeMapAttribute(dynamoDbClient, tableName, key, "Details",
"Dimensions");

            System.out.println("Updated map after removal: " +
removeResponse.attributes());

            // Example 5: Get the current value of a map attribute
            System.out.println("\nExample 5: Getting the current value of a map
attribute");
            Map<String, AttributeValue> currentMap =
getMapAttribute(dynamoDbClient, tableName, key, "Details");

            if (currentMap != null) {
                System.out.println("Current map attribute:");
                for (Map.Entry<String, AttributeValue> entry :
currentMap.entrySet()) {
                    System.out.println("  " + entry.getKey() + ": " +
entry.getValue());
                }
```

```
            } else {
                System.out.println("Map attribute not found");
            }

            // Explain map operations
            System.out.println("\nKey points about DynamoDB map operations:");
            System.out.println("1. Maps are unordered collections of name-value
pairs");
            System.out.println("2. Use dot notation (map.key) to access or update
specific attributes");
            System.out.println("3. You can update individual attributes without
overwriting the entire map");
            System.out.println("4. Maps can be nested to create complex data
structures");
            System.out.println("5. Use REMOVE to delete attributes from a map");
            System.out.println("6. You can create a map with multiple attributes
in a single operation");
            System.out.println("7. Map keys are case-sensitive");

        } catch (DynamoDbException e) {
            System.err.println("Error: " + e.getMessage());
            e.printStackTrace();
        }
    }
```

- Einzelheiten zur API finden Sie UpdateItemin der AWS SDK for Java 2.x API-Referenz.

JavaScript

SDK für JavaScript (v3)

Demonstrieren Sie Kartenoperationen mit AWS SDK für JavaScript.

```
/**
 * Example of updating map attributes in DynamoDB.
 *
 * This module demonstrates how to update map attributes that may not exist,
 * how to update nested attributes, and how to handle various map update
 scenarios.
 */
```

```
const { DynamoDBClient } = require("@aws-sdk/client-dynamodb");
const {
  DynamoDBDocumentClient,
  UpdateCommand,
  GetCommand
} = require("@aws-sdk/lib-dynamodb");

/**
 * Update a map attribute safely, handling the case where the map might not
 exist.
 *
 * This function demonstrates using the if_not_exists function to safely update
 * a map attribute that might not exist yet.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The key of the item to update
 * @param {string} mapName - The name of the map attribute
 * @param {string} mapKey - The key within the map to update
 * @param {any} value - The value to set
 * @returns {Promise<Object>} - The response from DynamoDB
 */
async function updateMapAttributeSafe(
  config,
  tableName,
  key,
  mapName,
  mapKey,
  value
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

  // Define the update parameters using SET with if_not_exists
  const params = {
    TableName: tableName,
    Key: key,
    UpdateExpression: `SET ${mapName}.${mapKey} = :value`,
    ExpressionAttributeValues: {
      ":value": value
    },
    ReturnValues: "UPDATED_NEW"
  };
```

```
  try {
    // Perform the update operation
    const response = await docClient.send(new UpdateCommand(params));
    return response;
  } catch (error) {
    // If the error is because the map doesn't exist, create it
    if (error.name === "ValidationException" &&
        error.message.includes("The document path provided in the update
 expression is invalid")) {

      // Create the map with the specified key-value pair
      const createParams = {
        TableName: tableName,
        Key: key,
        UpdateExpression: `SET ${mapName} = :map`,
        ExpressionAttributeValues: {
          ":map": { [mapKey]: value }
        },
        ReturnValues: "UPDATED_NEW"
      };

      return await docClient.send(new UpdateCommand(createParams));
    }

    // Re-throw other errors
    throw error;
  }
}

/**
 * Update a map attribute using the if_not_exists function.
 *
 * This function demonstrates a more elegant approach using if_not_exists
 * to handle the case where the map doesn't exist yet.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The key of the item to update
 * @param {string} mapName - The name of the map attribute
 * @param {string} mapKey - The key within the map to update
 * @param {any} value - The value to set
 * @returns {Promise<Object>} - The response from DynamoDB
 */
```

```
async function updateMapAttributeWithIfNotExists(
  config,
  tableName,
  key,
  mapName,
  mapKey,
  value
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

  // Define the update parameters using SET with if_not_exists
  const params = {
    TableName: tableName,
    Key: key,
    UpdateExpression: `SET ${mapName} = if_not_exists(${mapName}, :emptyMap),
 ${mapName}.${mapKey} = :value`,
    ExpressionAttributeValues: {
      ":emptyMap": {},
      ":value": value
    },
    ReturnValues: "UPDATED_NEW"
  };

  // Perform the update operation
  const response = await docClient.send(new UpdateCommand(params));

  return response;
}

/**
 * Add a value to a deeply nested map, creating parent maps if they don't exist.
 *
 * This function demonstrates how to update a deeply nested attribute,
 * creating any parent maps that don't exist along the way.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The key of the item to update
 * @param {string[]} path - The path to the nested attribute as an array of keys
 * @param {any} value - The value to set
 * @returns {Promise<Object>} - The response from DynamoDB
 */
```

```
async function addToNestedMap(
  config,
  tableName,
  key,
  path,
  value
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

  // Build the update expression and expression attribute values
  let updateExpression = "SET";
  const expressionAttributeValues = {};

  // For each level in the path, create a map if it doesn't exist
  for (let i = 0; i < path.length; i++) {
    const currentPath = path.slice(0, i + 1).join(".");
    const parentPath = i > 0 ? path.slice(0, i).join(".") : null;

    if (parentPath) {
      updateExpression += ` ${parentPath} =
 if_not_exists(${parentPath}, :emptyMap${i}),`;
      expressionAttributeValues[`:emptyMap${i}`] = {};
    }
  }

  // Set the final value
  const fullPath = path.join(".");
  updateExpression += ` ${fullPath} = :value`;
  expressionAttributeValues[":value"] = value;

  // Define the update parameters
  const params = {
    TableName: tableName,
    Key: key,
    UpdateExpression: updateExpression,
    ExpressionAttributeValues: expressionAttributeValues,
    ReturnValues: "UPDATED_NEW"
  };

  // Perform the update operation
  const response = await docClient.send(new UpdateCommand(params));
```

```
  return response;
}

/**
 * Update multiple fields in a map attribute in a single operation.
 *
 * This function demonstrates how to update multiple fields in a map
 * in a single DynamoDB operation.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The key of the item to update
 * @param {string} mapName - The name of the map attribute
 * @param {Object} updates - Object containing key-value pairs to update
 * @returns {Promise<Object>} - The response from DynamoDB
 */
async function updateMultipleMapFields(
  config,
  tableName,
  key,
  mapName,
  updates
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

  // Build the update expression and expression attribute values
  let updateExpression = `SET ${mapName} = if_not_exists(${mapName}, :emptyMap)`;
  const expressionAttributeValues = {
    ":emptyMap": {}
  };

  // Add each update to the expression
  Object.entries(updates).forEach(([field, value], index) => {
    updateExpression += `, ${mapName}.${field} = :val${index}`;
    expressionAttributeValues[`:val${index}`] = value;
  });

  // Define the update parameters
  const params = {
    TableName: tableName,
    Key: key,
    UpdateExpression: updateExpression,
```

```
    ExpressionAttributeValues: expressionAttributeValues,
    ReturnValues: "UPDATED_NEW"
  };

  // Perform the update operation
  const response = await docClient.send(new UpdateCommand(params));

  return response;
}

/**
 * Get the current value of an item.
 *
 * Helper function to retrieve the current value of an item.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The key of the item to get
 * @returns {Promise<Object|null>} - The item or null if not found
 */
async function getItem(
  config,
  tableName,
  key
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

  // Define the get parameters
  const params = {
    TableName: tableName,
    Key: key
  };

  // Perform the get operation
  const response = await docClient.send(new GetCommand(params));

  // Return the item if it exists, otherwise null
  return response.Item || null;
}

/**
 * Example of how to use the map attribute update functions.
```

```
 */
async function exampleUsage() {
  // Example parameters
  const config = { region: "us-west-2" };
  const tableName = "Users";
  const key = { UserId: "U12345" };

  console.log("Demonstrating different approaches to update map attributes in
 DynamoDB");

  try {
    // Example 1: Update a map attribute that might not exist (two-step approach)
    console.log("\nExample 1: Updating a map attribute that might not exist (two-
step approach)");
    const response1 = await updateMapAttributeSafe(
      config,
      tableName,
      key,
      "Preferences",
      "Theme",
      "Dark"
    );

    console.log("Updated preferences:", response1.Attributes);

    // Example 2: Update a map attribute using if_not_exists (elegant approach)
    console.log("\nExample 2: Updating a map attribute using if_not_exists
 (elegant approach)");
    const response2 = await updateMapAttributeWithIfNotExists(
      config,
      tableName,
      key,
      "Settings",
      "NotificationsEnabled",
      true
    );

    console.log("Updated settings:", response2.Attributes);

    // Example 3: Update a deeply nested attribute
    console.log("\nExample 3: Updating a deeply nested attribute");
    const response3 = await addToNestedMap(
      config,
      tableName,
```

```
      key,
      ["Profile", "Address", "City"],
      "Seattle"
    );

    console.log("Updated nested attribute:", response3.Attributes);

    // Example 4: Update multiple fields in a map
    console.log("\nExample 4: Updating multiple fields in a map");
    const response4 = await updateMultipleMapFields(
      config,
      tableName,
      key,
      "ContactInfo",
      {
        Email: "user@example.com",
        Phone: "555-123-4567",
        PreferredContact: "Email"
      }
    );

    console.log("Updated multiple fields:", response4.Attributes);

    // Get the final state of the item
    console.log("\nFinal state of the item:");
    const item = await getItem(config, tableName, key);
    console.log(JSON.stringify(item, null, 2));

    // Explain the benefits of different approaches
    console.log("\nKey points about updating map attributes:");
    console.log("1. Use if_not_exists to handle maps that might not exist");
    console.log("2. Multiple updates can be combined in a single operation");
    console.log("3. Deeply nested attributes require creating parent maps");
    console.log("4. DynamoDB expressions are atomic - the entire update succeeds
 or fails");
    console.log("5. Using a single operation is more efficient than multiple
 separate updates");

  } catch (error) {
    console.error("Error:", error);
  }
}

// Export the functions
```

```
module.exports = {
  updateMapAttributeSafe,
  updateMapAttributeWithIfNotExists,
  addToNestedMap,
  updateMultipleMapFields,
  getItem,
  exampleUsage
};

// Run the example if this file is executed directly
if (require.main === module) {
  exampleUsage();
}
```

- Einzelheiten zur API finden Sie UpdateItemin der AWS SDK für JavaScript API-Referenz.

Python

SDK für Python (Boto3)

Demonstrieren Sie Kartenoperationen mit AWS SDK für Python (Boto3).

```
"""
Example of updating map attributes in DynamoDB.

This module demonstrates how to update map attributes in DynamoDB, including
handling cases where the map attribute might not exist yet.
"""


import boto3
from typing import Any, Dict, Optional



def update_map_attribute_safe(
    table_name: str, key: Dict[str, Any], map_name: str, map_key: str, value: Any
) -> Dict[str, Any]:
    """
    Update a specific key in a map attribute, creating the map if it doesn't
 exist.
```

```
    This function demonstrates how to safely update a key within a map attribute,
    even if the map doesn't exist yet in the item.

    Args:
        table_name (str): The name of the DynamoDB table.
        key (Dict[str, Any]): The primary key of the item to update.
        map_name (str): The name of the map attribute.
        map_key (str): The key within the map to update.
        value (Any): The value to set for the map key.

    Returns:
        Dict[str, Any]: The response from DynamoDB containing the updated
 attribute values.
    """
    # Initialize the DynamoDB resource
    dynamodb = boto3.resource("dynamodb")
    table = dynamodb.Table(table_name)

    # Use SET with attribute_not_exists to safely update the map
    response = table.update_item(
        Key=key,
        UpdateExpression="SET #map.#key = :value",
        ExpressionAttributeNames={"#map": map_name, "#key": map_key},
        ExpressionAttributeValues={":value": value},
        ReturnValues="UPDATED_NEW",
    )

    return response


def add_to_nested_map(
    table_name: str, key: Dict[str, Any], path: str, value: Any
) -> Dict[str, Any]:
    """
    Add or update a value in a deeply nested map structure.

    This function demonstrates how to update a value at a specific path in a
    nested map structure, creating any intermediate maps as needed.

    Args:
        table_name (str): The name of the DynamoDB table.
        key (Dict[str, Any]): The primary key of the item to update.
```

```
        path (str): The path to the nested attribute (e.g.,
"user.preferences.theme").
        value (Any): The value to set at the specified path.

    Returns:
        Dict[str, Any]: The response from DynamoDB containing the updated
attribute values.
    """
    # Initialize the DynamoDB resource
    dynamodb = boto3.resource("dynamodb")
    table = dynamodb.Table(table_name)

    # Split the path into components
    path_parts = path.split(".")

    # Build the update expression and attribute names
    update_expression = "SET "
    expression_attribute_names = {}

    # Build the path expression
    path_expression = ""
    for i, part in enumerate(path_parts):
        name_placeholder = f"#attr{i}"
        expression_attribute_names[name_placeholder] = part

        if i == 0:
            path_expression = name_placeholder
        else:
            path_expression += f".{name_placeholder}"

    # Complete the update expression
    update_expression += f"{path_expression} = :value"

    # Execute the update
    response = table.update_item(
        Key=key,
        UpdateExpression=update_expression,
        ExpressionAttributeNames=expression_attribute_names,
        ExpressionAttributeValues={":value": value},
        ReturnValues="UPDATED_NEW",
    )

    return response
```

```
def update_map_with_if_not_exists(
    table_name: str,
    key: Dict[str, Any],
    map_name: str,
    map_key: str,
    value: Any,
    default_map: Optional[Dict[str, Any]] = None,
) -> Dict[str, Any]:
    """
    Update a key in a map, creating the map with default values if it doesn't
 exist.

    This function demonstrates how to use if_not_exists to initialize a map with
    default values if it doesn't exist yet, and then update a specific key.

    Args:
        table_name (str): The name of the DynamoDB table.
        key (Dict[str, Any]): The primary key of the item to update.
        map_name (str): The name of the map attribute.
        map_key (str): The key within the map to update.
        value (Any): The value to set for the map key.
        default_map (Optional[Dict[str, Any]]): Default map values if the map
 doesn't exist.

    Returns:
        Dict[str, Any]: The response from DynamoDB containing the updated
 attribute values.
    """
    # Initialize the DynamoDB resource
    dynamodb = boto3.resource("dynamodb")
    table = dynamodb.Table(table_name)

    # Set default map if not provided
    if default_map is None:
        default_map = {}

    # Create a map with the new key-value pair
    updated_map = default_map.copy()
    updated_map[map_key] = value

    # Use if_not_exists to initialize the map if it doesn't exist
    response = table.update_item(
        Key=key,
```

```
        UpdateExpression="SET #map = if_not_exists(#map, :default_map)",
        ExpressionAttributeNames={"#map": map_name},
        ExpressionAttributeValues={":default_map": updated_map},
        ReturnValues="UPDATED_NEW",
    )

    return response


def merge_into_map(
    table_name: str, key: Dict[str, Any], map_name: str, values_to_merge:
 Dict[str, Any]
) -> Dict[str, Any]:
    """
    Merge multiple key-value pairs into a map attribute.

    This function demonstrates how to update multiple keys in a map attribute
    in a single operation, without overwriting the entire map.

    Args:
        table_name (str): The name of the DynamoDB table.
        key (Dict[str, Any]): The primary key of the item to update.
        map_name (str): The name of the map attribute.
        values_to_merge (Dict[str, Any]): Key-value pairs to merge into the map.

    Returns:
        Dict[str, Any]: The response from DynamoDB containing the updated
 attribute values.
    """
    # Initialize the DynamoDB resource
    dynamodb = boto3.resource("dynamodb")
    table = dynamodb.Table(table_name)

    # Build the update expression for each key-value pair
    update_expression = "SET "
    expression_attribute_names = {"#map": map_name}
    expression_attribute_values = {}

    # Add each key-value pair to the update expression
    for i, (k, v) in enumerate(values_to_merge.items()):
        key_placeholder = f"#key{i}"
        value_placeholder = f":value{i}"

        expression_attribute_names[key_placeholder] = k
```

```
        expression_attribute_values[value_placeholder] = v

        if i > 0:
            update_expression += ", "
        update_expression += f"#map.{key_placeholder} = {value_placeholder}"

    # Execute the update
    response = table.update_item(
        Key=key,
        UpdateExpression=update_expression,
        ExpressionAttributeNames=expression_attribute_names,
        ExpressionAttributeValues=expression_attribute_values,
        ReturnValues="UPDATED_NEW",
    )

    return response




def example_usage():
    """Example of how to use the map attribute update functions."""
    # Example parameters
    table_name = "UserProfiles"
    key = {"UserId": "user123"}

    print("Example 1: Updating a specific key in a map attribute")
    try:
        response = update_map_attribute_safe(
            table_name=table_name, key=key, map_name="Preferences",
 map_key="Theme", value="Dark"
        )
        print(f"Map attribute updated successfully: {response.get('Attributes',
 {})}")
    except Exception as e:
        print(f"Error updating map attribute: {e}")

    print("\nExample 2: Adding a value to a deeply nested map")
    try:
        response = add_to_nested_map(
            table_name=table_name, key=key, path="Settings.Notifications.Email",
 value=True
        )
```

```
        print(f"Nested map updated successfully: {response.get('Attributes',
{})}")
    except Exception as e:
        print(f"Error updating nested map: {e}")

    print("\nExample 3: Initializing a map with default values if it doesn't
exist")
    try:
        default_map = {"Language": "English", "Currency": "USD", "TimeZone":
"UTC"}

        response = update_map_with_if_not_exists(
            table_name=table_name,
            key={"UserId": "newuser456"},
            map_name="Preferences",
            map_key="Theme",
            value="Light",
            default_map=default_map,
        )
        print(f"Map initialized with defaults: {response.get('Attributes', {})}")
    except Exception as e:
        print(f"Error initializing map: {e}")

    print("\nExample 4: Merging multiple values into a map")
    try:
        values_to_merge = {
            "NotificationsEnabled": True,
            "EmailFrequency": "Daily",
            "PushNotifications": False,
        }

        response = merge_into_map(
            table_name=table_name,
            key=key,
            map_name="NotificationSettings",
            values_to_merge=values_to_merge,
        )
        print(f"Multiple values merged into map: {response.get('Attributes',
{})}")
    except Exception as e:
        print(f"Error merging values into map: {e}")

    print("\nBest practices for working with map attributes in DynamoDB:")
    print("1. Use dot notation to access and update nested attributes")
```

```
    print("2. Use ExpressionAttributeNames to handle reserved words and special
 characters")
    print("3. Use if_not_exists() to handle cases where attributes might not
 exist")
    print("4. Update specific map keys rather than overwriting the entire map")
    print("5. Use a single update operation to modify multiple map keys for
 better performance")
    print("6. Consider your data model carefully to minimize the need for deeply
 nested attributes")



if __name__ == "__main__":
    example_usage()
```

- Einzelheiten zur API finden Sie UpdateItemin AWS SDK for Python (Boto3) API Reference.

Eine vollständige Liste der AWS SDK-Entwicklerhandbücher und Codebeispiele finden Sie unter. DynamoDB mit einem SDK verwenden AWS Dieses Thema enthält auch Informationen zu den ersten Schritten und Details zu früheren SDK-Versionen.

Führen Sie Set-Operationen in DynamoDB mit einem SDK durch AWS

In den folgenden Codebeispielen wird gezeigt, wie Set-Operationen in DynamoDB ausgeführt werden.

- Hinzufügen von Elementen zu einem Set-Attribut
- Entfernen von Elementen aus einem Set-Attribut
- Verwenden der Operationen ADD und DELETE mit Sätzen

Java

SDK für Java 2.x

Demonstrieren Sie festgelegte Operationen mit AWS SDK for Java 2.x.

```
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.AttributeValue;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
```

```
import software.amazon.awssdk.services.dynamodb.model.GetItemRequest;
import software.amazon.awssdk.services.dynamodb.model.GetItemResponse;
import software.amazon.awssdk.services.dynamodb.model.ReturnValue;
import software.amazon.awssdk.services.dynamodb.model.UpdateItemRequest;
import software.amazon.awssdk.services.dynamodb.model.UpdateItemResponse;

import java.util.HashMap;
import java.util.HashSet;
import java.util.Map;
import java.util.Set;

    /**
     * Adds values to a string set attribute.
     *
     * <p>This method demonstrates how to use the ADD operation to add values
     * to a string set attribute.
     *
     * @param dynamoDbClient The DynamoDB client
     * @param tableName The name of the DynamoDB table
     * @param key The key of the item to update
     * @param setAttributeName The name of the set attribute
     * @param valuesToAdd The values to add to the set
     * @return The response from DynamoDB
     * @throws DynamoDbException if an error occurs during the operation
     */
    public static UpdateItemResponse addToStringSet(
        DynamoDbClient dynamoDbClient,
        String tableName,
        Map<String, AttributeValue> key,
        String setAttributeName,
        Set<String> valuesToAdd) {

        // Create a string set value from the values to add
        AttributeValue setValue =
 AttributeValue.builder().ss(valuesToAdd).build();

        // Define the update parameters
        UpdateItemRequest request = UpdateItemRequest.builder()
            .tableName(tableName)
            .key(key)
            .updateExpression("ADD #setAttr :valuesToAdd")
            .expressionAttributeNames(Map.of("#setAttr", setAttributeName))
            .expressionAttributeValues(Map.of(":valuesToAdd", setValue))
            .returnValues(ReturnValue.UPDATED_NEW)
```

```
            .build();

        // Perform the update operation
        return dynamoDbClient.updateItem(request);
    }

    /**
     * Adds values to a number set attribute.
     *
     * <p>This method demonstrates how to use the ADD operation to add values
     * to a number set attribute.
     *
     * @param dynamoDbClient The DynamoDB client
     * @param tableName The name of the DynamoDB table
     * @param key The key of the item to update
     * @param setAttributeName The name of the set attribute
     * @param valuesToAdd The values to add to the set
     * @return The response from DynamoDB
     * @throws DynamoDbException if an error occurs during the operation
     */
    public static UpdateItemResponse addToNumberSet(
        DynamoDbClient dynamoDbClient,
        String tableName,
        Map<String, AttributeValue> key,
        String setAttributeName,
        Set<Number> valuesToAdd) {

        // Convert numbers to strings for DynamoDB
        Set<String> stringValues = new HashSet<>();
        for (Number value : valuesToAdd) {
            stringValues.add(value.toString());
        }

        // Create a number set value from the values to add
        AttributeValue setValue =
AttributeValue.builder().ns(stringValues).build();

        // Define the update parameters
        UpdateItemRequest request = UpdateItemRequest.builder()
            .tableName(tableName)
            .key(key)
            .updateExpression("ADD #setAttr :valuesToAdd")
            .expressionAttributeNames(Map.of("#setAttr", setAttributeName))
            .expressionAttributeValues(Map.of(":valuesToAdd", setValue))
```

```
            .returnValues(ReturnValue.UPDATED_NEW)
            .build();

        // Perform the update operation
        return dynamoDbClient.updateItem(request);
    }

    /**
     * Removes values from a set attribute.
     *
     * <p>This method demonstrates how to use the DELETE operation to remove
values
     * from a set attribute.
     *
     * @param dynamoDbClient The DynamoDB client
     * @param tableName The name of the DynamoDB table
     * @param key The key of the item to update
     * @param setAttributeName The name of the set attribute
     * @param valuesToRemove The values to remove from the set
     * @param isNumberSet Whether the set is a number set (true) or string set
(false)
     * @return The response from DynamoDB
     * @throws DynamoDbException if an error occurs during the operation
     */
    public static UpdateItemResponse removeFromSet(
        DynamoDbClient dynamoDbClient,
        String tableName,
        Map<String, AttributeValue> key,
        String setAttributeName,
        Set<?> valuesToRemove,
        boolean isNumberSet) {

        AttributeValue setValue;

        if (isNumberSet) {
            // Convert numbers to strings for DynamoDB
            Set<String> stringValues = new HashSet<>();
            for (Object value : valuesToRemove) {
                if (value instanceof Number) {
                    stringValues.add(value.toString());
                } else {
                    throw new IllegalArgumentException("Values must be numbers
for a number set");
                }
```

```
            }

            setValue = AttributeValue.builder().ns(stringValues).build();
        } else {
            // Convert objects to strings for DynamoDB
            Set<String> stringValues = new HashSet<>();
            for (Object value : valuesToRemove) {
                stringValues.add(value.toString());
            }

            setValue = AttributeValue.builder().ss(stringValues).build();
        }

        // Define the update parameters
        UpdateItemRequest request = UpdateItemRequest.builder()
            .tableName(tableName)
            .key(key)
            .updateExpression("DELETE #setAttr :valuesToRemove")
            .expressionAttributeNames(Map.of("#setAttr", setAttributeName))
            .expressionAttributeValues(Map.of(":valuesToRemove", setValue))
            .returnValues(ReturnValue.UPDATED_NEW)
            .build();

        // Perform the update operation
        return dynamoDbClient.updateItem(request);
    }

    /**
     * Checks if a value exists in a set attribute.
     *
     * <p>This method demonstrates how to use the contains function to check
     * if a value exists in a set attribute.
     *
     * @param dynamoDbClient The DynamoDB client
     * @param tableName The name of the DynamoDB table
     * @param key The key of the item to check
     * @param setAttributeName The name of the set attribute
     * @param valueToCheck The value to check for
     * @return Map containing the result of the check
     * @throws DynamoDbException if an error occurs during the operation
     */
    public static Map<String, Object> checkIfValueInSet(
        DynamoDbClient dynamoDbClient,
        String tableName,
```

```
        Map<String, AttributeValue> key,
        String setAttributeName,
        String valueToCheck) {

        Map<String, Object> result = new HashMap<>();

        try {
            // Define the update parameters with a condition expression
            UpdateItemRequest request = UpdateItemRequest.builder()
                .tableName(tableName)
                .key(key)
                .updateExpression("SET #tempAttr = :tempVal")
                .conditionExpression("contains(#setAttr, :valueToCheck)")
                .expressionAttributeNames(Map.of("#setAttr", setAttributeName,
"#tempAttr", "TempAttribute"))
                .expressionAttributeValues(Map.of(
                    ":valueToCheck",
AttributeValue.builder().s(valueToCheck).build(),
                    ":tempVal", AttributeValue.builder().s("TempValue").build()))
                .returnValues(ReturnValue.UPDATED_NEW)
                .build();

            // Attempt the update operation
            dynamoDbClient.updateItem(request);

            // If we get here, the condition was met
            result.put("exists", true);
            result.put("message", "Value '" + valueToCheck + "' exists in the
set");

            // Clean up the temporary attribute
            UpdateItemRequest cleanupRequest = UpdateItemRequest.builder()
                .tableName(tableName)
                .key(key)
                .updateExpression("REMOVE #tempAttr")
                .expressionAttributeNames(Map.of("#tempAttr", "TempAttribute"))
                .build();

            dynamoDbClient.updateItem(cleanupRequest);

        } catch (DynamoDbException e) {
            if (e.getMessage().contains("ConditionalCheckFailed")) {
                // The condition was not met
                result.put("exists", false);
```

```
                result.put("message", "Value '" + valueToCheck + "' does not
exist in the set");
            } else {
                // Some other error occurred
                result.put("exists", false);
                result.put("message", "Error checking set: " + e.getMessage());
                result.put("error", e.getClass().getSimpleName());
            }
        }

        return result;
    }

    /**
     * Creates a set with multiple values in a single operation.
     *
     * <p>This method demonstrates how to create a set with multiple values
     * in a single update operation.
     *
     * @param dynamoDbClient The DynamoDB client
     * @param tableName The name of the DynamoDB table
     * @param key The key of the item to update
     * @param setAttributeName The name of the set attribute
     * @param setValues The values to include in the set
     * @param isNumberSet Whether to create a number set (true) or string set
(false)
     * @return The response from DynamoDB
     * @throws DynamoDbException if an error occurs during the operation
     */
    public static UpdateItemResponse createSetWithValues(
        DynamoDbClient dynamoDbClient,
        String tableName,
        Map<String, AttributeValue> key,
        String setAttributeName,
        Set<?> setValues,
        boolean isNumberSet) {

        AttributeValue setValue;

        if (isNumberSet) {
            // Convert numbers to strings for DynamoDB
            Set<String> stringValues = new HashSet<>();
            for (Object value : setValues) {
                if (value instanceof Number) {
```

```
                    stringValues.add(value.toString());
                } else {
                    throw new IllegalArgumentException("Values must be numbers
for a number set");
                }
            }

            setValue = AttributeValue.builder().ns(stringValues).build();
        } else {
            // Convert objects to strings for DynamoDB
            Set<String> stringValues = new HashSet<>();
            for (Object value : setValues) {
                stringValues.add(value.toString());
            }

            setValue = AttributeValue.builder().ss(stringValues).build();
        }

        // Define the update parameters
        UpdateItemRequest request = UpdateItemRequest.builder()
            .tableName(tableName)
            .key(key)
            .updateExpression("SET #setAttr = :setValue")
            .expressionAttributeNames(Map.of("#setAttr", setAttributeName))
            .expressionAttributeValues(Map.of(":setValue", setValue))
            .returnValues(ReturnValue.UPDATED_NEW)
            .build();

        // Perform the update operation
        return dynamoDbClient.updateItem(request);
    }

    /**
     * Gets the current value of a set attribute.
     *
     * <p>Helper method to retrieve the current value of a set attribute.
     *
     * @param dynamoDbClient The DynamoDB client
     * @param tableName The name of the DynamoDB table
     * @param key The key of the item to get
     * @param setAttributeName The name of the set attribute
     * @return The set attribute value or null if not found
     * @throws DynamoDbException if an error occurs during the operation
     */
```

```
   public static AttributeValue getSetAttribute(
       DynamoDbClient dynamoDbClient, String tableName, Map<String,
AttributeValue> key, String setAttributeName) {

       // Define the get parameters
       GetItemRequest request = GetItemRequest.builder()
           .tableName(tableName)
           .key(key)
           .projectionExpression(setAttributeName)
           .build();

       try {
           // Perform the get operation
           GetItemResponse response = dynamoDbClient.getItem(request);

           // Return the set attribute if it exists, otherwise null
           if (response.item() != null &&
response.item().containsKey(setAttributeName)) {
               return response.item().get(setAttributeName);
           }

           return null;
       } catch (DynamoDbException e) {
           throw DynamoDbException.builder()
               .message("Failed to get set attribute: " + e.getMessage())
               .cause(e)
               .build();
       }
   }
```

Beispiel für die Verwendung von Mengenoperationen mit AWS SDK for Java 2.x.

```
   public static void exampleUsage(DynamoDbClient dynamoDbClient, String
tableName) {
       // Example key
       Map<String, AttributeValue> key = new HashMap<>();
       key.put("ProductId", AttributeValue.builder().s("P12345").build());

       System.out.println("Demonstrating set operations in DynamoDB");

       try {
           // Example 1: Create a string set with multiple values
```

```
            System.out.println("\nExample 1: Creating a string set with multiple
values");
            Set<String> tags = new HashSet<>();
            tags.add("Electronics");
            tags.add("Gadget");
            tags.add("Smartphone");

            UpdateItemResponse createResponse = createSetWithValues(
                dynamoDbClient, tableName, key, "Tags", tags, false // Not a
number set
                );

            System.out.println("Created set attribute: " +
createResponse.attributes());

            // Example 2: Add values to a string set
            System.out.println("\nExample 2: Adding values to a string set");
            Set<String> additionalTags = new HashSet<>();
            additionalTags.add("Mobile");
            additionalTags.add("Wireless");

            UpdateItemResponse addResponse = addToStringSet(dynamoDbClient,
tableName, key, "Tags", additionalTags);

            System.out.println("Updated set attribute: " +
addResponse.attributes());

            // Example 3: Create a number set with multiple values
            System.out.println("\nExample 3: Creating a number set with multiple
values");
            Set<Number> ratings = new HashSet<>();
            ratings.add(4);
            ratings.add(5);
            ratings.add(4.5);

            UpdateItemResponse createNumberSetResponse = createSetWithValues(
                dynamoDbClient, tableName, key, "Ratings", ratings, true // Is a
number set
                );

            System.out.println("Created number set attribute: " +
createNumberSetResponse.attributes());

            // Example 4: Add values to a number set
```

```
            System.out.println("\nExample 4: Adding values to a number set");
            Set<Number> additionalRatings = new HashSet<>();
            additionalRatings.add(3.5);
            additionalRatings.add(4.2);

            UpdateItemResponse addNumberResponse =
                addToNumberSet(dynamoDbClient, tableName, key, "Ratings",
additionalRatings);

            System.out.println("Updated number set attribute: " +
addNumberResponse.attributes());

            // Example 5: Remove values from a set
            System.out.println("\nExample 5: Removing values from a set");
            Set<String> tagsToRemove = new HashSet<>();
            tagsToRemove.add("Gadget");

            UpdateItemResponse removeResponse = removeFromSet(
                dynamoDbClient, tableName, key, "Tags", tagsToRemove, false //
Not a number set
                );

            System.out.println("Updated set after removal: " +
removeResponse.attributes());

            // Example 6: Check if a value exists in a set
            System.out.println("\nExample 6: Checking if a value exists in a
set");
            Map<String, Object> checkResult = checkIfValueInSet(dynamoDbClient,
tableName, key, "Tags", "Electronics");

            System.out.println("Check result: " + checkResult.get("message"));

            // Example 7: Get the current value of a set attribute
            System.out.println("\nExample 7: Getting the current value of a set
attribute");
            AttributeValue currentStringSet = getSetAttribute(dynamoDbClient,
tableName, key, "Tags");

            if (currentStringSet != null && currentStringSet.ss() != null) {
                System.out.println("Current string set values: " +
currentStringSet.ss());
            } else {
                System.out.println("String set attribute not found");
```

```
            }

            AttributeValue currentNumberSet = getSetAttribute(dynamoDbClient,
tableName, key, "Ratings");

            if (currentNumberSet != null && currentNumberSet.ns() != null) {
                System.out.println("Current number set values: " +
currentNumberSet.ns());
            } else {
                System.out.println("Number set attribute not found");
            }

            // Explain set operations
            System.out.println("\nKey points about DynamoDB set operations:");
            System.out.println(
                "1. DynamoDB supports three set types: string sets (SS), number
sets (NS), and binary sets (BS)");
            System.out.println("2. Sets can only contain elements of the same
type");
            System.out.println("3. Use ADD to add elements to a set");
            System.out.println("4. Use DELETE to remove elements from a set");
            System.out.println("5. Sets automatically remove duplicate values");
            System.out.println("6. Sets are unordered collections");
            System.out.println("7. Use the contains function to check if a value
exists in a set");
            System.out.println("8. You can create a set with multiple values in a
single operation");

        } catch (DynamoDbException e) {
            System.err.println("Error: " + e.getMessage());
            e.printStackTrace();
        }
    }
```

- Einzelheiten zur API finden Sie UpdateItemin der AWS SDK for Java 2.x API-Referenz.

JavaScript

SDK für JavaScript (v3)

Demonstrieren Sie die Bedienung von Sets mithilfe von AWS SDK für JavaScript.

```
const { DynamoDBClient } = require("@aws-sdk/client-dynamodb");
const {
  DynamoDBDocumentClient,
  UpdateCommand,
  GetCommand
} = require("@aws-sdk/lib-dynamodb");

/**
 * Add elements to a set attribute.
 *
 * This function demonstrates using the ADD operation to add elements to a set.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The key of the item to update
 * @param {string} setName - The name of the set attribute
 * @param {Array} values - The values to add to the set
 * @param {string} setType - The type of set ('string', 'number', or 'binary')
 * @returns {Promise<Object>} - The response from DynamoDB
 */
async function addToSet(
  config,
  tableName,
  key,
  setName,
  values,
  setType = 'string'
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

  // Create the appropriate set type
  let setValues;
  if (setType === 'string') {
    setValues = new Set(values.map(String));
  } else if (setType === 'number') {
    setValues = new Set(values.map(Number));
  } else if (setType === 'binary') {
    setValues = new Set(values);
  } else {
    throw new Error(`Unsupported set type: ${setType}`);
  }
```

```
  // Define the update parameters using ADD
  const params = {
    TableName: tableName,
    Key: key,
    UpdateExpression: `ADD ${setName} :values`,
    ExpressionAttributeValues: {
      ":values": setValues
    },
    ReturnValues: "UPDATED_NEW"
  };

  // Perform the update operation
  const response = await docClient.send(new UpdateCommand(params));

  return response;
}

/**
 * Remove elements from a set attribute.
 *
 * This function demonstrates using the DELETE operation to remove elements from
 a set.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The key of the item to update
 * @param {string} setName - The name of the set attribute
 * @param {Array} values - The values to remove from the set
 * @param {string} setType - The type of set ('string', 'number', or 'binary')
 * @returns {Promise<Object>} - The response from DynamoDB
 */
async function removeFromSet(
  config,
  tableName,
  key,
  setName,
  values,
  setType = 'string'
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);
```

```
  // Create the appropriate set type
  let setValues;
  if (setType === 'string') {
    setValues = new Set(values.map(String));
  } else if (setType === 'number') {
    setValues = new Set(values.map(Number));
  } else if (setType === 'binary') {
    setValues = new Set(values);
  } else {
    throw new Error(`Unsupported set type: ${setType}`);
  }

  // Define the update parameters using DELETE
  const params = {
    TableName: tableName,
    Key: key,
    UpdateExpression: `DELETE ${setName} :values`,
    ExpressionAttributeValues: {
      ":values": setValues
    },
    ReturnValues: "UPDATED_NEW"
  };

  // Perform the update operation
  const response = await docClient.send(new UpdateCommand(params));

  return response;
}

/**
 * Create a new set attribute with initial values.
 *
 * This function demonstrates using the SET operation to create a new set
 attribute.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The key of the item to update
 * @param {string} setName - The name of the set attribute
 * @param {Array} values - The initial values for the set
 * @param {string} setType - The type of set ('string', 'number', or 'binary')
 * @returns {Promise<Object>} - The response from DynamoDB
 */
async function createSet(
```

```
  config,
  tableName,
  key,
  setName,
  values,
  setType = 'string'
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

  // Create the appropriate set type
  let setValues;
  if (setType === 'string') {
    setValues = new Set(values.map(String));
  } else if (setType === 'number') {
    setValues = new Set(values.map(Number));
  } else if (setType === 'binary') {
    setValues = new Set(values);
  } else {
    throw new Error(`Unsupported set type: ${setType}`);
  }

  // Define the update parameters using SET
  const params = {
    TableName: tableName,
    Key: key,
    UpdateExpression: `SET ${setName} = :values`,
    ExpressionAttributeValues: {
      ":values": setValues
    },
    ReturnValues: "UPDATED_NEW"
  };

  // Perform the update operation
  const response = await docClient.send(new UpdateCommand(params));

  return response;
}

/**
 * Replace an entire set attribute with a new set of values.
 *
 * This function demonstrates using the SET operation to replace an entire set.
```

```
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The key of the item to update
 * @param {string} setName - The name of the set attribute
 * @param {Array} values - The new values for the set
 * @param {string} setType - The type of set ('string', 'number', or 'binary')
 * @returns {Promise<Object>} - The response from DynamoDB
 */
async function replaceSet(
  config,
  tableName,
  key,
  setName,
  values,
  setType = 'string'
) {
  // This is the same as createSet, but included for clarity of intent
  return await createSet(config, tableName, key, setName, values, setType);
}

/**
 * Remove the last element from a set and handle the empty set case.
 *
 * This function demonstrates what happens when you delete the last element of a
 set.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The key of the item to update
 * @param {string} setName - The name of the set attribute
 * @returns {Promise<Object>} - The result of the operation
 */
async function removeLastElementFromSet(
  config,
  tableName,
  key,
  setName
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

  // First, get the current item to check the set
```

```
  const currentItem = await getItem(config, tableName, key);

  // Check if the set exists and has elements
  if (!currentItem || !currentItem[setName] || currentItem[setName].size === 0) {
    return {
      success: false,
      message: "Set doesn't exist or is already empty",
      item: currentItem
    };
  }

  // Get the set values
  const setValues = Array.from(currentItem[setName]);

  // If there's only one element left, remove the attribute entirely
  if (setValues.length === 1) {
    // Define the update parameters to remove the attribute
    const params = {
      TableName: tableName,
      Key: key,
      UpdateExpression: `REMOVE ${setName}`,
      ReturnValues: "UPDATED_NEW"
    };

    // Perform the update operation
    await docClient.send(new UpdateCommand(params));

    return {
      success: true,
      message: "Last element removed, attribute has been deleted",
      removedValue: setValues[0]
    };
  } else {
    // Otherwise, remove just the last element
    // Create a set with just the last element
    const lastElement = setValues[setValues.length - 1];
    const setType = typeof lastElement === 'number' ? 'number' : 'string';

    // Remove the last element
    const response = await removeFromSet(
      config,
      tableName,
      key,
      setName,
```

```
      [lastElement],
      setType
    );

    return {
      success: true,
      message: "Last element removed, set still contains elements",
      removedValue: lastElement,
      remainingSet: response.Attributes[setName]
    };
  }
}

/**
 * Get the current value of an item.
 *
 * Helper function to retrieve the current value of an item.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The key of the item to get
 * @returns {Promise<Object|null>} - The item or null if not found
 */
async function getItem(
  config,
  tableName,
  key
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

  // Define the get parameters
  const params = {
    TableName: tableName,
    Key: key
  };

  // Perform the get operation
  const response = await docClient.send(new GetCommand(params));

  // Return the item if it exists, otherwise null
  return response.Item || null;
}
```

Beispiel für die Verwendung von Mengenoperationen mit AWS SDK für JavaScript.

```
/**
 * Example of how to work with sets in DynamoDB.
 */
async function exampleUsage() {
  // Example parameters
  const config = { region: "us-west-2" };
  const tableName = "Users";
  const key = { UserId: "U12345" };

  console.log("Demonstrating set operations in DynamoDB");

  try {
    // Example 1: Create a string set
    console.log("\nExample 1: Creating a string set");
    const response1 = await createSet(
      config,
      tableName,
      key,
      "Interests",
      ["Reading", "Hiking", "Cooking"],
      "string"
    );

    console.log("Created set:", response1.Attributes);

    // Example 2: Add elements to a set
    console.log("\nExample 2: Adding elements to a set");
    const response2 = await addToSet(
      config,
      tableName,
      key,
      "Interests",
      ["Photography", "Travel"],
      "string"
    );

    console.log("Updated set after adding elements:", response2.Attributes);

    // Example 3: Remove elements from a set
```

```
    console.log("\nExample 3: Removing elements from a set");
    const response3 = await removeFromSet(
      config,
      tableName,
      key,
      "Interests",
      ["Cooking"],
      "string"
    );

    console.log("Updated set after removing elements:", response3.Attributes);

    // Example 4: Create a number set
    console.log("\nExample 4: Creating a number set");
    const response4 = await createSet(
      config,
      tableName,
      key,
      "FavoriteNumbers",
      [7, 42, 99],
      "number"
    );

    console.log("Created number set:", response4.Attributes);

    // Example 5: Replace an entire set
    console.log("\nExample 5: Replacing an entire set");
    const response5 = await replaceSet(
      config,
      tableName,
      key,
      "Interests",
      ["Gaming", "Movies", "Music"],
      "string"
    );

    console.log("Replaced set:", response5.Attributes);

    // Example 6: Remove the last element from a set
    console.log("\nExample 6: Removing the last element from a set");

    // First, create a set with just one element
    await createSet(
      config,
```

```
      tableName,
      { UserId: "U67890" },
      "Tags",
      ["LastTag"],
      "string"
    );

    // Then, remove the last element
    const response6 = await removeLastElementFromSet(
      config,
      tableName,
      { UserId: "U67890" },
      "Tags"
    );

    console.log(response6.message);
    console.log("Removed value:", response6.removedValue);

    // Get the final state of the items
    console.log("\nFinal state of the items:");
    const item1 = await getItem(config, tableName, key);
    console.log("User U12345:", JSON.stringify(item1, null, 2));

    const item2 = await getItem(config, tableName, { UserId: "U67890" });
    console.log("User U67890:", JSON.stringify(item2, null, 2));

    // Explain set operations
    console.log("\nKey points about set operations in DynamoDB:");
    console.log("1. Use ADD to add elements to a set (duplicates are
 automatically removed)");
    console.log("2. Use DELETE to remove elements from a set");
    console.log("3. Use SET to create a new set or replace an existing one");
    console.log("4. DynamoDB supports three types of sets: string sets, number
 sets, and binary sets");
    console.log("5. When you delete the last element from a set, the attribute
 remains as an empty set");
    console.log("6. To remove an empty set, use the REMOVE operation");
    console.log("7. Sets automatically maintain unique values (no duplicates)");
    console.log("8. You cannot mix data types within a set");

  } catch (error) {
    console.error("Error:", error);
  }
}
```

- Einzelheiten zur API finden Sie UpdateItemin der AWS SDK für JavaScript API-Referenz.

Python

SDK für Python (Boto3)

Demonstrieren Sie festgelegte Operationen mit AWS SDK für Python (Boto3).

```
import boto3
from typing import Any, Dict, List


def create_set_attribute(
    table_name: str,
    key: Dict[str, Any],
    set_name: str,
    set_values: List[Any],
    set_type: str = "string",
) -> Dict[str, Any]:
    """
    Create a new set attribute or add elements to an existing set.

    This function demonstrates how to use the ADD operation to create a new set
    or add elements to an existing set.

    Args:
        table_name (str): The name of the DynamoDB table.
        key (Dict[str, Any]): The primary key of the item to update.
        set_name (str): The name of the set attribute.
        set_values (List[Any]): The values to add to the set.
        set_type (str, optional): The type of set to create: "string", "number",
 or "binary".
            Defaults to "string".

    Returns:
        Dict[str, Any]: The response from DynamoDB containing the updated
 attribute values.
    """
    # Initialize the DynamoDB resource
    dynamodb = boto3.resource("dynamodb")
    table = dynamodb.Table(table_name)
```

```
    # Convert the list to a DynamoDB set based on the specified type
    if set_type == "string":
        dynamo_set = set(str(value) for value in set_values)
    elif set_type == "number":
        # We need to use actual float values for the DynamoDB API
        # but mypy expects strings in sets, so we need to use type: ignore
        dynamo_set = set(float(value) for value in set_values)  # type: ignore
    else:  # binary set is not directly supported in high-level API, handled
 differently
        raise ValueError("Binary sets are not supported in this example")

    # Use the ADD operation to create or update the set
    response = table.update_item(
        Key=key,
        UpdateExpression="ADD #set_attr :set_values",
        ExpressionAttributeNames={"#set_attr": set_name},
        ExpressionAttributeValues={":set_values": dynamo_set},
        ReturnValues="UPDATED_NEW",
    )

    return response


def add_to_set(
    table_name: str, key: Dict[str, Any], set_name: str, values_to_add: List[Any]
) -> Dict[str, Any]:
    """
    Add elements to an existing set attribute.

    This function demonstrates how to use the ADD operation to add elements to an
 existing set.
    If the set doesn't exist, it will be created.

    Args:
        table_name (str): The name of the DynamoDB table.
        key (Dict[str, Any]): The primary key of the item to update.
        set_name (str): The name of the set attribute.
        values_to_add (List[Any]): The values to add to the set.

    Returns:
        Dict[str, Any]: The response from DynamoDB containing the updated
 attribute values.
    """
```

```
    # Initialize the DynamoDB resource
    dynamodb = boto3.resource("dynamodb")
    table = dynamodb.Table(table_name)

    # Convert the list to a set (assuming string set for simplicity)
    dynamo_set = set(str(value) for value in values_to_add)

    # Use the ADD operation to add values to the set
    response = table.update_item(
        Key=key,
        UpdateExpression="ADD #set_attr :values_to_add",
        ExpressionAttributeNames={"#set_attr": set_name},
        ExpressionAttributeValues={":values_to_add": dynamo_set},
        ReturnValues="UPDATED_NEW",
    )

    return response


def remove_from_set(
    table_name: str, key: Dict[str, Any], set_name: str, values_to_remove:
 List[Any]
) -> Dict[str, Any]:
    """
    Remove elements from a set attribute.

    This function demonstrates how to use the DELETE operation to remove elements
 from a set.
    If the last element is removed, the attribute will be deleted entirely.

    Args:
        table_name (str): The name of the DynamoDB table.
        key (Dict[str, Any]): The primary key of the item to update.
        set_name (str): The name of the set attribute.
        values_to_remove (List[Any]): The values to remove from the set.

    Returns:
        Dict[str, Any]: The response from DynamoDB containing the updated
 attribute values.
    """
    # Initialize the DynamoDB resource
    dynamodb = boto3.resource("dynamodb")
    table = dynamodb.Table(table_name)
```

```
    # Convert the list to a set (assuming string set for simplicity)
    dynamo_set = set(str(value) for value in values_to_remove)

    # Use the DELETE operation to remove values from the set
    response = table.update_item(
        Key=key,
        UpdateExpression="DELETE #set_attr :values_to_remove",
        ExpressionAttributeNames={"#set_attr": set_name},
        ExpressionAttributeValues={":values_to_remove": dynamo_set},
        ReturnValues="UPDATED_NEW",
    )

    return response


def check_if_set_exists(table_name: str, key: Dict[str, Any], set_name: str) ->
 bool:
    """
    Check if a set attribute exists in an item.

    This function demonstrates how to check if a set attribute exists after
    potentially removing all elements from it.

    Args:
        table_name (str): The name of the DynamoDB table.
        key (Dict[str, Any]): The primary key of the item to check.
        set_name (str): The name of the set attribute.

    Returns:
        bool: True if the set attribute exists, False otherwise.
    """
    # Initialize the DynamoDB resource
    dynamodb = boto3.resource("dynamodb")
    table = dynamodb.Table(table_name)

    # Get the item
    response = table.get_item(
        Key=key, ProjectionExpression="#set_attr",
 ExpressionAttributeNames={"#set_attr": set_name}
    )

    # Check if the item exists and has the set attribute
    return "Item" in response and set_name in response["Item"]
```

```
def demonstrate_last_element_removal(
    table_name: str, key: Dict[str, Any], set_name: str
) -> Dict[str, Any]:
    """
    Demonstrate what happens when you remove the last element from a set.

    This function creates a set with a single element, then removes that element,
    showing that the attribute is completely removed when the last element is
 deleted.

    Args:
        table_name (str): The name of the DynamoDB table.
        key (Dict[str, Any]): The primary key of the item to update.
        set_name (str): The name of the set attribute.

    Returns:
        Dict[str, Any]: A dictionary containing the results of the demonstration.
    """
    # Step 1: Create a set with a single element
    create_response = create_set_attribute(
        table_name=table_name,
        key=key,
        set_name=set_name,
        set_values=["last_element"],
        set_type="string",
    )

    # Step 2: Check that the set exists
    exists_before = check_if_set_exists(table_name, key, set_name)

    # Step 3: Remove the last element
    delete_response = remove_from_set(
        table_name=table_name, key=key, set_name=set_name,
 values_to_remove=["last_element"]
    )

    # Step 4: Check if the set still exists
    exists_after = check_if_set_exists(table_name, key, set_name)

    # Return the results
    return {
        "create_response": create_response,
        "exists_before": exists_before,
```

```
        "delete_response": delete_response,
        "exists_after": exists_after,
    }


def work_with_number_set(
    table_name: str,
    key: Dict[str, Any],
    set_name: str,
    initial_values: List[float],
    values_to_add: List[float],
    values_to_remove: List[float],
) -> Dict[str, Any]:
    """
    Demonstrate working with a number set in DynamoDB.

    This function shows how to create and manipulate a set of numbers.

    Args:
        table_name (str): The name of the DynamoDB table.
        key (Dict[str, Any]): The primary key of the item to update.
        set_name (str): The name of the set attribute.
        initial_values (List[float]): The initial values for the set.
        values_to_add (List[float]): Values to add to the set.
        values_to_remove (List[float]): Values to remove from the set.

    Returns:
        Dict[str, Any]: A dictionary containing the responses from each
 operation.
    """
    # Step 1: Create the number set
    create_response = create_set_attribute(
        table_name=table_name,
        key=key,
        set_name=set_name,
        set_values=initial_values,
        set_type="number",
    )

    # Step 2: Add more numbers to the set
    add_response = add_to_set(
        table_name=table_name, key=key, set_name=set_name,
 values_to_add=values_to_add
    )
```

```
    # Step 3: Remove some numbers from the set
    remove_response = remove_from_set(
        table_name=table_name, key=key, set_name=set_name,
 values_to_remove=values_to_remove
    )

    # Step 4: Get the final state
    dynamodb = boto3.resource("dynamodb")
    table = dynamodb.Table(table_name)

    get_response = table.get_item(
        Key=key,
        ProjectionExpression=f"#{set_name}",
        ExpressionAttributeNames={f"#{set_name}": set_name},
    )

    # Return all responses
    return {
        "create_response": create_response,
        "add_response": add_response,
        "remove_response": remove_response,
        "final_state": get_response.get("Item", {}),
    }
```

Beispiel für die Verwendung von Mengenoperationen mit AWS SDK für Python (Boto3).

```
def example_usage():
    """Example of how to use the set operations functions."""
    # Example parameters
    table_name = "UserPreferences"
    key = {"UserId": "user123"}

    print("Example 1: Creating a string set attribute")
    try:
        response = create_set_attribute(
            table_name=table_name,
            key=key,
            set_name="FavoriteTags",
            set_values=["AWS", "DynamoDB", "NoSQL"],
```

```
            set_type="string",
        )
        print(f"Set attribute created successfully: {response.get('Attributes',
{})}")
    except Exception as e:
        print(f"Error creating set attribute: {e}")

    print("\nExample 2: Adding elements to an existing set")
    try:
        response = add_to_set(
            table_name=table_name,
            key=key,
            set_name="FavoriteTags",
            values_to_add=["Database", "Serverless"],
        )
        print(f"Elements added to set successfully: {response.get('Attributes',
{})}")
    except Exception as e:
        print(f"Error adding to set: {e}")

    print("\nExample 3: Removing elements from a set")
    try:
        response = remove_from_set(
            table_name=table_name, key=key, set_name="FavoriteTags",
values_to_remove=["NoSQL"]
        )
        print(f"Elements removed from set successfully:
{response.get('Attributes', {})}")
    except Exception as e:
        print(f"Error removing from set: {e}")

    print("\nExample 4: Demonstrating what happens when you remove the last
element from a set")
    try:
        results = demonstrate_last_element_removal(
            table_name=table_name, key={"UserId": "tempUser"},
set_name="SingleElementSet"
        )

        print(f"Set exists before removal: {results['exists_before']}")
        print(f"Set exists after removal: {results['exists_after']}")

        if not results["exists_after"]:
```

```
            print("The set attribute was completely removed when the last element
was deleted.")
        else:
            print("The set attribute still exists after removing the last
element.")
    except Exception as e:
        print(f"Error in last element removal demonstration: {e}")

    print("\nExample 5: Working with a number set")
    try:
        results = work_with_number_set(
            table_name=table_name,
            key={"UserId": "user456"},
            set_name="LuckyNumbers",
            initial_values=[7, 13, 42],
            values_to_add=[99, 100],
            values_to_remove=[13],
        )

        print(f"Initial number set: {results['create_response'].get('Attributes',
{})}")
        print(f"After adding numbers: {results['add_response'].get('Attributes',
{})}")
        print(f"After removing numbers:
{results['remove_response'].get('Attributes', {})}")
        print(f"Final state: {results['final_state']}")
    except Exception as e:
        print(f"Error working with number set: {e}")

    print("\nKey Points About DynamoDB Sets:")
    print("1. Sets can only contain elements of the same type (string, number, or
binary)")
    print("2. Sets automatically eliminate duplicate values")
    print("3. The ADD operation creates a set if it doesn't exist")
    print("4. The DELETE operation removes specified elements from a set")
    print("5. When the last element is removed from a set, the entire attribute
is deleted")
    print("6. Empty sets are not allowed in DynamoDB")
    print("7. Sets are unordered collections")
    print("8. The ADD operation is atomic for sets")
```

- Einzelheiten zur API finden Sie UpdateItemin AWS SDK for Python (Boto3) API Reference.

Eine vollständige Liste der AWS SDK-Entwicklerhandbücher und Codebeispiele finden Sie unter. DynamoDB mit einem SDK verwenden AWS Dieses Thema enthält auch Informationen zu den ersten Schritten und Details zu früheren SDK-Versionen.

Abfragen einer DynamoDB-Tabelle mithilfe von Batches von PartiQL-Anweisungen und einem SDK AWS

Die folgenden Code-Beispiele veranschaulichen Folgendes:

- Abrufen eines Stapels von Elementen mithilfe mehrerer SELECT-Anweisungen.
- Hinzufügen eines Stapels von Elementen hinzu, indem mehrere INSERT-Anweisungen ausgeführt werden.
- Aktualisieren eines Stapels von Elementen mithilfe mehrerer UPDATE-Anweisungen.
- Löschen eines Stapels von Elementen mithilfe mehrerer DELETE-Anweisungen.

.NET

SDK für .NET

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
// Before you run this example, download 'movies.json' from
// https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/
GettingStarted.Js.02.html,
// and put it in the same folder as the example.

// Separator for the console display.
var SepBar = new string('-', 80);
const string tableName = "movie_table";
const string movieFileName = @"..\..\..\..\..\..\..\resources\sample_files
\movies.json";
```

```
DisplayInstructions();

// Create the table and wait for it to be active.
Console.WriteLine($"Creating the movie table: {tableName}");

var success = await DynamoDBMethods.CreateMovieTableAsync(tableName);
if (success)
{
    Console.WriteLine($"Successfully created table: {tableName}.");
}

WaitForEnter();

// Add movie information to the table from moviedata.json. See the
// instructions at the top of this file to download the JSON file.
Console.WriteLine($"Inserting movies into the new table. Please wait...");
success = await PartiQLBatchMethods.InsertMovies(tableName, movieFileName);
if (success)
{
    Console.WriteLine("Movies successfully added to the table.");
}
else
{
    Console.WriteLine("Movies could not be added to the table.");
}

WaitForEnter();

// Update multiple movies by using the BatchExecute statement.
var title1 = "Star Wars";
var year1 = 1977;
var title2 = "Wizard of Oz";
var year2 = 1939;

Console.WriteLine($"Updating two movies with producer information: {title1} and
 {title2}.");
success = await PartiQLBatchMethods.GetBatch(tableName, title1, title2, year1,
 year2);
if (success)
{
    Console.WriteLine($"Successfully retrieved {title1} and {title2}.");
}
else
```

```
{
    Console.WriteLine("Select statement failed.");
}

WaitForEnter();

// Update multiple movies by using the BatchExecute statement.
var producer1 = "LucasFilm";
var producer2 = "MGM";

Console.WriteLine($"Updating two movies with producer information: {title1} and
 {title2}.");
success = await PartiQLBatchMethods.UpdateBatch(tableName, producer1, title1,
 year1, producer2, title2, year2);
if (success)
{
    Console.WriteLine($"Successfully updated {title1} and {title2}.");
}
else
{
    Console.WriteLine("Update failed.");
}

WaitForEnter();

// Delete multiple movies by using the BatchExecute statement.
Console.WriteLine($"Now we will delete {title1} and {title2} from the table.");
success = await PartiQLBatchMethods.DeleteBatch(tableName, title1, year1, title2,
 year2);

if (success)
{
    Console.WriteLine($"Deleted {title1} and {title2}");
}
else
{
    Console.WriteLine($"could not delete {title1} or {title2}");
}

WaitForEnter();

// DNow that the PartiQL Batch scenario is complete, delete the movie table.
success = await DynamoDBMethods.DeleteTableAsync(tableName);
```

```
if (success)
{
    Console.WriteLine($"Successfully deleted {tableName}");
}
else
{
    Console.WriteLine($"Could not delete {tableName}");
}

/// <summary>
/// Displays the description of the application on the console.
/// </summary>
void DisplayInstructions()
{
    Console.Clear();
    Console.WriteLine();
    Console.Write(new string(' ', 24));
    Console.WriteLine("DynamoDB PartiQL Basics Example");
    Console.WriteLine(SepBar);
    Console.WriteLine("This demo application shows the basics of using Amazon
 DynamoDB with the AWS SDK for");
    Console.WriteLine(".NET version 3.7 and .NET 6.");
    Console.WriteLine(SepBar);
    Console.WriteLine("Creates a table by using the CreateTable method.");
    Console.WriteLine("Gets multiple movies by using a PartiQL SELECT
 statement.");
    Console.WriteLine("Updates multiple movies by using the ExecuteBatch
 method.");
    Console.WriteLine("Deletes multiple movies by using a PartiQL DELETE
 statement.");
    Console.WriteLine("Cleans up the resources created for the demo by deleting
 the table.");
    Console.WriteLine(SepBar);

    WaitForEnter();
}

/// <summary>
/// Simple method to wait for the <Enter> key to be pressed.
/// </summary>
void WaitForEnter()
{
    Console.WriteLine("\nPress <Enter> to continue.");
    Console.Write(SepBar);
```

```
    _ = Console.ReadLine();
}



        /// <summary>
        /// Gets movies from the movie table by
        /// using an Amazon DynamoDB PartiQL SELECT statement.
        /// </summary>
        /// <param name="tableName">The name of the table.</param>
        /// <param name="title1">The title of the first movie.</param>
        /// <param name="title2">The title of the second movie.</param>
        /// <param name="year1">The year of the first movie.</param>
        /// <param name="year2">The year of the second movie.</param>
        /// <returns>True if successful.</returns>
        public static async Task<bool> GetBatch(
            string tableName,
            string title1,
            string title2,
            int year1,
            int year2)
        {
            var getBatch = $"SELECT * FROM {tableName} WHERE title = ? AND year
 = ?";
            var statements = new List<BatchStatementRequest>
            {
                new BatchStatementRequest
                {
                    Statement = getBatch,
                    Parameters = new List<AttributeValue>
                    {
                        new AttributeValue { S = title1 },
                        new AttributeValue { N = year1.ToString() },
                    },
                },

                new BatchStatementRequest
                {
                    Statement = getBatch,
                    Parameters = new List<AttributeValue>
                    {
                        new AttributeValue { S = title2 },
                        new AttributeValue { N = year2.ToString() },
                    },
                }
```

```
            };

            var response = await Client.BatchExecuteStatementAsync(new
BatchExecuteStatementRequest
            {
                Statements = statements,
            });

            if (response.Responses.Count > 0)
            {
                response.Responses.ForEach(r =>
                {
                    if (r.Item.Any())
                    {

Console.WriteLine($"{r.Item["title"]}\t{r.Item["year"]}");
                    }
                });
                return true;
            }
            else
            {
                Console.WriteLine($"Couldn't find either {title1} or {title2}.");
                return false;
            }

        }

        /// <summary>
        /// Inserts movies imported from a JSON file into the movie table by
        /// using an Amazon DynamoDB PartiQL INSERT statement.
        /// </summary>
        /// <param name="tableName">The name of the table into which the movie
        /// information will be inserted.</param>
        /// <param name="movieFileName">The name of the JSON file that contains
        /// movie information.</param>
        /// <returns>A Boolean value that indicates the success or failure of
        /// the insert operation.</returns>
        public static async Task<bool> InsertMovies(string tableName, string
movieFileName)
        {
            // Get the list of movies from the JSON file.
            var movies = ImportMovies(movieFileName);
```

```
            var success = false;

            if (movies is not null)
            {
                // Insert the movies in a batch using PartiQL. Because the
                // batch can contain a maximum of 25 items, insert 25 movies
                // at a time.
                string insertBatch = $"INSERT INTO {tableName} VALUE
{{'title': ?, 'year': ?}}";
                var statements = new List<BatchStatementRequest>();

                try
                {
                    for (var indexOffset = 0; indexOffset < 250; indexOffset +=
25)
                    {
                        for (var i = indexOffset; i < indexOffset + 25; i++)
                        {
                            statements.Add(new BatchStatementRequest
                            {
                                Statement = insertBatch,
                                Parameters = new List<AttributeValue>
                                {
                                    new AttributeValue { S = movies[i].Title },
                                    new AttributeValue { N =
movies[i].Year.ToString() },
                                },
                            });
                        }

                        var response = await
Client.BatchExecuteStatementAsync(new BatchExecuteStatementRequest
                        {
                            Statements = statements,
                        });

                        // Wait between batches for movies to be successfully
added.
                        System.Threading.Thread.Sleep(3000);

                        success = response.HttpStatusCode ==
System.Net.HttpStatusCode.OK;

                        // Clear the list of statements for the next batch.
```

```
                    statements.Clear();
                }
            }
            catch (AmazonDynamoDBException ex)
            {
                Console.WriteLine(ex.Message);
            }
        }

        return success;
    }

    /// <summary>
    /// Loads the contents of a JSON file into a list of movies to be
    /// added to the DynamoDB table.
    /// </summary>
    /// <param name="movieFileName">The full path to the JSON file.</param>
    /// <returns>A generic list of movie objects.</returns>
    public static List<Movie> ImportMovies(string movieFileName)
    {
        if (!File.Exists(movieFileName))
        {
            return null!;
        }

        using var sr = new StreamReader(movieFileName);
        string json = sr.ReadToEnd();
        var allMovies = JsonConvert.DeserializeObject<List<Movie>>(json);

        if (allMovies is not null)
        {
            // Return the first 250 entries.
            return allMovies.GetRange(0, 250);
        }
        else
        {
            return null!;
        }
    }

    /// <summary>
    /// Updates information for multiple movies.
    /// </summary>
    /// <param name="tableName">The name of the table containing the
```

```
        /// movies to be updated.</param>
        /// <param name="producer1">The producer name for the first movie
        /// to update.</param>
        /// <param name="title1">The title of the first movie.</param>
        /// <param name="year1">The year that the first movie was released.</
param>
        /// <param name="producer2">The producer name for the second
        /// movie to update.</param>
        /// <param name="title2">The title of the second movie.</param>
        /// <param name="year2">The year that the second movie was released.</
param>
        /// <returns>A Boolean value that indicates the success of the update.</
returns>
        public static async Task<bool> UpdateBatch(
            string tableName,
            string producer1,
            string title1,
            int year1,
            string producer2,
            string title2,
            int year2)
        {

            string updateBatch = $"UPDATE {tableName} SET Producer=? WHERE title
 = ? AND year = ?";
            var statements = new List<BatchStatementRequest>
            {
                new BatchStatementRequest
                {
                    Statement = updateBatch,
                    Parameters = new List<AttributeValue>
                    {
                        new AttributeValue { S = producer1 },
                        new AttributeValue { S = title1 },
                        new AttributeValue { N = year1.ToString() },
                    },
                },

                new BatchStatementRequest
                {
                    Statement = updateBatch,
                    Parameters = new List<AttributeValue>
                    {
                        new AttributeValue { S = producer2 },
```

```
                        new AttributeValue { S = title2 },
                        new AttributeValue { N = year2.ToString() },
                    },
                }
            };

            var response = await Client.BatchExecuteStatementAsync(new
 BatchExecuteStatementRequest
            {
                Statements = statements,
            });

            return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
        }

        /// <summary>
        /// Deletes multiple movies using a PartiQL BatchExecuteAsync
        /// statement.
        /// </summary>
        /// <param name="tableName">The name of the table containing the
        /// moves that will be deleted.</param>
        /// <param name="title1">The title of the first movie.</param>
        /// <param name="year1">The year the first movie was released.</param>
        /// <param name="title2">The title of the second movie.</param>
        /// <param name="year2">The year the second movie was released.</param>
        /// <returns>A Boolean value indicating the success of the operation.</
returns>
        public static async Task<bool> DeleteBatch(
            string tableName,
            string title1,
            int year1,
            string title2,
            int year2)
        {

            string updateBatch = $"DELETE FROM {tableName} WHERE title = ? AND
 year = ?";
            var statements = new List<BatchStatementRequest>
            {
                new BatchStatementRequest
                {
                    Statement = updateBatch,
                    Parameters = new List<AttributeValue>
                    {
```

```
                        new AttributeValue { S = title1 },
                        new AttributeValue { N = year1.ToString() },
                    },
                },

                new BatchStatementRequest
                {
                    Statement = updateBatch,
                    Parameters = new List<AttributeValue>
                    {
                        new AttributeValue { S = title2 },
                        new AttributeValue { N = year2.ToString() },
                    },
                }
            };

            var response = await Client.BatchExecuteStatementAsync(new
BatchExecuteStatementRequest
            {
                Statements = statements,
            });

            return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
        }
```

- Einzelheiten zur API finden Sie BatchExecuteStatementin der AWS SDK für .NET API-Referenz.

C++

SDK für C++

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
        Aws::Client::ClientConfiguration clientConfig;
        //  1. Create a table. (CreateTable)
```

```
        if (AwsDoc::DynamoDB::createMoviesDynamoDBTable(clientConfig)) {

            AwsDoc::DynamoDB::partiqlBatchExecuteScenario(clientConfig);

            // 7. Delete the table. (DeleteTable)
            AwsDoc::DynamoDB::deleteMoviesDynamoDBTable(clientConfig);
        }

//! Scenario to modify and query a DynamoDB table using PartiQL batch statements.
/*!
  \sa partiqlBatchExecuteScenario()
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
 */
bool AwsDoc::DynamoDB::partiqlBatchExecuteScenario(
        const Aws::Client::ClientConfiguration &clientConfiguration) {

    // 2. Add multiple movies using "Insert" statements. (BatchExecuteStatement)
    Aws::DynamoDB::DynamoDBClient dynamoClient(clientConfiguration);

    std::vector<Aws::String> titles;
    std::vector<float> ratings;
    std::vector<int> years;
    std::vector<Aws::String> plots;
    Aws::String doAgain = "n";
    do {
        Aws::String aTitle = askQuestion(
                "Enter the title of a movie you want to add to the table: ");
        titles.push_back(aTitle);
        int aYear = askQuestionForInt("What year was it released? ");
        years.push_back(aYear);
        float aRating = askQuestionForFloatRange(
                "On a scale of 1 - 10, how do you rate it? ",
                1, 10);
        ratings.push_back(aRating);
        Aws::String aPlot = askQuestion("Summarize the plot for me: ");
        plots.push_back(aPlot);

        doAgain = askQuestion(Aws::String("Would you like to add more movies? (y/
n) "));
    } while (doAgain == "y");

    std::cout << "Adding " << titles.size()
              << (titles.size() == 1 ? " movie " : " movies ")
```

```
            << "to the table using a batch \"INSERT\" statement." << std::endl;

    {
        Aws::Vector<Aws::DynamoDB::Model::BatchStatementRequest> statements(
                titles.size());

        std::stringstream sqlStream;
        sqlStream << "INSERT INTO \"" << MOVIE_TABLE_NAME << "\" VALUE {'"
                  << TITLE_KEY << "': ?, '" << YEAR_KEY << "': ?, '"
                  << INFO_KEY << "': ?}";

        std::string sql(sqlStream.str());

        for (size_t i = 0; i < statements.size(); ++i) {
            statements[i].SetStatement(sql);

            Aws::Vector<Aws::DynamoDB::Model::AttributeValue> attributes;
            attributes.push_back(
                    Aws::DynamoDB::Model::AttributeValue().SetS(titles[i]));

attributes.push_back(Aws::DynamoDB::Model::AttributeValue().SetN(years[i]));

            // Create attribute for the info map.
            Aws::DynamoDB::Model::AttributeValue infoMapAttribute;

            std::shared_ptr<Aws::DynamoDB::Model::AttributeValue> ratingAttribute
= Aws::MakeShared<Aws::DynamoDB::Model::AttributeValue>(
                    ALLOCATION_TAG.c_str());
            ratingAttribute->SetN(ratings[i]);
            infoMapAttribute.AddMEntry(RATING_KEY, ratingAttribute);

            std::shared_ptr<Aws::DynamoDB::Model::AttributeValue> plotAttribute =
Aws::MakeShared<Aws::DynamoDB::Model::AttributeValue>(
                    ALLOCATION_TAG.c_str());
            plotAttribute->SetS(plots[i]);
            infoMapAttribute.AddMEntry(PLOT_KEY, plotAttribute);
            attributes.push_back(infoMapAttribute);
            statements[i].SetParameters(attributes);
        }

        Aws::DynamoDB::Model::BatchExecuteStatementRequest request;

        request.SetStatements(statements);
```

```
       Aws::DynamoDB::Model::BatchExecuteStatementOutcome outcome =
dynamoClient.BatchExecuteStatement(
               request);
       if (!outcome.IsSuccess()) {
           std::cerr << "Failed to add the movies: " <<
outcome.GetError().GetMessage()
                     << std::endl;
           return false;
       }
   }

   std::cout << "Retrieving the movie data with a batch \"SELECT\" statement."
             << std::endl;

   // 3. Get the data for multiple movies using "Select" statements.
(BatchExecuteStatement)
   {
       Aws::Vector<Aws::DynamoDB::Model::BatchStatementRequest> statements(
               titles.size());
       std::stringstream sqlStream;
       sqlStream << "SELECT * FROM  \"" << MOVIE_TABLE_NAME << "\" WHERE "
                 << TITLE_KEY << "=? and " << YEAR_KEY << "=?";

       std::string sql(sqlStream.str());

       for (size_t i = 0; i < statements.size(); ++i) {
           statements[i].SetStatement(sql);
           Aws::Vector<Aws::DynamoDB::Model::AttributeValue> attributes;
           attributes.push_back(
                   Aws::DynamoDB::Model::AttributeValue().SetS(titles[i]));

attributes.push_back(Aws::DynamoDB::Model::AttributeValue().SetN(years[i]));
           statements[i].SetParameters(attributes);
       }

       Aws::DynamoDB::Model::BatchExecuteStatementRequest request;

       request.SetStatements(statements);

       Aws::DynamoDB::Model::BatchExecuteStatementOutcome outcome =
dynamoClient.BatchExecuteStatement(
               request);
       if (outcome.IsSuccess()) {
```

```
            const Aws::DynamoDB::Model::BatchExecuteStatementResult &result =
outcome.GetResult();

            const Aws::Vector<Aws::DynamoDB::Model::BatchStatementResponse>
&responses = result.GetResponses();

            for (const Aws::DynamoDB::Model::BatchStatementResponse &response:
responses) {
                const Aws::Map<Aws::String, Aws::DynamoDB::Model::AttributeValue>
&item = response.GetItem();

                printMovieInfo(item);
            }
        }
        else {
            std::cerr << "Failed to retrieve the movie information: "
                      << outcome.GetError().GetMessage() << std::endl;
            return false;
        }
    }

    // 4. Update the data for multiple movies using "Update" statements.
(BatchExecuteStatement)

    for (size_t i = 0; i < titles.size(); ++i) {
        ratings[i] = askQuestionForFloatRange(
                Aws::String("\nLet's update your the movie, \"") + titles[i] +
                ".\nYou rated it  " + std::to_string(ratings[i])
                + ", what new rating would you give it? ", 1, 10);
    }

    std::cout << "Updating the movie with a batch \"UPDATE\" statement." <<
std::endl;

    {
        Aws::Vector<Aws::DynamoDB::Model::BatchStatementRequest> statements(
                titles.size());

        std::stringstream sqlStream;
        sqlStream << "UPDATE \"" << MOVIE_TABLE_NAME << "\" SET "
                  << INFO_KEY << "." << RATING_KEY << "=? WHERE "
                  << TITLE_KEY << "=? AND " << YEAR_KEY << "=?";
```

```
        std::string sql(sqlStream.str());

        for (size_t i = 0; i < statements.size(); ++i) {
            statements[i].SetStatement(sql);

            Aws::Vector<Aws::DynamoDB::Model::AttributeValue> attributes;
            attributes.push_back(
                    Aws::DynamoDB::Model::AttributeValue().SetN(ratings[i]));
            attributes.push_back(
                    Aws::DynamoDB::Model::AttributeValue().SetS(titles[i]));

attributes.push_back(Aws::DynamoDB::Model::AttributeValue().SetN(years[i]));
            statements[i].SetParameters(attributes);
        }

        Aws::DynamoDB::Model::BatchExecuteStatementRequest request;

        request.SetStatements(statements);
        Aws::DynamoDB::Model::BatchExecuteStatementOutcome outcome =
dynamoClient.BatchExecuteStatement(
                request);
        if (!outcome.IsSuccess()) {
            std::cerr << "Failed to update movie information: "
                      << outcome.GetError().GetMessage() << std::endl;
            return false;
        }
    }

    std::cout << "Retrieving the updated movie data with a batch \"SELECT\"
statement."
              << std::endl;

    // 5. Get the updated data for multiple movies using "Select" statements.
(BatchExecuteStatement)
    {
        Aws::Vector<Aws::DynamoDB::Model::BatchStatementRequest> statements(
                titles.size());
        std::stringstream sqlStream;
        sqlStream << "SELECT * FROM  \"" << MOVIE_TABLE_NAME << "\" WHERE "
                  << TITLE_KEY << "=? and " << YEAR_KEY << "=?";

        std::string sql(sqlStream.str());

        for (size_t i = 0; i < statements.size(); ++i) {
```

```
            statements[i].SetStatement(sql);
            Aws::Vector<Aws::DynamoDB::Model::AttributeValue> attributes;
            attributes.push_back(
                    Aws::DynamoDB::Model::AttributeValue().SetS(titles[i]));

attributes.push_back(Aws::DynamoDB::Model::AttributeValue().SetN(years[i]));
            statements[i].SetParameters(attributes);
        }

        Aws::DynamoDB::Model::BatchExecuteStatementRequest request;

        request.SetStatements(statements);

        Aws::DynamoDB::Model::BatchExecuteStatementOutcome outcome =
dynamoClient.BatchExecuteStatement(
                request);
        if (outcome.IsSuccess()) {
            const Aws::DynamoDB::Model::BatchExecuteStatementResult &result =
outcome.GetResult();

            const Aws::Vector<Aws::DynamoDB::Model::BatchStatementResponse>
&responses = result.GetResponses();

            for (const Aws::DynamoDB::Model::BatchStatementResponse &response:
responses) {
                const Aws::Map<Aws::String, Aws::DynamoDB::Model::AttributeValue>
&item = response.GetItem();

                printMovieInfo(item);
            }
        }
        else {
            std::cerr << "Failed to retrieve the movies information: "
                      << outcome.GetError().GetMessage() << std::endl;
            return false;
        }
    }

    std::cout << "Deleting the movie data with a batch \"DELETE\" statement."
              << std::endl;

    // 6. Delete multiple movies using "Delete" statements.
(BatchExecuteStatement)
    {
```

```
        Aws::Vector<Aws::DynamoDB::Model::BatchStatementRequest> statements(
                titles.size());
        std::stringstream sqlStream;
        sqlStream << "DELETE FROM  \"" << MOVIE_TABLE_NAME << "\" WHERE "
                  << TITLE_KEY << "=? and " << YEAR_KEY << "=?";

        std::string sql(sqlStream.str());

        for (size_t i = 0; i < statements.size(); ++i) {
            statements[i].SetStatement(sql);
            Aws::Vector<Aws::DynamoDB::Model::AttributeValue> attributes;
            attributes.push_back(
                    Aws::DynamoDB::Model::AttributeValue().SetS(titles[i]));

 attributes.push_back(Aws::DynamoDB::Model::AttributeValue().SetN(years[i]));
            statements[i].SetParameters(attributes);
        }

        Aws::DynamoDB::Model::BatchExecuteStatementRequest request;

        request.SetStatements(statements);

        Aws::DynamoDB::Model::BatchExecuteStatementOutcome outcome =
 dynamoClient.BatchExecuteStatement(
                request);

        if (!outcome.IsSuccess()) {
            std::cerr << "Failed to delete the movies: "
                      << outcome.GetError().GetMessage() << std::endl;
            return false;
        }
    }

    return true;
}

//! Create a DynamoDB table to be used in sample code scenarios.
/*!
  \sa createMoviesDynamoDBTable()
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
*/
bool AwsDoc::DynamoDB::createMoviesDynamoDBTable(
        const Aws::Client::ClientConfiguration &clientConfiguration) {
```

```
    Aws::DynamoDB::DynamoDBClient dynamoClient(clientConfiguration);

    bool movieTableAlreadyExisted = false;

    {
        Aws::DynamoDB::Model::CreateTableRequest request;

        Aws::DynamoDB::Model::AttributeDefinition yearAttributeDefinition;
        yearAttributeDefinition.SetAttributeName(YEAR_KEY);
        yearAttributeDefinition.SetAttributeType(
                Aws::DynamoDB::Model::ScalarAttributeType::N);
        request.AddAttributeDefinitions(yearAttributeDefinition);

        Aws::DynamoDB::Model::AttributeDefinition titleAttributeDefinition;
        yearAttributeDefinition.SetAttributeName(TITLE_KEY);
        yearAttributeDefinition.SetAttributeType(
                Aws::DynamoDB::Model::ScalarAttributeType::S);
        request.AddAttributeDefinitions(yearAttributeDefinition);

        Aws::DynamoDB::Model::KeySchemaElement yearKeySchema;
        yearKeySchema.WithAttributeName(YEAR_KEY).WithKeyType(
                Aws::DynamoDB::Model::KeyType::HASH);
        request.AddKeySchema(yearKeySchema);

        Aws::DynamoDB::Model::KeySchemaElement titleKeySchema;
        yearKeySchema.WithAttributeName(TITLE_KEY).WithKeyType(
                Aws::DynamoDB::Model::KeyType::RANGE);
        request.AddKeySchema(yearKeySchema);

        Aws::DynamoDB::Model::ProvisionedThroughput throughput;
        throughput.WithReadCapacityUnits(
                PROVISIONED_THROUGHPUT_UNITS).WithWriteCapacityUnits(
                PROVISIONED_THROUGHPUT_UNITS);
        request.SetProvisionedThroughput(throughput);
        request.SetTableName(MOVIE_TABLE_NAME);

        std::cout << "Creating table '" << MOVIE_TABLE_NAME << "'..." <<
std::endl;
        const Aws::DynamoDB::Model::CreateTableOutcome &result =
dynamoClient.CreateTable(
                request);
        if (!result.IsSuccess()) {
            if (result.GetError().GetErrorType() ==
                Aws::DynamoDB::DynamoDBErrors::RESOURCE_IN_USE) {
```

```
                std::cout << "Table already exists." << std::endl;
                movieTableAlreadyExisted = true;
            }
            else {
                std::cerr << "Failed to create table: "
                          << result.GetError().GetMessage();
                return false;
            }
        }
    }

    // Wait for table to become active.
    if (!movieTableAlreadyExisted) {
        std::cout << "Waiting for table '" << MOVIE_TABLE_NAME
                  << "' to become active...." << std::endl;
        if (!AwsDoc::DynamoDB::waitTableActive(MOVIE_TABLE_NAME,
 clientConfiguration)) {
            return false;
        }
        std::cout << "Table '" << MOVIE_TABLE_NAME << "' created and active."
                  << std::endl;
    }

    return true;
}

//! Delete the DynamoDB table used for sample code scenarios.
/*!
  \sa deleteMoviesDynamoDBTable()
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
*/
bool AwsDoc::DynamoDB::deleteMoviesDynamoDBTable(
        const Aws::Client::ClientConfiguration &clientConfiguration) {
    Aws::DynamoDB::DynamoDBClient dynamoClient(clientConfiguration);

    Aws::DynamoDB::Model::DeleteTableRequest request;
    request.SetTableName(MOVIE_TABLE_NAME);

    const Aws::DynamoDB::Model::DeleteTableOutcome &result =
 dynamoClient.DeleteTable(
            request);
    if (result.IsSuccess()) {
        std::cout << "Your table \""
```

```
                  << result.GetResult().GetTableDescription().GetTableName()
                  << " was deleted.\n";
    }
    else {
        std::cerr << "Failed to delete table: " << result.GetError().GetMessage()
                  << std::endl;
    }

    return result.IsSuccess();
}

//! Query a newly created DynamoDB table until it is active.
/*!
  \sa waitTableActive()
  \param waitTableActive: The DynamoDB table's name.
  \param dynamoClient: A DynamoDB client.
  \return bool: Function succeeded.
*/
bool AwsDoc::DynamoDB::waitTableActive(const Aws::String &tableName,
                                       const Aws::DynamoDB::DynamoDBClient
 &dynamoClient) {

    // Repeatedly call DescribeTable until table is ACTIVE.
    const int MAX_QUERIES = 20;
    Aws::DynamoDB::Model::DescribeTableRequest request;
    request.SetTableName(tableName);

    int count = 0;
    while (count < MAX_QUERIES) {
        const Aws::DynamoDB::Model::DescribeTableOutcome &result =
 dynamoClient.DescribeTable(
                request);
        if (result.IsSuccess()) {
            Aws::DynamoDB::Model::TableStatus status =
 result.GetResult().GetTable().GetTableStatus();

            if (Aws::DynamoDB::Model::TableStatus::ACTIVE != status) {
                std::this_thread::sleep_for(std::chrono::seconds(1));
            }
            else {
                return true;
            }
        }
        else {
```

```
            std::cerr << "Error DynamoDB::waitTableActive "
                      << result.GetError().GetMessage() << std::endl;
            return false;
        }
        count++;
    }
    return false;
}
```

- Einzelheiten zur API finden Sie BatchExecuteStatementin der AWS SDK für C++ API-Referenz.

Go

SDK für Go V2

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

Führen Sie ein Szenario aus, das eine Tabelle erstellt und Stapel von PartiQL-Abfragen ausführt.

```
import (
 "context"
 "fmt"
 "log"
 "strings"
 "time"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb"
 "github.com/awsdocs/aws-doc-sdk-examples/gov2/dynamodb/actions"
)

// RunPartiQLBatchScenario shows you how to use the AWS SDK for Go
```

```
// to run batches of PartiQL statements to query a table that stores data about
 movies.
//
//   - Use batches of PartiQL statements to add, get, update, and delete data for
//     individual movies.
//
// This example creates an Amazon DynamoDB service client from the specified
 sdkConfig so that
// you can replace it with a mocked or stubbed config for unit testing.
//
// This example creates and deletes a DynamoDB table to use during the scenario.
func RunPartiQLBatchScenario(ctx context.Context, sdkConfig aws.Config, tableName
 string) {
 defer func() {
  if r := recover(); r != nil {
   fmt.Printf("Something went wrong with the demo.")
  }
 }()

 log.Println(strings.Repeat("-", 88))
 log.Println("Welcome to the Amazon DynamoDB PartiQL batch demo.")
 log.Println(strings.Repeat("-", 88))

 tableBasics := actions.TableBasics{
  DynamoDbClient: dynamodb.NewFromConfig(sdkConfig),
  TableName:      tableName,
 }
 runner := actions.PartiQLRunner{
  DynamoDbClient: dynamodb.NewFromConfig(sdkConfig),
  TableName:      tableName,
 }

 exists, err := tableBasics.TableExists(ctx)
 if err != nil {
  panic(err)
 }
 if !exists {
  log.Printf("Creating table %v...\n", tableName)
  _, err = tableBasics.CreateMovieTable(ctx)
  if err != nil {
   panic(err)
  } else {
   log.Printf("Created table %v.\n", tableName)
  }
```

```
} else {
 log.Printf("Table %v already exists.\n", tableName)
}
log.Println(strings.Repeat("-", 88))

currentYear, _, _ := time.Now().Date()
customMovies := []actions.Movie{{
 Title: "House PartiQL",
 Year:  currentYear - 5,
 Info: map[string]interface{}{
  "plot":   "Wacky high jinks result from querying a mysterious database.",
  "rating": 8.5}}, {
 Title: "House PartiQL 2",
 Year:  currentYear - 3,
 Info: map[string]interface{}{
  "plot":   "Moderate high jinks result from querying another mysterious
database.",
  "rating": 6.5}}, {
 Title: "House PartiQL 3",
 Year:  currentYear - 1,
 Info: map[string]interface{}{
  "plot":   "Tepid high jinks result from querying yet another mysterious
database.",
  "rating": 2.5},
},
}

log.Printf("Inserting a batch of movies into table '%v'.\n", tableName)
err = runner.AddMovieBatch(ctx, customMovies)
if err == nil {
 log.Printf("Added %v movies to the table.\n", len(customMovies))
}
log.Println(strings.Repeat("-", 88))

log.Println("Getting data for a batch of movies.")
movies, err := runner.GetMovieBatch(ctx, customMovies)
if err == nil {
 for _, movie := range movies {
  log.Println(movie)
 }
}
log.Println(strings.Repeat("-", 88))

newRatings := []float64{7.7, 4.4, 1.1}
```

```
 log.Println("Updating a batch of movies with new ratings.")
 err = runner.UpdateMovieBatch(ctx, customMovies, newRatings)
 if err == nil {
  log.Printf("Updated %v movies with new ratings.\n", len(customMovies))
 }
 log.Println(strings.Repeat("-", 88))

 log.Println("Getting projected data from the table to verify our update.")
 log.Println("Using a page size of 2 to demonstrate paging.")
 projections, err := runner.GetAllMovies(ctx, 2)
 if err == nil {
  log.Println("All movies:")
  for _, projection := range projections {
   log.Println(projection)
  }
 }
 log.Println(strings.Repeat("-", 88))

 log.Println("Deleting a batch of movies.")
 err = runner.DeleteMovieBatch(ctx, customMovies)
 if err == nil {
  log.Printf("Deleted %v movies.\n", len(customMovies))
 }

 err = tableBasics.DeleteTable(ctx)
 if err == nil {
  log.Printf("Deleted table %v.\n", tableBasics.TableName)
 }

 log.Println(strings.Repeat("-", 88))
 log.Println("Thanks for watching!")
 log.Println(strings.Repeat("-", 88))
}
```

Definieren Sie eine Movie-Struktur, die in diesem Beispiel verwendet wird.

```
import (
 "archive/zip"
 "bytes"
 "encoding/json"
```

```
 "fmt"
 "io"
 "log"
 "net/http"

 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/attributevalue"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb/types"
)

// Movie encapsulates data about a movie. Title and Year are the composite
 primary key
// of the movie in Amazon DynamoDB. Title is the sort key, Year is the partition
 key,
// and Info is additional data.
type Movie struct {
 Title string                `dynamodbav:"title"`
 Year  int                   `dynamodbav:"year"`
 Info  map[string]interface{} `dynamodbav:"info"`
}

// GetKey returns the composite primary key of the movie in a format that can be
// sent to DynamoDB.
func (movie Movie) GetKey() map[string]types.AttributeValue {
 title, err := attributevalue.Marshal(movie.Title)
 if err != nil {
  panic(err)
 }
 year, err := attributevalue.Marshal(movie.Year)
 if err != nil {
  panic(err)
 }
 return map[string]types.AttributeValue{"title": title, "year": year}
}

// String returns the title, year, rating, and plot of a movie, formatted for the
 example.
func (movie Movie) String() string {
 return fmt.Sprintf("%v\n\tReleased: %v\n\tRating: %v\n\tPlot: %v\n",
  movie.Title, movie.Year, movie.Info["rating"], movie.Info["plot"])
}
```

Erstellen Sie eine Struktur und Methoden, die PartiQL-Anweisungen ausführen.

```
import (
 "context"
 "fmt"
 "log"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/attributevalue"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb/types"
)

// PartiQLRunner encapsulates the Amazon DynamoDB service actions used in the
// PartiQL examples. It contains a DynamoDB service client that is used to act on
 the
// specified table.
type PartiQLRunner struct {
 DynamoDbClient *dynamodb.Client
 TableName      string
}



// AddMovieBatch runs a batch of PartiQL INSERT statements to add multiple movies
 to the
// DynamoDB table.
func (runner PartiQLRunner) AddMovieBatch(ctx context.Context, movies []Movie)
 error {
 statementRequests := make([]types.BatchStatementRequest, len(movies))
 for index, movie := range movies {
  params, err := attributevalue.MarshalList([]interface{}{movie.Title,
 movie.Year, movie.Info})
  if err != nil {
   panic(err)
  }
  statementRequests[index] = types.BatchStatementRequest{
   Statement: aws.String(fmt.Sprintf(
    "INSERT INTO \"%v\" VALUE {'title': ?, 'year': ?, 'info': ?}",
 runner.TableName)),
   Parameters: params,
  }
 }
```

```
 _, err := runner.DynamoDbClient.BatchExecuteStatement(ctx,
 &dynamodb.BatchExecuteStatementInput{
  Statements: statementRequests,
 })
 if err != nil {
  log.Printf("Couldn't insert a batch of items with PartiQL. Here's why: %v\n",
 err)
 }
 return err
}



// GetMovieBatch runs a batch of PartiQL SELECT statements to get multiple movies
 from
// the DynamoDB table by title and year.
func (runner PartiQLRunner) GetMovieBatch(ctx context.Context, movies []Movie)
 ([]Movie, error) {
 statementRequests := make([]types.BatchStatementRequest, len(movies))
 for index, movie := range movies {
  params, err := attributevalue.MarshalList([]interface{}{movie.Title,
 movie.Year})
  if err != nil {
   panic(err)
  }
  statementRequests[index] = types.BatchStatementRequest{
   Statement: aws.String(
    fmt.Sprintf("SELECT * FROM \"%v\" WHERE title=? AND year=?",
 runner.TableName)),
   Parameters: params,
  }
 }

 output, err := runner.DynamoDbClient.BatchExecuteStatement(ctx,
 &dynamodb.BatchExecuteStatementInput{
  Statements: statementRequests,
 })
 var outMovies []Movie
 if err != nil {
  log.Printf("Couldn't get a batch of items with PartiQL. Here's why: %v\n", err)
 } else {
  for _, response := range output.Responses {
   var movie Movie
```

```
    err = attributevalue.UnmarshalMap(response.Item, &movie)
    if err != nil {
     log.Printf("Couldn't unmarshal response. Here's why: %v\n", err)
    } else {
     outMovies = append(outMovies, movie)
    }
   }
  }
  return outMovies, err
}



// GetAllMovies runs a PartiQL SELECT statement to get all movies from the
 DynamoDB table.
// pageSize is not typically required and is used to show how to paginate the
 results.
// The results are projected to return only the title and rating of each movie.
func (runner PartiQLRunner) GetAllMovies(ctx context.Context, pageSize int32)
 ([]map[string]interface{}, error) {
 var output []map[string]interface{}
 var response *dynamodb.ExecuteStatementOutput
 var err error
 var nextToken *string
 for moreData := true; moreData; {
  response, err = runner.DynamoDbClient.ExecuteStatement(ctx,
 &dynamodb.ExecuteStatementInput{
   Statement: aws.String(
    fmt.Sprintf("SELECT title, info.rating FROM \"%v\"", runner.TableName)),
   Limit:     aws.Int32(pageSize),
   NextToken: nextToken,
  })
  if err != nil {
   log.Printf("Couldn't get movies. Here's why: %v\n", err)
   moreData = false
  } else {
   var pageOutput []map[string]interface{}
   err = attributevalue.UnmarshalListOfMaps(response.Items, &pageOutput)
   if err != nil {
    log.Printf("Couldn't unmarshal response. Here's why: %v\n", err)
   } else {
    log.Printf("Got a page of length %v.\n", len(response.Items))
    output = append(output, pageOutput...)
   }
```

```
   nextToken = response.NextToken
   moreData = nextToken != nil
  }
 }
 return output, err
}



// UpdateMovieBatch runs a batch of PartiQL UPDATE statements to update the
 rating of
// multiple movies that already exist in the DynamoDB table.
func (runner PartiQLRunner) UpdateMovieBatch(ctx context.Context, movies []Movie,
 ratings []float64) error {
 statementRequests := make([]types.BatchStatementRequest, len(movies))
 for index, movie := range movies {
  params, err := attributevalue.MarshalList([]interface{}{ratings[index],
 movie.Title, movie.Year})
  if err != nil {
   panic(err)
  }
  statementRequests[index] = types.BatchStatementRequest{
   Statement: aws.String(
    fmt.Sprintf("UPDATE \"%v\" SET info.rating=? WHERE title=? AND year=?",
 runner.TableName)),
   Parameters: params,
  }
 }

 _, err := runner.DynamoDbClient.BatchExecuteStatement(ctx,
 &dynamodb.BatchExecuteStatementInput{
  Statements: statementRequests,
 })
 if err != nil {
  log.Printf("Couldn't update the batch of movies. Here's why: %v\n", err)
 }
 return err
}



// DeleteMovieBatch runs a batch of PartiQL DELETE statements to remove multiple
 movies
// from the DynamoDB table.
```

```
func (runner PartiQLRunner) DeleteMovieBatch(ctx context.Context, movies []Movie)
 error {
 statementRequests := make([]types.BatchStatementRequest, len(movies))
 for index, movie := range movies {
  params, err := attributevalue.MarshalList([]interface{}{movie.Title,
 movie.Year})
  if err != nil {
   panic(err)
  }
  statementRequests[index] = types.BatchStatementRequest{
   Statement: aws.String(
    fmt.Sprintf("DELETE FROM \"%v\" WHERE title=? AND year=?",
 runner.TableName)),
   Parameters: params,
  }
 }

 _, err := runner.DynamoDbClient.BatchExecuteStatement(ctx,
 &dynamodb.BatchExecuteStatementInput{
  Statements: statementRequests,
 })
 if err != nil {
  log.Printf("Couldn't delete the batch of movies. Here's why: %v\n", err)
 }
 return err
}
```

- Einzelheiten zur API finden Sie BatchExecuteStatementin der AWS SDK für Go API-Referenz.

Java

SDK für Java 2.x

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
public class ScenarioPartiQLBatch {
    public static void main(String[] args) throws IOException {
        String tableName = "MoviesPartiQBatch";
        Region region = Region.US_EAST_1;
        DynamoDbClient ddb = DynamoDbClient.builder()
            .region(region)
            .build();

        System.out.println("Creating an Amazon DynamoDB table named " + tableName
            + " with a key named year and a sort key named title.");
        createTable(ddb, tableName);

        System.out.println("Adding multiple records into the " + tableName
            + " table using a batch command.");
        putRecordBatch(ddb);

        // Update multiple movies by using the BatchExecute statement.
        String title1 = "Star Wars";
        int year1 = 1977;
        String title2 = "Wizard of Oz";
        int year2 = 1939;

        System.out.println("Query two movies.");
        getBatch(ddb, tableName, title1, title2, year1, year2);

        System.out.println("Updating multiple records using a batch command.");
        updateTableItemBatch(ddb);

        System.out.println("Deleting multiple records using a batch command.");
        deleteItemBatch(ddb);

        System.out.println("Deleting the Amazon DynamoDB table.");
        deleteDynamoDBTable(ddb, tableName);
        ddb.close();
    }

    public static boolean getBatch(DynamoDbClient ddb, String tableName, String
 title1, String title2, int year1, int year2) {
        String getBatch = "SELECT * FROM " + tableName + " WHERE title = ? AND
 year = ?";

        List<BatchStatementRequest> statements = new ArrayList<>();
        statements.add(BatchStatementRequest.builder()
```

```
            .statement(getBatch)
            .parameters(AttributeValue.builder().s(title1).build(),
                AttributeValue.builder().n(String.valueOf(year1)).build())
            .build());
        statements.add(BatchStatementRequest.builder()
            .statement(getBatch)
            .parameters(AttributeValue.builder().s(title2).build(),
                AttributeValue.builder().n(String.valueOf(year2)).build())
            .build());

        BatchExecuteStatementRequest batchExecuteStatementRequest =
BatchExecuteStatementRequest.builder()
            .statements(statements)
            .build();

        try {
            BatchExecuteStatementResponse response =
ddb.batchExecuteStatement(batchExecuteStatementRequest);
            if (!response.responses().isEmpty()) {
                response.responses().forEach(r -> {
                    System.out.println(r.item().get("title") + "\\t" +
r.item().get("year"));
                });
                return true;
            } else {
                System.out.println("Couldn't find either " + title1 + " or " +
title2 + ".");
                return false;
            }
        } catch (DynamoDbException e) {
            System.err.println(e.getMessage());
            return false;
        }
    }

    public static void createTable(DynamoDbClient ddb, String tableName) {
        DynamoDbWaiter dbWaiter = ddb.waiter();
        ArrayList<AttributeDefinition> attributeDefinitions = new ArrayList<>();

        // Define attributes.
        attributeDefinitions.add(AttributeDefinition.builder()
            .attributeName("year")
            .attributeType("N")
            .build());
```

```
        attributeDefinitions.add(AttributeDefinition.builder()
            .attributeName("title")
            .attributeType("S")
            .build());

        ArrayList<KeySchemaElement> tableKey = new ArrayList<>();
        KeySchemaElement key = KeySchemaElement.builder()
            .attributeName("year")
            .keyType(KeyType.HASH)
            .build();

        KeySchemaElement key2 = KeySchemaElement.builder()
            .attributeName("title")
            .keyType(KeyType.RANGE) // Sort
            .build();

        // Add KeySchemaElement objects to the list.
        tableKey.add(key);
        tableKey.add(key2);

        CreateTableRequest request = CreateTableRequest.builder()
            .keySchema(tableKey)
            .billingMode(BillingMode.PAY_PER_REQUEST) //  DynamoDB automatically
scales based on traffic.
            .attributeDefinitions(attributeDefinitions)
            .tableName(tableName)
            .build();

        try {
            CreateTableResponse response = ddb.createTable(request);
            DescribeTableRequest tableRequest = DescribeTableRequest.builder()
                .tableName(tableName)
                .build();

            // Wait until the Amazon DynamoDB table is created.
            WaiterResponse<DescribeTableResponse> waiterResponse = dbWaiter
                .waitUntilTableExists(tableRequest);
            waiterResponse.matched().response().ifPresent(System.out::println);
            String newTable = response.tableDescription().tableName();
            System.out.println("The " + newTable + " was successfully created.");

        } catch (DynamoDbException e) {
            System.err.println(e.getMessage());
```

```
            System.exit(1);
        }
    }

    public static void putRecordBatch(DynamoDbClient ddb) {
        String sqlStatement = "INSERT INTO MoviesPartiQBatch VALUE {'year':?,
'title' : ?, 'info' : ?}";
        try {
            // Create three movies to add to the Amazon DynamoDB table.
            // Set data for Movie 1.
            List<AttributeValue> parameters = new ArrayList<>();

            AttributeValue att1 = AttributeValue.builder()
                .n("1977")
                .build();

            AttributeValue att2 = AttributeValue.builder()
                .s("Star Wars")
                .build();

            AttributeValue att3 = AttributeValue.builder()
                .s("No Information")
                .build();

            parameters.add(att1);
            parameters.add(att2);
            parameters.add(att3);

            BatchStatementRequest statementRequestMovie1 =
BatchStatementRequest.builder()
                .statement(sqlStatement)
                .parameters(parameters)
                .build();

            // Set data for Movie 2.
            List<AttributeValue> parametersMovie2 = new ArrayList<>();
            AttributeValue attMovie2 = AttributeValue.builder()
                .n("1939")
                .build();

            AttributeValue attMovie2A = AttributeValue.builder()
                .s("Wizard of Oz")
                .build();
```

```
            AttributeValue attMovie2B = AttributeValue.builder()
                .s("No Information")
                .build();

            parametersMovie2.add(attMovie2);
            parametersMovie2.add(attMovie2A);
            parametersMovie2.add(attMovie2B);

            BatchStatementRequest statementRequestMovie2 =
BatchStatementRequest.builder()
                .statement(sqlStatement)
                .parameters(parametersMovie2)
                .build();

            // Set data for Movie 3.
            List<AttributeValue> parametersMovie3 = new ArrayList<>();
            AttributeValue attMovie3 = AttributeValue.builder()
                .n(String.valueOf("2022"))
                .build();

            AttributeValue attMovie3A = AttributeValue.builder()
                .s("My Movie 3")
                .build();

            AttributeValue attMovie3B = AttributeValue.builder()
                .s("No Information")
                .build();

            parametersMovie3.add(attMovie3);
            parametersMovie3.add(attMovie3A);
            parametersMovie3.add(attMovie3B);

            BatchStatementRequest statementRequestMovie3 =
BatchStatementRequest.builder()
                .statement(sqlStatement)
                .parameters(parametersMovie3)
                .build();

            // Add all three movies to the list.
            List<BatchStatementRequest> myBatchStatementList = new ArrayList<>();
            myBatchStatementList.add(statementRequestMovie1);
            myBatchStatementList.add(statementRequestMovie2);
            myBatchStatementList.add(statementRequestMovie3);
```

```
            BatchExecuteStatementRequest batchRequest =
 BatchExecuteStatementRequest.builder()
                .statements(myBatchStatementList)
                .build();

            BatchExecuteStatementResponse response =
 ddb.batchExecuteStatement(batchRequest);
            System.out.println("ExecuteStatement successful: " +
 response.toString());
            System.out.println("Added new movies using a batch command.");

        } catch (DynamoDbException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
    }

    public static void updateTableItemBatch(DynamoDbClient ddb) {
        String sqlStatement = "UPDATE MoviesPartiQBatch SET info = 'directors\":
[\"Merian C. Cooper\",\"Ernest B. Schoedsack' where year=? and title=?";
        List<AttributeValue> parametersRec1 = new ArrayList<>();

        // Update three records.
        AttributeValue att1 = AttributeValue.builder()
            .n(String.valueOf("2022"))
            .build();

        AttributeValue att2 = AttributeValue.builder()
            .s("My Movie 1")
            .build();

        parametersRec1.add(att1);
        parametersRec1.add(att2);

        BatchStatementRequest statementRequestRec1 =
 BatchStatementRequest.builder()
            .statement(sqlStatement)
            .parameters(parametersRec1)
            .build();

        // Update record 2.
        List<AttributeValue> parametersRec2 = new ArrayList<>();
        AttributeValue attRec2 = AttributeValue.builder()
            .n(String.valueOf("2022"))
```

```
            .build();

        AttributeValue attRec2a = AttributeValue.builder()
            .s("My Movie 2")
            .build();

        parametersRec2.add(attRec2);
        parametersRec2.add(attRec2a);
        BatchStatementRequest statementRequestRec2 =
BatchStatementRequest.builder()
            .statement(sqlStatement)
            .parameters(parametersRec2)
            .build();

        // Update record 3.
        List<AttributeValue> parametersRec3 = new ArrayList<>();
        AttributeValue attRec3 = AttributeValue.builder()
            .n(String.valueOf("2022"))
            .build();

        AttributeValue attRec3a = AttributeValue.builder()
            .s("My Movie 3")
            .build();

        parametersRec3.add(attRec3);
        parametersRec3.add(attRec3a);
        BatchStatementRequest statementRequestRec3 =
BatchStatementRequest.builder()
            .statement(sqlStatement)
            .parameters(parametersRec3)
            .build();

        // Add all three movies to the list.
        List<BatchStatementRequest> myBatchStatementList = new ArrayList<>();
        myBatchStatementList.add(statementRequestRec1);
        myBatchStatementList.add(statementRequestRec2);
        myBatchStatementList.add(statementRequestRec3);

        BatchExecuteStatementRequest batchRequest =
BatchExecuteStatementRequest.builder()
            .statements(myBatchStatementList)
            .build();

        try {
```

```
            BatchExecuteStatementResponse response =
ddb.batchExecuteStatement(batchRequest);
            System.out.println("ExecuteStatement successful: " +
response.toString());
            System.out.println("Updated three movies using a batch command.");

        } catch (DynamoDbException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
        System.out.println("Item was updated!");
    }

    public static void deleteItemBatch(DynamoDbClient ddb) {
        String sqlStatement = "DELETE FROM MoviesPartiQBatch WHERE year = ? and
title=?";
        List<AttributeValue> parametersRec1 = new ArrayList<>();

        // Specify three records to delete.
        AttributeValue att1 = AttributeValue.builder()
            .n(String.valueOf("2022"))
            .build();

        AttributeValue att2 = AttributeValue.builder()
            .s("My Movie 1")
            .build();

        parametersRec1.add(att1);
        parametersRec1.add(att2);

        BatchStatementRequest statementRequestRec1 =
BatchStatementRequest.builder()
            .statement(sqlStatement)
            .parameters(parametersRec1)
            .build();

        // Specify record 2.
        List<AttributeValue> parametersRec2 = new ArrayList<>();
        AttributeValue attRec2 = AttributeValue.builder()
            .n(String.valueOf("2022"))
            .build();

        AttributeValue attRec2a = AttributeValue.builder()
            .s("My Movie 2")
```

```
            .build();

        parametersRec2.add(attRec2);
        parametersRec2.add(attRec2a);
        BatchStatementRequest statementRequestRec2 =
BatchStatementRequest.builder()
            .statement(sqlStatement)
            .parameters(parametersRec2)
            .build();

        // Specify record 3.
        List<AttributeValue> parametersRec3 = new ArrayList<>();
        AttributeValue attRec3 = AttributeValue.builder()
            .n(String.valueOf("2022"))
            .build();

        AttributeValue attRec3a = AttributeValue.builder()
            .s("My Movie 3")
            .build();

        parametersRec3.add(attRec3);
        parametersRec3.add(attRec3a);

        BatchStatementRequest statementRequestRec3 =
BatchStatementRequest.builder()
            .statement(sqlStatement)
            .parameters(parametersRec3)
            .build();

        // Add all three movies to the list.
        List<BatchStatementRequest> myBatchStatementList = new ArrayList<>();
        myBatchStatementList.add(statementRequestRec1);
        myBatchStatementList.add(statementRequestRec2);
        myBatchStatementList.add(statementRequestRec3);

        BatchExecuteStatementRequest batchRequest =
BatchExecuteStatementRequest.builder()
            .statements(myBatchStatementList)
            .build();

        try {
            ddb.batchExecuteStatement(batchRequest);
            System.out.println("Deleted three movies using a batch command.");
```

```
        } catch (DynamoDbException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
    }

    public static void deleteDynamoDBTable(DynamoDbClient ddb, String tableName)
 {
        DeleteTableRequest request = DeleteTableRequest.builder()
            .tableName(tableName)
            .build();

        try {
            ddb.deleteTable(request);

        } catch (DynamoDbException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
        System.out.println(tableName + " was successfully deleted!");
    }

    private static ExecuteStatementResponse
 executeStatementRequest(DynamoDbClient ddb, String statement,

 List<AttributeValue> parameters) {
        ExecuteStatementRequest request = ExecuteStatementRequest.builder()
            .statement(statement)
            .parameters(parameters)
            .build();

        return ddb.executeStatement(request);
    }
}
```

- Einzelheiten zur API finden Sie BatchExecuteStatementin der AWS SDK for Java 2.x API-Referenz.

JavaScript

SDK für JavaScript (v3)

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

Führen Sie Batch-PartiQL-Anweisungen aus.

```
import {
  BillingMode,
  CreateTableCommand,
  DeleteTableCommand,
  DescribeTableCommand,
  DynamoDBClient,
  waitUntilTableExists,
} from "@aws-sdk/client-dynamodb";
import {
  DynamoDBDocumentClient,
  BatchExecuteStatementCommand,
} from "@aws-sdk/lib-dynamodb";
import { ScenarioInput } from "@aws-doc-sdk-examples/lib/scenario";

const client = new DynamoDBClient({});
const docClient = DynamoDBDocumentClient.from(client);

const log = (msg) => console.log(`[SCENARIO] ${msg}`);
const tableName = "Cities";

export const main = async (confirmAll = false) => {
  /**
   * Delete table if it exists.
   */
  try {
    await client.send(new DescribeTableCommand({ TableName: tableName }));
    // If no error was thrown, the table exists.
    const input = new ScenarioInput(
      "deleteTable",
      `A table named ${tableName} already exists. If you choose not to delete
this table, the scenario cannot continue. Delete it?`,
```

```
      { type: "confirm", confirmAll },
    );
    const deleteTable = await input.handle({}, { confirmAll });
    if (deleteTable) {
      await client.send(new DeleteTableCommand({ tableName }));
    } else {
      console.warn(
        "Scenario could not run. Either delete ${tableName} or provide a unique
 table name.",
      );
      return;
    }
  } catch (caught) {
    if (
      caught instanceof Error &&
      caught.name === "ResourceNotFoundException"
    ) {
      // Do nothing. This means the table is not there.
    } else {
      throw caught;
    }
  }

  /**
   * Create a table.
   */

  log("Creating a table.");
  const createTableCommand = new CreateTableCommand({
    TableName: tableName,
    // This example performs a large write to the database.
    // Set the billing mode to PAY_PER_REQUEST to
    // avoid throttling the large write.
    BillingMode: BillingMode.PAY_PER_REQUEST,
    // Define the attributes that are necessary for the key schema.
    AttributeDefinitions: [
      {
        AttributeName: "name",
        // 'S' is a data type descriptor that represents a number type.
        // For a list of all data type descriptors, see the following link.
        // https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/
Programming.LowLevelAPI.html#Programming.LowLevelAPI.DataTypeDescriptors
        AttributeType: "S",
      },
```

```
    ],
    // The KeySchema defines the primary key. The primary key can be
    // a partition key, or a combination of a partition key and a sort key.
    // Key schema design is important. For more info, see
    // https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/best-
practices.html
    KeySchema: [{ AttributeName: "name", KeyType: "HASH" }],
  });
  await client.send(createTableCommand);
  log(`Table created: ${tableName}.`);

  /**
   * Wait until the table is active.
   */

  // This polls with DescribeTableCommand until the requested table is 'ACTIVE'.
  // You can't write to a table before it's active.
  log("Waiting for the table to be active.");
  await waitUntilTableExists({ client }, { TableName: tableName });
  log("Table active.");

  /**
   * Insert items.
   */

  log("Inserting cities into the table.");
  const addItemsStatementCommand = new BatchExecuteStatementCommand({
    // https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/ql-
reference.insert.html
    Statements: [
      {
        Statement: `INSERT INTO ${tableName} value {'name':?, 'population':?}`,
        Parameters: ["Alachua", 10712],
      },
      {
        Statement: `INSERT INTO ${tableName} value {'name':?, 'population':?}`,
        Parameters: ["High Springs", 6415],
      },
    ],
  });
  await docClient.send(addItemsStatementCommand);
  log("Cities inserted.");

  /**
```

```
   * Select items.
   */

  log("Selecting cities from the table.");
  const selectItemsStatementCommand = new BatchExecuteStatementCommand({
    // https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/ql-
reference.select.html
    Statements: [
      {
        Statement: `SELECT * FROM ${tableName} WHERE name=?`,
        Parameters: ["Alachua"],
      },
      {
        Statement: `SELECT * FROM ${tableName} WHERE name=?`,
        Parameters: ["High Springs"],
      },
    ],
  });
  const selectItemResponse = await docClient.send(selectItemsStatementCommand);
  log(
    `Got cities: ${selectItemResponse.Responses.map(
      (r) => `${r.Item.name} (${r.Item.population})`,
    ).join(", ")}`,
  );

  /**
   * Update items.
   */

  log("Modifying the populations.");
  const updateItemStatementCommand = new BatchExecuteStatementCommand({
    // https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/ql-
reference.update.html
    Statements: [
      {
        Statement: `UPDATE ${tableName} SET population=? WHERE name=?`,
        Parameters: [10, "Alachua"],
      },
      {
        Statement: `UPDATE ${tableName} SET population=? WHERE name=?`,
        Parameters: [5, "High Springs"],
      },
    ],
  });
```

```
  await docClient.send(updateItemStatementCommand);
  log("Updated cities.");

  /**
   * Delete the items.
   */

  log("Deleting the cities.");
  const deleteItemStatementCommand = new BatchExecuteStatementCommand({
    // https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/ql-
reference.delete.html
    Statements: [
      {
        Statement: `DELETE FROM ${tableName} WHERE name=?`,
        Parameters: ["Alachua"],
      },
      {
        Statement: `DELETE FROM ${tableName} WHERE name=?`,
        Parameters: ["High Springs"],
      },
    ],
  });
  await docClient.send(deleteItemStatementCommand);
  log("Cities deleted.");

  /**
   * Delete the table.
   */

  log("Deleting the table.");
  const deleteTableCommand = new DeleteTableCommand({ TableName: tableName });
  await client.send(deleteTableCommand);
  log("Table deleted.");
};
```

- Einzelheiten zur API finden Sie BatchExecuteStatementin der AWS SDK für JavaScript API-Referenz.

Kotlin

SDK für Kotlin

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
suspend fun main() {
    val ddb = DynamoDbClient.fromEnvironment { region = "us-east-1" }
    val tableName = "MoviesPartiQBatch"
    println("Creating an Amazon DynamoDB table named $tableName with a key named
 id and a sort key named title.")
    createTablePartiQLBatch(ddb, tableName, "year")
    putRecordBatch(ddb)
    updateTableItemBatchBatch(ddb)
    deleteItemsBatch(ddb)
    deleteTablePartiQLBatch(tableName)
}

suspend fun createTablePartiQLBatch(
    ddb: DynamoDbClient,
    tableNameVal: String,
    key: String,
) {
    val attDef =
        AttributeDefinition {
            attributeName = key
            attributeType = ScalarAttributeType.N
        }

    val attDef1 =
        AttributeDefinition {
            attributeName = "title"
            attributeType = ScalarAttributeType.S
        }

    val keySchemaVal =
        KeySchemaElement {
            attributeName = key
```

```
            keyType = KeyType.Hash
        }

    val keySchemaVal1 =
        KeySchemaElement {
            attributeName = "title"
            keyType = KeyType.Range
        }

    val request =
        CreateTableRequest {
            attributeDefinitions = listOf(attDef, attDef1)
            keySchema = listOf(keySchemaVal, keySchemaVal1)
            billingMode = BillingMode.PayPerRequest
            tableName = tableNameVal
        }

    val response = ddb.createTable(request)
    ddb.waitUntilTableExists {
        // suspend call
        tableName = tableNameVal
    }
    println("The table was successfully created
 ${response.tableDescription?.tableArn}")
}

suspend fun putRecordBatch(ddb: DynamoDbClient) {
    val sqlStatement = "INSERT INTO MoviesPartiQBatch VALUE {'year':?,
 'title' : ?, 'info' : ?}"

    // Create three movies to add to the Amazon DynamoDB table.
    val parametersMovie1 = mutableListOf<AttributeValue>()
    parametersMovie1.add(AttributeValue.N("2022"))
    parametersMovie1.add(AttributeValue.S("My Movie 1"))
    parametersMovie1.add(AttributeValue.S("No Information"))

    val statementRequestMovie1 =
        BatchStatementRequest {
            statement = sqlStatement
            parameters = parametersMovie1
        }

    // Set data for Movie 2.
    val parametersMovie2 = mutableListOf<AttributeValue>()
```

```
    parametersMovie2.add(AttributeValue.N("2022"))
    parametersMovie2.add(AttributeValue.S("My Movie 2"))
    parametersMovie2.add(AttributeValue.S("No Information"))

    val statementRequestMovie2 =
        BatchStatementRequest {
            statement = sqlStatement
            parameters = parametersMovie2
        }

    // Set data for Movie 3.
    val parametersMovie3 = mutableListOf<AttributeValue>()
    parametersMovie3.add(AttributeValue.N("2022"))
    parametersMovie3.add(AttributeValue.S("My Movie 3"))
    parametersMovie3.add(AttributeValue.S("No Information"))

    val statementRequestMovie3 =
        BatchStatementRequest {
            statement = sqlStatement
            parameters = parametersMovie3
        }

    // Add all three movies to the list.
    val myBatchStatementList = mutableListOf<BatchStatementRequest>()
    myBatchStatementList.add(statementRequestMovie1)
    myBatchStatementList.add(statementRequestMovie2)
    myBatchStatementList.add(statementRequestMovie3)

    val batchRequest =
        BatchExecuteStatementRequest {
            statements = myBatchStatementList
        }
    val response = ddb.batchExecuteStatement(batchRequest)
    println("ExecuteStatement successful: " + response.toString())
    println("Added new movies using a batch command.")
}

suspend fun updateTableItemBatchBatch(ddb: DynamoDbClient) {
    val sqlStatement =
        "UPDATE MoviesPartiQBatch SET info = 'directors\":[\"Merian C. Cooper\",
\"Ernest B. Schoedsack' where year=? and title=?"
    val parametersRec1 = mutableListOf<AttributeValue>()
    parametersRec1.add(AttributeValue.N("2022"))
    parametersRec1.add(AttributeValue.S("My Movie 1"))
```

```
    val statementRequestRec1 =
        BatchStatementRequest {
            statement = sqlStatement
            parameters = parametersRec1
        }

    // Update record 2.
    val parametersRec2 = mutableListOf<AttributeValue>()
    parametersRec2.add(AttributeValue.N("2022"))
    parametersRec2.add(AttributeValue.S("My Movie 2"))
    val statementRequestRec2 =
        BatchStatementRequest {
            statement = sqlStatement
            parameters = parametersRec2
        }

    // Update record 3.
    val parametersRec3 = mutableListOf<AttributeValue>()
    parametersRec3.add(AttributeValue.N("2022"))
    parametersRec3.add(AttributeValue.S("My Movie 3"))
    val statementRequestRec3 =
        BatchStatementRequest {
            statement = sqlStatement
            parameters = parametersRec3
        }

    // Add all three movies to the list.
    val myBatchStatementList = mutableListOf<BatchStatementRequest>()
    myBatchStatementList.add(statementRequestRec1)
    myBatchStatementList.add(statementRequestRec2)
    myBatchStatementList.add(statementRequestRec3)

    val batchRequest =
        BatchExecuteStatementRequest {
            statements = myBatchStatementList
        }

    val response = ddb.batchExecuteStatement(batchRequest)
    println("ExecuteStatement successful: $response")
    println("Updated three movies using a batch command.")
    println("Items were updated!")
}

suspend fun deleteItemsBatch(ddb: DynamoDbClient) {
```

```
    // Specify three records to delete.
    val sqlStatement = "DELETE FROM MoviesPartiQBatch WHERE year = ? and title=?"
    val parametersRec1 = mutableListOf<AttributeValue>()
    parametersRec1.add(AttributeValue.N("2022"))
    parametersRec1.add(AttributeValue.S("My Movie 1"))

    val statementRequestRec1 =
        BatchStatementRequest {
            statement = sqlStatement
            parameters = parametersRec1
        }

    // Specify record 2.
    val parametersRec2 = mutableListOf<AttributeValue>()
    parametersRec2.add(AttributeValue.N("2022"))
    parametersRec2.add(AttributeValue.S("My Movie 2"))
    val statementRequestRec2 =
        BatchStatementRequest {
            statement = sqlStatement
            parameters = parametersRec2
        }

    // Specify record 3.
    val parametersRec3 = mutableListOf<AttributeValue>()
    parametersRec3.add(AttributeValue.N("2022"))
    parametersRec3.add(AttributeValue.S("My Movie 3"))
    val statementRequestRec3 =
        BatchStatementRequest {
            statement = sqlStatement
            parameters = parametersRec3
        }

    // Add all three movies to the list.
    val myBatchStatementList = mutableListOf<BatchStatementRequest>()
    myBatchStatementList.add(statementRequestRec1)
    myBatchStatementList.add(statementRequestRec2)
    myBatchStatementList.add(statementRequestRec3)

    val batchRequest =
        BatchExecuteStatementRequest {
            statements = myBatchStatementList
        }

    ddb.batchExecuteStatement(batchRequest)
```

```
    println("Deleted three movies using a batch command.")
}

suspend fun deleteTablePartiQLBatch(tableNameVal: String) {
    val request =
        DeleteTableRequest {
            tableName = tableNameVal
        }

    DynamoDbClient { region = "us-east-1" }.use { ddb ->
        ddb.deleteTable(request)
        println("$tableNameVal was deleted")
    }
}
```

- Einzelheiten zur API finden Sie BatchExecuteStatementin der API-Referenz zum AWS SDK für Kotlin.

PHP

SDK für PHP

> **Note**
>
> Es gibt noch mehr dazu. GitHub Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
namespace DynamoDb\PartiQL_Basics;

use Aws\DynamoDb\Marshaler;
use DynamoDb;
use DynamoDb\DynamoDBAttribute;

use function AwsUtilities\loadMovieData;
use function AwsUtilities\testable_readline;

class GettingStartedWithPartiQLBatch
{
    public function run()
```

```
    {
        echo("\n");
        echo("--------------------------------------\n");
        print("Welcome to the Amazon DynamoDB - PartiQL getting started demo
using PHP!\n");
        echo("--------------------------------------\n");

        $uuid = uniqid();
        $service = new DynamoDb\DynamoDBService();

        $tableName = "partiql_demo_table_$uuid";
        $service->createTable(
            $tableName,
            [
                new DynamoDBAttribute('year', 'N', 'HASH'),
                new DynamoDBAttribute('title', 'S', 'RANGE')
            ]
        );

        echo "Waiting for table...";
        $service->dynamoDbClient->waitUntil("TableExists", ['TableName' =>
$tableName]);
        echo "table $tableName found!\n";

        echo "What's the name of the last movie you watched?\n";
        while (empty($movieName)) {
            $movieName = testable_readline("Movie name: ");
        }
        echo "And what year was it released?\n";
        $movieYear = "year";
        while (!is_numeric($movieYear) || intval($movieYear) != $movieYear) {
            $movieYear = testable_readline("Year released: ");
        }
        $key = [
            'Item' => [
                'year' => [
                    'N' => "$movieYear",
                ],
                'title' => [
                    'S' => $movieName,
                ],
            ],
        ];
```

```
        list($statement, $parameters) = $service-
>buildStatementAndParameters("INSERT", $tableName, $key);
        $service->insertItemByPartiQLBatch($statement, $parameters);

        echo "How would you rate the movie from 1-10?\n";
        $rating = 0;
        while (!is_numeric($rating) || intval($rating) != $rating || $rating < 1
 || $rating > 10) {
            $rating = testable_readline("Rating (1-10): ");
        }
        echo "What was the movie about?\n";
        while (empty($plot)) {
            $plot = testable_readline("Plot summary: ");
        }
        $attributes = [
            new DynamoDBAttribute('rating', 'N', 'HASH', $rating),
            new DynamoDBAttribute('plot', 'S', 'RANGE', $plot),
        ];

        list($statement, $parameters) = $service-
>buildStatementAndParameters("UPDATE", $tableName, $key, $attributes);
        $service->updateItemByPartiQLBatch($statement, $parameters);
        echo "Movie added and updated.\n";

        $batch = json_decode(loadMovieData());

        $service->writeBatch($tableName, $batch);

        $movie = $service->getItemByPartiQLBatch($tableName, [$key]);
        echo "\nThe movie {$movie['Responses'][0]['Item']['title']['S']}
        was released in {$movie['Responses'][0]['Item']['year']['N']}.\n";
        echo "What rating would you like to give {$movie['Responses'][0]['Item']
['title']['S']}?\n";
        $rating = 0;
        while (!is_numeric($rating) || intval($rating) != $rating || $rating < 1
 || $rating > 10) {
            $rating = testable_readline("Rating (1-10): ");
        }
        $attributes = [
            new DynamoDBAttribute('rating', 'N', 'HASH', $rating),
            new DynamoDBAttribute('plot', 'S', 'RANGE', $plot)
        ];
        list($statement, $parameters) = $service-
>buildStatementAndParameters("UPDATE", $tableName, $key, $attributes);
```

```
        $service->updateItemByPartiQLBatch($statement, $parameters);

        $movie = $service->getItemByPartiQLBatch($tableName, [$key]);
        echo "Okay, you have rated {$movie['Responses'][0]['Item']['title']
['S']}
        as a {$movie['Responses'][0]['Item']['rating']['N']}\n";

        $service->deleteItemByPartiQLBatch($statement, $parameters);
        echo "But, bad news, this was a trap. That movie has now been deleted
 because of your rating...harsh.\n";

        echo "That's okay though. The book was better. Now, for something
 lighter, in what year were you born?\n";
        $birthYear = "not a number";
        while (!is_numeric($birthYear) || $birthYear >= date("Y")) {
            $birthYear = testable_readline("Birth year: ");
        }
        $birthKey = [
            'Key' => [
                'year' => [
                    'N' => "$birthYear",
                ],
            ],
        ];
        $result = $service->query($tableName, $birthKey);
        $marshal = new Marshaler();
        echo "Here are the movies in our collection released the year you were
 born:\n";
        $oops = "Oops! There were no movies released in that year (that we know
 of).\n";
        $display = "";
        foreach ($result['Items'] as $movie) {
            $movie = $marshal->unmarshalItem($movie);
            $display .= $movie['title'] . "\n";
        }
        echo ($display) ?: $oops;

        $yearsKey = [
            'Key' => [
                'year' => [
                    'N' => [
                        'minRange' => 1990,
                        'maxRange' => 1999,
                    ],
```

```
                ],
            ],
        ];
        $filter = "year between 1990 and 1999";
        echo "\nHere's a list of all the movies released in the 90s:\n";
        $result = $service->scan($tableName, $yearsKey, $filter);
        foreach ($result['Items'] as $movie) {
            $movie = $marshal->unmarshalItem($movie);
            echo $movie['title'] . "\n";
        }

        echo "\nCleaning up this demo by deleting table $tableName...\n";
        $service->deleteTable($tableName);
    }
}

    public function insertItemByPartiQLBatch(string $statement, array
 $parameters)
    {
        $this->dynamoDbClient->batchExecuteStatement([
            'Statements' => [
                [
                    'Statement' => "$statement",
                    'Parameters' => $parameters,
                ],
            ],
        ]);
    }

    public function getItemByPartiQLBatch(string $tableName, array $keys): Result
    {
        $statements = [];
        foreach ($keys as $key) {
            list($statement, $parameters) = $this-
>buildStatementAndParameters("SELECT", $tableName, $key['Item']);
            $statements[] = [
                'Statement' => "$statement",
                'Parameters' => $parameters,
            ];
        }

        return $this->dynamoDbClient->batchExecuteStatement([
            'Statements' => $statements,
        ]);
```

```
    }

    public function updateItemByPartiQLBatch(string $statement, array
$parameters)
    {
        $this->dynamoDbClient->batchExecuteStatement([
            'Statements' => [
                [
                    'Statement' => "$statement",
                    'Parameters' => $parameters,
                ],
            ],
        ]);
    }

    public function deleteItemByPartiQLBatch(string $statement, array
$parameters)
    {
        $this->dynamoDbClient->batchExecuteStatement([
            'Statements' => [
                [
                    'Statement' => "$statement",
                    'Parameters' => $parameters,
                ],
            ],
        ]);
    }
```

- Einzelheiten zur API finden Sie BatchExecuteStatementin der AWS SDK für PHP API-Referenz.

Python

SDK für Python (Boto3)

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

Erstellen Sie eine Klasse, die Stapel von PartiQL-Anweisungen ausführen kann.

```
from datetime import datetime
from decimal import Decimal
import logging
from pprint import pprint

import boto3
from botocore.exceptions import ClientError

from scaffold import Scaffold

logger = logging.getLogger(__name__)

class PartiQLBatchWrapper:
    """
    Encapsulates a DynamoDB resource to run PartiQL statements.
    """

    def __init__(self, dyn_resource):
        """
        :param dyn_resource: A Boto3 DynamoDB resource.
        """
        self.dyn_resource = dyn_resource


    def run_partiql(self, statements, param_list):
        """
        Runs a PartiQL statement. A Boto3 resource is used even though
        `execute_statement` is called on the underlying `client` object because
 the
        resource transforms input and output from plain old Python objects
 (POPOs) to
        the DynamoDB format. If you create the client directly, you must do these
        transforms yourself.

        :param statements: The batch of PartiQL statements.
        :param param_list: The batch of PartiQL parameters that are associated
 with
                           each statement. This list must be in the same order as
 the
                           statements.
        :return: The responses returned from running the statements, if any.
        """
```

```
        try:
            output = self.dyn_resource.meta.client.batch_execute_statement(
                Statements=[
                    {"Statement": statement, "Parameters": params}
                    for statement, params in zip(statements, param_list)
                ]
            )
        except ClientError as err:
            if err.response["Error"]["Code"] == "ResourceNotFoundException":
                logger.error(
                    "Couldn't execute batch of PartiQL statements because the
 table "
                    "does not exist."
                )
            else:
                logger.error(
                    "Couldn't execute batch of PartiQL statements. Here's why:
 %s: %s",
                    err.response["Error"]["Code"],
                    err.response["Error"]["Message"],
                )
            raise
        else:
            return output
```

Führen Sie ein Szenario aus, das eine Tabelle erstellt und PartiQL-Abfragen in Stapeln ausführt.

```
def run_scenario(scaffold, wrapper, table_name):
    logging.basicConfig(level=logging.INFO, format="%(levelname)s: %(message)s")

    print("-" * 88)
    print("Welcome to the Amazon DynamoDB PartiQL batch statement demo.")
    print("-" * 88)

    print(f"Creating table '{table_name}' for the demo...")
    scaffold.create_table(table_name)
    print("-" * 88)
```

```
    movie_data = [
        {
            "title": f"House PartiQL",
            "year": datetime.now().year - 5,
            "info": {
                "plot": "Wacky high jinks result from querying a mysterious
database.",
                "rating": Decimal("8.5"),
            },
        },
        {
            "title": f"House PartiQL 2",
            "year": datetime.now().year - 3,
            "info": {
                "plot": "Moderate high jinks result from querying another
mysterious database.",
                "rating": Decimal("6.5"),
            },
        },
        {
            "title": f"House PartiQL 3",
            "year": datetime.now().year - 1,
            "info": {
                "plot": "Tepid high jinks result from querying yet another
mysterious database.",
                "rating": Decimal("2.5"),
            },
        },
    ]

    print(f"Inserting a batch of movies into table '{table_name}.")
    statements = [
        f'INSERT INTO "{table_name}" ' f"VALUE {{'title': ?, 'year': ?,
'info': ?}}"
    ] * len(movie_data)
    params = [list(movie.values()) for movie in movie_data]
    wrapper.run_partiql(statements, params)
    print("Success!")
    print("-" * 88)

    print(f"Getting data for a batch of movies.")
    statements = [f'SELECT * FROM "{table_name}" WHERE title=? AND year=?'] *
len(
        movie_data
```

```
    )
    params = [[movie["title"], movie["year"]] for movie in movie_data]
    output = wrapper.run_partiql(statements, params)
    for item in output["Responses"]:
        print(f"\n{item['Item']['title']}, {item['Item']['year']}")
        pprint(item["Item"])
    print("-" * 88)

    ratings = [Decimal("7.7"), Decimal("5.5"), Decimal("1.3")]
    print(f"Updating a batch of movies with new ratings.")
    statements = [
        f'UPDATE "{table_name}" SET info.rating=? ' f"WHERE title=? AND year=?"
    ] * len(movie_data)
    params = [
        [rating, movie["title"], movie["year"]]
        for rating, movie in zip(ratings, movie_data)
    ]
    wrapper.run_partiql(statements, params)
    print("Success!")
    print("-" * 88)

    print(f"Getting projected data from the table to verify our update.")
    output = wrapper.dyn_resource.meta.client.execute_statement(
        Statement=f'SELECT title, info.rating FROM "{table_name}"'
    )
    pprint(output["Items"])
    print("-" * 88)

    print(f"Deleting a batch of movies from the table.")
    statements = [f'DELETE FROM "{table_name}" WHERE title=? AND year=?'] * len(
        movie_data
    )
    params = [[movie["title"], movie["year"]] for movie in movie_data]
    wrapper.run_partiql(statements, params)
    print("Success!")
    print("-" * 88)

    print(f"Deleting table '{table_name}'...")
    scaffold.delete_table()
    print("-" * 88)

    print("\nThanks for watching!")
    print("-" * 88)
```

```
if __name__ == "__main__":
    try:
        dyn_res = boto3.resource("dynamodb")
        scaffold = Scaffold(dyn_res)
        movies = PartiQLBatchWrapper(dyn_res)
        run_scenario(scaffold, movies, "doc-example-table-partiql-movies")
    except Exception as e:
        print(f"Something went wrong with the demo! Here's what: {e}")
```

- Einzelheiten zur API finden Sie BatchExecuteStatementin AWS SDK for Python (Boto3) API Reference.

Ruby

SDK für Ruby

> **Note**
>
> Es gibt noch mehr dazu. GitHub Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

Führen Sie ein Szenario aus, in dem eine Tabelle erstellt wird und PartiQL-Stapelabfragen ausgeführt werden.

```
table_name = "doc-example-table-movies-partiql-#{rand(10**4)}"
scaffold = Scaffold.new(table_name)
sdk = DynamoDBPartiQLBatch.new(table_name)

new_step(1, 'Create a new DynamoDB table if none already exists.')
unless scaffold.exists?(table_name)
  puts("\nNo such table: #{table_name}. Creating it...")
  scaffold.create_table(table_name)
  print "Done!\n".green
end

new_step(2, 'Populate DynamoDB table with movie data.')
download_file = 'moviedata.json'
puts("Downloading movie database to #{download_file}...")
```

```
  movie_data = scaffold.fetch_movie_data(download_file)
  puts("Writing movie data from #{download_file} into your table...")
  scaffold.write_batch(movie_data)
  puts("Records added: #{movie_data.length}.")
  print "Done!\n".green

  new_step(3, 'Select a batch of items from the movies table.')
  puts "Let's select some popular movies for side-by-side comparison."
  response = sdk.batch_execute_select([['Mean Girls', 2004], ['Goodfellas',
 1977], ['The Prancing of the Lambs', 2005]])
  puts("Items selected: #{response['responses'].length}\n")
  print "\nDone!\n".green

  new_step(4, 'Delete a batch of items from the movies table.')
  sdk.batch_execute_write([['Mean Girls', 2004], ['Goodfellas', 1977], ['The
 Prancing of the Lambs', 2005]])
  print "\nDone!\n".green

  new_step(5, 'Delete the table.')
  return unless scaffold.exists?(table_name)

  scaffold.delete_table
end
```

- Einzelheiten zur API finden Sie BatchExecuteStatementin der AWS SDK für Ruby API-Referenz.

Eine vollständige Liste der AWS SDK-Entwicklerhandbücher und Codebeispiele finden Sie unterDynamoDB mit einem SDK verwenden AWS. Dieses Thema enthält auch Informationen zu den ersten Schritten und Details zu früheren SDK-Versionen.

Fragen Sie eine DynamoDB-Tabelle mit PartiQL und einem SDK ab AWS

Die folgenden Code-Beispiele veranschaulichen Folgendes:

- Abrufen eines Elementes durch Ausführen einer SELECT-Anweisung.
- Hinzufügen eines Elementes durch Ausführung einer INSERT-Anweisung.
- Aktualisieren eines Elementes durch Ausführung einer UPDATE-Anweisung.
- Löschen eines Elementes durch Ausführung einer DELETE-Anweisung.

.NET

SDK für .NET

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
namespace PartiQL_Basics_Scenario
{
    public class PartiQLMethods
    {
        private static readonly AmazonDynamoDBClient Client = new
 AmazonDynamoDBClient();


        /// <summary>
        /// Inserts movies imported from a JSON file into the movie table by
        /// using an Amazon DynamoDB PartiQL INSERT statement.
        /// </summary>
        /// <param name="tableName">The name of the table where the movie
        /// information will be inserted.</param>
        /// <param name="movieFileName">The name of the JSON file that contains
        /// movie information.</param>
        /// <returns>A Boolean value that indicates the success or failure of
        /// the insert operation.</returns>
        public static async Task<bool> InsertMovies(string tableName, string
 movieFileName)
        {
            // Get the list of movies from the JSON file.
            var movies = ImportMovies(movieFileName);

            var success = false;

            if (movies is not null)
            {
                // Insert the movies in a batch using PartiQL. Because the
                // batch can contain a maximum of 25 items, insert 25 movies
                // at a time.
```

```
                string insertBatch = $"INSERT INTO {tableName} VALUE
{{'title': ?, 'year': ?}}";
                var statements = new List<BatchStatementRequest>();

                try
                {
                    for (var indexOffset = 0; indexOffset < 250; indexOffset +=
25)
                    {
                        for (var i = indexOffset; i < indexOffset + 25; i++)
                        {
                            statements.Add(new BatchStatementRequest
                            {
                                Statement = insertBatch,
                                Parameters = new List<AttributeValue>
                                {
                                    new AttributeValue { S = movies[i].Title },
                                    new AttributeValue { N =
movies[i].Year.ToString() },
                                },
                            });
                        }

                        var response = await
Client.BatchExecuteStatementAsync(new BatchExecuteStatementRequest
                        {
                            Statements = statements,
                        });

                        // Wait between batches for movies to be successfully
added.
                        System.Threading.Thread.Sleep(3000);

                        success = response.HttpStatusCode ==
System.Net.HttpStatusCode.OK;

                        // Clear the list of statements for the next batch.
                        statements.Clear();
                    }
                }
                catch (AmazonDynamoDBException ex)
                {
                    Console.WriteLine(ex.Message);
                }
```

```
            }

            return success;
        }

        /// <summary>
        /// Loads the contents of a JSON file into a list of movies to be
        /// added to the DynamoDB table.
        /// </summary>
        /// <param name="movieFileName">The full path to the JSON file.</param>
        /// <returns>A generic list of movie objects.</returns>
        public static List<Movie> ImportMovies(string movieFileName)
        {
            if (!File.Exists(movieFileName))
            {
                return null!;
            }

            using var sr = new StreamReader(movieFileName);
            string json = sr.ReadToEnd();
            var allMovies = JsonConvert.DeserializeObject<List<Movie>>(json);

            if (allMovies is not null)
            {
                // Return the first 250 entries.
                return allMovies.GetRange(0, 250);
            }
            else
            {
                return null!;
            }
        }



        /// <summary>
        /// Uses a PartiQL SELECT statement to retrieve a single movie from the
        /// movie database.
        /// </summary>
        /// <param name="tableName">The name of the movie table.</param>
        /// <param name="movieTitle">The title of the movie to retrieve.</param>
        /// <returns>A list of movie data. If no movie matches the supplied
        /// title, the list is empty.</returns>
```

```
        public static async Task<List<Dictionary<string, AttributeValue>>>
GetSingleMovie(string tableName, string movieTitle)
        {
            string selectSingle = $"SELECT * FROM {tableName} WHERE title = ?";
            var parameters = new List<AttributeValue>
            {
                new AttributeValue { S = movieTitle },
            };

            var response = await Client.ExecuteStatementAsync(new
ExecuteStatementRequest
            {
                Statement = selectSingle,
                Parameters = parameters,
            });

            return response.Items;
        }



        /// <summary>
        /// Retrieve multiple movies by year using a SELECT statement.
        /// </summary>
        /// <param name="tableName">The name of the movie table.</param>
        /// <param name="year">The year the movies were released.</param>
        /// <returns></returns>
        public static async Task<List<Dictionary<string, AttributeValue>>>
GetMovies(string tableName, int year)
        {
            string selectSingle = $"SELECT * FROM {tableName} WHERE year = ?";
            var parameters = new List<AttributeValue>
            {
                new AttributeValue { N = year.ToString() },
            };

            var response = await Client.ExecuteStatementAsync(new
ExecuteStatementRequest
            {
                Statement = selectSingle,
                Parameters = parameters,
            });

            return response.Items;
```

```
        }


        /// <summary>
        /// Inserts a single movie into the movies table.
        /// </summary>
        /// <param name="tableName">The name of the table.</param>
        /// <param name="movieTitle">The title of the movie to insert.</param>
        /// <param name="year">The year that the movie was released.</param>
        /// <returns>A Boolean value that indicates the success or failure of
        /// the INSERT operation.</returns>
        public static async Task<bool> InsertSingleMovie(string tableName, string
movieTitle, int year)
        {
            string insertBatch = $"INSERT INTO {tableName} VALUE {{'title': ?,
'year': ?}}";

            var response = await Client.ExecuteStatementAsync(new
ExecuteStatementRequest
            {
                Statement = insertBatch,
                Parameters = new List<AttributeValue>
                {
                    new AttributeValue { S = movieTitle },
                    new AttributeValue { N = year.ToString() },
                },
            });

            return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
        }



        /// <summary>
        /// Updates a single movie in the table, adding information for the
        /// producer.
        /// </summary>
        /// <param name="tableName">the name of the table.</param>
        /// <param name="producer">The name of the producer.</param>
        /// <param name="movieTitle">The movie title.</param>
        /// <param name="year">The year the movie was released.</param>
        /// <returns>A Boolean value that indicates the success of the
        /// UPDATE operation.</returns>
```

```
        public static async Task<bool> UpdateSingleMovie(string tableName, string
producer, string movieTitle, int year)
        {
            string insertSingle = $"UPDATE {tableName} SET Producer=? WHERE title
= ? AND year = ?";

            var response = await Client.ExecuteStatementAsync(new
ExecuteStatementRequest
            {
                Statement = insertSingle,
                Parameters = new List<AttributeValue>
                {
                    new AttributeValue { S = producer },
                    new AttributeValue { S = movieTitle },
                    new AttributeValue { N = year.ToString() },
                },
            });

            return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
        }



        /// <summary>
        /// Deletes a single movie from the table.
        /// </summary>
        /// <param name="tableName">The name of the table.</param>
        /// <param name="movieTitle">The title of the movie to delete.</param>
        /// <param name="year">The year that the movie was released.</param>
        /// <returns>A Boolean value that indicates the success of the
        /// DELETE operation.</returns>
        public static async Task<bool> DeleteSingleMovie(string tableName, string
movieTitle, int year)
        {
            var deleteSingle = $"DELETE FROM {tableName} WHERE title = ? AND year
= ?";

            var response = await Client.ExecuteStatementAsync(new
ExecuteStatementRequest
            {
                Statement = deleteSingle,
                Parameters = new List<AttributeValue>
                {
                    new AttributeValue { S = movieTitle },
```

```
                    new AttributeValue { N = year.ToString() },
                },
            });

            return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
        }


        /// <summary>
        /// Displays the list of movies returned from a database query.
        /// </summary>
        /// <param name="items">The list of movie information to display.</param>
        private static void DisplayMovies(List<Dictionary<string,
 AttributeValue>> items)
        {
            if (items.Count > 0)
            {
                Console.WriteLine($"Found {items.Count} movies.");
                items.ForEach(item =>
 Console.WriteLine($"{item["year"].N}\t{item["title"].S}"));
            }
            else
            {
                Console.WriteLine($"Didn't find a movie that matched the supplied
 criteria.");
            }
        }


    }
}



        /// <summary>
        /// Uses a PartiQL SELECT statement to retrieve a single movie from the
        /// movie database.
        /// </summary>
        /// <param name="tableName">The name of the movie table.</param>
        /// <param name="movieTitle">The title of the movie to retrieve.</param>
        /// <returns>A list of movie data. If no movie matches the supplied
        /// title, the list is empty.</returns>
        public static async Task<List<Dictionary<string, AttributeValue>>>
 GetSingleMovie(string tableName, string movieTitle)
```

```
        {
            string selectSingle = $"SELECT * FROM {tableName} WHERE title = ?";
            var parameters = new List<AttributeValue>
            {
                new AttributeValue { S = movieTitle },
            };

            var response = await Client.ExecuteStatementAsync(new
ExecuteStatementRequest
            {
                Statement = selectSingle,
                Parameters = parameters,
            });

            return response.Items;
        }



        /// <summary>
        /// Inserts a single movie into the movies table.
        /// </summary>
        /// <param name="tableName">The name of the table.</param>
        /// <param name="movieTitle">The title of the movie to insert.</param>
        /// <param name="year">The year that the movie was released.</param>
        /// <returns>A Boolean value that indicates the success or failure of
        /// the INSERT operation.</returns>
        public static async Task<bool> InsertSingleMovie(string tableName, string
movieTitle, int year)
        {
            string insertBatch = $"INSERT INTO {tableName} VALUE {{'title': ?,
'year': ?}}";

            var response = await Client.ExecuteStatementAsync(new
ExecuteStatementRequest
            {
                Statement = insertBatch,
                Parameters = new List<AttributeValue>
                {
                    new AttributeValue { S = movieTitle },
                    new AttributeValue { N = year.ToString() },
                },
            });
```

```
            return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
        }



        /// <summary>
        /// Updates a single movie in the table, adding information for the
        /// producer.
        /// </summary>
        /// <param name="tableName">the name of the table.</param>
        /// <param name="producer">The name of the producer.</param>
        /// <param name="movieTitle">The movie title.</param>
        /// <param name="year">The year the movie was released.</param>
        /// <returns>A Boolean value that indicates the success of the
        /// UPDATE operation.</returns>
        public static async Task<bool> UpdateSingleMovie(string tableName, string
producer, string movieTitle, int year)
        {
            string insertSingle = $"UPDATE {tableName} SET Producer=? WHERE title
= ? AND year = ?";

            var response = await Client.ExecuteStatementAsync(new
ExecuteStatementRequest
            {
                Statement = insertSingle,
                Parameters = new List<AttributeValue>
                {
                    new AttributeValue { S = producer },
                    new AttributeValue { S = movieTitle },
                    new AttributeValue { N = year.ToString() },
                },
            });

            return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
        }



        /// <summary>
        /// Deletes a single movie from the table.
        /// </summary>
        /// <param name="tableName">The name of the table.</param>
        /// <param name="movieTitle">The title of the movie to delete.</param>
        /// <param name="year">The year that the movie was released.</param>
```

```
        /// <returns>A Boolean value that indicates the success of the
        /// DELETE operation.</returns>
        public static async Task<bool> DeleteSingleMovie(string tableName, string
movieTitle, int year)
        {
            var deleteSingle = $"DELETE FROM {tableName} WHERE title = ? AND year
= ?";

            var response = await Client.ExecuteStatementAsync(new
ExecuteStatementRequest
            {
                Statement = deleteSingle,
                Parameters = new List<AttributeValue>
                {
                    new AttributeValue { S = movieTitle },
                    new AttributeValue { N = year.ToString() },
                },
            });

            return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
        }
```

- Einzelheiten zur API finden Sie ExecuteStatementin der AWS SDK für .NET API-Referenz.

C++

SDK für C++

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
        //  1. Create a table. (CreateTable)
        if (AwsDoc::DynamoDB::createMoviesDynamoDBTable(clientConfig)) {

            AwsDoc::DynamoDB::partiqlExecuteScenario(clientConfig);

            // 7. Delete the table. (DeleteTable)
```

```
            AwsDoc::DynamoDB::deleteMoviesDynamoDBTable(clientConfig);
        }

//! Scenario to modify and query a DynamoDB table using single PartiQL
 statements.
/*!
  \sa partiqlExecuteScenario()
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
 */
bool
AwsDoc::DynamoDB::partiqlExecuteScenario(
        const Aws::Client::ClientConfiguration &clientConfiguration) {
    Aws::DynamoDB::DynamoDBClient dynamoClient(clientConfiguration);

    // 2. Add a new movie using an "Insert" statement. (ExecuteStatement)
    Aws::String title;
    float rating;
    int year;
    Aws::String plot;
    {
        title = askQuestion(
                "Enter the title of a movie you want to add to the table: ");
        year = askQuestionForInt("What year was it released? ");
        rating = askQuestionForFloatRange("On a scale of 1 - 10, how do you rate
 it? ",
                                          1, 10);
        plot = askQuestion("Summarize the plot for me: ");

        Aws::DynamoDB::Model::ExecuteStatementRequest request;
        std::stringstream sqlStream;
        sqlStream << "INSERT INTO \"" << MOVIE_TABLE_NAME << "\" VALUE {'"
                  << TITLE_KEY << "': ?, '" << YEAR_KEY << "': ?, '"
                  << INFO_KEY << "': ?}";

        request.SetStatement(sqlStream.str());

        // Create the parameter attributes.
        Aws::Vector<Aws::DynamoDB::Model::AttributeValue> attributes;
        attributes.push_back(Aws::DynamoDB::Model::AttributeValue().SetS(title));
        attributes.push_back(Aws::DynamoDB::Model::AttributeValue().SetN(year));

        Aws::DynamoDB::Model::AttributeValue infoMapAttribute;
```

```
        std::shared_ptr<Aws::DynamoDB::Model::AttributeValue> ratingAttribute =
Aws::MakeShared<Aws::DynamoDB::Model::AttributeValue>(
                ALLOCATION_TAG.c_str());
        ratingAttribute->SetN(rating);
        infoMapAttribute.AddMEntry(RATING_KEY, ratingAttribute);

        std::shared_ptr<Aws::DynamoDB::Model::AttributeValue> plotAttribute =
Aws::MakeShared<Aws::DynamoDB::Model::AttributeValue>(
                ALLOCATION_TAG.c_str());
        plotAttribute->SetS(plot);
        infoMapAttribute.AddMEntry(PLOT_KEY, plotAttribute);
        attributes.push_back(infoMapAttribute);
        request.SetParameters(attributes);

        Aws::DynamoDB::Model::ExecuteStatementOutcome outcome =
dynamoClient.ExecuteStatement(
                request);

        if (!outcome.IsSuccess()) {
            std::cerr << "Failed to add a movie: " <<
outcome.GetError().GetMessage()
                      << std::endl;
            return false;
        }
    }

    std::cout << "\nAdded '" << title << "' to '" << MOVIE_TABLE_NAME << "'."
              << std::endl;

    //  3. Get the data for the movie using a "Select" statement.
(ExecuteStatement)
    {
        Aws::DynamoDB::Model::ExecuteStatementRequest request;
        std::stringstream sqlStream;
        sqlStream << "SELECT * FROM  \"" << MOVIE_TABLE_NAME << "\" WHERE "
                  << TITLE_KEY << "=? and " << YEAR_KEY << "=?";

        request.SetStatement(sqlStream.str());

        Aws::Vector<Aws::DynamoDB::Model::AttributeValue> attributes;
        attributes.push_back(Aws::DynamoDB::Model::AttributeValue().SetS(title));
        attributes.push_back(Aws::DynamoDB::Model::AttributeValue().SetN(year));
        request.SetParameters(attributes);
```

```
        Aws::DynamoDB::Model::ExecuteStatementOutcome outcome =
dynamoClient.ExecuteStatement(
                request);

        if (!outcome.IsSuccess()) {
            std::cerr << "Failed to retrieve movie information: "
                      << outcome.GetError().GetMessage() << std::endl;
            return false;
        }
        else {
            // Print the retrieved movie information.
            const Aws::DynamoDB::Model::ExecuteStatementResult &result =
outcome.GetResult();

            const Aws::Vector<Aws::Map<Aws::String,
Aws::DynamoDB::Model::AttributeValue>> &items = result.GetItems();

            if (items.size() == 1) {
                printMovieInfo(items[0]);
            }
            else {
                std::cerr << "Error: " << items.size() << " movies were
retrieved. "
                          << " There should be only one movie." << std::endl;
            }
        }
    }

    //  4. Update the data for the movie using an "Update" statement.
(ExecuteStatement)
    {
        rating = askQuestionForFloatRange(
                Aws::String("\nLet's update your movie.\nYou rated it  ") +
                std::to_string(rating)
                + ", what new rating would you give it? ", 1, 10);

        Aws::DynamoDB::Model::ExecuteStatementRequest request;
        std::stringstream sqlStream;
        sqlStream << "UPDATE \"" << MOVIE_TABLE_NAME << "\" SET "
                  << INFO_KEY << "." << RATING_KEY << "=? WHERE "
                  << TITLE_KEY << "=? AND " << YEAR_KEY << "=?";

        request.SetStatement(sqlStream.str());
```

```
        Aws::Vector<Aws::DynamoDB::Model::AttributeValue> attributes;

attributes.push_back(Aws::DynamoDB::Model::AttributeValue().SetN(rating));
        attributes.push_back(Aws::DynamoDB::Model::AttributeValue().SetS(title));
        attributes.push_back(Aws::DynamoDB::Model::AttributeValue().SetN(year));

        request.SetParameters(attributes);

        Aws::DynamoDB::Model::ExecuteStatementOutcome outcome =
dynamoClient.ExecuteStatement(
                request);

        if (!outcome.IsSuccess()) {
            std::cerr << "Failed to update a movie: "
                      << outcome.GetError().GetMessage();
            return false;
        }
    }

    std::cout << "\nUpdated '" << title << "' with new attributes:" << std::endl;

    //  5. Get the updated data for the movie using a "Select" statement.
(ExecuteStatement)
    {
        Aws::DynamoDB::Model::ExecuteStatementRequest request;
        std::stringstream sqlStream;
        sqlStream << "SELECT * FROM  \"" << MOVIE_TABLE_NAME << "\" WHERE "
                  << TITLE_KEY << "=? and " << YEAR_KEY << "=?";

        request.SetStatement(sqlStream.str());

        Aws::Vector<Aws::DynamoDB::Model::AttributeValue> attributes;
        attributes.push_back(Aws::DynamoDB::Model::AttributeValue().SetS(title));
        attributes.push_back(Aws::DynamoDB::Model::AttributeValue().SetN(year));
        request.SetParameters(attributes);

        Aws::DynamoDB::Model::ExecuteStatementOutcome outcome =
dynamoClient.ExecuteStatement(
                request);
        if (!outcome.IsSuccess()) {
            std::cerr << "Failed to retrieve the movie information: "
                      << outcome.GetError().GetMessage() << std::endl;
            return false;
        }
```

```
        else {
            const Aws::DynamoDB::Model::ExecuteStatementResult &result =
outcome.GetResult();

            const Aws::Vector<Aws::Map<Aws::String,
Aws::DynamoDB::Model::AttributeValue>> &items = result.GetItems();

            if (items.size() == 1) {
                printMovieInfo(items[0]);
            }
            else {
                std::cerr << "Error: " << items.size() << " movies were
retrieved. "
                          << " There should be only one movie." << std::endl;
            }
        }
    }

    std::cout << "Deleting the movie" << std::endl;

    // 6. Delete the movie using a "Delete" statement. (ExecuteStatement)
    {
        Aws::DynamoDB::Model::ExecuteStatementRequest request;
        std::stringstream sqlStream;
        sqlStream << "DELETE FROM  \"" << MOVIE_TABLE_NAME << "\" WHERE "
                  << TITLE_KEY << "=? and " << YEAR_KEY << "=?";

        request.SetStatement(sqlStream.str());

        Aws::Vector<Aws::DynamoDB::Model::AttributeValue> attributes;
        attributes.push_back(Aws::DynamoDB::Model::AttributeValue().SetS(title));
        attributes.push_back(Aws::DynamoDB::Model::AttributeValue().SetN(year));
        request.SetParameters(attributes);

        Aws::DynamoDB::Model::ExecuteStatementOutcome outcome =
dynamoClient.ExecuteStatement(
                request);
        if (!outcome.IsSuccess()) {
            std::cerr << "Failed to delete the movie: "
                      << outcome.GetError().GetMessage() << std::endl;
            return false;
        }
    }
```

```
    std::cout << "Movie successfully deleted." << std::endl;
    return true;
}

//! Create a DynamoDB table to be used in sample code scenarios.
/*!
  \sa createMoviesDynamoDBTable()
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
*/
bool AwsDoc::DynamoDB::createMoviesDynamoDBTable(
        const Aws::Client::ClientConfiguration &clientConfiguration) {
    Aws::DynamoDB::DynamoDBClient dynamoClient(clientConfiguration);

    bool movieTableAlreadyExisted = false;

    {
        Aws::DynamoDB::Model::CreateTableRequest request;

        Aws::DynamoDB::Model::AttributeDefinition yearAttributeDefinition;
        yearAttributeDefinition.SetAttributeName(YEAR_KEY);
        yearAttributeDefinition.SetAttributeType(
                Aws::DynamoDB::Model::ScalarAttributeType::N);
        request.AddAttributeDefinitions(yearAttributeDefinition);

        Aws::DynamoDB::Model::AttributeDefinition titleAttributeDefinition;
        yearAttributeDefinition.SetAttributeName(TITLE_KEY);
        yearAttributeDefinition.SetAttributeType(
                Aws::DynamoDB::Model::ScalarAttributeType::S);
        request.AddAttributeDefinitions(yearAttributeDefinition);

        Aws::DynamoDB::Model::KeySchemaElement yearKeySchema;
        yearKeySchema.WithAttributeName(YEAR_KEY).WithKeyType(
                Aws::DynamoDB::Model::KeyType::HASH);
        request.AddKeySchema(yearKeySchema);

        Aws::DynamoDB::Model::KeySchemaElement titleKeySchema;
        yearKeySchema.WithAttributeName(TITLE_KEY).WithKeyType(
                Aws::DynamoDB::Model::KeyType::RANGE);
        request.AddKeySchema(yearKeySchema);

        Aws::DynamoDB::Model::ProvisionedThroughput throughput;
        throughput.WithReadCapacityUnits(
                PROVISIONED_THROUGHPUT_UNITS).WithWriteCapacityUnits(
```

```
                PROVISIONED_THROUGHPUT_UNITS);
        request.SetProvisionedThroughput(throughput);
        request.SetTableName(MOVIE_TABLE_NAME);

        std::cout << "Creating table '" << MOVIE_TABLE_NAME << "'..." <<
 std::endl;
        const Aws::DynamoDB::Model::CreateTableOutcome &result =
 dynamoClient.CreateTable(
                request);
        if (!result.IsSuccess()) {
            if (result.GetError().GetErrorType() ==
                Aws::DynamoDB::DynamoDBErrors::RESOURCE_IN_USE) {
                std::cout << "Table already exists." << std::endl;
                movieTableAlreadyExisted = true;
            }
            else {
                std::cerr << "Failed to create table: "
                          << result.GetError().GetMessage();
                return false;
            }
        }
    }

    // Wait for table to become active.
    if (!movieTableAlreadyExisted) {
        std::cout << "Waiting for table '" << MOVIE_TABLE_NAME
                  << "' to become active...." << std::endl;
        if (!AwsDoc::DynamoDB::waitTableActive(MOVIE_TABLE_NAME,
 clientConfiguration)) {
            return false;
        }
        std::cout << "Table '" << MOVIE_TABLE_NAME << "' created and active."
                  << std::endl;
    }

    return true;
}

//! Delete the DynamoDB table used for sample code scenarios.
/*!
  \sa deleteMoviesDynamoDBTable()
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
*/
```

```
bool AwsDoc::DynamoDB::deleteMoviesDynamoDBTable(
        const Aws::Client::ClientConfiguration &clientConfiguration) {
    Aws::DynamoDB::DynamoDBClient dynamoClient(clientConfiguration);

    Aws::DynamoDB::Model::DeleteTableRequest request;
    request.SetTableName(MOVIE_TABLE_NAME);

    const Aws::DynamoDB::Model::DeleteTableOutcome &result =
 dynamoClient.DeleteTable(
            request);
    if (result.IsSuccess()) {
        std::cout << "Your table \""
                  << result.GetResult().GetTableDescription().GetTableName()
                  << " was deleted.\n";
    }
    else {
        std::cerr << "Failed to delete table: " << result.GetError().GetMessage()
                  << std::endl;
    }

    return result.IsSuccess();
}

//! Query a newly created DynamoDB table until it is active.
/*!
  \sa waitTableActive()
  \param waitTableActive: The DynamoDB table's name.
  \param dynamoClient: A DynamoDB client.
  \return bool: Function succeeded.
*/
bool AwsDoc::DynamoDB::waitTableActive(const Aws::String &tableName,
                                       const Aws::DynamoDB::DynamoDBClient
 &dynamoClient) {

    // Repeatedly call DescribeTable until table is ACTIVE.
    const int MAX_QUERIES = 20;
    Aws::DynamoDB::Model::DescribeTableRequest request;
    request.SetTableName(tableName);

    int count = 0;
    while (count < MAX_QUERIES) {
        const Aws::DynamoDB::Model::DescribeTableOutcome &result =
 dynamoClient.DescribeTable(
                request);
```

```
        if (result.IsSuccess()) {
            Aws::DynamoDB::Model::TableStatus status =
 result.GetResult().GetTable().GetTableStatus();

            if (Aws::DynamoDB::Model::TableStatus::ACTIVE != status) {
                std::this_thread::sleep_for(std::chrono::seconds(1));
            }
            else {
                return true;
            }
        }
        else {
            std::cerr << "Error DynamoDB::waitTableActive "
                      << result.GetError().GetMessage() << std::endl;
            return false;
        }
        count++;
    }
    return false;
}
```

- Einzelheiten zur API finden Sie ExecuteStatementin der AWS SDK für C++ API-Referenz.

Go

SDK für Go V2

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

Führen Sie ein Szenario aus, das eine Tabelle erstellt und PartiQL-Abfragen ausführt.

```
import (
 "context"
 "fmt"
 "log"
 "strings"
```

```
 "time"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb"
 "github.com/awsdocs/aws-doc-sdk-examples/gov2/dynamodb/actions"
)

// RunPartiQLSingleScenario shows you how to use the AWS SDK for Go
// to use PartiQL to query a table that stores data about movies.
//
// * Use PartiQL statements to add, get, update, and delete data for individual
 movies.
//
// This example creates an Amazon DynamoDB service client from the specified
 sdkConfig so that
// you can replace it with a mocked or stubbed config for unit testing.
//
// This example creates and deletes a DynamoDB table to use during the scenario.
func RunPartiQLSingleScenario(ctx context.Context, sdkConfig aws.Config,
 tableName string) {
 defer func() {
  if r := recover(); r != nil {
   fmt.Printf("Something went wrong with the demo.")
  }
 }()

 log.Println(strings.Repeat("-", 88))
 log.Println("Welcome to the Amazon DynamoDB PartiQL single action demo.")
 log.Println(strings.Repeat("-", 88))

 tableBasics := actions.TableBasics{
  DynamoDbClient: dynamodb.NewFromConfig(sdkConfig),
  TableName:      tableName,
 }
 runner := actions.PartiQLRunner{
  DynamoDbClient: dynamodb.NewFromConfig(sdkConfig),
  TableName:      tableName,
 }

 exists, err := tableBasics.TableExists(ctx)
 if err != nil {
  panic(err)
 }
 if !exists {
```

```
 log.Printf("Creating table %v...\n", tableName)
 _, err = tableBasics.CreateMovieTable(ctx)
 if err != nil {
  panic(err)
 } else {
  log.Printf("Created table %v.\n", tableName)
 }
} else {
 log.Printf("Table %v already exists.\n", tableName)
}
log.Println(strings.Repeat("-", 88))

currentYear, _, _ := time.Now().Date()
customMovie := actions.Movie{
 Title: "24 Hour PartiQL People",
 Year:  currentYear,
 Info: map[string]interface{}{
  "plot":   "A group of data developers discover a new query language they can't
stop using.",
  "rating": 9.9,
 },
}

log.Printf("Inserting movie '%v' released in %v.", customMovie.Title,
customMovie.Year)
err = runner.AddMovie(ctx, customMovie)
if err == nil {
 log.Printf("Added %v to the movie table.\n", customMovie.Title)
}
log.Println(strings.Repeat("-", 88))

log.Printf("Getting data for movie '%v' released in %v.", customMovie.Title,
customMovie.Year)
movie, err := runner.GetMovie(ctx, customMovie.Title, customMovie.Year)
if err == nil {
 log.Println(movie)
}
log.Println(strings.Repeat("-", 88))

newRating := 6.6
log.Printf("Updating movie '%v' with a rating of %v.", customMovie.Title,
newRating)
err = runner.UpdateMovie(ctx, customMovie, newRating)
if err == nil {
```

```
  log.Printf("Updated %v with a new rating.\n", customMovie.Title)
 }
 log.Println(strings.Repeat("-", 88))

 log.Printf("Getting data again to verify the update.")
 movie, err = runner.GetMovie(ctx, customMovie.Title, customMovie.Year)
 if err == nil {
  log.Println(movie)
 }
 log.Println(strings.Repeat("-", 88))

 log.Printf("Deleting movie '%v'.\n", customMovie.Title)
 err = runner.DeleteMovie(ctx, customMovie)
 if err == nil {
  log.Printf("Deleted %v.\n", customMovie.Title)
 }

 err = tableBasics.DeleteTable(ctx)
 if err == nil {
  log.Printf("Deleted table %v.\n", tableBasics.TableName)
 }

 log.Println(strings.Repeat("-", 88))
 log.Println("Thanks for watching!")
 log.Println(strings.Repeat("-", 88))
}
```

Definieren Sie eine Movie-Struktur, die in diesem Beispiel verwendet wird.

```
import (
 "archive/zip"
 "bytes"
 "encoding/json"
 "fmt"
 "io"
 "log"
 "net/http"

 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/attributevalue"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb/types"
```

```
)

// Movie encapsulates data about a movie. Title and Year are the composite
 primary key
// of the movie in Amazon DynamoDB. Title is the sort key, Year is the partition
 key,
// and Info is additional data.
type Movie struct {
 Title string                `dynamodbav:"title"`
 Year  int                   `dynamodbav:"year"`
 Info  map[string]interface{} `dynamodbav:"info"`
}

// GetKey returns the composite primary key of the movie in a format that can be
// sent to DynamoDB.
func (movie Movie) GetKey() map[string]types.AttributeValue {
 title, err := attributevalue.Marshal(movie.Title)
 if err != nil {
  panic(err)
 }
 year, err := attributevalue.Marshal(movie.Year)
 if err != nil {
  panic(err)
 }
 return map[string]types.AttributeValue{"title": title, "year": year}
}

// String returns the title, year, rating, and plot of a movie, formatted for the
 example.
func (movie Movie) String() string {
 return fmt.Sprintf("%v\n\tReleased: %v\n\tRating: %v\n\tPlot: %v\n",
  movie.Title, movie.Year, movie.Info["rating"], movie.Info["plot"])
}
```

Erstellen Sie eine Struktur und Methoden, die PartiQL-Anweisungen ausführen.

```
import (
 "context"
 "fmt"
 "log"
```

```
 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/attributevalue"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb/types"
)

// PartiQLRunner encapsulates the Amazon DynamoDB service actions used in the
// PartiQL examples. It contains a DynamoDB service client that is used to act on
 the
// specified table.
type PartiQLRunner struct {
 DynamoDbClient *dynamodb.Client
 TableName      string
}



// AddMovie runs a PartiQL INSERT statement to add a movie to the DynamoDB table.
func (runner PartiQLRunner) AddMovie(ctx context.Context, movie Movie) error {
 params, err := attributevalue.MarshalList([]interface{}{movie.Title, movie.Year,
 movie.Info})
 if err != nil {
  panic(err)
 }
 _, err = runner.DynamoDbClient.ExecuteStatement(ctx,
 &dynamodb.ExecuteStatementInput{
  Statement: aws.String(
   fmt.Sprintf("INSERT INTO \"%v\" VALUE {'title': ?, 'year': ?, 'info': ?}",
    runner.TableName)),
  Parameters: params,
 })
 if err != nil {
  log.Printf("Couldn't insert an item with PartiQL. Here's why: %v\n", err)
 }
 return err
}



// GetMovie runs a PartiQL SELECT statement to get a movie from the DynamoDB
 table by
// title and year.
```

```
func (runner PartiQLRunner) GetMovie(ctx context.Context, title string, year int)
 (Movie, error) {
 var movie Movie
 params, err := attributevalue.MarshalList([]interface{}{title, year})
 if err != nil {
  panic(err)
 }
 response, err := runner.DynamoDbClient.ExecuteStatement(ctx,
 &dynamodb.ExecuteStatementInput{
  Statement: aws.String(
   fmt.Sprintf("SELECT * FROM \"%v\" WHERE title=? AND year=?",
    runner.TableName)),
  Parameters: params,
 })
 if err != nil {
  log.Printf("Couldn't get info about %v. Here's why: %v\n", title, err)
 } else {
  err = attributevalue.UnmarshalMap(response.Items[0], &movie)
  if err != nil {
   log.Printf("Couldn't unmarshal response. Here's why: %v\n", err)
  }
 }
 return movie, err
}



// UpdateMovie runs a PartiQL UPDATE statement to update the rating of a movie
 that
// already exists in the DynamoDB table.
func (runner PartiQLRunner) UpdateMovie(ctx context.Context, movie Movie, rating
 float64) error {
 params, err := attributevalue.MarshalList([]interface{}{rating, movie.Title,
 movie.Year})
 if err != nil {
  panic(err)
 }
 _, err = runner.DynamoDbClient.ExecuteStatement(ctx,
 &dynamodb.ExecuteStatementInput{
  Statement: aws.String(
   fmt.Sprintf("UPDATE \"%v\" SET info.rating=? WHERE title=? AND year=?",
    runner.TableName)),
  Parameters: params,
 })
```

```
 if err != nil {
  log.Printf("Couldn't update movie %v. Here's why: %v\n", movie.Title, err)
 }
 return err
}



// DeleteMovie runs a PartiQL DELETE statement to remove a movie from the
 DynamoDB table.
func (runner PartiQLRunner) DeleteMovie(ctx context.Context, movie Movie) error {
 params, err := attributevalue.MarshalList([]interface{}{movie.Title,
 movie.Year})
 if err != nil {
  panic(err)
 }
 _, err = runner.DynamoDbClient.ExecuteStatement(ctx,
 &dynamodb.ExecuteStatementInput{
  Statement: aws.String(
   fmt.Sprintf("DELETE FROM \"%v\" WHERE title=? AND year=?",
    runner.TableName)),
  Parameters: params,
 })
 if err != nil {
  log.Printf("Couldn't delete %v from the table. Here's why: %v\n", movie.Title,
 err)
 }
 return err
}
```

- Einzelheiten zur API finden Sie ExecuteStatementin der AWS SDK für Go API-Referenz.

Java

SDK für Java 2.x

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
public class ScenarioPartiQ {
    public static void main(String[] args) throws IOException {
        String fileName = "../../../resources/sample_files/movies.json";
        String tableName = "MoviesPartiQ";
        Region region = Region.US_EAST_1;
        DynamoDbClient ddb = DynamoDbClient.builder()
            .region(region)
            .build();

        System.out.println(
            "******* Creating an Amazon DynamoDB table named MoviesPartiQ with a
 key named year and a sort key named title.");
        createTable(ddb, tableName);

        System.out.println("Loading data into the MoviesPartiQ table.");
        loadData(ddb, fileName);

        System.out.println("Getting data from the MoviesPartiQ table.");
        getItem(ddb);

        System.out.println("Putting a record into the MoviesPartiQ table.");
        putRecord(ddb);

        System.out.println("Updating a record.");
        updateTableItem(ddb);

        System.out.println("Querying the movies released in 2013.");
        queryTable(ddb);

        System.out.println("Deleting the Amazon DynamoDB table.");
        deleteDynamoDBTable(ddb, tableName);
        ddb.close();
    }

    public static void createTable(DynamoDbClient ddb, String tableName) {
        DynamoDbWaiter dbWaiter = ddb.waiter();
        ArrayList<AttributeDefinition> attributeDefinitions = new ArrayList<>();

        // Define attributes.
        attributeDefinitions.add(AttributeDefinition.builder()
            .attributeName("year")
            .attributeType("N")
            .build());
```

```
        attributeDefinitions.add(AttributeDefinition.builder()
            .attributeName("title")
            .attributeType("S")
            .build());

        ArrayList<KeySchemaElement> tableKey = new ArrayList<>();
        KeySchemaElement key = KeySchemaElement.builder()
            .attributeName("year")
            .keyType(KeyType.HASH)
            .build();

        KeySchemaElement key2 = KeySchemaElement.builder()
            .attributeName("title")
            .keyType(KeyType.RANGE) // Sort
            .build();

        // Add KeySchemaElement objects to the list.
        tableKey.add(key);
        tableKey.add(key2);

        CreateTableRequest request = CreateTableRequest.builder()
            .keySchema(tableKey)
            .billingMode(BillingMode.PAY_PER_REQUEST) //Scales based on traffic.
            .attributeDefinitions(attributeDefinitions)
            .tableName(tableName)
            .build();

        try {
            CreateTableResponse response = ddb.createTable(request);
            DescribeTableRequest tableRequest = DescribeTableRequest.builder()
                .tableName(tableName)
                .build();

            // Wait until the Amazon DynamoDB table is created.
            WaiterResponse<DescribeTableResponse> waiterResponse =
dbWaiter.waitUntilTableExists(tableRequest);
            waiterResponse.matched().response().ifPresent(System.out::println);
            String newTable = response.tableDescription().tableName();
            System.out.println("The " + newTable + " was successfully created.");

        } catch (DynamoDbException e) {
            System.err.println(e.getMessage());
            System.exit(1);
```

```
        }
    }

    // Load data into the table.
    public static void loadData(DynamoDbClient ddb, String fileName) throws
IOException {

        String sqlStatement = "INSERT INTO MoviesPartiQ VALUE {'year':?,
'title' : ?, 'info' : ?}";
        JsonParser parser = new JsonFactory().createParser(new File(fileName));
        com.fasterxml.jackson.databind.JsonNode rootNode = new
ObjectMapper().readTree(parser);
        Iterator<JsonNode> iter = rootNode.iterator();
        ObjectNode currentNode;
        int t = 0;
        List<AttributeValue> parameters = new ArrayList<>();
        while (iter.hasNext()) {

            // Add 200 movies to the table.
            if (t == 200)
                break;
            currentNode = (ObjectNode) iter.next();

            int year = currentNode.path("year").asInt();
            String title = currentNode.path("title").asText();
            String info = currentNode.path("info").toString();

            AttributeValue att1 = AttributeValue.builder()
                .n(String.valueOf(year))
                .build();

            AttributeValue att2 = AttributeValue.builder()
                .s(title)
                .build();

            AttributeValue att3 = AttributeValue.builder()
                .s(info)
                .build();

            parameters.add(att1);
            parameters.add(att2);
            parameters.add(att3);

            // Insert the movie into the Amazon DynamoDB table.
```

```
            executeStatementRequest(ddb, sqlStatement, parameters);
            System.out.println("Added Movie " + title);

            parameters.remove(att1);
            parameters.remove(att2);
            parameters.remove(att3);
            t++;
        }
    }

    public static void getItem(DynamoDbClient ddb) {

        String sqlStatement = "SELECT * FROM MoviesPartiQ where year=? and
title=?";
        List<AttributeValue> parameters = new ArrayList<>();
        AttributeValue att1 = AttributeValue.builder()
            .n("2012")
            .build();

        AttributeValue att2 = AttributeValue.builder()
            .s("The Perks of Being a Wallflower")
            .build();

        parameters.add(att1);
        parameters.add(att2);

        try {
            ExecuteStatementResponse response = executeStatementRequest(ddb,
sqlStatement, parameters);
            System.out.println("ExecuteStatement successful: " +
response.toString());

        } catch (DynamoDbException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
    }

    public static void putRecord(DynamoDbClient ddb) {

        String sqlStatement = "INSERT INTO MoviesPartiQ VALUE {'year':?,
'title' : ?, 'info' : ?}";
        try {
            List<AttributeValue> parameters = new ArrayList<>();
```

```
            AttributeValue att1 = AttributeValue.builder()
                .n(String.valueOf("2020"))
                .build();

            AttributeValue att2 = AttributeValue.builder()
                .s("My Movie")
                .build();

            AttributeValue att3 = AttributeValue.builder()
                .s("No Information")
                .build();

            parameters.add(att1);
            parameters.add(att2);
            parameters.add(att3);

            executeStatementRequest(ddb, sqlStatement, parameters);
            System.out.println("Added new movie.");

        } catch (DynamoDbException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
    }

    public static void updateTableItem(DynamoDbClient ddb) {

        String sqlStatement = "UPDATE MoviesPartiQ SET info = 'directors\":
[\"Merian C. Cooper\",\"Ernest B. Schoedsack' where year=? and title=?";
        List<AttributeValue> parameters = new ArrayList<>();
        AttributeValue att1 = AttributeValue.builder()
            .n(String.valueOf("2013"))
            .build();

        AttributeValue att2 = AttributeValue.builder()
            .s("The East")
            .build();

        parameters.add(att1);
        parameters.add(att2);

        try {
            executeStatementRequest(ddb, sqlStatement, parameters);
```

```
        } catch (DynamoDbException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
        System.out.println("Item was updated!");
    }

    // Query the table where the year is 2013.
    public static void queryTable(DynamoDbClient ddb) {
        String sqlStatement = "SELECT * FROM MoviesPartiQ where year = ? ORDER BY
year";
        try {

            List<AttributeValue> parameters = new ArrayList<>();
            AttributeValue att1 = AttributeValue.builder()
                .n(String.valueOf("2013"))
                .build();
            parameters.add(att1);

            // Get items in the table and write out the ID value.
            ExecuteStatementResponse response = executeStatementRequest(ddb,
sqlStatement, parameters);
            System.out.println("ExecuteStatement successful: " +
response.toString());

        } catch (DynamoDbException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
    }

    public static void deleteDynamoDBTable(DynamoDbClient ddb, String tableName)
{

        DeleteTableRequest request = DeleteTableRequest.builder()
            .tableName(tableName)
            .build();

        try {
            ddb.deleteTable(request);

        } catch (DynamoDbException e) {
            System.err.println(e.getMessage());
```

```
            System.exit(1);
        }
        System.out.println(tableName + " was successfully deleted!");
    }

    private static ExecuteStatementResponse
 executeStatementRequest(DynamoDbClient ddb, String statement,

 List<AttributeValue> parameters) {
        ExecuteStatementRequest request = ExecuteStatementRequest.builder()
            .statement(statement)
            .parameters(parameters)
            .build();

        return ddb.executeStatement(request);
    }

    private static void processResults(ExecuteStatementResponse
 executeStatementResult) {
        System.out.println("ExecuteStatement successful: " +
 executeStatementResult.toString());
    }
}
```

- Einzelheiten zur API finden Sie ExecuteStatementin der AWS SDK for Java 2.x API-Referenz.

JavaScript

SDK für JavaScript (v3)

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

Führen Sie einzelne PartiQL-Anweisungen aus.

```
import {
  BillingMode,
```

```
  CreateTableCommand,
  DeleteTableCommand,
  DescribeTableCommand,
  DynamoDBClient,
  waitUntilTableExists,
} from "@aws-sdk/client-dynamodb";
import {
  DynamoDBDocumentClient,
  ExecuteStatementCommand,
} from "@aws-sdk/lib-dynamodb";
import { ScenarioInput } from "@aws-doc-sdk-examples/lib/scenario";

const client = new DynamoDBClient({});
const docClient = DynamoDBDocumentClient.from(client);

const log = (msg) => console.log(`[SCENARIO] ${msg}`);
const tableName = "SingleOriginCoffees";

export const main = async (confirmAll = false) => {
  /**
   * Delete table if it exists.
   */
  try {
    await client.send(new DescribeTableCommand({ TableName: tableName }));
    // If no error was thrown, the table exists.
    const input = new ScenarioInput(
      "deleteTable",
      `A table named ${tableName} already exists. If you choose not to delete
this table, the scenario cannot continue. Delete it?`,
      { type: "confirm", confirmAll },
    );
    const deleteTable = await input.handle({});
    if (deleteTable) {
      await client.send(new DeleteTableCommand({ tableName }));
    } else {
      console.warn(
        "Scenario could not run. Either delete ${tableName} or provide a unique
 table name.",
      );
      return;
    }
  } catch (caught) {
    if (
      caught instanceof Error &&
```

```
      caught.name === "ResourceNotFoundException"
    ) {
      // Do nothing. This means the table is not there.
    } else {
      throw caught;
    }
  }

  /**
   * Create a table.
   */

  log("Creating a table.");
  const createTableCommand = new CreateTableCommand({
    TableName: tableName,
    // This example performs a large write to the database.
    // Set the billing mode to PAY_PER_REQUEST to
    // avoid throttling the large write.
    BillingMode: BillingMode.PAY_PER_REQUEST,
    // Define the attributes that are necessary for the key schema.
    AttributeDefinitions: [
      {
        AttributeName: "varietal",
        // 'S' is a data type descriptor that represents a number type.
        // For a list of all data type descriptors, see the following link.
        // https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/
Programming.LowLevelAPI.html#Programming.LowLevelAPI.DataTypeDescriptors
        AttributeType: "S",
      },
    ],
    // The KeySchema defines the primary key. The primary key can be
    // a partition key, or a combination of a partition key and a sort key.
    // Key schema design is important. For more info, see
    // https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/best-
practices.html
    KeySchema: [{ AttributeName: "varietal", KeyType: "HASH" }],
  });
  await client.send(createTableCommand);
  log(`Table created: ${tableName}.`);

  /**
   * Wait until the table is active.
   */
```

```
  // This polls with DescribeTableCommand until the requested table is 'ACTIVE'.
  // You can't write to a table before it's active.
  log("Waiting for the table to be active.");
  await waitUntilTableExists({ client }, { TableName: tableName });
  log("Table active.");

  /**
   * Insert an item.
   */

  log("Inserting a coffee into the table.");
  const addItemStatementCommand = new ExecuteStatementCommand({
    // https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/ql-
reference.insert.html
    Statement: `INSERT INTO ${tableName} value {'varietal':?, 'profile':?}`,
    Parameters: ["arabica", ["chocolate", "floral"]],
  });
  await client.send(addItemStatementCommand);
  log("Coffee inserted.");

  /**
   * Select an item.
   */

  log("Selecting the coffee from the table.");
  const selectItemStatementCommand = new ExecuteStatementCommand({
    // https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/ql-
reference.select.html
    Statement: `SELECT * FROM ${tableName} WHERE varietal=?`,
    Parameters: ["arabica"],
  });
  const selectItemResponse = await docClient.send(selectItemStatementCommand);
  log(`Got coffee: ${JSON.stringify(selectItemResponse.Items[0])}`);

  /**
   * Update the item.
   */

  log("Add a flavor profile to the coffee.");
  const updateItemStatementCommand = new ExecuteStatementCommand({
    // https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/ql-
reference.update.html
    Statement: `UPDATE ${tableName} SET profile=list_append(profile, ?) WHERE
 varietal=?`,
```

```
    Parameters: [["fruity"], "arabica"],
  });
  await client.send(updateItemStatementCommand);
  log("Updated coffee");

  /**
   * Delete the item.
   */

  log("Deleting the coffee.");
  const deleteItemStatementCommand = new ExecuteStatementCommand({
    // https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/ql-
reference.delete.html
    Statement: `DELETE FROM ${tableName} WHERE varietal=?`,
    Parameters: ["arabica"],
  });
  await docClient.send(deleteItemStatementCommand);
  log("Coffee deleted.");

  /**
   * Delete the table.
   */

  log("Deleting the table.");
  const deleteTableCommand = new DeleteTableCommand({ TableName: tableName });
  await client.send(deleteTableCommand);
  log("Table deleted.");
};
```

- Einzelheiten zur API finden Sie ExecuteStatementin der AWS SDK für JavaScript API-Referenz.

Kotlin

SDK für Kotlin

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
suspend fun main() {
    val ddb = DynamoDbClient.fromEnvironment { region = "us-east-1" }
    val tableName = "MoviesPartiQ"
    val fileName = "../../../resources/sample_files/movies.json"
    println("Creating an Amazon DynamoDB table named MoviesPartiQ with a key
 named id and a sort key named title.")
    createTablePartiQL(ddb, tableName, "year")
    loadDataPartiQL(ddb, fileName)

    println("******* Getting data from the MoviesPartiQ table.")
    getMoviePartiQL(ddb)

    println("******* Putting a record into the MoviesPartiQ table.")
    putRecordPartiQL(ddb)

    println("******* Updating a record.")
    updateTableItemPartiQL(ddb)

    println("******* Querying the movies released in 2013.")
    queryTablePartiQL(ddb)

    println("******* Deleting the MoviesPartiQ table.")
    deleteTablePartiQL(tableName)
}

suspend fun createTablePartiQL(
    ddb: DynamoDbClient,
    tableNameVal: String,
    key: String,
) {
    val attDef =
        AttributeDefinition {
            attributeName = key
            attributeType = ScalarAttributeType.N
        }

    val attDef1 =
        AttributeDefinition {
            attributeName = "title"
            attributeType = ScalarAttributeType.S
        }

    val keySchemaVal =
```

```
        KeySchemaElement {
            attributeName = key
            keyType = KeyType.Hash
        }

    val keySchemaVal1 =
        KeySchemaElement {
            attributeName = "title"
            keyType = KeyType.Range
        }

    val request =
        CreateTableRequest {
            attributeDefinitions = listOf(attDef, attDef1)
            keySchema = listOf(keySchemaVal, keySchemaVal1)
            billingMode = BillingMode.PayPerRequest
            tableName = tableNameVal
        }

    val response = ddb.createTable(request)
    ddb.waitUntilTableExists {
        // suspend call
        tableName = tableNameVal
    }
    println("The table was successfully created
 ${response.tableDescription?.tableArn}")
}

suspend fun loadDataPartiQL(
    ddb: DynamoDbClient,
    fileName: String,
) {
    val sqlStatement = "INSERT INTO MoviesPartiQ VALUE {'year':?, 'title' : ?,
 'info' : ?}"
    val parser = JsonFactory().createParser(File(fileName))
    val rootNode = ObjectMapper().readTree<JsonNode>(parser)
    val iter: Iterator<JsonNode> = rootNode.iterator()
    var currentNode: ObjectNode
    var t = 0

    while (iter.hasNext()) {
        if (t == 200) {
            break
        }
```

```
        currentNode = iter.next() as ObjectNode
        val year = currentNode.path("year").asInt()
        val title = currentNode.path("title").asText()
        val info = currentNode.path("info").toString()

        val parameters: MutableList<AttributeValue> = ArrayList<AttributeValue>()
        parameters.add(AttributeValue.N(year.toString()))
        parameters.add(AttributeValue.S(title))
        parameters.add(AttributeValue.S(info))

        executeStatementPartiQL(ddb, sqlStatement, parameters)
        println("Added Movie $title")
        parameters.clear()
        t++
    }
}

suspend fun getMoviePartiQL(ddb: DynamoDbClient) {
    val sqlStatement = "SELECT * FROM MoviesPartiQ where year=? and title=?"
    val parameters: MutableList<AttributeValue> = ArrayList<AttributeValue>()
    parameters.add(AttributeValue.N("2012"))
    parameters.add(AttributeValue.S("The Perks of Being a Wallflower"))
    val response = executeStatementPartiQL(ddb, sqlStatement, parameters)
    println("ExecuteStatement successful: $response")
}

suspend fun putRecordPartiQL(ddb: DynamoDbClient) {
    val sqlStatement = "INSERT INTO MoviesPartiQ VALUE {'year':?, 'title' : ?,
 'info' : ?}"
    val parameters: MutableList<AttributeValue> = java.util.ArrayList()
    parameters.add(AttributeValue.N("2020"))
    parameters.add(AttributeValue.S("My Movie"))
    parameters.add(AttributeValue.S("No Info"))
    executeStatementPartiQL(ddb, sqlStatement, parameters)
    println("Added new movie.")
}

suspend fun updateTableItemPartiQL(ddb: DynamoDbClient) {
    val sqlStatement = "UPDATE MoviesPartiQ SET info = 'directors\":[\"Merian C.
 Cooper\",\"Ernest B. Schoedsack\' where year=? and title=?"
    val parameters: MutableList<AttributeValue> = java.util.ArrayList()
    parameters.add(AttributeValue.N("2013"))
    parameters.add(AttributeValue.S("The East"))
```

```
    executeStatementPartiQL(ddb, sqlStatement, parameters)
    println("Item was updated!")
}

// Query the table where the year is 2013.
suspend fun queryTablePartiQL(ddb: DynamoDbClient) {
    val sqlStatement = "SELECT * FROM MoviesPartiQ where year = ?"
    val parameters: MutableList<AttributeValue> = java.util.ArrayList()
    parameters.add(AttributeValue.N("2013"))
    val response = executeStatementPartiQL(ddb, sqlStatement, parameters)
    println("ExecuteStatement successful: $response")
}

suspend fun deleteTablePartiQL(tableNameVal: String) {
    val request =
        DeleteTableRequest {
            tableName = tableNameVal
        }

    DynamoDbClient { region = "us-east-1" }.use { ddb ->
        ddb.deleteTable(request)
        println("$tableNameVal was deleted")
    }
}

suspend fun executeStatementPartiQL(
    ddb: DynamoDbClient,
    statementVal: String,
    parametersVal: List<AttributeValue>,
): ExecuteStatementResponse {
    val request =
        ExecuteStatementRequest {
            statement = statementVal
            parameters = parametersVal
        }

    return ddb.executeStatement(request)
}
```

- Einzelheiten zur API finden Sie ExecuteStatementin der API-Referenz zum AWS SDK für Kotlin.

PHP

SDK für PHP

> **Note**
>
> Es gibt noch mehr dazu. GitHub Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
namespace DynamoDb\PartiQL_Basics;

use Aws\DynamoDb\Marshaler;
use DynamoDb;
use DynamoDb\DynamoDBAttribute;

use function AwsUtilities\testable_readline;
use function AwsUtilities\loadMovieData;

class GettingStartedWithPartiQL
{
    public function run()
    {
        echo("\n");
        echo("--------------------------------------\n");
        print("Welcome to the Amazon DynamoDB - PartiQL getting started demo
 using PHP!\n");
        echo("--------------------------------------\n");

        $uuid = uniqid();
        $service = new DynamoDb\DynamoDBService();

        $tableName = "partiql_demo_table_$uuid";
        $service->createTable(
            $tableName,
            [
                new DynamoDBAttribute('year', 'N', 'HASH'),
                new DynamoDBAttribute('title', 'S', 'RANGE')
            ]
        );

        echo "Waiting for table...";
```

```
        $service->dynamoDbClient->waitUntil("TableExists", ['TableName' =>
 $tableName]);
        echo "table $tableName found!\n";

        echo "What's the name of the last movie you watched?\n";
        while (empty($movieName)) {
            $movieName = testable_readline("Movie name: ");
        }
        echo "And what year was it released?\n";
        $movieYear = "year";
        while (!is_numeric($movieYear) || intval($movieYear) != $movieYear) {
            $movieYear = testable_readline("Year released: ");
        }
        $key = [
            'Item' => [
                'year' => [
                    'N' => "$movieYear",
                ],
                'title' => [
                    'S' => $movieName,
                ],
            ],
        ];
        list($statement, $parameters) = $service-
>buildStatementAndParameters("INSERT", $tableName, $key);
        $service->insertItemByPartiQL($statement, $parameters);

        echo "How would you rate the movie from 1-10?\n";
        $rating = 0;
        while (!is_numeric($rating) || intval($rating) != $rating || $rating < 1
 || $rating > 10) {
            $rating = testable_readline("Rating (1-10): ");
        }
        echo "What was the movie about?\n";
        while (empty($plot)) {
            $plot = testable_readline("Plot summary: ");
        }
        $attributes = [
            new DynamoDBAttribute('rating', 'N', 'HASH', $rating),
            new DynamoDBAttribute('plot', 'S', 'RANGE', $plot),
        ];

        list($statement, $parameters) = $service-
>buildStatementAndParameters("UPDATE", $tableName, $key, $attributes);
```

```
        $service->updateItemByPartiQL($statement, $parameters);
        echo "Movie added and updated.\n";



        $batch = json_decode(loadMovieData());

        $service->writeBatch($tableName, $batch);

        $movie = $service->getItemByPartiQL($tableName, $key);
        echo "\nThe movie {$movie['Items'][0]['title']['S']} was released in
 {$movie['Items'][0]['year']['N']}.\n";
        echo "What rating would you like to give {$movie['Items'][0]['title']
['S']}?\n";
        $rating = 0;
        while (!is_numeric($rating) || intval($rating) != $rating || $rating < 1
 || $rating > 10) {
            $rating = testable_readline("Rating (1-10): ");
        }
        $attributes = [
            new DynamoDBAttribute('rating', 'N', 'HASH', $rating),
            new DynamoDBAttribute('plot', 'S', 'RANGE', $plot)
        ];
        list($statement, $parameters) = $service-
>buildStatementAndParameters("UPDATE", $tableName, $key, $attributes);
        $service->updateItemByPartiQL($statement, $parameters);

        $movie = $service->getItemByPartiQL($tableName, $key);
        echo "Okay, you have rated {$movie['Items'][0]['title']['S']} as a
 {$movie['Items'][0]['rating']['N']}\n";

        $service->deleteItemByPartiQL($statement, $parameters);
        echo "But, bad news, this was a trap. That movie has now been deleted
 because of your rating...harsh.\n";

        echo "That's okay though. The book was better. Now, for something
 lighter, in what year were you born?\n";
        $birthYear = "not a number";
        while (!is_numeric($birthYear) || $birthYear >= date("Y")) {
            $birthYear = testable_readline("Birth year: ");
        }
        $birthKey = [
            'Key' => [
                'year' => [
```

```
                    'N' => "$birthYear",
                ],
            ],
        ];
        $result = $service->query($tableName, $birthKey);
        $marshal = new Marshaler();
        echo "Here are the movies in our collection released the year you were
 born:\n";
        $oops = "Oops! There were no movies released in that year (that we know
 of).\n";
        $display = "";
        foreach ($result['Items'] as $movie) {
            $movie = $marshal->unmarshalItem($movie);
            $display .= $movie['title'] . "\n";
        }
        echo ($display) ?: $oops;

        $yearsKey = [
            'Key' => [
                'year' => [
                    'N' => [
                        'minRange' => 1990,
                        'maxRange' => 1999,
                    ],
                ],
            ],
        ];
        $filter = "year between 1990 and 1999";
        echo "\nHere's a list of all the movies released in the 90s:\n";
        $result = $service->scan($tableName, $yearsKey, $filter);
        foreach ($result['Items'] as $movie) {
            $movie = $marshal->unmarshalItem($movie);
            echo $movie['title'] . "\n";
        }

        echo "\nCleaning up this demo by deleting table $tableName...\n";
        $service->deleteTable($tableName);
    }
}

    public function insertItemByPartiQL(string $statement, array $parameters)
    {
        $this->dynamoDbClient->executeStatement([
            'Statement' => "$statement",
```

```
            'Parameters' => $parameters,
        ]);
    }

    public function getItemByPartiQL(string $tableName, array $key): Result
    {
        list($statement, $parameters) = $this-
>buildStatementAndParameters("SELECT", $tableName, $key['Item']);

        return $this->dynamoDbClient->executeStatement([
            'Parameters' => $parameters,
            'Statement' => $statement,
        ]);
    }

    public function updateItemByPartiQL(string $statement, array $parameters)
    {
        $this->dynamoDbClient->executeStatement([
            'Statement' => $statement,
            'Parameters' => $parameters,
        ]);
    }

    public function deleteItemByPartiQL(string $statement, array $parameters)
    {
        $this->dynamoDbClient->executeStatement([
            'Statement' => $statement,
            'Parameters' => $parameters,
        ]);
    }
```

- Einzelheiten zur API finden Sie ExecuteStatementin der AWS SDK für PHP API-Referenz.

Python

SDK für Python (Boto3)

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

Erstellen Sie eine Klasse, die PartiQL-Anweisungen ausführen kann.

```
from datetime import datetime
from decimal import Decimal
import logging
from pprint import pprint

import boto3
from botocore.exceptions import ClientError

from scaffold import Scaffold

logger = logging.getLogger(__name__)

class PartiQLWrapper:
    """
    Encapsulates a DynamoDB resource to run PartiQL statements.
    """

    def __init__(self, dyn_resource):
        """
        :param dyn_resource: A Boto3 DynamoDB resource.
        """
        self.dyn_resource = dyn_resource


    def run_partiql(self, statement, params):
        """
        Runs a PartiQL statement. A Boto3 resource is used even though
        `execute_statement` is called on the underlying `client` object because
 the
        resource transforms input and output from plain old Python objects
 (POPOs) to
        the DynamoDB format. If you create the client directly, you must do these
        transforms yourself.

        :param statement: The PartiQL statement.
        :param params: The list of PartiQL parameters. These are applied to the
                       statement in the order they are listed.
        :return: The items returned from the statement, if any.
        """
        try:
            output = self.dyn_resource.meta.client.execute_statement(
                Statement=statement, Parameters=params
```

```
            )
        except ClientError as err:
            if err.response["Error"]["Code"] == "ResourceNotFoundException":
                logger.error(
                    "Couldn't execute PartiQL '%s' because the table does not
 exist.",
                    statement,
                )
            else:
                logger.error(
                    "Couldn't execute PartiQL '%s'. Here's why: %s: %s",
                    statement,
                    err.response["Error"]["Code"],
                    err.response["Error"]["Message"],
                )
            raise
        else:
            return output
```

Führen Sie ein Szenario aus, das eine Tabelle erstellt und PartiQL-Abfragen ausführt.

```
def run_scenario(scaffold, wrapper, table_name):
    logging.basicConfig(level=logging.INFO, format="%(levelname)s: %(message)s")

    print("-" * 88)
    print("Welcome to the Amazon DynamoDB PartiQL single statement demo.")
    print("-" * 88)

    print(f"Creating table '{table_name}' for the demo...")
    scaffold.create_table(table_name)
    print("-" * 88)

    title = "24 Hour PartiQL People"
    year = datetime.now().year
    plot = "A group of data developers discover a new query language they can't
 stop using."
    rating = Decimal("9.9")

    print(f"Inserting movie '{title}' released in {year}.")
    wrapper.run_partiql(
```

```
        f"INSERT INTO \"{table_name}\" VALUE {{'title': ?, 'year': ?,
 'info': ?}}",
        [title, year, {"plot": plot, "rating": rating}],
    )
    print("Success!")
    print("-" * 88)

    print(f"Getting data for movie '{title}' released in {year}.")
    output = wrapper.run_partiql(
        f'SELECT * FROM "{table_name}" WHERE title=? AND year=?', [title, year]
    )
    for item in output["Items"]:
        print(f"\n{item['title']}, {item['year']}")
        pprint(output["Items"])
    print("-" * 88)

    rating = Decimal("2.4")
    print(f"Updating movie '{title}' with a rating of {float(rating)}.")
    wrapper.run_partiql(
        f'UPDATE "{table_name}" SET info.rating=? WHERE title=? AND year=?',
        [rating, title, year],
    )
    print("Success!")
    print("-" * 88)

    print(f"Getting data again to verify our update.")
    output = wrapper.run_partiql(
        f'SELECT * FROM "{table_name}" WHERE title=? AND year=?', [title, year]
    )
    for item in output["Items"]:
        print(f"\n{item['title']}, {item['year']}")
        pprint(output["Items"])
    print("-" * 88)

    print(f"Deleting movie '{title}' released in {year}.")
    wrapper.run_partiql(
        f'DELETE FROM "{table_name}" WHERE title=? AND year=?', [title, year]
    )
    print("Success!")
    print("-" * 88)

    print(f"Deleting table '{table_name}'...")
    scaffold.delete_table()
    print("-" * 88)
```

```
    print("\nThanks for watching!")
    print("-" * 88)


if __name__ == "__main__":
    try:
        dyn_res = boto3.resource("dynamodb")
        scaffold = Scaffold(dyn_res)
        movies = PartiQLWrapper(dyn_res)
        run_scenario(scaffold, movies, "doc-example-table-partiql-movies")
    except Exception as e:
        print(f"Something went wrong with the demo! Here's what: {e}")
```

- Einzelheiten zur API finden Sie ExecuteStatementin AWS SDK for Python (Boto3) API Reference.

Ruby

SDK für Ruby

> **Note**
>
> Es gibt noch mehr dazu. GitHub Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

Führen Sie ein Szenario aus, das eine Tabelle erstellt und PartiQL-Abfragen ausführt.

```
  table_name = "doc-example-table-movies-partiql-#{rand(10**8)}"
  scaffold = Scaffold.new(table_name)
  sdk = DynamoDBPartiQLSingle.new(table_name)

  new_step(1, 'Create a new DynamoDB table if none already exists.')
  unless scaffold.exists?(table_name)
    puts("\nNo such table: #{table_name}. Creating it...")
    scaffold.create_table(table_name)
    print "Done!\n".green
  end
```

```
  new_step(2, 'Populate DynamoDB table with movie data.')
  download_file = 'moviedata.json'
  puts("Downloading movie database to #{download_file}...")
  movie_data = scaffold.fetch_movie_data(download_file)
  puts("Writing movie data from #{download_file} into your table...")
  scaffold.write_batch(movie_data)
  puts("Records added: #{movie_data.length}.")
  print "Done!\n".green

  new_step(3, 'Select a single item from the movies table.')
  response = sdk.select_item_by_title('Star Wars')
  puts("Items selected for title 'Star Wars': #{response.items.length}\n")
  print response.items.first.to_s.yellow
  print "\n\nDone!\n".green

  new_step(4, 'Update a single item from the movies table.')
  puts "Let's correct the rating on The Big Lebowski to 10.0."
  sdk.update_rating_by_title('The Big Lebowski', 1998, 10.0)
  print "\nDone!\n".green

  new_step(5, 'Delete a single item from the movies table.')
  puts "Let's delete The Silence of the Lambs because it's just too scary."
  sdk.delete_item_by_title('The Silence of the Lambs', 1991)
  print "\nDone!\n".green

  new_step(6, 'Insert a new item into the movies table.')
  puts "Let's create a less-scary movie called The Prancing of the Lambs."
  sdk.insert_item('The Prancing of the Lambs', 2005, 'A movie about happy
 livestock.', 5.0)
  print "\nDone!\n".green

  new_step(7, 'Delete the table.')
  return unless scaffold.exists?(table_name)

  scaffold.delete_table
end
```

- Einzelheiten zur API finden Sie ExecuteStatementin der AWS SDK für Ruby API-Referenz.

Rust

SDK für Rust

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
async fn make_table(
    client: &Client,
    table: &str,
    key: &str,
) -> Result<(), SdkError<CreateTableError>> {
    let ad = AttributeDefinition::builder()
        .attribute_name(key)
        .attribute_type(ScalarAttributeType::S)
        .build()
        .expect("creating AttributeDefinition");

    let ks = KeySchemaElement::builder()
        .attribute_name(key)
        .key_type(KeyType::Hash)
        .build()
        .expect("creating KeySchemaElement");

    match client
        .create_table()
        .table_name(table)
        .key_schema(ks)
        .attribute_definitions(ad)
        .billing_mode(BillingMode::PayPerRequest)
        .send()
        .await
    {
        Ok(_) => Ok(()),
        Err(e) => Err(e),
    }
}
```

```
async fn add_item(client: &Client, item: Item) -> Result<(),
 SdkError<ExecuteStatementError>> {
    match client
        .execute_statement()
        .statement(format!(
            r#"INSERT INTO "{}" VALUE {{
                "{}": ?,
                "acount_type": ?,
                "age": ?,
                "first_name": ?,
                "last_name": ?
        }} "#,
            item.table, item.key
        ))
        .set_parameters(Some(vec![
            AttributeValue::S(item.utype),
            AttributeValue::S(item.age),
            AttributeValue::S(item.first_name),
            AttributeValue::S(item.last_name),
        ]))
        .send()
        .await
    {
        Ok(_) => Ok(()),
        Err(e) => Err(e),
    }
}

async fn query_item(client: &Client, item: Item) -> bool {
    match client
        .execute_statement()
        .statement(format!(
            r#"SELECT * FROM "{}" WHERE "{}" = ?"#,
            item.table, item.key
        ))
        .set_parameters(Some(vec![AttributeValue::S(item.value)]))
        .send()
        .await
    {
        Ok(resp) => {
            if !resp.items().is_empty() {
                println!("Found a matching entry in the table:");
                println!("{:?}", resp.items.unwrap_or_default().pop());
                true
```

```
            } else {
                println!("Did not find a match.");
                false
            }
        }
        Err(e) => {
            println!("Got an error querying table:");
            println!("{}", e);
            process::exit(1);
        }
    }
}

async fn remove_item(client: &Client, table: &str, key: &str, value: String) ->
 Result<(), Error> {
    client
        .execute_statement()
        .statement(format!(r#"DELETE FROM "{table}" WHERE "{key}" = ?"#))
        .set_parameters(Some(vec![AttributeValue::S(value)]))
        .send()
        .await?;

    println!("Deleted item.");

    Ok(())
}

async fn remove_table(client: &Client, table: &str) -> Result<(), Error> {
    client.delete_table().table_name(table).send().await?;

    Ok(())
}
```

- Einzelheiten zur API finden Sie ExecuteStatementin der API-Referenz zum AWS SDK für Rust.

Eine vollständige Liste der AWS SDK-Entwicklerhandbücher und Codebeispiele finden Sie unterDynamoDB mit einem SDK verwenden AWS. Dieses Thema enthält auch Informationen zu den ersten Schritten und Details zu früheren SDK-Versionen.

Abfragen einer DynamoDB-Tabelle mithilfe eines globalen sekundären Indexes mit einem SDK AWS

In den folgenden Codebeispielen wird gezeigt, wie eine Tabelle mit einem globalen sekundären Index abgefragt wird.

- Abfragen einer DynamoDB-Tabelle mit ihrem Primärschlüssel
- Abfragen eines globalen sekundären Index (GSI) für alternative Zugriffsmuster
- Vergleichen von Tabellenabfragen und GSI-Abfragen

Java

SDK für Java 2.x

Fragen Sie eine DynamoDB-Tabelle mit ihrem Primärschlüssel und einem Global Secondary Index (GSI) mit ab. AWS SDK for Java 2.x

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.AttributeValue;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.model.QueryRequest;
import software.amazon.awssdk.services.dynamodb.model.QueryResponse;
import software.amazon.awssdk.services.dynamodb.model.ResourceNotFoundException;

import java.util.HashMap;
import java.util.Map;

    public QueryResponse queryTable(
        final String tableName, final String partitionKeyName, final String
 partitionKeyValue) {

        CodeSampleUtils.validateTableParameters(tableName, partitionKeyName,
 partitionKeyValue);

        // Create expression attribute names for the column names
        final Map<String, String> expressionAttributeNames = new HashMap<>();
        expressionAttributeNames.put(EXPRESSION_ATTRIBUTE_NAME_PK,
 partitionKeyName);

        // Create expression attribute values for the column values
```

```
        final Map<String, AttributeValue> expressionAttributeValues = new
HashMap<>();
        expressionAttributeValues.put(
            EXPRESSION_ATTRIBUTE_VALUE_PK,
            AttributeValue.builder().s(partitionKeyValue).build());

        // Create the query request
        final QueryRequest queryRequest = QueryRequest.builder()
            .tableName(tableName)
            .keyConditionExpression(KEY_CONDITION_EXPRESSION)
            .expressionAttributeNames(expressionAttributeNames)
            .expressionAttributeValues(expressionAttributeValues)
            .build();

        try {
            final QueryResponse response = dynamoDbClient.query(queryRequest);
            System.out.println("Query on base table successful. Found " +
response.count() + " items");
            return response;
        } catch (ResourceNotFoundException e) {
            System.err.format("Error: The Amazon DynamoDB table \"%s\" can't be
found.\n", tableName);
            throw new DynamoDbQueryException("Table not found: " + tableName, e);
        } catch (DynamoDbException e) {
            System.err.println("Error querying base table: " + e.getMessage());
            throw new DynamoDbQueryException("Failed to execute query on base
table", e);
        }
    }

    /**
     * Queries a DynamoDB Global Secondary Index (GSI) by partition key.
     *
     * @param tableName         The name of the DynamoDB table
     * @param indexName         The name of the GSI
     * @param partitionKeyName  The name of the GSI partition key attribute
     * @param partitionKeyValue The value of the GSI partition key to query
     * @return The query response from DynamoDB
     * @throws ResourceNotFoundException if the table or index doesn't exist
     * @throws DynamoDbException if the query fails
     */
    public QueryResponse queryGlobalSecondaryIndex(
        final String tableName, final String indexName, final String
partitionKeyName, final String partitionKeyValue) {
```

```
        CodeSampleUtils.validateTableParameters(tableName, partitionKeyName,
partitionKeyValue);
        CodeSampleUtils.validateStringParameter("Index name", indexName);

        // Create expression attribute names for the column names
        final Map<String, String> expressionAttributeNames = new HashMap<>();
        expressionAttributeNames.put(EXPRESSION_ATTRIBUTE_NAME_IK,
partitionKeyName);

        // Create expression attribute values for the column values
        final Map<String, AttributeValue> expressionAttributeValues = new
HashMap<>();
        expressionAttributeValues.put(
            EXPRESSION_ATTRIBUTE_VALUE_IK,
            AttributeValue.builder().s(partitionKeyValue).build());

        // Create the query request
        final QueryRequest queryRequest = QueryRequest.builder()
            .tableName(tableName)
            .indexName(indexName)
            .keyConditionExpression(GSI_KEY_CONDITION_EXPRESSION)
            .expressionAttributeNames(expressionAttributeNames)
            .expressionAttributeValues(expressionAttributeValues)
            .build();

        try {
            final QueryResponse response = dynamoDbClient.query(queryRequest);
            System.out.println("Query on GSI successful. Found " +
response.count() + " items");
            return response;
        } catch (ResourceNotFoundException e) {
            System.err.format(
                "Error: The Amazon DynamoDB table \"%s\" or index \"%s\" can't be
found.\n", tableName, indexName);
            throw new DynamoDbQueryException("Table or index not found: " +
tableName + "/" + indexName, e);
        } catch (DynamoDbException e) {
            System.err.println("Error querying GSI: " + e.getMessage());
            throw new DynamoDbQueryException("Failed to execute query on GSI",
e);
        }
    }
```

Vergleichen Sie die direkte Abfrage einer Tabelle mit der Abfrage einer GSI mit. AWS SDK for Java 2.x

```
   public static void main(String[] args) {
       final String usage =
           """
               Usage:
                   <tableName> <basePartitionKeyName> <basePartitionKeyValue>
<gsiName> <gsiPartitionKeyName> <gsiPartitionKeyValue> [region]
               Where:
                   tableName - The Amazon DynamoDB table to query.
                   basePartitionKeyName - The name of the base table partition
key attribute.
                   basePartitionKeyValue - The value of the base table partition
key to query.
                   gsiName - The name of the Global Secondary Index.
                   gsiPartitionKeyName - The name of the GSI partition key
attribute.
                   gsiPartitionKeyValue - The value of the GSI partition key to
query.
                   region (optional) - The AWS region where the table exists.
(Default: us-east-1)
               """;

       if (args.length < 6) {
           System.out.println(usage);
           System.exit(1);
       }

       final String tableName = args[0];
       final String basePartitionKeyName = args[1];
       final String basePartitionKeyValue = args[2];
       final String gsiName = args[3];
       final String gsiPartitionKeyName = args[4];
       final String gsiPartitionKeyValue = args[5];
       final Region region = args.length > 6 ? Region.of(args[6]) :
Region.US_EAST_1;

       try (DynamoDbClient ddb =
DynamoDbClient.builder().region(region).build()) {
           final QueryTableAndGSI queryHelper = new QueryTableAndGSI(ddb);
```

```
            // Query the base table
            System.out.println("Querying base table where " +
basePartitionKeyName + " = " + basePartitionKeyValue);
            final QueryResponse tableResponse =
                queryHelper.queryTable(tableName, basePartitionKeyName,
basePartitionKeyValue);

            System.out.println("Found " + tableResponse.count() + " items in base
table:");
            tableResponse.items().forEach(item -> System.out.println(item));

            // Query the GSI
            System.out.println(
                "\nQuerying GSI '" + gsiName + "' where " + gsiPartitionKeyName +
" = " + gsiPartitionKeyValue);
            final QueryResponse gsiResponse =
                queryHelper.queryGlobalSecondaryIndex(tableName, gsiName,
gsiPartitionKeyName, gsiPartitionKeyValue);

            System.out.println("Found " + gsiResponse.count() + " items in
GSI:");
            gsiResponse.items().forEach(item -> System.out.println(item));

            // Explain the differences between querying a table and a GSI
            System.out.println("\nKey differences between querying a table and a
GSI:");
            System.out.println("1. When querying a GSI, you must specify the
indexName parameter");
            System.out.println("2. GSIs may not contain all attributes from the
base table (projection)");
            System.out.println("3. GSIs consume read capacity units from the
GSI's capacity, not the base table's");
            System.out.println("4. GSIs may have eventually consistent data
(cannot use ConsistentRead=true)");

        } catch (IllegalArgumentException e) {
            System.err.println("Invalid input: " + e.getMessage());
            System.exit(1);
        } catch (ResourceNotFoundException e) {
            System.err.println("Table or index not found: " + e.getMessage());
            System.exit(1);
        } catch (DynamoDbException e) {
            System.err.println("DynamoDB error: " + e.getMessage());
```

```
            System.exit(1);
        } catch (Exception e) {
            System.err.println("Unexpected error: " + e.getMessage());
            System.exit(1);
        }
    }
```

- Weitere API-Informationen finden Sie unter Abfragen in der AWS SDK for Java 2.x -API-Referenz.

JavaScript

SDK für JavaScript (v3)

Fragen Sie eine DynamoDB-Tabelle mithilfe des Primärschlüssels mit ab. AWS SDK für JavaScript

```
const { DynamoDBClient, QueryCommand } = require("@aws-sdk/client-dynamodb");

/**
 * Queries a DynamoDB table using the primary key
 *
 * @param {Object} config - AWS SDK configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {string} userId - The user ID to query by (partition key)
 * @returns {Promise<Object>} - The query response
 */
async function queryTable(
  config,
  tableName,
  userId
) {
  try {
    // Create DynamoDB client
    const client = new DynamoDBClient(config);

    // Construct the query input for the base table
    const input = {
      TableName: tableName,
      KeyConditionExpression: "user_id = :userId",
      ExpressionAttributeValues: {
```

```
        ":userId": { S: userId }
      }
    };

    // Execute the query
    const command = new QueryCommand(input);
    return await client.send(command);
  } catch (error) {
    console.error(`Error querying table: ${error}`);
    throw error;
  }
}
```

Fragen Sie einen DynamoDB Global Secondary Index (GSI) mit ab. AWS SDK für JavaScript

```
/**
 * Queries a DynamoDB Global Secondary Index (GSI)
 *
 * @param {Object} config - AWS SDK configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {string} indexName - The name of the GSI to query
 * @param {string} gameId - The game ID to query by (GSI partition key)
 * @returns {Promise<Object>} - The query response
 */
async function queryGSI(
  config,
  tableName,
  indexName,
  gameId
) {
  try {
    // Create DynamoDB client
    const client = new DynamoDBClient(config);

    // Construct the query input for the GSI
    const input = {
      TableName: tableName,
      IndexName: indexName,
      KeyConditionExpression: "game_id = :gameId",
      ExpressionAttributeValues: {
        ":gameId": { S: gameId }
      }
```

```
    };

    // Execute the query
    const command = new QueryCommand(input);
    return await client.send(command);
  } catch (error) {
    console.error(`Error querying GSI: ${error}`);
    throw error;
  }
}
```

- Weitere API-Informationen finden Sie unter Abfragen in der AWS SDK für JavaScript -API-Referenz.

Python

SDK für Python (Boto3)

Fragen Sie eine DynamoDB-Tabelle mit ihrem Primärschlüssel und einem Global Secondary Index (GSI) mit ab. AWS SDK für Python (Boto3)

```
import boto3
from boto3.dynamodb.conditions import Key


def query_table(table_name, partition_key_name, partition_key_value):
    """
    Query a DynamoDB table using its primary key.

    Args:
        table_name (str): The name of the DynamoDB table.
        partition_key_name (str): The name of the partition key attribute.
        partition_key_value (str): The value of the partition key to query.

    Returns:
        dict: The response from DynamoDB containing the query results.
    """
    # Initialize the DynamoDB resource
    dynamodb = boto3.resource("dynamodb")
    table = dynamodb.Table(table_name)
```

```
    # Perform the query on the table's primary key
    response =
 table.query(KeyConditionExpression=Key(partition_key_name).eq(partition_key_value))

    return response


def query_gsi(table_name, index_name, partition_key_name, partition_key_value):
    """
    Query a Global Secondary Index (GSI) on a DynamoDB table.

    Args:
        table_name (str): The name of the DynamoDB table.
        index_name (str): The name of the Global Secondary Index.
        partition_key_name (str): The name of the GSI's partition key attribute.
        partition_key_value (str): The value of the GSI's partition key to query.

    Returns:
        dict: The response from DynamoDB containing the query results.
    """
    # Initialize the DynamoDB resource
    dynamodb = boto3.resource("dynamodb")
    table = dynamodb.Table(table_name)

    # Perform the query on the GSI
    response = table.query(
        IndexName=index_name,
 KeyConditionExpression=Key(partition_key_name).eq(partition_key_value)
    )

    return response
```

- Weitere API-Informationen finden Sie unter Abfragen in der API-Referenz zum AWS SDK für Python (Boto3).

Eine vollständige Liste der AWS SDK-Entwicklerhandbücher und Codebeispiele finden Sie unter. DynamoDB mit einem SDK verwenden AWS Dieses Thema enthält auch Informationen zu den ersten Schritten und Details zu früheren SDK-Versionen.

Fragen Sie eine DynamoDB-Tabelle mithilfe einer Begins_with-Bedingung mit einem SDK ab AWS

In den folgenden Codebeispielen wird gezeigt, wie eine Tabelle mit einer „begins_with“-Bedingung abgefragt wird.

- Verwenden der Funktion „begins_with“ in einem Schlüsselbedingungsausdruck
- Filtern der Elemente auf Grundlage eines Präfixmusters für den Sortierschlüssel

Java

SDK für Java 2.x

Fragen Sie mit AWS SDK for Java 2.x eine DynamoDB-Tabelle mithilfe einer „begins_with“-Bedingung für den Sortierschlüssel ab.

```
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.AttributeValue;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.model.QueryRequest;
import software.amazon.awssdk.services.dynamodb.model.QueryResponse;
import software.amazon.awssdk.services.dynamodb.model.ResourceNotFoundException;

import java.util.HashMap;
import java.util.Map;
import java.util.logging.Level;
import java.util.logging.Logger;

    public QueryResponse queryWithBeginsWithCondition(
        final String tableName,
        final String partitionKeyName,
        final String partitionKeyValue,
        final String sortKeyName,
        final String sortKeyPrefix) {

        CodeSampleUtils.validateTableParameters(tableName, partitionKeyName,
 partitionKeyValue);
        CodeSampleUtils.validateStringParameter("Sort key name", sortKeyName);
        CodeSampleUtils.validateStringParameter("Sort key prefix",
 sortKeyPrefix);
```

```
        // Create expression attribute names for the column names
        final Map<String, String> expressionAttributeNames = new HashMap<>();
        expressionAttributeNames.put(EXPRESSION_ATTRIBUTE_NAME_PK,
partitionKeyName);
        expressionAttributeNames.put(EXPRESSION_ATTRIBUTE_NAME_SK, sortKeyName);

        // Create expression attribute values for the column values
        final Map<String, AttributeValue> expressionAttributeValues = new
HashMap<>();
        expressionAttributeValues.put(
            EXPRESSION_ATTRIBUTE_VALUE_PK,
            AttributeValue.builder().s(partitionKeyValue).build());
        expressionAttributeValues.put(
            EXPRESSION_ATTRIBUTE_VALUE_SK_PREFIX,
            AttributeValue.builder().s(sortKeyPrefix).build());

        // Create the query request
        final QueryRequest queryRequest = QueryRequest.builder()
            .tableName(tableName)
            .keyConditionExpression(KEY_CONDITION_EXPRESSION)
            .expressionAttributeNames(expressionAttributeNames)
            .expressionAttributeValues(expressionAttributeValues)
            .build();

        try {
            final QueryResponse response = dynamoDbClient.query(queryRequest);
            LOGGER.log(Level.INFO, "Query with begins_with condition successful.
Found {0} items", response.count());
            return response;
        } catch (ResourceNotFoundException e) {
            LOGGER.log(Level.SEVERE, "Table not found: {0}", tableName);
            throw e;
        } catch (DynamoDbException e) {
            LOGGER.log(Level.SEVERE, "Error querying with begins_with condition",
e);
            throw e;
        }
    }
```

Demonstrieren Sie mit AWS SDK for Java 2.x die Verwendung von „begins_with“ mit unterschiedlichen Präfixlängen.

```
   public static void main(String[] args) {
       try {
           CodeSampleUtils.BeginsWithQueryConfig config =
CodeSampleUtils.BeginsWithQueryConfig.fromArgs(args);
           LOGGER.log(Level.INFO, "Querying items where {0} = {1} and {2} begins
with ''{3}''", new Object[] {
               config.getPartitionKeyName(),
               config.getPartitionKeyValue(),
               config.getSortKeyName(),
               config.getSortKeyPrefix()
           });

           // Using the builder pattern to create and execute the query
           final QueryResponse response = new BeginsWithQueryBuilder()
               .withTableName(config.getTableName())
               .withPartitionKeyName(config.getPartitionKeyName())
               .withPartitionKeyValue(config.getPartitionKeyValue())
               .withSortKeyName(config.getSortKeyName())
               .withSortKeyPrefix(config.getSortKeyPrefix())
               .withRegion(config.getRegion())
               .execute();

           // Process the results
           LOGGER.log(Level.INFO, "Found {0} items:", response.count());
           response.items().forEach(item -> LOGGER.info(item.toString()));

           // Demonstrate with a different prefix
           if (!config.getSortKeyPrefix().isEmpty()) {
               String shorterPrefix = config.getSortKeyPrefix()
                   .substring(0, Math.max(1,
config.getSortKeyPrefix().length() / 2));
               LOGGER.log(Level.INFO, "\nNow querying with a shorter prefix:
''{0}''", shorterPrefix);

               final QueryResponse response2 = new BeginsWithQueryBuilder()
                   .withTableName(config.getTableName())
                   .withPartitionKeyName(config.getPartitionKeyName())
                   .withPartitionKeyValue(config.getPartitionKeyValue())
                   .withSortKeyName(config.getSortKeyName())
                   .withSortKeyPrefix(shorterPrefix)
                   .withRegion(config.getRegion())
                   .execute();
```

```
                LOGGER.log(Level.INFO, "Found {0} items with shorter prefix:",
 response2.count());
                response2.items().forEach(item -> LOGGER.info(item.toString()));
            }
        } catch (IllegalArgumentException e) {
            LOGGER.log(Level.SEVERE, "Invalid input: {0}", e.getMessage());
            printUsage();
        } catch (ResourceNotFoundException e) {
            LOGGER.log(Level.SEVERE, "Table not found", e);
        } catch (DynamoDbException e) {
            LOGGER.log(Level.SEVERE, "DynamoDB error", e);
        } catch (Exception e) {
            LOGGER.log(Level.SEVERE, "Unexpected error", e);
        }
    }
```

- Weitere API-Informationen finden Sie unter Abfragen in der AWS SDK for Java 2.x -API-Referenz.

JavaScript

SDK für JavaScript (v3)

Fragen Sie mit AWS SDK für JavaScript eine DynamoDB-Tabelle mithilfe einer „begins_with“-Bedingung im Sortierschlüssel ab.

```
const { DynamoDBClient, QueryCommand } = require("@aws-sdk/client-dynamodb");

/**
 * Queries a DynamoDB table for items where the sort key begins with a specific
 prefix
 *
 * @param {Object} config - AWS SDK configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {string} partitionKeyName - The name of the partition key
 * @param {string} partitionKeyValue - The value of the partition key
 * @param {string} sortKeyName - The name of the sort key
 * @param {string} prefix - The prefix to match at the beginning of the sort key
 * @returns {Promise<Object>} - The query response
 */
async function queryWithBeginsWith(
```

```
  config,
  tableName,
  partitionKeyName,
  partitionKeyValue,
  sortKeyName,
  prefix
) {
  try {
    // Create DynamoDB client
    const client = new DynamoDBClient(config);

    // Construct the query input
    const input = {
      TableName: tableName,
      KeyConditionExpression: "#pk = :pkValue AND begins_with(#sk, :prefix)",
      ExpressionAttributeNames: {
        "#pk": partitionKeyName,
        "#sk": sortKeyName
      },
      ExpressionAttributeValues: {
        ":pkValue": { S: partitionKeyValue },
        ":prefix": { S: prefix }
      }
    };

    // Execute the query
    const command = new QueryCommand(input);
    return await client.send(command);
  } catch (error) {
    console.error(`Error querying with begins_with: ${error}`);
    throw error;
  }
}
```

- Weitere API-Informationen finden Sie unter Abfragen in der AWS SDK für JavaScript -API-Referenz.

Python

SDK für Python (Boto3)

Fragen Sie mit AWS SDK für Python (Boto3) eine DynamoDB-Tabelle mithilfe einer „begins_with“-Bedingung im Sortierschlüssel ab.

```
import boto3
from boto3.dynamodb.conditions import Key


def query_with_begins_with(
    table_name, partition_key_name, partition_key_value, sort_key_name, prefix
):
    """
    Query a DynamoDB table with a begins_with condition on the sort key.

    Args:
        table_name (str): The name of the DynamoDB table.
        partition_key_name (str): The name of the partition key attribute.
        partition_key_value (str): The value of the partition key to query.
        sort_key_name (str): The name of the sort key attribute.
        prefix (str): The prefix to match at the beginning of the sort key.

    Returns:
        dict: The response from DynamoDB containing the query results.
    """
    # Initialize the DynamoDB resource
    dynamodb = boto3.resource("dynamodb")
    table = dynamodb.Table(table_name)

    # Perform the query with a begins_with condition on the sort key
    key_condition = Key(partition_key_name).eq(partition_key_value) & Key(
        sort_key_name
    ).begins_with(prefix)
    response = table.query(KeyConditionExpression=key_condition)

    return response
```

- Weitere API-Informationen finden Sie unter Abfragen in der API-Referenz zum AWS SDK für Python (Boto3).

Eine vollständige Liste der AWS SDK-Entwicklerhandbücher und Codebeispiele finden Sie unterDynamoDB mit einem SDK verwenden AWS. Dieses Thema enthält auch Informationen zu den ersten Schritten und Details zu früheren SDK-Versionen.

Fragen Sie eine DynamoDB-Tabelle mithilfe eines Datumsbereichs im Sortierschlüssel mit einem SDK ab AWS

In den folgenden Codebeispielen wird gezeigt, wie eine Tabelle mithilfe eines Datumsbereichs im Sortierschlüssel abgefragt wird.

- Abfragen von Elementen innerhalb eines bestimmten Datumsbereichs
- Verwenden von Vergleichsoperatoren für Sortierschlüssel im Datumsformat

Java

SDK für Java 2.x

Fragen Sie eine DynamoDB-Tabelle nach Elementen innerhalb eines Datumsbereichs mit ab. AWS SDK for Java 2.x

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.AttributeValue;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.model.QueryRequest;
import software.amazon.awssdk.services.dynamodb.model.QueryResponse;
import software.amazon.awssdk.services.dynamodb.model.ResourceNotFoundException;

import java.time.LocalDate;
import java.util.HashMap;
import java.util.Map;
import java.util.logging.Level;
import java.util.logging.Logger;

    public QueryResponse queryWithDateRange(
        final String tableName,
        final String partitionKeyName,
```

```
        final String partitionKeyValue,
        final String dateKeyName,
        final LocalDate startDate,
        final LocalDate endDate) {

        // Focus on query logic, assuming parameters are valid
        if (startDate == null || endDate == null) {
            throw new IllegalArgumentException("Start date and end date cannot be
null");
        }

        if (endDate.isBefore(startDate)) {
            throw new IllegalArgumentException("End date must be after start
date");
        }

        // Format dates as ISO strings for DynamoDB (using just the date part)
        final String formattedStartDate = startDate.toString();
        final String formattedEndDate = endDate.toString();

        // Create expression attribute names for the column names
        final Map<String, String> expressionAttributeNames = new HashMap<>();
        expressionAttributeNames.put(EXPRESSION_ATTRIBUTE_NAME_PK,
partitionKeyName);
        expressionAttributeNames.put(EXPRESSION_ATTRIBUTE_NAME_SK, dateKeyName);

        // Create expression attribute values for the column values
        final Map<String, AttributeValue> expressionAttributeValues = new
HashMap<>();
        expressionAttributeValues.put(
            EXPRESSION_ATTRIBUTE_VALUE_PK,
            AttributeValue.builder().s(partitionKeyValue).build());
        expressionAttributeValues.put(
            EXPRESSION_ATTRIBUTE_VALUE_START_DATE,
            AttributeValue.builder().s(formattedStartDate).build());
        expressionAttributeValues.put(
            EXPRESSION_ATTRIBUTE_VALUE_END_DATE,
            AttributeValue.builder().s(formattedEndDate).build());

        // Create the query request
        final QueryRequest queryRequest = QueryRequest.builder()
            .tableName(tableName)
            .keyConditionExpression(KEY_CONDITION_EXPRESSION)
            .expressionAttributeNames(expressionAttributeNames)
```

```
            .expressionAttributeValues(expressionAttributeValues)
            .build();

        try {
            final QueryResponse response = dynamoDbClient.query(queryRequest);
            LOGGER.log(Level.INFO, "Query by date range successful. Found {0}
items", response.count());
            return response;
        } catch (ResourceNotFoundException e) {
            LOGGER.log(Level.SEVERE, "Table not found: {0}", tableName);
            throw e;
        } catch (DynamoDbException e) {
            LOGGER.log(Level.SEVERE, "Error querying by date range: {0}",
e.getMessage());
            throw e;
        }
    }
```

Zeigt, wie eine DynamoDB-Tabelle mit Datumsbereichsfilterung abgefragt wird.

```
    public static void main(String[] args) {
        final String usage =
            """
                Usage:
                    <tableName> <partitionKeyName> <partitionKeyValue>
<dateKeyName> <startDate> <endDate> [region]
                Where:
                    tableName - The Amazon DynamoDB table to query.
                    partitionKeyName - The name of the partition key attribute.
                    partitionKeyValue - The value of the partition key to query.
                    dateKeyName - The name of the date attribute to filter on.
                    startDate - The start date for the range query (YYYY-MM-DD).
                    endDate - The end date for the range query (YYYY-MM-DD).
                    region (optional) - The AWS region where the table exists.
(Default: us-east-1)
                """;

        if (args.length < 6) {
            System.out.println(usage);
            System.exit(1);
        }
```

```
        try {
            // Parse command line arguments into a config object
            CodeSampleUtils.DateRangeQueryConfig config =
CodeSampleUtils.DateRangeQueryConfig.fromArgs(args);

            LOGGER.log(
                Level.INFO, "Querying items from {0} to {1}", new Object[]
{config.getStartDate(), config.getEndDate()
                });

            // Using the builder pattern to create and execute the query
            final QueryResponse response = new DateRangeQueryBuilder()
                .withTableName(config.getTableName())
                .withPartitionKeyName(config.getPartitionKeyName())
                .withPartitionKeyValue(config.getPartitionKeyValue())
                .withDateKeyName(config.getDateKeyName())
                .withStartDate(config.getStartDate())
                .withEndDate(config.getEndDate())
                .withRegion(config.getRegion())
                .execute();

            // Process the results
            LOGGER.log(Level.INFO, "Found {0} items:", response.count());
            response.items().forEach(item -> {
                LOGGER.info(item.toString());

                // Extract and display the date attribute for clarity
                if (item.containsKey(config.getDateKeyName())) {
                    LOGGER.log(
                        Level.INFO,
                        "  Date attribute: {0}",
                        item.get(config.getDateKeyName()).s());
                }
            });

            // Demonstrate with a different date range
            LocalDate narrowerStartDate = config.getStartDate().plusDays(1);
            LocalDate narrowerEndDate = config.getEndDate().minusDays(1);

            if (!narrowerStartDate.isAfter(narrowerEndDate)) {
                LOGGER.log(Level.INFO, "\nNow querying with a narrower date
range: {0} to {1}", new Object[] {
                    narrowerStartDate, narrowerEndDate
                });
```

```
                final QueryResponse response2 = new DateRangeQueryBuilder()
                    .withTableName(config.getTableName())
                    .withPartitionKeyName(config.getPartitionKeyName())
                    .withPartitionKeyValue(config.getPartitionKeyValue())
                    .withDateKeyName(config.getDateKeyName())
                    .withStartDate(narrowerStartDate)
                    .withEndDate(narrowerEndDate)
                    .withRegion(config.getRegion())
                    .execute();

                LOGGER.log(Level.INFO, "Found {0} items with narrower date
range:", response2.count());
                response2.items().forEach(item -> LOGGER.info(item.toString()));
            }

            LOGGER.info("\nNote: When storing dates in DynamoDB:");
            LOGGER.info("1. Use ISO format (YYYY-MM-DD) for lexicographical
ordering");
            LOGGER.info("2. Use the BETWEEN operator for inclusive date range
queries");
            LOGGER.info("3. Consider using ISO-8601 format for timestamps with
time components");

        } catch (IllegalArgumentException e) {
            LOGGER.log(Level.SEVERE, "Invalid input: {0}", e.getMessage());
            System.exit(1);
        } catch (ResourceNotFoundException e) {
            LOGGER.log(Level.SEVERE, "Table not found: {0}", e.getMessage());
            System.exit(1);
        } catch (DynamoDbException e) {
            LOGGER.log(Level.SEVERE, "DynamoDB error: {0}", e.getMessage());
            System.exit(1);
        } catch (Exception e) {
            LOGGER.log(Level.SEVERE, "Unexpected error: {0}", e.getMessage());
            System.exit(1);
        }
    }
```

- Weitere API-Informationen finden Sie unter Abfragen in der AWS SDK for Java 2.x -API-Referenz.

JavaScript

SDK für JavaScript (v3)

Fragen Sie eine DynamoDB-Tabelle nach Elementen innerhalb eines Datumsbereichs mit ab. AWS SDK für JavaScript

```
const { DynamoDBClient, QueryCommand } = require("@aws-sdk/client-dynamodb");

/**
 * Queries a DynamoDB table for items within a specific date range on the sort
 key
 *
 * @param {Object} config - AWS SDK configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {string} partitionKeyName - The name of the partition key
 * @param {string} partitionKeyValue - The value of the partition key
 * @param {string} sortKeyName - The name of the sort key (must be a date/time
 attribute)
 * @param {Date} startDate - The start date for the range query
 * @param {Date} endDate - The end date for the range query
 * @returns {Promise<Object>} - The query response
 */
async function queryByDateRangeOnSortKey(
  config,
  tableName,
  partitionKeyName,
  partitionKeyValue,
  sortKeyName,
  startDate,
  endDate
) {
  try {
    // Create DynamoDB client
    const client = new DynamoDBClient(config);

    // Format dates as ISO strings for DynamoDB
    const formattedStartDate = startDate.toISOString();
    const formattedEndDate = endDate.toISOString();

    // Construct the query input
    const input = {
      TableName: tableName,
```

```
      KeyConditionExpression: '#pk = :pkValue AND #sk BETWEEN :startDate
 AND :endDate',
      ExpressionAttributeNames: {
        "#pk": partitionKeyName,
        "#sk": sortKeyName
      },
      ExpressionAttributeValues: {
        ":pkValue": { S: partitionKeyValue },
        ":startDate": { S: formattedStartDate },
        ":endDate": { S: formattedEndDate }
      }
    };

    // Execute the query
    const command = new QueryCommand(input);
    return await client.send(command);
  } catch (error) {
    console.error(`Error querying by date range on sort key: ${error}`);
    throw error;
  }
}
```

- Weitere API-Informationen finden Sie unter Abfragen in der AWS SDK für JavaScript -API-Referenz.

Python

SDK für Python (Boto3)

Fragen Sie eine DynamoDB-Tabelle nach Elementen innerhalb eines Datumsbereichs mit ab. AWS SDK für Python (Boto3)

```
from datetime import datetime, timedelta

import boto3
from boto3.dynamodb.conditions import Key


def query_with_date_range(
    table_name, partition_key_name, partition_key_value, sort_key_name,
 start_date, end_date
```

```
):
    """
    Query a DynamoDB table with a date range on the sort key.

    Args:
        table_name (str): The name of the DynamoDB table.
        partition_key_name (str): The name of the partition key attribute.
        partition_key_value (str): The value of the partition key to query.
        sort_key_name (str): The name of the sort key attribute (containing date
 values).
        start_date (datetime): The start date for the query range.
        end_date (datetime): The end date for the query range.

    Returns:
        dict: The response from DynamoDB containing the query results.
    """
    # Initialize the DynamoDB resource
    dynamodb = boto3.resource("dynamodb")
    table = dynamodb.Table(table_name)

    # Format the date values as ISO 8601 strings
    # DynamoDB works well with ISO format for date values
    start_date_str = start_date.isoformat()
    end_date_str = end_date.isoformat()

    # Perform the query with a date range on the sort key using BETWEEN operator
    key_condition = Key(partition_key_name).eq(partition_key_value) &
 Key(sort_key_name).between(
        start_date_str, end_date_str
    )

    response = table.query(
        KeyConditionExpression=key_condition,
        ExpressionAttributeValues={
            ":pk_val": partition_key_value,
            ":start_date": start_date_str,
            ":end_date": end_date_str,
        },
    )

    return response


def query_with_date_range_by_month(
```

```
    table_name, partition_key_name, partition_key_value, sort_key_name, year,
 month
):
    """
    Query a DynamoDB table for a specific month's data.

    Args:
        table_name (str): The name of the DynamoDB table.
        partition_key_name (str): The name of the partition key attribute.
        partition_key_value (str): The value of the partition key to query.
        sort_key_name (str): The name of the sort key attribute (containing date
 values).
        year (int): The year to query.
        month (int): The month to query (1-12).

    Returns:
        dict: The response from DynamoDB containing the query results.
    """
    # Initialize the DynamoDB resource
    dynamodb = boto3.resource("dynamodb")
    table = dynamodb.Table(table_name)

    # Calculate the start and end dates for the specified month
    if month == 12:
        next_year = year + 1
        next_month = 1
    else:
        next_year = year
        next_month = month + 1

    start_date = datetime(year, month, 1)
    end_date = datetime(next_year, next_month, 1) - timedelta(microseconds=1)

    # Format the date values as ISO 8601 strings
    start_date_str = start_date.isoformat()
    end_date_str = end_date.isoformat()

    # Perform the query with a date range on the sort key
    key_condition = Key(partition_key_name).eq(partition_key_value) &
 Key(sort_key_name).between(
        start_date_str, end_date_str
    )

    response = table.query(KeyConditionExpression=key_condition)
```

```
    return response
```

- Weitere API-Informationen finden Sie unter Abfragen in der API-Referenz zum AWS SDK für Python (Boto3).

Eine vollständige Liste der AWS SDK-Entwicklerhandbücher und Codebeispiele finden Sie unter. DynamoDB mit einem SDK verwenden AWS Dieses Thema enthält auch Informationen zu den ersten Schritten und Details zu früheren SDK-Versionen.

Fragen Sie eine DynamoDB-Tabelle mit einem komplexen Filterausdruck mit einem SDK ab AWS

In den folgenden Codebeispielen wird gezeigt, wie eine Tabelle mit einem komplexen Filterausdruck abgefragt wird.

- Anwenden von komplexen Filterausdrücken auf Abfrageergebnisse
- Kombinieren von mehreren Bedingungen über logische Operatoren
- Filtern von Elementen auf der Grundlage von Nicht-Schlüsselattributen.

Java

SDK für Java 2.x

Fragen Sie eine DynamoDB-Tabelle mit einem komplexen Filterausdruck ab mit. AWS SDK for Java 2.x

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.AttributeValue;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.model.QueryRequest;
import software.amazon.awssdk.services.dynamodb.model.QueryResponse;
import software.amazon.awssdk.services.dynamodb.model.ResourceNotFoundException;

import java.util.HashMap;
```

```
import java.util.Map;
import java.util.logging.Level;
import java.util.logging.Logger;

    public QueryResponse queryWithComplexFilter(
        final String tableName,
        final String partitionKeyName,
        final String partitionKeyValue,
        final String statusAttrName,
        final String activeStatus,
        final String pendingStatus,
        final String priceAttrName,
        final double minPrice,
        final double maxPrice,
        final String categoryAttrName) {

        // Validate parameters
        CodeSampleUtils.validateTableParameters(tableName, partitionKeyName,
 partitionKeyValue);
        CodeSampleUtils.validateStringParameter("Status attribute name",
 statusAttrName);
        CodeSampleUtils.validateStringParameter("Active status", activeStatus);
        CodeSampleUtils.validateStringParameter("Pending status", pendingStatus);
        CodeSampleUtils.validateStringParameter("Price attribute name",
 priceAttrName);
        CodeSampleUtils.validateStringParameter("Category attribute name",
 categoryAttrName);
        CodeSampleUtils.validateNumericRange("Minimum price", minPrice, 0.0,
 Double.MAX_VALUE);
        CodeSampleUtils.validateNumericRange("Maximum price", maxPrice, minPrice,
 Double.MAX_VALUE);

        // Create expression attribute names for the column names
        final Map<String, String> expressionAttributeNames = new HashMap<>();
        expressionAttributeNames.put("#pk", partitionKeyName);
        expressionAttributeNames.put(EXPRESSION_ATTRIBUTE_NAME_STATUS,
 statusAttrName);
        expressionAttributeNames.put(EXPRESSION_ATTRIBUTE_NAME_PRICE,
 priceAttrName);
        expressionAttributeNames.put(EXPRESSION_ATTRIBUTE_NAME_CATEGORY,
 categoryAttrName);

        // Create expression attribute values for the column values
```

```
        final Map<String, AttributeValue> expressionAttributeValues = new
 HashMap<>();
        expressionAttributeValues.put(
            ":pkValue", AttributeValue.builder().s(partitionKeyValue).build());
        expressionAttributeValues.put(
            EXPRESSION_ATTRIBUTE_VALUE_ACTIVE,
            AttributeValue.builder().s(activeStatus).build());
        expressionAttributeValues.put(
            EXPRESSION_ATTRIBUTE_VALUE_PENDING,
            AttributeValue.builder().s(pendingStatus).build());
        expressionAttributeValues.put(
            EXPRESSION_ATTRIBUTE_VALUE_MIN_PRICE,
            AttributeValue.builder().n(String.valueOf(minPrice)).build());
        expressionAttributeValues.put(
            EXPRESSION_ATTRIBUTE_VALUE_MAX_PRICE,
            AttributeValue.builder().n(String.valueOf(maxPrice)).build());

        // Create the query request
        final QueryRequest queryRequest = QueryRequest.builder()
            .tableName(tableName)
            .keyConditionExpression(KEY_CONDITION_EXPRESSION)
            .filterExpression(FILTER_EXPRESSION)
            .expressionAttributeNames(expressionAttributeNames)
            .expressionAttributeValues(expressionAttributeValues)
            .build();

        return dynamoDbClient.query(queryRequest);
    }
```

- Weitere API-Informationen finden Sie unter Abfragen in der AWS SDK for Java 2.x -API-Referenz.

JavaScript

SDK für JavaScript (v3)

Fragen Sie eine DynamoDB-Tabelle mit einem komplexen Filterausdruck ab mit. AWS SDK für JavaScript

```
const { DynamoDBClient, QueryCommand } = require("@aws-sdk/client-dynamodb");
```

```
/**
 * Queries a DynamoDB table with a complex filter expression
 *
 * @param {Object} config - AWS SDK configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {string} partitionKeyName - The name of the partition key
 * @param {string} partitionKeyValue - The value of the partition key
 * @param {number|string} minViews - Minimum number of views for filtering
 * @param {number|string} minReplies - Minimum number of replies for filtering
 * @param {string} requiredTag - Tag that must be present in the item's tags set
 * @returns {Promise<Object>} - The query response
 */
async function queryWithComplexFilter(
  config,
  tableName,
  partitionKeyName,
  partitionKeyValue,
  minViews,
  minReplies,
  requiredTag
) {
  try {
    // Create DynamoDB client
    const client = new DynamoDBClient(config);

    // Construct the query input
    const input = {
      TableName: tableName,
      KeyConditionExpression: "#pk = :pkValue",
      FilterExpression: "views >= :minViews AND replies >= :minReplies AND
 contains(tags, :tag)",
      ExpressionAttributeNames: {
        "#pk": partitionKeyName
      },
      ExpressionAttributeValues: {
        ":pkValue": { S: partitionKeyValue },
        ":minViews": { N: minViews.toString() },
        ":minReplies": { N: minReplies.toString() },
        ":tag": { S: requiredTag }
      }
    };

    // Execute the query
    const command = new QueryCommand(input);
```

```
    return await client.send(command);
  } catch (error) {
    console.error(`Error querying with complex filter: ${error}`);
    throw error;
  }
}
```

- Weitere API-Informationen finden Sie unter Abfragen in der AWS SDK für JavaScript -API-Referenz.

Python

SDK für Python (Boto3)

Fragen Sie eine DynamoDB-Tabelle mit einem komplexen Filterausdruck ab mit. AWS SDK für Python (Boto3)

```
import boto3
from boto3.dynamodb.conditions import Attr, Key


def query_with_complex_filter(
    table_name,
    partition_key_name,
    partition_key_value,
    min_rating=None,
    status_list=None,
    max_price=None,
):
    """
    Query a DynamoDB table with a complex filter expression.

    Args:
        table_name (str): The name of the DynamoDB table.
        partition_key_name (str): The name of the partition key attribute.
        partition_key_value (str): The value of the partition key to query.
        min_rating (float, optional): Minimum rating value for filtering.
        status_list (list, optional): List of status values to include.
        max_price (float, optional): Maximum price value for filtering.

    Returns:
```

```
        dict: The response from DynamoDB containing the query results.
    """
    # Initialize the DynamoDB resource
    dynamodb = boto3.resource("dynamodb")
    table = dynamodb.Table(table_name)

    # Start with the key condition expression
    key_condition = Key(partition_key_name).eq(partition_key_value)

    # Initialize the filter expression and expression attribute values
    filter_expression = None
    expression_attribute_values = {}

    # Build the filter expression based on provided parameters
    if min_rating is not None:
        filter_expression = Attr("rating").gte(min_rating)
        expression_attribute_values[":min_rating"] = min_rating

    if status_list and len(status_list) > 0:
        status_condition = None
        for i, status in enumerate(status_list):
            status_value_name = f":status{i}"
            expression_attribute_values[status_value_name] = status

            if status_condition is None:
                status_condition = Attr("status").eq(status)
            else:
                status_condition = status_condition | Attr("status").eq(status)

        if filter_expression is None:
            filter_expression = status_condition
        else:
            filter_expression = filter_expression & status_condition

    if max_price is not None:
        price_condition = Attr("price").lte(max_price)
        expression_attribute_values[":max_price"] = max_price

        if filter_expression is None:
            filter_expression = price_condition
        else:
            filter_expression = filter_expression & price_condition

    # Prepare the query parameters
```

```
    query_params = {"KeyConditionExpression": key_condition}

    if filter_expression:
        query_params["FilterExpression"] = filter_expression
        if expression_attribute_values:
            query_params["ExpressionAttributeValues"] =
 expression_attribute_values

    # Execute the query
    response = table.query(**query_params)
    return response


def query_with_complex_filter_and_or(
    table_name,
    partition_key_name,
    partition_key_value,
    category=None,
    min_rating=None,
    max_price=None,
):
    """
    Query a DynamoDB table with a complex filter expression using AND and OR
 operators.

    Args:
        table_name (str): The name of the DynamoDB table.
        partition_key_name (str): The name of the partition key attribute.
        partition_key_value (str): The value of the partition key to query.
        category (str, optional): Category value for filtering.
        min_rating (float, optional): Minimum rating value for filtering.
        max_price (float, optional): Maximum price value for filtering.

    Returns:
        dict: The response from DynamoDB containing the query results.
    """
    # Initialize the DynamoDB resource
    dynamodb = boto3.resource("dynamodb")
    table = dynamodb.Table(table_name)

    # Start with the key condition expression
    key_condition = Key(partition_key_name).eq(partition_key_value)

    # Build a complex filter expression with AND and OR operators
```

```
    filter_expression = None
    expression_attribute_values = {}

    # Build the category condition
    if category:
        filter_expression = Attr("category").eq(category)
        expression_attribute_values[":category"] = category

    # Build the rating and price condition (rating >= min_rating OR price <=
max_price)
    rating_price_condition = None

    if min_rating is not None:
        rating_price_condition = Attr("rating").gte(min_rating)
        expression_attribute_values[":min_rating"] = min_rating

    if max_price is not None:
        price_condition = Attr("price").lte(max_price)
        expression_attribute_values[":max_price"] = max_price

        if rating_price_condition is None:
            rating_price_condition = price_condition
        else:
            rating_price_condition = rating_price_condition | price_condition

    # Combine the conditions
    if rating_price_condition:
        if filter_expression is None:
            filter_expression = rating_price_condition
        else:
            filter_expression = filter_expression & rating_price_condition

    # Prepare the query parameters
    query_params = {"KeyConditionExpression": key_condition}

    if filter_expression:
        query_params["FilterExpression"] = filter_expression
        if expression_attribute_values:
            query_params["ExpressionAttributeValues"] =
expression_attribute_values

    # Execute the query
    response = table.query(**query_params)
    return response
```

- Weitere API-Informationen finden Sie unter Abfragen in der API-Referenz zum AWS SDK für Python (Boto3).

Eine vollständige Liste der AWS SDK-Entwicklerhandbücher und Codebeispiele finden Sie unter. DynamoDB mit einem SDK verwenden AWS Dieses Thema enthält auch Informationen zu den ersten Schritten und Details zu früheren SDK-Versionen.

Fragen Sie eine DynamoDB-Tabelle mit einem dynamischen Filterausdruck mit einem SDK ab AWS

In den folgenden Codebeispielen wird gezeigt, wie eine Tabelle mit einem dynamischen Filterausdruck abgefragt wird.

- Erstellen von Filterausdrücken dynamisch zur Laufzeit
- Konstruieren von Filterbedingungen auf der Grundlage von Benutzereingaben oder Anwendungsstatus
- Hinzufügen oder Entfernen von Filterkriterien unter bestimmten Bedingungen

Java

SDK für Java 2.x

Fragen Sie eine DynamoDB-Tabelle mit einem dynamisch erstellten Filterausdruck ab mit. AWS SDK for Java 2.x

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.AttributeValue;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.model.QueryRequest;
import software.amazon.awssdk.services.dynamodb.model.QueryResponse;
import software.amazon.awssdk.services.dynamodb.model.ResourceNotFoundException;

import java.util.HashMap;
import java.util.Map;
```

```
   public static QueryResponse queryWithDynamicFilter(
       final String tableName,
       final String partitionKeyName,
       final String partitionKeyValue,
       final Map<String, Object> filterCriteria,
       final Region region,
       final DynamoDbClient dynamoDbClient) {

       validateParameters(tableName, partitionKeyName, partitionKeyValue,
filterCriteria);

       DynamoDbClient ddbClient = dynamoDbClient;
       boolean shouldClose = false;

       try {
           if (ddbClient == null) {
               ddbClient = createClient(region);
               shouldClose = true;
           }

           final QueryWithDynamicFilter queryHelper = new
QueryWithDynamicFilter(ddbClient);
           return queryHelper.queryWithDynamicFilter(tableName,
partitionKeyName, partitionKeyValue, filterCriteria);
       } catch (ResourceNotFoundException e) {
           System.err.println("Table not found: " + tableName);
           throw e;
       } catch (DynamoDbException e) {
           System.err.println("Failed to execute dynamic filter query: " +
e.getMessage());
           throw e;
       } catch (Exception e) {
           System.err.println("Unexpected error during query: " +
e.getMessage());
           throw e;
       } finally {
           if (shouldClose && ddbClient != null) {
               ddbClient.close();
           }
       }
   }
```

Demonstriert, wie dynamische Filterausdrücke mit verwendet werden. AWS SDK for Java 2.x

```
   public static void main(String[] args) {
       final String usage =
           """
               Usage:
                   <tableName> <partitionKeyName> <partitionKeyValue>
<filterAttrName> <filterAttrValue> [region]
               Where:
                   tableName - The Amazon DynamoDB table to query.
                   partitionKeyName - The name of the partition key attribute.
                   partitionKeyValue - The value of the partition key to query.
                   filterAttrName - The name of the attribute to filter on.
                   filterAttrValue - The value to filter by.
                   region (optional) - The AWS region where the table exists.
(Default: us-east-1)
               """;

       if (args.length < 5) {
           System.out.println(usage);
           System.exit(1);
       }

       final String tableName = args[0];
       final String partitionKeyName = args[1];
       final String partitionKeyValue = args[2];
       final String filterAttrName = args[3];
       final String filterAttrValue = args[4];
       final Region region = args.length > 5 ? Region.of(args[5]) :
Region.US_EAST_1;

       System.out.println("Querying items with dynamic filter: " +
filterAttrName + " = " + filterAttrValue);

       try {
           // Using the builder pattern to create and execute the query
           final QueryResponse response = new DynamicFilterQueryBuilder()
               .withTableName(tableName)
               .withPartitionKeyName(partitionKeyName)
               .withPartitionKeyValue(partitionKeyValue)
               .withFilterCriterion(filterAttrName, filterAttrValue)
               .withRegion(region)
               .execute();
```

```
            // Process the results
            System.out.println("Found " + response.count() + " items:");
            response.items().forEach(item -> System.out.println(item));

            // Demonstrate multiple filter criteria
            System.out.println("\nNow querying with multiple filter criteria:");

            Map<String, Object> multipleFilters = new HashMap<>();
            multipleFilters.put(filterAttrName, filterAttrValue);
            multipleFilters.put("status", "active");

            final QueryResponse multiFilterResponse = new
DynamicFilterQueryBuilder()
                .withTableName(tableName)
                .withPartitionKeyName(partitionKeyName)
                .withPartitionKeyValue(partitionKeyValue)
                .withFilterCriteria(multipleFilters)
                .withRegion(region)
                .execute();

            System.out.println("Found " + multiFilterResponse.count() + " items
with multiple filters:");
            multiFilterResponse.items().forEach(item ->
System.out.println(item));

        } catch (IllegalArgumentException e) {
            System.err.println("Invalid input: " + e.getMessage());
            System.exit(1);
        } catch (ResourceNotFoundException e) {
            System.err.println("Table not found: " + tableName);
            System.exit(1);
        } catch (DynamoDbException e) {
            System.err.println("DynamoDB error: " + e.getMessage());
            System.exit(1);
        } catch (Exception e) {
            System.err.println("Unexpected error: " + e.getMessage());
            System.exit(1);
        }
    }
```

- Weitere API-Informationen finden Sie unter Abfragen in der AWS SDK for Java 2.x -API-Referenz.

JavaScript

SDK für JavaScript (v3)

Fragen Sie eine DynamoDB-Tabelle mit einem dynamisch erstellten Filterausdruck ab mit. AWS SDK für JavaScript

```
const { DynamoDBClient, QueryCommand } = require("@aws-sdk/client-dynamodb");

async function queryWithDynamicFilter(
  config,
  tableName,
  partitionKeyName,
  partitionKeyValue,
  sortKeyName,
  sortKeyValue,
  filterParams = {}
) {
  try {
    // Create DynamoDB client
    const client = new DynamoDBClient(config);

    // Initialize filter expression components
    let filterExpressions = [];
    const expressionAttributeValues = {
      ":pkValue": { S: partitionKeyValue },
      ":skValue": { S: sortKeyValue }
    };
    const expressionAttributeNames = {
      "#pk": partitionKeyName,
      "#sk": sortKeyName
    };

    // Add status filter if provided
    if (filterParams.status) {
      filterExpressions.push("status = :status");
      expressionAttributeValues[":status"] = { S: filterParams.status };
    }

    // Add minimum views filter if provided
    if (filterParams.minViews !== undefined) {
      filterExpressions.push("views >= :minViews");
      expressionAttributeValues[":minViews"] = { N:
 filterParams.minViews.toString() };
```

```
    }

    // Add author filter if provided
    if (filterParams.author) {
      filterExpressions.push("author = :author");
      expressionAttributeValues[":author"] = { S: filterParams.author };
    }

    // Construct the query input
    const input = {
      TableName: tableName,
      KeyConditionExpression: "#pk = :pkValue AND #sk = :skValue"
    };

    // Add filter expression if any filters were provided
    if (filterExpressions.length > 0) {
      input.FilterExpression = filterExpressions.join(" AND ");
    }

    // Add expression attribute names and values
    input.ExpressionAttributeNames = expressionAttributeNames;
    input.ExpressionAttributeValues = expressionAttributeValues;

    // Execute the query
    const command = new QueryCommand(input);
    return await client.send(command);
  } catch (error) {
    console.error(`Error querying with dynamic filter: ${error}`);
    throw error;
  }
}
```

- Weitere API-Informationen finden Sie unter Abfragen in der AWS SDK für JavaScript -API-Referenz.

Python

SDK für Python (Boto3)

Fragen Sie eine DynamoDB-Tabelle mit einem dynamisch erstellten Filterausdruck ab mit. AWS SDK für Python (Boto3)

```
import boto3
from boto3.dynamodb.conditions import Attr, Key


def query_with_dynamic_filter(
    table_name, partition_key_name, partition_key_value, filter_conditions=None
):
    """
    Query a DynamoDB table with a dynamically constructed filter expression.

    Args:
        table_name (str): The name of the DynamoDB table.
        partition_key_name (str): The name of the partition key attribute.
        partition_key_value (str): The value of the partition key to query.
        filter_conditions (dict, optional): A dictionary of filter conditions
 where
            keys are attribute names and values are dictionaries with 'operator'
 and 'value'.
            Example: {'rating': {'operator': '>=', 'value': 4}, 'status':
{'operator': '=', 'value': 'active'}}

    Returns:
        dict: The response from DynamoDB containing the query results.
    """
    # Initialize the DynamoDB resource
    dynamodb = boto3.resource("dynamodb")
    table = dynamodb.Table(table_name)

    # Start with the key condition expression
    key_condition = Key(partition_key_name).eq(partition_key_value)

    # Initialize variables for the filter expression and attribute values
    filter_expression = None
    expression_attribute_values = {":pk_val": partition_key_value}

    # Dynamically build the filter expression if filter conditions are provided
    if filter_conditions:
        for attr_name, condition in filter_conditions.items():
            operator = condition.get("operator")
            value = condition.get("value")
            attr_value_name = f":{attr_name}"
            expression_attribute_values[attr_value_name] = value
```

```
            # Create the appropriate filter expression based on the operator
            current_condition = None
            if operator == "=":
                current_condition = Attr(attr_name).eq(value)
            elif operator == "!=":
                current_condition = Attr(attr_name).ne(value)
            elif operator == ">":
                current_condition = Attr(attr_name).gt(value)
            elif operator == ">=":
                current_condition = Attr(attr_name).gte(value)
            elif operator == "<":
                current_condition = Attr(attr_name).lt(value)
            elif operator == "<=":
                current_condition = Attr(attr_name).lte(value)
            elif operator == "contains":
                current_condition = Attr(attr_name).contains(value)
            elif operator == "begins_with":
                current_condition = Attr(attr_name).begins_with(value)

            # Combine with existing filter expression using AND
            if current_condition:
                if filter_expression is None:
                    filter_expression = current_condition
                else:
                    filter_expression = filter_expression & current_condition

    # Perform the query with the dynamically built filter expression
    query_params = {"KeyConditionExpression": key_condition}

    if filter_expression:
        query_params["FilterExpression"] = filter_expression

    response = table.query(**query_params)
    return response
```

Demonstriert, wie dynamische Filterausdrücke mit verwendet werden. AWS SDK für Python (Boto3)

```
def example_usage():
    """Example of how to use the query_with_dynamic_filter function."""
```

```
    # Example parameters
    table_name = "Products"
    partition_key_name = "Category"
    partition_key_value = "Electronics"

    # Define dynamic filter conditions based on user input or runtime conditions
    user_min_rating = 4  # This could come from user input
    user_status_filter = "active"  # This could come from user input

    filter_conditions = {}

    # Only add conditions that are actually specified
    if user_min_rating is not None:
        filter_conditions["rating"] = {"operator": ">=", "value":
user_min_rating}

    if user_status_filter:
        filter_conditions["status"] = {"operator": "=", "value":
user_status_filter}

    print(
        f"Querying products in category '{partition_key_value}' with filter
conditions: {filter_conditions}"
    )

    # Execute the query with dynamic filter
    response = query_with_dynamic_filter(
        table_name, partition_key_name, partition_key_value, filter_conditions
    )

    # Process the results
    items = response.get("Items", [])
    print(f"Found {len(items)} items")

    for item in items:
        print(f"Product: {item}")
```

- Weitere API-Informationen finden Sie unter Abfragen in der API-Referenz zum AWS SDK für Python (Boto3).

Eine vollständige Liste der AWS SDK-Entwicklerhandbücher und Codebeispiele finden Sie unterDynamoDB mit einem SDK verwenden AWS. Dieses Thema enthält auch Informationen zu den ersten Schritten und Details zu früheren SDK-Versionen.

Fragen Sie eine DynamoDB-Tabelle mit einem Filterausdruck und einem Limit mit einem SDK ab AWS

In den folgenden Codebeispielen wird gezeigt, wie eine Tabelle mit einem Filterausdruck und Limit abgefragt wird.

- Anwenden von Filterausdrücken auf Abfrageergebnisse mit einem Limit für die ausgewerteten Elemente
- Verstehen, wie sich Limits auf gefilterte Abfrageergebnisse auswirken
- Steuern der maximalen Anzahl der in einer Abfrage verarbeiteten Elemente

Java

SDK für Java 2.x

Fragen Sie eine DynamoDB-Tabelle mit einem Filterausdruck ab und verwenden Sie das Limit. AWS SDK for Java 2.x

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.AttributeValue;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.model.QueryRequest;
import software.amazon.awssdk.services.dynamodb.model.QueryResponse;
import software.amazon.awssdk.services.dynamodb.model.ResourceNotFoundException;

import java.util.HashMap;
import java.util.Map;
import java.util.logging.Level;
import java.util.logging.Logger;

    public QueryResponse queryWithFilterAndLimit(
        final String tableName,
        final String partitionKeyName,
        final String partitionKeyValue,
        final String filterAttrName,
```

```
        final String filterAttrValue,
        final int limit) {

        CodeSampleUtils.validateTableParameters(tableName, partitionKeyName,
partitionKeyValue);
        CodeSampleUtils.validateStringParameter("Filter attribute name",
filterAttrName);
        CodeSampleUtils.validateStringParameter("Filter attribute value",
filterAttrValue);
        CodeSampleUtils.validatePositiveInteger("Limit", limit);

        // Create expression attribute names for the column names
        final Map<String, String> expressionAttributeNames = new HashMap<>();
        expressionAttributeNames.put(EXPRESSION_ATTRIBUTE_NAME_PK,
partitionKeyName);
        expressionAttributeNames.put(EXPRESSION_ATTRIBUTE_NAME_FILTER,
filterAttrName);

        // Create expression attribute values for the column values
        final Map<String, AttributeValue> expressionAttributeValues = new
HashMap<>();
        expressionAttributeValues.put(
            EXPRESSION_ATTRIBUTE_VALUE_PK,
            AttributeValue.builder().s(partitionKeyValue).build());
        expressionAttributeValues.put(
            EXPRESSION_ATTRIBUTE_VALUE_FILTER,
            AttributeValue.builder().s(filterAttrValue).build());

        // Create the filter expression
        final String filterExpression = "#filterAttr = :filterValue";

        // Create the query request
        final QueryRequest queryRequest = QueryRequest.builder()
            .tableName(tableName)
            .keyConditionExpression(KEY_CONDITION_EXPRESSION)
            .filterExpression(filterExpression)
            .expressionAttributeNames(expressionAttributeNames)
            .expressionAttributeValues(expressionAttributeValues)
            .limit(limit)
            .build();

        try {
            final QueryResponse response = dynamoDbClient.query(queryRequest);
```

```
            LOGGER.log(Level.INFO, "Query with filter and limit successful. Found
 {0} items", response.count());
            LOGGER.log(
                Level.INFO, "ScannedCount: {0} (total items evaluated before
 filtering)", response.scannedCount());
            return response;
        } catch (ResourceNotFoundException e) {
            LOGGER.log(Level.SEVERE, "Table not found: {0}", tableName);
            throw e;
        } catch (DynamoDbException e) {
            LOGGER.log(Level.SEVERE, "Error querying with filter and limit: {0}",
 e.getMessage());
            throw e;
        }
    }
```

- Weitere API-Informationen finden Sie unter Abfragen in der AWS SDK for Java 2.x -API-Referenz.

Python

SDK für Python (Boto3)

Fragen Sie eine DynamoDB-Tabelle mit einem Filterausdruck ab und verwenden Sie das Limit. AWS SDK für Python (Boto3)

```
import boto3
from boto3.dynamodb.conditions import Attr, Key


def query_with_filter_and_limit(
    table_name,
    partition_key_name,
    partition_key_value,
    filter_attribute=None,
    filter_value=None,
    limit=10,
):
    """
    Query a DynamoDB table with a filter expression and limit the number of
 results.
```

```
    Args:
        table_name (str): The name of the DynamoDB table.
        partition_key_name (str): The name of the partition key attribute.
        partition_key_value (str): The value of the partition key to query.
        filter_attribute (str, optional): The attribute name to filter on.
        filter_value (any, optional): The value to compare against in the filter.
        limit (int, optional): The maximum number of items to evaluate. Defaults
 to 10.

    Returns:
        dict: The response from DynamoDB containing the query results.
    """
    # Initialize the DynamoDB resource
    dynamodb = boto3.resource("dynamodb")
    table = dynamodb.Table(table_name)

    # Build the key condition expression
    key_condition = Key(partition_key_name).eq(partition_key_value)

    # Prepare the query parameters
    query_params = {"KeyConditionExpression": key_condition, "Limit": limit}

    # Add the filter expression if filter attributes are provided
    if filter_attribute and filter_value is not None:
        query_params["FilterExpression"] =
 Attr(filter_attribute).gt(filter_value)
        query_params["ExpressionAttributeValues"] = {":filter_value":
 filter_value}

    # Execute the query
    response = table.query(**query_params)
    return response
```

Demonstriert die Verwendung von Filterausdrücken mit Grenzwerten in. AWS SDK für Python (Boto3)

```
def example_usage():
    """Example of how to use the query_with_filter_and_limit function."""
    # Example parameters
```

```
    table_name = "ProductReviews"
    partition_key_name = "ProductId"
    partition_key_value = "P123456"
    filter_attribute = "Rating"
    filter_value = 3  # Filter for ratings > 3
    limit = 5

    print(f"Querying reviews for product '{partition_key_value}' with rating >
{filter_value}")
    print(f"Limiting to {limit} evaluated items")

    # Execute the query with filter and limit
    response = query_with_filter_and_limit(
        table_name, partition_key_name, partition_key_value, filter_attribute,
filter_value, limit
    )

    # Process the results
    items = response.get("Items", [])
    print(f"\nReturned {len(items)} items that passed the filter")

    for item in items:
        print(f"Review: {item}")

    # Explain the difference between Limit and actual results
    explain_limit_vs_results(response)

    # Check if there are more results
    if "LastEvaluatedKey" in response:
        print("\nThere are more results available. Use the LastEvaluatedKey for
pagination.")
    else:
        print("\nAll matching results have been retrieved.")
```

- Weitere API-Informationen finden Sie unter Abfragen in der API-Referenz zum AWS SDK für Python (Boto3).

Eine vollständige Liste der AWS SDK-Entwicklerhandbücher und Codebeispiele finden Sie unterDynamoDB mit einem SDK verwenden AWS. Dieses Thema enthält auch Informationen zu den ersten Schritten und Details zu früheren SDK-Versionen.

Abfragen einer DynamoDB-Tabelle mit verschachtelten Attributen mithilfe eines SDK AWS

In den folgenden Codebeispielen wird gezeigt, wie eine Tabelle mit verschachtelten Attributen abgefragt wird.

- Zugreifen und Filtern nach verschachtelten Attributen in DynamoDB-Elementen
- Verwenden von Dokumentpfadausdrücken, um auf verschachtelte Elemente zu verweisen

Java

SDK für Java 2.x

Fragen Sie eine DynamoDB-Tabelle mit verschachtelten Attributen ab mit. AWS SDK for Java 2.x

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.AttributeValue;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.model.QueryRequest;
import software.amazon.awssdk.services.dynamodb.model.QueryResponse;
import software.amazon.awssdk.services.dynamodb.model.ResourceNotFoundException;

import java.util.HashMap;
import java.util.Map;

    public QueryResponse queryWithNestedAttributes(
        final String tableName,
        final String partitionKeyName,
        final String partitionKeyValue,
        final String nestedPath,
        final String nestedAttr,
        final String nestedValue) {

        CodeSampleUtils.validateTableParameters(tableName, partitionKeyName,
 partitionKeyValue);
```

```
        CodeSampleUtils.validateStringParameter("Nested path", nestedPath);
        CodeSampleUtils.validateStringParameter("Nested attribute", nestedAttr);
        CodeSampleUtils.validateStringParameter("Nested value", nestedValue);

        // Split the nested path into components
        final String[] pathComponents = nestedPath.split("\\.");

        // Create expression attribute names for the column names
        final Map<String, String> expressionAttributeNames = new HashMap<>();
        expressionAttributeNames.put(EXPRESSION_ATTRIBUTE_NAME_PK,
partitionKeyName);

        // Build the nested attribute reference using document path notation
        final StringBuilder nestedAttributeRef = new StringBuilder();
        for (int i = 0; i < pathComponents.length; i++) {
            final String aliasName = "#n" + i;
            expressionAttributeNames.put(aliasName, pathComponents[i]);

            if (i > 0) {
                nestedAttributeRef.append(".");
            }
            nestedAttributeRef.append(aliasName);
        }

        // Create expression attribute values for the column values
        final Map<String, AttributeValue> expressionAttributeValues = new
HashMap<>();
        expressionAttributeValues.put(
            EXPRESSION_ATTRIBUTE_VALUE_PK,
            AttributeValue.builder().s(partitionKeyValue).build());
        expressionAttributeValues.put(
            EXPRESSION_ATTRIBUTE_VALUE_NESTED,
            AttributeValue.builder().s(nestedValue).build());

        // Create the filter expression using the nested attribute reference
        final String filterExpression = nestedAttributeRef + " = :nestedValue";

        // Create the query request
        final QueryRequest queryRequest = QueryRequest.builder()
            .tableName(tableName)
            .keyConditionExpression(KEY_CONDITION_EXPRESSION)
            .filterExpression(filterExpression)
            .expressionAttributeNames(expressionAttributeNames)
            .expressionAttributeValues(expressionAttributeValues)
```

```
            .build();

        try {
            final QueryResponse response = dynamoDbClient.query(queryRequest);
            System.out.println("Query with nested attribute filter successful.
Found " + response.count() + " items");
            return response;
        } catch (ResourceNotFoundException e) {
            System.err.format("Error: The Amazon DynamoDB table \"%s\" can't be
found.\n", tableName);
            throw e;
        } catch (DynamoDbException e) {
            System.err.println("Error querying with nested attribute filter: " +
e.getMessage());
            throw e;
        }
    }
```

Demonstriert, wie eine DynamoDB-Tabelle mit verschachtelten Attributen abgefragt wird.

```
    public static void main(String[] args) {
        final String usage =
            """
                Usage:
                    <tableName> <partitionKeyName> <partitionKeyValue>
<nestedPath> <nestedAttr> <nestedValue> [region]
                Where:
                    tableName - The Amazon DynamoDB table to query.
                    partitionKeyName - The name of the partition key attribute.
                    partitionKeyValue - The value of the partition key to query.
                    nestedPath - The path to the nested map attribute (e.g.,
"address").
                    nestedAttr - The name of the nested attribute (e.g., "city").
                    nestedValue - The value to filter by (e.g., "Seattle").
                    region (optional) - The AWS region where the table exists.
(Default: us-east-1)
                """;

        if (args.length < 6) {
            System.out.println(usage);
            System.exit(1);
        }
```

```
        final String tableName = args[0];
        final String partitionKeyName = args[1];
        final String partitionKeyValue = args[2];
        final String nestedPath = args[3];
        final String nestedAttr = args[4];
        final String nestedValue = args[5];
        final Region region = args.length > 6 ? Region.of(args[6]) :
Region.US_EAST_1;

        System.out.println("Querying items where " + partitionKeyName + " = " +
partitionKeyValue + " and " + nestedPath
            + "." + nestedAttr + " = " + nestedValue);

        try {
            // Using the builder pattern to create and execute the query
            final QueryResponse response = new NestedAttributeQueryBuilder()
                .withTableName(tableName)
                .withPartitionKeyName(partitionKeyName)
                .withPartitionKeyValue(partitionKeyValue)
                .withNestedPath(nestedPath)
                .withNestedAttribute(nestedAttr)
                .withNestedValue(nestedValue)
                .withRegion(region)
                .execute();

            // Process the results
            System.out.println("Found " + response.count() + " items:");
            response.items().forEach(item -> {
                System.out.println(item);

                // Extract and display the nested attribute for clarity
                if (item.containsKey(nestedPath) && item.get(nestedPath).hasM())
{
                    Map<String, AttributeValue> nestedMap =
item.get(nestedPath).m();
                    if (nestedMap.containsKey(nestedAttr)) {
                        System.out.println("  Nested attribute " + nestedPath +
"." + nestedAttr + ": "
                                + formatAttributeValue(nestedMap.get(nestedAttr)));
                    }
                }
            });
```

```
            System.out.println("\nNote: When working with nested attributes in
 DynamoDB:");
            System.out.println("1. Use dot notation in filter expressions to
 access nested attributes");
            System.out.println("2. Use expression attribute names for each
 component of the path");
            System.out.println("3. Check if the nested attribute exists before
 accessing it");

        } catch (IllegalArgumentException e) {
            System.err.println("Invalid input: " + e.getMessage());
            System.exit(1);
        } catch (ResourceNotFoundException e) {
            System.err.println("Table not found: " + tableName);
            System.exit(1);
        } catch (DynamoDbException e) {
            System.err.println("DynamoDB error: " + e.getMessage());
            System.exit(1);
        } catch (Exception e) {
            System.err.println("Unexpected error: " + e.getMessage());
            System.exit(1);
        }
    }
```

- Weitere API-Informationen finden Sie unter Abfragen in der AWS SDK for Java 2.x -API-Referenz.

JavaScript

SDK für JavaScript (v3)

Fragen Sie eine DynamoDB-Tabelle mit verschachtelten Attributen ab mit. AWS SDK für JavaScript

```
const { DynamoDBClient, QueryCommand } = require("@aws-sdk/client-dynamodb");

/**
 * Queries a DynamoDB table filtering on a nested attribute
 *
 * @param {Object} config - AWS SDK configuration object
 * @param {string} tableName - The name of the DynamoDB table
```

```
 * @param {string} productId - The product ID to query by (partition key)
 * @param {string} category - The category to filter by (nested attribute)
 * @returns {Promise<Object>} - The query response
 */
async function queryWithNestedAttribute(
  config,
  tableName,
  productId,
  category
) {
  try {
    // Create DynamoDB client
    const client = new DynamoDBClient(config);

    // Construct the query input
    const input = {
      TableName: tableName,
      KeyConditionExpression: "product_id = :productId",
      FilterExpression: "details.category = :category",
      ExpressionAttributeValues: {
        ":productId": { S: productId },
        ":category": { S: category }
      }
    };

    // Execute the query
    const command = new QueryCommand(input);
    return await client.send(command);
  } catch (error) {
    console.error(`Error querying with nested attribute: ${error}`);
    throw error;
  }
}
```

- Weitere API-Informationen finden Sie unter Abfragen in der AWS SDK für JavaScript -API-Referenz.

Python

SDK für Python (Boto3)

Fragen Sie eine DynamoDB-Tabelle mit verschachtelten Attributen ab mit. AWS SDK für Python (Boto3)

```
from typing import Any, Dict, List

import boto3
from boto3.dynamodb.conditions import Attr, Key


def query_with_nested_attributes(
    table_name: str,
    partition_key_name: str,
    partition_key_value: str,
    nested_path: str,
    comparison_operator: str,
    comparison_value: Any,
) -> Dict[str, Any]:
    """
    Query a DynamoDB table and filter by nested attributes.

    Args:
        table_name (str): The name of the DynamoDB table.
        partition_key_name (str): The name of the partition key attribute.
        partition_key_value (str): The value of the partition key to query.
        nested_path (str): The path to the nested attribute (e.g.,
 'specs.weight').
        comparison_operator (str): The comparison operator to use ('=', '!=',
 '<', '<=', '>', '>=').
        comparison_value (any): The value to compare against.

    Returns:
        dict: The response from DynamoDB containing the query results.
    """
    # Initialize the DynamoDB resource
    dynamodb = boto3.resource("dynamodb")
    table = dynamodb.Table(table_name)

    # Build the key condition expression
    key_condition = Key(partition_key_name).eq(partition_key_value)
```

```
    # Build the filter expression based on the nested attribute path and
 comparison operator
    filter_expression = None
    if comparison_operator == "=":
        filter_expression = Attr(nested_path).eq(comparison_value)
    elif comparison_operator == "!=":
        filter_expression = Attr(nested_path).ne(comparison_value)
    elif comparison_operator == "<":
        filter_expression = Attr(nested_path).lt(comparison_value)
    elif comparison_operator == "<=":
        filter_expression = Attr(nested_path).lte(comparison_value)
    elif comparison_operator == ">":
        filter_expression = Attr(nested_path).gt(comparison_value)
    elif comparison_operator == ">=":
        filter_expression = Attr(nested_path).gte(comparison_value)
    elif comparison_operator == "contains":
        filter_expression = Attr(nested_path).contains(comparison_value)
    elif comparison_operator == "begins_with":
        filter_expression = Attr(nested_path).begins_with(comparison_value)

    # Execute the query with the filter expression
    response = table.query(KeyConditionExpression=key_condition,
 FilterExpression=filter_expression)

    return response


def query_with_multiple_nested_attributes(
    table_name: str,
    partition_key_name: str,
    partition_key_value: str,
    nested_conditions: List[Dict[str, Any]],
) -> Dict[str, Any]:
    """
    Query a DynamoDB table and filter by multiple nested attributes.

    Args:
        table_name (str): The name of the DynamoDB table.
        partition_key_name (str): The name of the partition key attribute.
        partition_key_value (str): The value of the partition key to query.
        nested_conditions (list): A list of dictionaries, each containing:
            - path (str): The path to the nested attribute
            - operator (str): The comparison operator
            - value (any): The value to compare against
```

```
    Returns:
        dict: The response from DynamoDB containing the query results.
    """
    # Initialize the DynamoDB resource
    dynamodb = boto3.resource("dynamodb")
    table = dynamodb.Table(table_name)

    # Build the key condition expression
    key_condition = Key(partition_key_name).eq(partition_key_value)

    # Build the combined filter expression for all nested attributes
    combined_filter = None

    for condition in nested_conditions:
        if not isinstance(condition, dict):
            continue
        path = condition.get("path", "")
        operator = condition.get("operator", "")
        value = condition.get("value")

        if not path or not operator:
            continue

        # Build the individual filter expression
        current_filter = None
        if operator == "=":
            current_filter = Attr(path).eq(value)
        elif operator == "!=":
            current_filter = Attr(path).ne(value)
        elif operator == "<":
            current_filter = Attr(path).lt(value)
        elif operator == "<=":
            current_filter = Attr(path).lte(value)
        elif operator == ">":
            current_filter = Attr(path).gt(value)
        elif operator == ">=":
            current_filter = Attr(path).gte(value)
        elif operator == "contains":
            current_filter = Attr(path).contains(value)
        elif operator == "begins_with":
            current_filter = Attr(path).begins_with(value)

        # Combine with the existing filter using AND
```

```
        if current_filter:
            if combined_filter is None:
                combined_filter = current_filter
            else:
                combined_filter = combined_filter & current_filter

    # Execute the query with the combined filter expression
    response = table.query(KeyConditionExpression=key_condition,
FilterExpression=combined_filter)

    return response
```

- Weitere API-Informationen finden Sie unter Abfragen in der API-Referenz zum AWS SDK für Python (Boto3).

Eine vollständige Liste der AWS SDK-Entwicklerhandbücher und Codebeispiele finden Sie unter. DynamoDB mit einem SDK verwenden AWS Dieses Thema enthält auch Informationen zu den ersten Schritten und Details zu früheren SDK-Versionen.

Abfragen einer DynamoDB-Tabelle mit Paginierung mithilfe eines SDK AWS

In den folgenden Codebeispielen wird gezeigt, wie eine Tabelle mit Paginierung abgefragt wird.

- Implementieren der Paginierung für DynamoDB-Abfrageergebnisse
- Verwenden Sie das LastEvaluatedKey , um nachfolgende Seiten abzurufen.
- Steuern der Anzahl der Elemente pro Seite mit dem Parameter „Limit“

Java

SDK für Java 2.x

Fragen Sie eine DynamoDB-Tabelle mit Paginierung ab mit. AWS SDK for Java 2.x

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.AttributeValue;
```

```
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.model.QueryRequest;
import software.amazon.awssdk.services.dynamodb.model.QueryResponse;
import software.amazon.awssdk.services.dynamodb.model.ResourceNotFoundException;

import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;

    public List<Map<String, AttributeValue>> queryWithPagination(
        final String tableName, final String partitionKeyName, final String
 partitionKeyValue, final int pageSize) {

        CodeSampleUtils.validateTableParameters(tableName, partitionKeyName,
 partitionKeyValue);
        CodeSampleUtils.validatePositiveInteger("Page size", pageSize);

        // Create expression attribute names for the column names
        final Map<String, String> expressionAttributeNames = new HashMap<>();
        expressionAttributeNames.put(EXPRESSION_ATTRIBUTE_NAME_PK,
 partitionKeyName);

        // Create expression attribute values for the column values
        final Map<String, AttributeValue> expressionAttributeValues = new
 HashMap<>();
        expressionAttributeValues.put(
            EXPRESSION_ATTRIBUTE_VALUE_PK,
            AttributeValue.builder().s(partitionKeyValue).build());

        // Create the query request
        QueryRequest.Builder queryRequestBuilder = QueryRequest.builder()
            .tableName(tableName)
            .keyConditionExpression(KEY_CONDITION_EXPRESSION)
            .expressionAttributeNames(expressionAttributeNames)
            .expressionAttributeValues(expressionAttributeValues)
            .limit(pageSize);

        // List to store all items from all pages
        final List<Map<String, AttributeValue>> allItems = new ArrayList<>();

        // Map to store the last evaluated key for pagination
        Map<String, AttributeValue> lastEvaluatedKey = null;
        int pageNumber = 1;
```

```
        try {
            do {
                // If we have a last evaluated key, use it for the next page
                if (lastEvaluatedKey != null) {
                    queryRequestBuilder.exclusiveStartKey(lastEvaluatedKey);
                }

                // Execute the query
                final QueryResponse response =
dynamoDbClient.query(queryRequestBuilder.build());

                // Process the current page of results
                final List<Map<String, AttributeValue>> pageItems =
response.items();
                allItems.addAll(pageItems);

                // Get the last evaluated key for the next page
                lastEvaluatedKey = response.lastEvaluatedKey();
                if (lastEvaluatedKey != null && lastEvaluatedKey.isEmpty()) {
                    lastEvaluatedKey = null;
                }

                System.out.println("Page " + pageNumber + ": Retrieved " +
pageItems.size() + " items (Running total: "
                    + allItems.size() + ")");

                pageNumber++;

            } while (lastEvaluatedKey != null);

            System.out.println("Query with pagination complete. Retrieved a total
of " + allItems.size()
                + " items across " + (pageNumber - 1) + " pages");

            return allItems;
        } catch (ResourceNotFoundException e) {
            System.err.format("Error: The Amazon DynamoDB table \"%s\" can't be
found.\n", tableName);
            throw e;
        } catch (DynamoDbException e) {
            System.err.println("Error querying with pagination: " +
e.getMessage());
            throw e;
```

```
        }
    }
```

Demonstriert, wie eine DynamoDB-Tabelle mit Paginierung abgefragt wird.

```
    public static void main(String[] args) {
        final String usage =
            """
                Usage:
                    <tableName> <partitionKeyName> <partitionKeyValue> [pageSize]
[region]
                Where:
                    tableName - The Amazon DynamoDB table to query.
                    partitionKeyName - The name of the partition key attribute.
                    partitionKeyValue - The value of the partition key to query.
                    pageSize (optional) - The maximum number of items to return
per page. (Default: 10)
                    region (optional) - The AWS region where the table exists.
(Default: us-east-1)
                """;

        if (args.length < 3) {
            System.out.println(usage);
            System.exit(1);
        }

        final String tableName = args[0];
        final String partitionKeyName = args[1];
        final String partitionKeyValue = args[2];
        final int pageSize = args.length > 3 ? Integer.parseInt(args[3]) : 10;
        final Region region = args.length > 4 ? Region.of(args[4]) :
Region.US_EAST_1;

        System.out.println("Querying items with pagination (page size: " +
pageSize + ")");

        try {
            // Using the builder pattern to create and execute the query
            final List<Map<String, AttributeValue>> allItems = new
PaginationQueryBuilder()
                .withTableName(tableName)
                .withPartitionKeyName(partitionKeyName)
```

```
                .withPartitionKeyValue(partitionKeyValue)
                .withPageSize(pageSize)
                .withRegion(region)
                .executeWithPagination();

            // Process the results
            System.out.println("\nSummary: Retrieved a total of " +
allItems.size() + " items");

            // Display the first few items as a sample
            final int sampleSize = Math.min(5, allItems.size());
            if (sampleSize > 0) {
                System.out.println("\nSample of retrieved items (first " +
sampleSize + "):");
                for (int i = 0; i < sampleSize; i++) {
                    System.out.println(allItems.get(i));
                }

                if (allItems.size() > sampleSize) {
                    System.out.println("... and " + (allItems.size() -
sampleSize) + " more items");
                }
            }
        } catch (IllegalArgumentException e) {
            System.err.println("Invalid input: " + e.getMessage());
            System.exit(1);
        } catch (ResourceNotFoundException e) {
            System.err.println("Table not found: " + tableName);
            System.exit(1);
        } catch (DynamoDbException e) {
            System.err.println("DynamoDB error: " + e.getMessage());
            System.exit(1);
        } catch (Exception e) {
            System.err.println("Unexpected error: " + e.getMessage());
            System.exit(1);
        }
    }
```

- Weitere API-Informationen finden Sie unter Abfragen in der AWS SDK for Java 2.x -API-Referenz.

JavaScript

SDK für JavaScript (v3)

Fragen Sie eine DynamoDB-Tabelle mit Paginierung ab mit. AWS SDK für JavaScript

```
/**
 * Example demonstrating how to handle large query result sets in DynamoDB using
 pagination
 *
 * This example shows:
 * - How to use pagination to handle large result sets
 * - How to use LastEvaluatedKey to retrieve the next page of results
 * - How to construct subsequent query requests using ExclusiveStartKey
 */
const { DynamoDBClient, QueryCommand } = require("@aws-sdk/client-dynamodb");

/**
 * Queries a DynamoDB table with pagination to handle large result sets
 *
 * @param {Object} config - AWS SDK configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {string} partitionKeyName - The name of the partition key
 * @param {string} partitionKeyValue - The value of the partition key
 * @param {number} pageSize - Number of items per page
 * @returns {Promise<Array>} - All items from the query
 */
async function queryWithPagination(
  config,
  tableName,
  partitionKeyName,
  partitionKeyValue,
  pageSize = 25
) {
  try {
    // Create DynamoDB client
    const client = new DynamoDBClient(config);

    // Initialize variables for pagination
    let lastEvaluatedKey = undefined;
    const allItems = [];
    let pageCount = 0;
```

```
   // Loop until all pages are retrieved
   do {
     // Construct the query input
     const input = {
       TableName: tableName,
       KeyConditionExpression: "#pk = :pkValue",
       Limit: pageSize,
       ExpressionAttributeNames: {
         "#pk": partitionKeyName
       },
       ExpressionAttributeValues: {
         ":pkValue": { S: partitionKeyValue }
       }
     };

     // Add ExclusiveStartKey if we have a LastEvaluatedKey from a previous
query
     if (lastEvaluatedKey) {
       input.ExclusiveStartKey = lastEvaluatedKey;
     }

     // Execute the query
     const command = new QueryCommand(input);
     const response = await client.send(command);

     // Process the current page of results
     pageCount++;
     console.log(`Processing page ${pageCount} with ${response.Items.length}
items`);

     // Add the items from this page to our collection
     if (response.Items && response.Items.length > 0) {
       allItems.push(...response.Items);
     }

     // Get the LastEvaluatedKey for the next page
     lastEvaluatedKey = response.LastEvaluatedKey;

   } while (lastEvaluatedKey); // Continue until there are no more pages

   console.log(`Query complete. Retrieved ${allItems.length} items in
${pageCount} pages.`);
   return allItems;
 } catch (error) {
```

```
    console.error(`Error querying with pagination: ${error}`);
    throw error;
  }
}

/**
 * Example usage:
 *
 * // Query all items in the "AWS DynamoDB" forum with pagination
 * const allItems = await queryWithPagination(
 *   { region: "us-west-2" },
 *   "ForumThreads",
 *   "ForumName",
 *   "AWS DynamoDB",
 *   25 // 25 items per page
 * );
 *
 * console.log(`Total items retrieved: ${allItems.length}`);
 *
 * // Notes on pagination:
 * // - LastEvaluatedKey contains the primary key of the last evaluated item
 * // - When LastEvaluatedKey is undefined/null, there are no more items to
 retrieve
 * // - ExclusiveStartKey tells DynamoDB where to start the next page
 * // - Pagination helps manage memory usage for large result sets
 * // - Each page requires a separate network request to DynamoDB
 */

module.exports = { queryWithPagination };
```

- Weitere API-Informationen finden Sie unter Abfragen in der AWS SDK für JavaScript -API-Referenz.

Python

SDK für Python (Boto3)

Fragen Sie eine DynamoDB-Tabelle mit Paginierung ab mit. AWS SDK für Python (Boto3)

```
import boto3
from boto3.dynamodb.conditions import Key
```

```
def query_with_pagination(
    table_name, partition_key_name, partition_key_value, page_size=25,
 max_pages=None
):
    """
    Query a DynamoDB table with pagination to handle large result sets.

    Args:
        table_name (str): The name of the DynamoDB table.
        partition_key_name (str): The name of the partition key attribute.
        partition_key_value (str): The value of the partition key to query.
        page_size (int, optional): The number of items to return per page.
 Defaults to 25.
        max_pages (int, optional): The maximum number of pages to retrieve. If
 None, retrieves all pages.

    Returns:
        list: All items retrieved from the query across all pages.
    """
    # Initialize the DynamoDB resource
    dynamodb = boto3.resource("dynamodb")
    table = dynamodb.Table(table_name)

    # Initialize variables for pagination
    last_evaluated_key = None
    page_count = 0
    all_items = []

    # Paginate through the results
    while True:
        # Check if we've reached the maximum number of pages
        if max_pages is not None and page_count >= max_pages:
            break

        # Prepare the query parameters
        query_params = {
            "KeyConditionExpression":
 Key(partition_key_name).eq(partition_key_value),
            "Limit": page_size,
        }

        # Add the ExclusiveStartKey if we have a LastEvaluatedKey from a previous
 query
```

```
        if last_evaluated_key:
            query_params["ExclusiveStartKey"] = last_evaluated_key

        # Execute the query
        response = table.query(**query_params)

        # Process the current page of results
        items = response.get("Items", [])
        all_items.extend(items)

        # Update pagination tracking
        page_count += 1

        # Get the LastEvaluatedKey for the next page, if any
        last_evaluated_key = response.get("LastEvaluatedKey")

        # If there's no LastEvaluatedKey, we've reached the end of the results
        if not last_evaluated_key:
            break

    return all_items


def query_with_pagination_generator(
    table_name, partition_key_name, partition_key_value, page_size=25
):
    """
    Query a DynamoDB table with pagination using a generator to handle large
 result sets.
    This approach is memory-efficient as it yields one page at a time.

    Args:
        table_name (str): The name of the DynamoDB table.
        partition_key_name (str): The name of the partition key attribute.
        partition_key_value (str): The value of the partition key to query.
        page_size (int, optional): The number of items to return per page.
 Defaults to 25.

    Yields:
        tuple: A tuple containing (items, page_number, last_page) where:
            - items is a list of items for the current page
            - page_number is the current page number (starting from 1)
            - last_page is a boolean indicating if this is the last page
    """
```

```
    # Initialize the DynamoDB resource
    dynamodb = boto3.resource("dynamodb")
    table = dynamodb.Table(table_name)

    # Initialize variables for pagination
    last_evaluated_key = None
    page_number = 0

    # Paginate through the results
    while True:
        # Prepare the query parameters
        query_params = {
            "KeyConditionExpression":
Key(partition_key_name).eq(partition_key_value),
            "Limit": page_size,
        }

        # Add the ExclusiveStartKey if we have a LastEvaluatedKey from a previous
query
        if last_evaluated_key:
            query_params["ExclusiveStartKey"] = last_evaluated_key

        # Execute the query
        response = table.query(**query_params)

        # Get the current page of results
        items = response.get("Items", [])
        page_number += 1

        # Get the LastEvaluatedKey for the next page, if any
        last_evaluated_key = response.get("LastEvaluatedKey")

        # Determine if this is the last page
        is_last_page = last_evaluated_key is None

        # Yield the current page of results
        yield (items, page_number, is_last_page)

        # If there's no LastEvaluatedKey, we've reached the end of the results
        if is_last_page:
            break
```

- Weitere API-Informationen finden Sie unter Abfragen in der API-Referenz zum AWS SDK für Python (Boto3).

Eine vollständige Liste der AWS SDK-Entwicklerhandbücher und Codebeispiele finden Sie unter. DynamoDB mit einem SDK verwenden AWS Dieses Thema enthält auch Informationen zu den ersten Schritten und Details zu früheren SDK-Versionen.

Abfragen einer DynamoDB-Tabelle mit stark konsistenten Lesevorgängen mithilfe eines SDK AWS

In den folgenden Codebeispielen wird gezeigt, wie eine Tabelle mit strikt konsistenten Lesevorgängen abgefragt wird.

- Konfigurieren des Konsistenzniveaus für DynamoDB-Abfragen
- Verwenden Sie stark konsistente Lesevorgänge, um die meisten up-to-date Daten zu erhalten.
- Erkennen der Kompromisse zwischen letztendlicher Konsistenz und starker Konsistenz

Java

SDK für Java 2.x

Fragen Sie eine DynamoDB-Tabelle mit konfigurierbarer Lesekonsistenz ab mit. AWS SDK for Java 2.x

```
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.AttributeValue;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.model.QueryRequest;
import software.amazon.awssdk.services.dynamodb.model.QueryResponse;
import software.amazon.awssdk.services.dynamodb.model.ResourceNotFoundException;

import java.util.HashMap;
import java.util.Map;
import java.util.logging.Level;
import java.util.logging.Logger;

    public QueryResponse queryWithConsistentReads(
```

```
        final String tableName,
        final String partitionKeyName,
        final String partitionKeyValue,
        final boolean useConsistentRead) {

        CodeSampleUtils.validateTableParameters(tableName, partitionKeyName,
partitionKeyValue);

        // Create expression attribute names for the column names
        final Map<String, String> expressionAttributeNames = new HashMap<>();
        expressionAttributeNames.put(EXPRESSION_ATTRIBUTE_NAME_PK,
partitionKeyName);

        // Create expression attribute values for the column values
        final Map<String, AttributeValue> expressionAttributeValues = new
HashMap<>();
        expressionAttributeValues.put(
            EXPRESSION_ATTRIBUTE_VALUE_PK,
            AttributeValue.builder().s(partitionKeyValue).build());

        // Create the query request
        final QueryRequest queryRequest = QueryRequest.builder()
            .tableName(tableName)
            .keyConditionExpression(KEY_CONDITION_EXPRESSION)
            .expressionAttributeNames(expressionAttributeNames)
            .expressionAttributeValues(expressionAttributeValues)
            .consistentRead(useConsistentRead)
            .build();

        try {
            final QueryResponse response = dynamoDbClient.query(queryRequest);
            LOGGER.log(Level.INFO, "Query successful. Found {0} items",
response.count());
            return response;
        } catch (ResourceNotFoundException e) {
            LOGGER.log(Level.SEVERE, "Table not found: {0}", tableName);
            throw e;
        } catch (DynamoDbException e) {
            LOGGER.log(Level.SEVERE, "Error querying with consistent reads", e);
            throw e;
        }
    }
```

- Weitere API-Informationen finden Sie unter Abfragen in der AWS SDK for Java 2.x -API-Referenz.

JavaScript

SDK für JavaScript (v3)

Fragen Sie eine DynamoDB-Tabelle mit konfigurierbarer Lesekonsistenz ab mit. AWS SDK für JavaScript

```
const { DynamoDBClient, QueryCommand } = require("@aws-sdk/client-dynamodb");

/**
 * Queries a DynamoDB table with configurable read consistency
 *
 * @param {Object} config - AWS SDK configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {string} partitionKeyName - The name of the partition key
 * @param {string} partitionKeyValue - The value of the partition key
 * @param {boolean} useConsistentRead - Whether to use strongly consistent reads
 * @returns {Promise<Object>} - The query response
 */
async function queryWithConsistentRead(
  config,
  tableName,
  partitionKeyName,
  partitionKeyValue,
  useConsistentRead = false
) {
  try {
    // Create DynamoDB client
    const client = new DynamoDBClient(config);

    // Construct the query input
    const input = {
      TableName: tableName,
      KeyConditionExpression: "#pk = :pkValue",
      ExpressionAttributeNames: {
        "#pk": partitionKeyName
      },
      ExpressionAttributeValues: {
        ":pkValue": { S: partitionKeyValue }
      },
```

```
      ConsistentRead: useConsistentRead
    };

    // Execute the query
    const command = new QueryCommand(input);
    return await client.send(command);
  } catch (error) {
    console.error(`Error querying with consistent read: ${error}`);
    throw error;
  }
}
```

- Weitere API-Informationen finden Sie unter Abfragen in der AWS SDK für JavaScript -API-Referenz.

Python

SDK für Python (Boto3)

Fragen Sie eine DynamoDB-Tabelle mit der Option für stark konsistente Lesevorgänge mit ab. AWS SDK für Python (Boto3)

```
import time

import boto3
from boto3.dynamodb.conditions import Key


def query_with_consistent_read(
    table_name,
    partition_key_name,
    partition_key_value,
    sort_key_name=None,
    sort_key_value=None,
    consistent_read=True,
):
    """
    Query a DynamoDB table with the option for strongly consistent reads.

    Args:
        table_name (str): The name of the DynamoDB table.
```

```
        partition_key_name (str): The name of the partition key attribute.
        partition_key_value (str): The value of the partition key to query.
        sort_key_name (str, optional): The name of the sort key attribute.
        sort_key_value (str, optional): The value of the sort key to query.
        consistent_read (bool, optional): Whether to use strongly consistent
reads. Defaults to True.

    Returns:
        dict: The response from DynamoDB containing the query results.
    """
    # Initialize the DynamoDB resource
    dynamodb = boto3.resource("dynamodb")
    table = dynamodb.Table(table_name)

    # Build the key condition expression
    key_condition = Key(partition_key_name).eq(partition_key_value)

    if sort_key_name and sort_key_value:
        key_condition = key_condition & Key(sort_key_name).eq(sort_key_value)

    # Perform the query with the consistent read option
    response = table.query(KeyConditionExpression=key_condition,
ConsistentRead=consistent_read)

    return response
```

- Weitere API-Informationen finden Sie unter Abfragen in der API-Referenz zum AWS SDK für Python (Boto3).

Eine vollständige Liste der AWS SDK-Entwicklerhandbücher und Codebeispiele finden Sie unter. DynamoDB mit einem SDK verwenden AWS Dieses Thema enthält auch Informationen zu den ersten Schritten und Details zu früheren SDK-Versionen.

Abfragen von DynamoDB-Daten mithilfe von PartiQL SELECT-Anweisungen mit einem SDK AWS

Im folgenden Codebeispiel wird gezeigt, wie Sie Daten mit PartiQL-SELECT-Anweisungen abfragen.

JavaScript

SDK für JavaScript (v3)

Fragen Sie Elemente aus einer DynamoDB-Tabelle mithilfe von PartiQL SELECT-Anweisungen mit ab. AWS SDK für JavaScript

```
/**
 * This example demonstrates how to query items from a DynamoDB table using
 PartiQL.
 * It shows different ways to select data with various index types.
 */
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";
import {
  DynamoDBDocumentClient,
  ExecuteStatementCommand,
  BatchExecuteStatementCommand,
} from "@aws-sdk/lib-dynamodb";

/**
 * Select all items from a DynamoDB table using PartiQL.
 * Note: This should be used with caution on large tables.
 *
 * @param tableName - The name of the DynamoDB table
 * @returns The response from the ExecuteStatementCommand
 */
export const selectAllItems = async (tableName: string) => {
  const client = new DynamoDBClient({});
  const docClient = DynamoDBDocumentClient.from(client);

  const params = {
    Statement: `SELECT * FROM "${tableName}"`,
  };

  try {
    const data = await docClient.send(new ExecuteStatementCommand(params));
    console.log("Items retrieved successfully");
    return data;
  } catch (err) {
    console.error("Error retrieving items:", err);
    throw err;
  }
};
```

```
/**
 * Select an item by its primary key using PartiQL.
 *
 * @param tableName - The name of the DynamoDB table
 * @param partitionKeyName - The name of the partition key attribute
 * @param partitionKeyValue - The value of the partition key
 * @returns The response from the ExecuteStatementCommand
 */
export const selectItemByPartitionKey = async (
  tableName: string,
  partitionKeyName: string,
  partitionKeyValue: string | number
) => {
  const client = new DynamoDBClient({});
  const docClient = DynamoDBDocumentClient.from(client);

  const params = {
    Statement: `SELECT * FROM "${tableName}" WHERE ${partitionKeyName} = ?`,
    Parameters: [partitionKeyValue],
  };

  try {
    const data = await docClient.send(new ExecuteStatementCommand(params));
    console.log("Item retrieved successfully");
    return data;
  } catch (err) {
    console.error("Error retrieving item:", err);
    throw err;
  }
};

/**
 * Select an item by its composite key (partition key + sort key) using PartiQL.
 *
 * @param tableName - The name of the DynamoDB table
 * @param partitionKeyName - The name of the partition key attribute
 * @param partitionKeyValue - The value of the partition key
 * @param sortKeyName - The name of the sort key attribute
 * @param sortKeyValue - The value of the sort key
 * @returns The response from the ExecuteStatementCommand
 */
export const selectItemByCompositeKey = async (
  tableName: string,
```

```
  partitionKeyName: string,
  partitionKeyValue: string | number,
  sortKeyName: string,
  sortKeyValue: string | number
) => {
  const client = new DynamoDBClient({});
  const docClient = DynamoDBDocumentClient.from(client);

  const params = {
    Statement: `SELECT * FROM "${tableName}" WHERE ${partitionKeyName} = ? AND
 ${sortKeyName} = ?`,
    Parameters: [partitionKeyValue, sortKeyValue],
  };

  try {
    const data = await docClient.send(new ExecuteStatementCommand(params));
    console.log("Item retrieved successfully");
    return data;
  } catch (err) {
    console.error("Error retrieving item:", err);
    throw err;
  }
};

/**
 * Select items using a filter condition with PartiQL.
 *
 * @param tableName - The name of the DynamoDB table
 * @param filterAttribute - The attribute to filter on
 * @param filterValue - The value to filter by
 * @returns The response from the ExecuteStatementCommand
 */
export const selectItemsWithFilter = async (
  tableName: string,
  filterAttribute: string,
  filterValue: string | number
) => {
  const client = new DynamoDBClient({});
  const docClient = DynamoDBDocumentClient.from(client);

  const params = {
    Statement: `SELECT * FROM "${tableName}" WHERE ${filterAttribute} = ?`,
    Parameters: [filterValue],
  };
```

```
  try {
    const data = await docClient.send(new ExecuteStatementCommand(params));
    console.log("Items retrieved successfully");
    return data;
  } catch (err) {
    console.error("Error retrieving items:", err);
    throw err;
  }
};

/**
 * Select items using a begins_with function for prefix matching.
 * This is useful for querying hierarchical data.
 *
 * @param tableName - The name of the DynamoDB table
 * @param attributeName - The attribute to check for prefix
 * @param prefix - The prefix to match
 * @returns The response from the ExecuteStatementCommand
 */
export const selectItemsByPrefix = async (
  tableName: string,
  attributeName: string,
  prefix: string
) => {
  const client = new DynamoDBClient({});
  const docClient = DynamoDBDocumentClient.from(client);

  const params = {
    Statement: `SELECT * FROM "${tableName}" WHERE
 begins_with(${attributeName}, ?)`,
    Parameters: [prefix],
  };

  try {
    const data = await docClient.send(new ExecuteStatementCommand(params));
    console.log("Items retrieved successfully");
    return data;
  } catch (err) {
    console.error("Error retrieving items:", err);
    throw err;
  }
};
```

```
/**
 * Select items using a between condition for range queries.
 *
 * @param tableName - The name of the DynamoDB table
 * @param attributeName - The attribute to check for range
 * @param startValue - The start value of the range
 * @param endValue - The end value of the range
 * @returns The response from the ExecuteStatementCommand
 */
export const selectItemsByRange = async (
  tableName: string,
  attributeName: string,
  startValue: number | string,
  endValue: number | string
) => {
  const client = new DynamoDBClient({});
  const docClient = DynamoDBDocumentClient.from(client);

  const params = {
    Statement: `SELECT * FROM "${tableName}" WHERE ${attributeName} BETWEEN ?
 AND ?`,
    Parameters: [startValue, endValue],
  };

  try {
    const data = await docClient.send(new ExecuteStatementCommand(params));
    console.log("Items retrieved successfully");
    return data;
  } catch (err) {
    console.error("Error retrieving items:", err);
    throw err;
  }
};

/**
 * Example usage showing how to select items with different index types
 */
export const selectExamples = async () => {
  // Select all items from a table (use with caution on large tables)
  await selectAllItems("UsersTable");

  // Select by partition key (simple primary key)
  await selectItemByPartitionKey("UsersTable", "userId", "user123");
```

```
  // Select by composite key (partition key + sort key)
  await selectItemByCompositeKey("OrdersTable", "orderId", "order456",
 "productId", "prod789");

  // Select with a filter condition (can use any attribute)
  await selectItemsWithFilter("UsersTable", "userType", "premium");

  // Select items with a prefix (useful for hierarchical data)
  await selectItemsByPrefix("ProductsTable", "category", "electronics");

  // Select items within a range (useful for numeric or date ranges)
  await selectItemsByRange("OrdersTable", "orderDate", "2023-01-01",
 "2023-12-31");
};
```

- Weitere API-Informationen finden Sie in den folgenden Themen der AWS SDK für JavaScript -API-Referenz.
  - BatchExecuteStatement
  - ExecuteStatement

Eine vollständige Liste der AWS SDK-Entwicklerhandbücher und Codebeispiele finden Sie unter. DynamoDB mit einem SDK verwenden AWS Dieses Thema enthält auch Informationen zu den ersten Schritten und Details zu früheren SDK-Versionen.

Abfragen einer DynamoDB-Tabelle nach TTL-Elementen mithilfe eines SDK AWS

In den folgenden Codebeispielen wird gezeigt, wie TTL-Elemente abgefragt werden.

Java

SDK für Java 2.x

Abfragen eines gefilterten Ausdrucks zum Sammeln von TTL-Elementen in einer DynamoDB-Tabelle mithilfe von. AWS SDK for Java 2.x

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.AttributeValue;
```

```
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.model.QueryRequest;
import software.amazon.awssdk.services.dynamodb.model.QueryResponse;
import software.amazon.awssdk.services.dynamodb.model.ResourceNotFoundException;

import java.util.Map;
import java.util.Optional;

        final QueryRequest request = QueryRequest.builder()
            .tableName(tableName)
            .keyConditionExpression(KEY_CONDITION_EXPRESSION)
            .filterExpression(FILTER_EXPRESSION)
            .expressionAttributeNames(expressionAttributeNames)
            .expressionAttributeValues(expressionAttributeValues)
            .build();

        try (DynamoDbClient ddb = dynamoDbClient != null
            ? dynamoDbClient
            : DynamoDbClient.builder().region(region).build()) {
            final QueryResponse response = ddb.query(request);
            System.out.println("Query successful. Found " + response.count() + "
 items that have not expired yet.");

            // Print each item
            response.items().forEach(item -> {
                System.out.println("Item: " + item);
            });

            return 0;
        } catch (ResourceNotFoundException e) {
            System.err.format(TABLE_NOT_FOUND_ERROR, tableName);
            throw e;
        } catch (DynamoDbException e) {
            System.err.println(e.getMessage());
            throw e;
        }
```

- Weitere API-Informationen finden Sie unter Abfragen in der AWS SDK for Java 2.x -API-Referenz.

JavaScript

SDK für JavaScript (v3)

Abfragen eines gefilterten Ausdrucks zum Sammeln von TTL-Elementen in einer DynamoDB-Tabelle mithilfe von. AWS SDK für JavaScript

```
import { DynamoDBClient, QueryCommand } from "@aws-sdk/client-dynamodb";
import { marshall, unmarshall } from "@aws-sdk/util-dynamodb";

export const queryFiltered = async (tableName, primaryKey, region = 'us-east-1')
 => {
    const client = new DynamoDBClient({
        region: region,
        endpoint: `https://dynamodb.${region}.amazonaws.com`
    });

    const currentTime = Math.floor(Date.now() / 1000);

    const params = {
        TableName: tableName,
        KeyConditionExpression: "#pk = :pk",
        FilterExpression: "#ea > :ea",
        ExpressionAttributeNames: {
            "#pk": "primaryKey",
            "#ea": "expireAt"
        },
        ExpressionAttributeValues: marshall({
            ":pk": primaryKey,
            ":ea": currentTime
        })
    };

    try {
        const { Items } = await client.send(new QueryCommand(params));
        Items.forEach(item => {
            console.log(unmarshall(item))
        });
        return Items;
    } catch (err) {
        console.error(`Error querying items: ${err}`);
        throw err;
    }
}
```

```
// Example usage (commented out for testing)
// queryFiltered('your-table-name', 'your-partition-key-value');
```

- Weitere API-Informationen finden Sie unter Abfragen in der AWS SDK für JavaScript -API-Referenz.

Python

SDK für Python (Boto3)

Abfragen eines gefilterten Ausdrucks zum Sammeln von TTL-Elementen in einer DynamoDB-Tabelle mithilfe von. AWS SDK für Python (Boto3)

```
from datetime import datetime

import boto3


def query_dynamodb_items(table_name, partition_key):
    """

    :param table_name: Name of the DynamoDB table
    :param partition_key:
    :return:
    """
    try:
        # Initialize a DynamoDB resource
        dynamodb = boto3.resource("dynamodb", region_name="us-east-1")

        # Specify your table
        table = dynamodb.Table(table_name)

        # Get the current time in epoch format
        current_time = int(datetime.now().timestamp())

        # Perform the query operation with a filter expression to exclude expired
 items
        # response = table.query(
        #
 KeyConditionExpression=boto3.dynamodb.conditions.Key('partitionKey').eq(partition_key),
```

```
        #
 FilterExpression=boto3.dynamodb.conditions.Attr('expireAt').gt(current_time)
        # )
        response = table.query(

 KeyConditionExpression=dynamodb.conditions.Key("partitionKey").eq(partition_key),

 FilterExpression=dynamodb.conditions.Attr("expireAt").gt(current_time),
        )

        # Print the items that are not expired
        for item in response["Items"]:
            print(item)

    except Exception as e:
        print(f"Error querying items: {e}")


# Call the function with your values
query_dynamodb_items("Music", "your-partition-key-value")
```

- Weitere API-Informationen finden Sie unter Abfragen in der API-Referenz zum AWS SDK für Python (Boto3).

Eine vollständige Liste der AWS SDK-Entwicklerhandbücher und Codebeispiele finden Sie unter. DynamoDB mit einem SDK verwenden AWS Dieses Thema enthält auch Informationen zu den ersten Schritten und Details zu früheren SDK-Versionen.

Abfragen von DynamoDB-Tabellen mithilfe von Datums- und Uhrzeitmustern mit einem SDK AWS

In den folgenden Codebeispielen wird gezeigt, wie Datums- und Uhrzeitmuster zum Abfragen von Tabellen verwendet werden.

- date/time Werte in DynamoDB speichern und abfragen.
- Implementieren von Datumsbereichsabfragen mithilfe von Sortierschlüsseln
- Formatieren von Datumszeichenfolgen für effektive Abfragen

Java

SDK für Java 2.x

Abfrage anhand von Datumsbereichen in Sortierschlüsseln mit AWS SDK for Java 2.x.

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.AttributeValue;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.model.QueryRequest;
import software.amazon.awssdk.services.dynamodb.model.QueryResponse;
import software.amazon.awssdk.services.dynamodb.model.ResourceNotFoundException;

import java.time.LocalDate;
import java.util.HashMap;
import java.util.Map;
import java.util.logging.Level;
import java.util.logging.Logger;

    public QueryResponse queryWithDateRange(
        final String tableName,
        final String partitionKeyName,
        final String partitionKeyValue,
        final String dateKeyName,
        final LocalDate startDate,
        final LocalDate endDate) {

        // Focus on query logic, assuming parameters are valid
        if (startDate == null || endDate == null) {
            throw new IllegalArgumentException("Start date and end date cannot be
 null");
        }

        if (endDate.isBefore(startDate)) {
            throw new IllegalArgumentException("End date must be after start
 date");
        }

        // Format dates as ISO strings for DynamoDB (using just the date part)
        final String formattedStartDate = startDate.toString();
        final String formattedEndDate = endDate.toString();

        // Create expression attribute names for the column names
```

```
        final Map<String, String> expressionAttributeNames = new HashMap<>();
        expressionAttributeNames.put(EXPRESSION_ATTRIBUTE_NAME_PK,
partitionKeyName);
        expressionAttributeNames.put(EXPRESSION_ATTRIBUTE_NAME_SK, dateKeyName);

        // Create expression attribute values for the column values
        final Map<String, AttributeValue> expressionAttributeValues = new
HashMap<>();
        expressionAttributeValues.put(
            EXPRESSION_ATTRIBUTE_VALUE_PK,
            AttributeValue.builder().s(partitionKeyValue).build());
        expressionAttributeValues.put(
            EXPRESSION_ATTRIBUTE_VALUE_START_DATE,
            AttributeValue.builder().s(formattedStartDate).build());
        expressionAttributeValues.put(
            EXPRESSION_ATTRIBUTE_VALUE_END_DATE,
            AttributeValue.builder().s(formattedEndDate).build());

        // Create the query request
        final QueryRequest queryRequest = QueryRequest.builder()
            .tableName(tableName)
            .keyConditionExpression(KEY_CONDITION_EXPRESSION)
            .expressionAttributeNames(expressionAttributeNames)
            .expressionAttributeValues(expressionAttributeValues)
            .build();

        try {
            final QueryResponse response = dynamoDbClient.query(queryRequest);
            LOGGER.log(Level.INFO, "Query by date range successful. Found {0}
items", response.count());
            return response;
        } catch (ResourceNotFoundException e) {
            LOGGER.log(Level.SEVERE, "Table not found: {0}", tableName);
            throw e;
        } catch (DynamoDbException e) {
            LOGGER.log(Level.SEVERE, "Error querying by date range: {0}",
e.getMessage());
            throw e;
        }
    }
```

Abfrage mit Datums- und Uhrzeitvariablen mit. AWS SDK for Java 2.x

```
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.AttributeValue;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.model.QueryRequest;
import software.amazon.awssdk.services.dynamodb.model.QueryResponse;
import software.amazon.awssdk.services.dynamodb.model.ResourceNotFoundException;

import java.time.Instant;
import java.time.LocalDateTime;
import java.time.ZoneOffset;
import java.util.HashMap;
import java.util.Map;

    public QueryResponse queryWithDateTime(
        final String tableName,
        final String partitionKeyName,
        final String partitionKeyValue,
        final String dateKeyName,
        final String startDate,
        final String endDate) {

        CodeSampleUtils.validateTableParameters(tableName, partitionKeyName,
 partitionKeyValue);
        CodeSampleUtils.validateDateRangeParameters(dateKeyName, startDate,
 endDate);
        CodeSampleUtils.validateDateFormat("Start date", startDate);
        CodeSampleUtils.validateDateFormat("End date", endDate);

        // Create expression attribute names for the column names
        final Map<String, String> expressionAttributeNames = new HashMap<>();
        expressionAttributeNames.put(EXPRESSION_ATTRIBUTE_NAME_PK,
 partitionKeyName);
        expressionAttributeNames.put("#dateKey", dateKeyName);

        // Create expression attribute values for the column values
        final Map<String, AttributeValue> expressionAttributeValues = new
 HashMap<>();
        expressionAttributeValues.put(
            EXPRESSION_ATTRIBUTE_VALUE_PK,
            AttributeValue.builder().s(partitionKeyValue).build());
        expressionAttributeValues.put(
            ":startDate", AttributeValue.builder().s(startDate).build());
        expressionAttributeValues.put(
```

```
            ":endDate", AttributeValue.builder().s(endDate).build());

        // Create the query request
        final QueryRequest queryRequest = QueryRequest.builder()
            .tableName(tableName)
            .keyConditionExpression(KEY_CONDITION_EXPRESSION)
            .expressionAttributeNames(expressionAttributeNames)
            .expressionAttributeValues(expressionAttributeValues)
            .build();

        try {
            final QueryResponse response = dynamoDbClient.query(queryRequest);
            System.out.println("Query successful. Found " + response.count() + "
items");
            return response;
        } catch (ResourceNotFoundException e) {
            System.err.format("Error: The Amazon DynamoDB table \"%s\" can't be
found.\n", tableName);
            throw e;
        } catch (DynamoDbException e) {
            System.err.println("Error querying with date range: " +
e.getMessage());
            throw e;
        }
    }
```

Führen Sie mit AWS SDK for Java 2.x Abfragen innerhalb von Datumsbereichen in Unix-Epochenzeitstempeln aus.

```
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.AttributeValue;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.model.QueryRequest;
import software.amazon.awssdk.services.dynamodb.model.QueryResponse;
import software.amazon.awssdk.services.dynamodb.model.ResourceNotFoundException;

import java.time.Instant;
import java.time.LocalDateTime;
import java.time.ZoneOffset;
import java.util.HashMap;
import java.util.Map;
```

```
   public QueryResponse queryWithDateTimeEpoch(
       final String tableName,
       final String partitionKeyName,
       final String partitionKeyValue,
       final String dateKeyName,
       final long startEpoch,
       final long endEpoch) {

       CodeSampleUtils.validateTableParameters(tableName, partitionKeyName,
partitionKeyValue);
       CodeSampleUtils.validateStringParameter("Date key name", dateKeyName);
       CodeSampleUtils.validateEpochTimestamp("Start epoch", startEpoch);
       CodeSampleUtils.validateEpochTimestamp("End epoch", endEpoch);

       // Create expression attribute names for the column names
       final Map<String, String> expressionAttributeNames = new HashMap<>();
       expressionAttributeNames.put(EXPRESSION_ATTRIBUTE_NAME_PK,
partitionKeyName);
       expressionAttributeNames.put("#dateKey", dateKeyName);

       // Create expression attribute values for the column values
       final Map<String, AttributeValue> expressionAttributeValues = new
HashMap<>();
       expressionAttributeValues.put(
           EXPRESSION_ATTRIBUTE_VALUE_PK,
           AttributeValue.builder().s(partitionKeyValue).build());
       expressionAttributeValues.put(
           ":startDate",
AttributeValue.builder().n(String.valueOf(startEpoch)).build());
       expressionAttributeValues.put(
           ":endDate",
AttributeValue.builder().n(String.valueOf(endEpoch)).build());

       // Create the query request
       final QueryRequest queryRequest = QueryRequest.builder()
           .tableName(tableName)
           .keyConditionExpression(KEY_CONDITION_EXPRESSION)
           .expressionAttributeNames(expressionAttributeNames)
           .expressionAttributeValues(expressionAttributeValues)
           .build();

       try {
           final QueryResponse response = dynamoDbClient.query(queryRequest);
```

```
            System.out.println("Query successful. Found " + response.count() + "
 items");
            return response;
        } catch (ResourceNotFoundException e) {
            System.err.format("Error: The Amazon DynamoDB table \"%s\" can't be
 found.\n", tableName);
            throw e;
        } catch (DynamoDbException e) {
            System.err.println("Error querying with epoch timestamps: " +
 e.getMessage());
            throw e;
        }
    }
```

Abfrage innerhalb von Datumsbereichen mithilfe von LocalDateTime Objekten mit. AWS SDK for Java 2.x

```
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.AttributeValue;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.model.QueryRequest;
import software.amazon.awssdk.services.dynamodb.model.QueryResponse;
import software.amazon.awssdk.services.dynamodb.model.ResourceNotFoundException;

import java.time.Instant;
import java.time.LocalDateTime;
import java.time.ZoneOffset;
import java.util.HashMap;
import java.util.Map;

    public QueryResponse queryWithDateTimeLocalDateTime(
        final String tableName,
        final String partitionKeyName,
        final String partitionKeyValue,
        final String dateKeyName,
        final LocalDateTime startDateTime,
        final LocalDateTime endDateTime) {

        CodeSampleUtils.validateTableParameters(tableName, partitionKeyName,
 partitionKeyValue);
        CodeSampleUtils.validateStringParameter("Date key name", dateKeyName);
        if (startDateTime == null || endDateTime == null) {
```

```
            throw new IllegalArgumentException("Start and end LocalDateTime must
 not be null");
        }

        // Convert LocalDateTime to ISO-8601 strings in UTC with the correct
 format
        final String startDate =
 startDateTime.atZone(ZoneOffset.UTC).format(DATE_TIME_FORMATTER);
        final String endDate =
 endDateTime.atZone(ZoneOffset.UTC).format(DATE_TIME_FORMATTER);

        return queryWithDateTime(tableName, partitionKeyName, partitionKeyValue,
 dateKeyName, startDate, endDate);
    }
```

- Weitere API-Informationen finden Sie unter Abfragen in der AWS SDK for Java 2.x -API-Referenz.

JavaScript

SDK für JavaScript (v3)

Abfrage unter Verwendung von Datumsbereichen in Sortierschlüsseln mit AWS SDK für JavaScript.

```
const { DynamoDBClient, QueryCommand } = require("@aws-sdk/client-dynamodb");

/**
 * Queries a DynamoDB table for items within a specific date range on the sort
 key
 *
 * @param {Object} config - AWS SDK configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {string} partitionKeyName - The name of the partition key
 * @param {string} partitionKeyValue - The value of the partition key
 * @param {string} sortKeyName - The name of the sort key (must be a date/time
 attribute)
 * @param {Date} startDate - The start date for the range query
 * @param {Date} endDate - The end date for the range query
 * @returns {Promise<Object>} - The query response
 */
```

```
async function queryByDateRangeOnSortKey(
  config,
  tableName,
  partitionKeyName,
  partitionKeyValue,
  sortKeyName,
  startDate,
  endDate
) {
  try {
    // Create DynamoDB client
    const client = new DynamoDBClient(config);

    // Format dates as ISO strings for DynamoDB
    const formattedStartDate = startDate.toISOString();
    const formattedEndDate = endDate.toISOString();

    // Construct the query input
    const input = {
      TableName: tableName,
      KeyConditionExpression: '#pk = :pkValue AND #sk BETWEEN :startDate
 AND :endDate',
      ExpressionAttributeNames: {
        "#pk": partitionKeyName,
        "#sk": sortKeyName
      },
      ExpressionAttributeValues: {
        ":pkValue": { S: partitionKeyValue },
        ":startDate": { S: formattedStartDate },
        ":endDate": { S: formattedEndDate }
      }
    };

    // Execute the query
    const command = new QueryCommand(input);
    return await client.send(command);
  } catch (error) {
    console.error(`Error querying by date range on sort key: ${error}`);
    throw error;
  }
}
```

Abfrage mit Datums- und Uhrzeitvariablen mit. AWS SDK für JavaScript

```
const { DynamoDBClient, QueryCommand } = require("@aws-sdk/client-dynamodb");

/**
 * Queries a DynamoDB table for items within a specific date range
 *
 * @param {Object} config - AWS SDK configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {string} partitionKeyName - The name of the partition key
 * @param {string} partitionKeyValue - The value of the partition key
 * @param {string} dateKeyName - The name of the date attribute to filter on
 * @param {Date} startDate - The start date for the range query
 * @param {Date} endDate - The end date for the range query
 * @returns {Promise<Object>} - The query response
 */
async function queryByDateRange(
  config,
  tableName,
  partitionKeyName,
  partitionKeyValue,
  dateKeyName,
  startDate,
  endDate
) {
  try {
    // Create DynamoDB client
    const client = new DynamoDBClient(config);

    // Format dates as ISO strings for DynamoDB
    const formattedStartDate = startDate.toISOString();
    const formattedEndDate = endDate.toISOString();

    // Construct the query input
    const input = {
      TableName: tableName,
      KeyConditionExpression: `#pk = :pkValue AND #dateAttr BETWEEN :startDate
 AND :endDate`,
      ExpressionAttributeNames: {
        "#pk": partitionKeyName,
        "#dateAttr": dateKeyName
      },
      ExpressionAttributeValues: {
        ":pkValue": { S: partitionKeyValue },
```

```
        ":startDate": { S: formattedStartDate },
        ":endDate": { S: formattedEndDate }
      }
    };

    // Execute the query
    const command = new QueryCommand(input);
    return await client.send(command);
  } catch (error) {
    console.error(`Error querying by date range: ${error}`);
    throw error;
  }
}
```

- Weitere API-Informationen finden Sie unter Abfragen in der AWS SDK für JavaScript -API-Referenz.

Python

SDK für Python (Boto3)

Abfrage unter Verwendung von Datumsbereichen in Sortierschlüsseln mit. AWS SDK für Python (Boto3)

```
from datetime import datetime, timedelta

import boto3
from boto3.dynamodb.conditions import Key


def query_with_date_range(
    table_name, partition_key_name, partition_key_value, sort_key_name,
 start_date, end_date
):
    """
    Query a DynamoDB table with a date range on the sort key.

    Args:
        table_name (str): The name of the DynamoDB table.
        partition_key_name (str): The name of the partition key attribute.
        partition_key_value (str): The value of the partition key to query.
```

```
        sort_key_name (str): The name of the sort key attribute (containing date
 values).
        start_date (datetime): The start date for the query range.
        end_date (datetime): The end date for the query range.

    Returns:
        dict: The response from DynamoDB containing the query results.
    """
    # Initialize the DynamoDB resource
    dynamodb = boto3.resource("dynamodb")
    table = dynamodb.Table(table_name)

    # Format the date values as ISO 8601 strings
    # DynamoDB works well with ISO format for date values
    start_date_str = start_date.isoformat()
    end_date_str = end_date.isoformat()

    # Perform the query with a date range on the sort key using BETWEEN operator
    key_condition = Key(partition_key_name).eq(partition_key_value) &
 Key(sort_key_name).between(
        start_date_str, end_date_str
    )

    response = table.query(
        KeyConditionExpression=key_condition,
        ExpressionAttributeValues={
            ":pk_val": partition_key_value,
            ":start_date": start_date_str,
            ":end_date": end_date_str,
        },
    )

    return response


def query_with_date_range_by_month(
    table_name, partition_key_name, partition_key_value, sort_key_name, year,
 month
):
    """
    Query a DynamoDB table for a specific month's data.

    Args:
        table_name (str): The name of the DynamoDB table.
```

```
        partition_key_name (str): The name of the partition key attribute.
        partition_key_value (str): The value of the partition key to query.
        sort_key_name (str): The name of the sort key attribute (containing date
values).
        year (int): The year to query.
        month (int): The month to query (1-12).

    Returns:
        dict: The response from DynamoDB containing the query results.
    """
    # Initialize the DynamoDB resource
    dynamodb = boto3.resource("dynamodb")
    table = dynamodb.Table(table_name)

    # Calculate the start and end dates for the specified month
    if month == 12:
        next_year = year + 1
        next_month = 1
    else:
        next_year = year
        next_month = month + 1

    start_date = datetime(year, month, 1)
    end_date = datetime(next_year, next_month, 1) - timedelta(microseconds=1)

    # Format the date values as ISO 8601 strings
    start_date_str = start_date.isoformat()
    end_date_str = end_date.isoformat()

    # Perform the query with a date range on the sort key
    key_condition = Key(partition_key_name).eq(partition_key_value) &
Key(sort_key_name).between(
        start_date_str, end_date_str
    )

    response = table.query(KeyConditionExpression=key_condition)

    return response
```

Abfrage mit Datums- und Uhrzeitvariablen mit. AWS SDK für Python (Boto3)

```
from datetime import datetime, timedelta

import boto3
from boto3.dynamodb.conditions import Key


def query_with_datetime(
    table_name, partition_key_name, partition_key_value, sort_key_name,
 start_date, end_date
):
    """
    Query a DynamoDB table with a date range filter on the sort key.

    Args:
        table_name (str): The name of the DynamoDB table.
        partition_key_name (str): The name of the partition key attribute.
        partition_key_value (str): The value of the partition key to query.
        sort_key_name (str): The name of the sort key attribute (containing date/
time values).
        start_date (datetime): The start date/time for the query range.
        end_date (datetime): The end date/time for the query range.

    Returns:
        dict: The response from DynamoDB containing the query results.
    """
    # Initialize the DynamoDB resource
    dynamodb = boto3.resource("dynamodb")
    table = dynamodb.Table(table_name)

    # Format the date/time values as ISO 8601 strings
    # DynamoDB works well with ISO format for date/time values
    start_date_str = start_date.isoformat()
    end_date_str = end_date.isoformat()

    # Perform the query with a date range on the sort key
    key_condition = Key(partition_key_name).eq(partition_key_value) &
 Key(sort_key_name).between(
        start_date_str, end_date_str
    )

    response = table.query(
        KeyConditionExpression=key_condition,
        ExpressionAttributeValues={
```

```
            ":pk_val": partition_key_value,
            ":start_date": start_date_str,
            ":end_date": end_date_str,
        },
    )

    return response



def example_usage():
    """Example of how to use the query_with_datetime function."""
    # Example parameters
    table_name = "Events"
    partition_key_name = "EventType"
    partition_key_value = "UserLogin"
    sort_key_name = "Timestamp"

    # Create date/time variables for the query
    end_date = datetime.now()
    start_date = end_date - timedelta(days=7)  # Query events from the last 7
 days

    print(f"Querying events from {start_date.isoformat()} to
 {end_date.isoformat()}")

    # Execute the query
    response = query_with_datetime(
        table_name, partition_key_name, partition_key_value, sort_key_name,
 start_date, end_date
    )

    # Process the results
    items = response.get("Items", [])
    print(f"Found {len(items)} items")

    for item in items:
        print(f"Event: {item}")
```

- Weitere API-Informationen finden Sie unter Abfragen in der API-Referenz zum AWS SDK für Python (Boto3).

Eine vollständige Liste der AWS SDK-Entwicklerhandbücher und Codebeispiele finden Sie unter. DynamoDB mit einem SDK verwenden AWS Dieses Thema enthält auch Informationen zu den ersten Schritten und Details zu früheren SDK-Versionen.

Speichern Sie EXIF und andere Bildinformationen mit einem SDK AWS

Wie das aussehen kann, sehen Sie am nachfolgenden Beispielcode:

- Rufen Sie EXIF-Informationen aus einer JPG-, JPEG- oder PNG-Datei ab.
- Laden Sie die Bilddatei in einen Amazon-S3-Bucket hoch.
- Verwenden Sie Amazon Rekognition, um die drei wichtigsten Attribute (Labels) in der Datei zu identifizieren.
- Fügen Sie die EXIF- und Label-Informationen einer Amazon-DynamoDB-Tabelle in der Region hinzu.

Rust

SDK für Rust

Rufen Sie EXIF-Informationen aus einer JPG-, JPEG- oder PNG-Datei ab, laden Sie die Bilddatei in einen Amazon-S3-Bucket hoch und identifizieren Sie mit Amazon Rekognition die drei wichtigsten Attribute (Labels in Amazon Rekognition) in der Datei. Fügen Sie die EXIF- und Labelinformationen dann einer Amazon-DynamoDB-Tabelle in der Region hinzu.

Den vollständigen Quellcode und Anweisungen zur Einrichtung und Ausführung finden Sie im vollständigen Beispiel unter GitHub.

In diesem Beispiel verwendete Dienste

- DynamoDB
- Amazon Rekognition
- Amazon S3

Eine vollständige Liste der AWS SDK-Entwicklerhandbücher und Codebeispiele finden Sie unterDynamoDB mit einem SDK verwenden AWS. Dieses Thema enthält auch Informationen zu den ersten Schritten und Details zu früheren SDK-Versionen.

Richten Sie die attributebasierte Zugriffskontrolle für DynamoDB mit v2 ein AWS Command Line Interface

Im folgenden Codebeispiel wird gezeigt, wie Sie die attributbasierte Zugriffskontrolle (ABAC) für DynamoDB implementieren.

- Erstellen Sie eine IAM-Richtlinie für ABAC.
- Erstellen Sie Tabellen mit Tags für verschiedene Abteilungen.
- Listen Sie Tabellen auf der Grundlage von Tags auf und filtern Sie sie.

Bash

AWS CLI mit Bash-Skript

Erstellen Sie eine IAM-Richtlinie für ABAC.

```
# Step 1: Create a policy document for ABAC
cat > abac-policy.json << 'EOF'
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "dynamodb:GetItem",
        "dynamodb:BatchGetItem",
        "dynamodb:Query",
        "dynamodb:Scan"
      ],
      "Resource": "arn:aws:dynamodb:*:*:table/*",
      "Condition": {
        "StringEquals": {
          "aws:ResourceTag/Department": "${aws:PrincipalTag/Department}"
        }
      }
    },
    {
```

```
      "Effect": "Allow",
      "Action": [
        "dynamodb:PutItem",
        "dynamodb:UpdateItem",
        "dynamodb:DeleteItem",
        "dynamodb:BatchWriteItem"
      ],
      "Resource": "arn:aws:dynamodb:*:*:table/*",
      "Condition": {
        "StringEquals": {
          "aws:ResourceTag/Department": "${aws:PrincipalTag/Department}",
          "aws:ResourceTag/Environment": "Development"
        }
      }
    }
  ]
}
EOF

# Step 2: Create the IAM policy
aws iam create-policy \
    --policy-name DynamoDBDepartmentBasedAccess \
    --policy-document file://abac-policy.json
```

Erstellen Sie Tabellen mit Tags für verschiedene Abteilungen.

```
# Create a DynamoDB table with tags for ABAC
aws dynamodb create-table \
    --table-name FinanceData \
    --attribute-definitions \
        AttributeName=RecordID,AttributeType=S \
    --key-schema \
        AttributeName=RecordID,KeyType=HASH \
    --billing-mode PAY_PER_REQUEST \
    --tags \
        Key=Department,Value=Finance \
        Key=Environment,Value=Development

# Create another table with different tags
aws dynamodb create-table \
    --table-name MarketingData \
    --attribute-definitions \
```

```
        AttributeName=RecordID,AttributeType=S \
    --key-schema \
        AttributeName=RecordID,KeyType=HASH \
    --billing-mode PAY_PER_REQUEST \
    --tags \
        Key=Department,Value=Marketing \
        Key=Environment,Value=Production
```

Listen Sie Tabellen auf der Grundlage von Tags auf und filtern Sie sie.

```
# List all DynamoDB tables
echo "Listing all tables:"
aws dynamodb list-tables

# Get ARNs for all tables
echo -e "\nGetting ARNs for all tables:"
TABLE_ARNS=$(aws dynamodb list-tables --query "TableNames[*]" --output text |
 xargs -I {} aws dynamodb describe-table --table-name {} --query "Table.TableArn"
 --output text)

# For each table ARN, list its tags
echo -e "\nListing tags for each table:"
for ARN in $TABLE_ARNS; do
    TABLE_NAME=$(echo $ARN | awk -F/ '{print $2}')
    echo -e "\nTags for table: $TABLE_NAME"
    aws dynamodb list-tags-of-resource --resource-arn $ARN
done

# Example: Find tables with a specific tag
echo -e "\nFinding tables with Environment=Production tag:"
for ARN in $TABLE_ARNS; do
    TABLE_NAME=$(echo $ARN | awk -F/ '{print $2}')
    TAGS=$(aws dynamodb list-tags-of-resource --resource-arn $ARN --query "Tags[?
Key=='Environment' && Value=='Production']" --output text)
    if [ ! -z "$TAGS" ]; then
        echo "Table with Production tag: $TABLE_NAME"
    fi
done
```

- Weitere API-Informationen finden Sie in den folgenden Themen der AWS CLI - Befehlsreferenz.

- CreatePolicy
- CreateTable
- ListTables

Eine vollständige Liste der AWS SDK-Entwicklerhandbücher und Codebeispiele finden Sie unterDynamoDB mit einem SDK verwenden AWS. Dieses Thema enthält auch Informationen zu den ersten Schritten und Details zu früheren SDK-Versionen.

Die Reihenfolge der Aktualisierungsausdrücke in DynamoDB mit einem SDK verstehen AWS

In den folgenden Codebeispielen wird die Reihenfolge der Aktualisierungsausdrücke erklärt.

- Erfahren, wie DynamoDB Aktualisierungsausdrücke verarbeitet
- Verstehen der Reihenfolge der Operationen in Aktualisierungsausdrücken
- Vermeiden von unerwarteten Ergebnissen durch Verständnis der Ausdrucksauswertung

Java

SDK für Java 2.x

Demonstrieren Sie die Reihenfolge der Aktualisierungsausdrücke mit AWS SDK for Java 2.x.

```
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.AttributeValue;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.model.GetItemRequest;
import software.amazon.awssdk.services.dynamodb.model.GetItemResponse;
import software.amazon.awssdk.services.dynamodb.model.ReturnValue;
import software.amazon.awssdk.services.dynamodb.model.UpdateItemRequest;
import software.amazon.awssdk.services.dynamodb.model.UpdateItemResponse;

import java.util.HashMap;
import java.util.Map;

    /**
     * Demonstrates the effect of update expression order.
     *
     * <p>This method shows how the order of operations in an update expression
```

```
     * affects the result of the update.
     *
     * @param dynamoDbClient The DynamoDB client
     * @param tableName The name of the DynamoDB table
     * @param key The key of the item to update
     * @return Map containing the results of different update orders
     * @throws DynamoDbException if an error occurs during the operation
     */
    public static Map<String, Object> demonstrateUpdateOrder(
        DynamoDbClient dynamoDbClient, String tableName, Map<String,
AttributeValue> key) {

        Map<String, Object> results = new HashMap<>();

        try {
            // Initialize the item with a counter
            UpdateItemRequest initRequest = UpdateItemRequest.builder()
                .tableName(tableName)
                .key(key)
                .updateExpression("SET Counter = :zero, OldCounter = :zero")
                .expressionAttributeValues(
                    Map.of(":zero", AttributeValue.builder().n("0").build()))
                .returnValues(ReturnValue.UPDATED_NEW)
                .build();

            dynamoDbClient.updateItem(initRequest);

            // Example 1: SET first, then ADD
            UpdateItemRequest setFirstRequest = UpdateItemRequest.builder()
                .tableName(tableName)
                .key(key)
                .updateExpression("SET Counter = :value ADD
OldCounter :increment")
                .expressionAttributeValues(Map.of(
                    ":value", AttributeValue.builder().n("10").build(),
                    ":increment", AttributeValue.builder().n("5").build()))
                .returnValues(ReturnValue.UPDATED_NEW)
                .build();

            UpdateItemResponse setFirstResponse =
dynamoDbClient.updateItem(setFirstRequest);
            results.put("setFirstResponse", setFirstResponse);

            // Reset the item
```

```
            dynamoDbClient.updateItem(initRequest);

            // Example 2: ADD first, then SET
            UpdateItemRequest addFirstRequest = UpdateItemRequest.builder()
                .tableName(tableName)
                .key(key)
                .updateExpression("ADD Counter :increment SET OldCounter
= :value")
                .expressionAttributeValues(Map.of(
                    ":value", AttributeValue.builder().n("10").build(),
                    ":increment", AttributeValue.builder().n("5").build()))
                .returnValues(ReturnValue.UPDATED_NEW)
                .build();

            UpdateItemResponse addFirstResponse =
dynamoDbClient.updateItem(addFirstRequest);
            results.put("addFirstResponse", addFirstResponse);

            // Reset the item
            dynamoDbClient.updateItem(initRequest);

            // Example 3: SET with multiple attributes
            UpdateItemRequest multiSetRequest = UpdateItemRequest.builder()
                .tableName(tableName)
                .key(key)
                .updateExpression("SET Counter = :value, OldCounter = Counter")
                .expressionAttributeValues(
                    Map.of(":value", AttributeValue.builder().n("10").build()))
                .returnValues(ReturnValue.UPDATED_NEW)
                .build();

            UpdateItemResponse multiSetResponse =
dynamoDbClient.updateItem(multiSetRequest);
            results.put("multiSetResponse", multiSetResponse);

            // Reset the item
            dynamoDbClient.updateItem(initRequest);

            // Example 4: SET with expression using the same attribute
            UpdateItemRequest selfReferenceRequest = UpdateItemRequest.builder()
                .tableName(tableName)
                .key(key)
                .updateExpression("SET Counter = Counter + :increment, OldCounter
= Counter")
```

```
                .expressionAttributeValues(
                    Map.of(":increment",
AttributeValue.builder().n("5").build()))
                .returnValues(ReturnValue.UPDATED_NEW)
                .build();

            UpdateItemResponse selfReferenceResponse =
dynamoDbClient.updateItem(selfReferenceRequest);
            results.put("selfReferenceResponse", selfReferenceResponse);

            results.put("success", true);

        } catch (DynamoDbException e) {
            results.put("success", false);
            results.put("error", e.getMessage());
        }

        return results;
    }

    /**
     * Updates an item with SET first, then REMOVE.
     *
     * <p>This method demonstrates updating an item with SET operation first,
     * followed by a REMOVE operation.
     *
     * @param dynamoDbClient The DynamoDB client
     * @param tableName The name of the DynamoDB table
     * @param key The key of the item to update
     * @param attributeToSet The attribute to set
     * @param setValue The value to set
     * @param attributeToRemove The attribute to remove
     * @return The response from DynamoDB
     * @throws DynamoDbException if an error occurs during the operation
     */
    public static UpdateItemResponse updateWithSetFirst(
        DynamoDbClient dynamoDbClient,
        String tableName,
        Map<String, AttributeValue> key,
        String attributeToSet,
        AttributeValue setValue,
        String attributeToRemove) {

        // Define the update parameters
```

```
        UpdateItemRequest request = UpdateItemRequest.builder()
            .tableName(tableName)
            .key(key)
            .updateExpression("SET #setAttr = :setValue REMOVE #removeAttr")
            .expressionAttributeNames(Map.of(
                "#setAttr", attributeToSet,
                "#removeAttr", attributeToRemove))
            .expressionAttributeValues(Map.of(":setValue", setValue))
            .returnValues(ReturnValue.UPDATED_NEW)
            .build();

        // Perform the update operation
        try {
            return dynamoDbClient.updateItem(request);
        } catch (DynamoDbException e) {
            throw DynamoDbException.builder()
                .message("Failed to update item with SET first: " +
e.getMessage())
                .cause(e)
                .build();
        }
    }

    /**
     * Updates an item with REMOVE first, then SET.
     *
     * <p>This method demonstrates updating an item with REMOVE operation first,
     * followed by a SET operation.
     *
     * @param dynamoDbClient The DynamoDB client
     * @param tableName The name of the DynamoDB table
     * @param key The key of the item to update
     * @param attributeToSet The attribute to set
     * @param setValue The value to set
     * @param attributeToRemove The attribute to remove
     * @return The response from DynamoDB
     * @throws DynamoDbException if an error occurs during the operation
     */
    public static UpdateItemResponse updateWithRemoveFirst(
        DynamoDbClient dynamoDbClient,
        String tableName,
        Map<String, AttributeValue> key,
        String attributeToSet,
        AttributeValue setValue,
```

```
        String attributeToRemove) {

        // Define the update parameters
        UpdateItemRequest request = UpdateItemRequest.builder()
            .tableName(tableName)
            .key(key)
            .updateExpression("REMOVE #removeAttr SET #setAttr = :setValue")
            .expressionAttributeNames(Map.of(
                "#setAttr", attributeToSet,
                "#removeAttr", attributeToRemove))
            .expressionAttributeValues(Map.of(":setValue", setValue))
            .returnValues(ReturnValue.UPDATED_NEW)
            .build();

        // Perform the update operation
        try {
            return dynamoDbClient.updateItem(request);
        } catch (DynamoDbException e) {
            throw DynamoDbException.builder()
                .message("Failed to update item with REMOVE first: " +
e.getMessage())
                .cause(e)
                .build();
        }
    }

    /**
     * Updates an item with all operation types in a specific order.
     *
     * <p>This method demonstrates using all operation types (SET, REMOVE, ADD,
DELETE)
     * in a specific order in a single update expression.
     *
     * @param dynamoDbClient The DynamoDB client
     * @param tableName The name of the DynamoDB table
     * @param key The key of the item to update
     * @return The response from DynamoDB
     * @throws DynamoDbException if an error occurs during the operation
     */
    public static UpdateItemResponse updateWithAllOperationTypes(
        DynamoDbClient dynamoDbClient, String tableName, Map<String,
AttributeValue> key) {

        // Define the update parameters
```

```
        UpdateItemRequest request = UpdateItemRequest.builder()
            .tableName(tableName)
            .key(key)
            .updateExpression("SET #stringAttr = :stringVal, #mapAttr.#nestedAttr
= :nestedVal " + "REMOVE #oldAttr "
                + "ADD #counterAttr :increment "
                + "DELETE #stringSetAttr :stringSetVal")
            .expressionAttributeNames(Map.of(
                "#stringAttr", "StringAttribute",
                "#mapAttr", "MapAttribute",
                "#nestedAttr", "NestedAttribute",
                "#oldAttr", "OldAttribute",
                "#counterAttr", "CounterAttribute",
                "#stringSetAttr", "StringSetAttribute"))
            .expressionAttributeValues(Map.of(
                ":stringVal", AttributeValue.builder().s("New Value").build(),
                ":nestedVal", AttributeValue.builder().s("Nested Value").build(),
                ":increment", AttributeValue.builder().n("1").build(),
                ":stringSetVal", AttributeValue.builder().ss("Value1").build()))
            .returnValues(ReturnValue.UPDATED_NEW)
            .build();

        // Perform the update operation
        try {
            return dynamoDbClient.updateItem(request);
        } catch (DynamoDbException e) {
            throw DynamoDbException.builder()
                .message("Failed to update item with all operation types: " +
e.getMessage())
                .cause(e)
                .build();
        }
    }

    /**
     * Gets the current state of an item.
     *
     * <p>Helper method to retrieve the current state of an item.
     *
     * @param dynamoDbClient The DynamoDB client
     * @param tableName The name of the DynamoDB table
     * @param key The key of the item to get
     * @return The item or null if not found
     * @throws DynamoDbException if an error occurs during the operation
```

```
     */
    public static Map<String, AttributeValue> getItem(
        DynamoDbClient dynamoDbClient, String tableName, Map<String,
AttributeValue> key) {

        // Define the get parameters
        GetItemRequest request =
            GetItemRequest.builder().tableName(tableName).key(key).build();

        // Perform the get operation
        try {
            GetItemResponse response = dynamoDbClient.getItem(request);

            // Return the item if it exists, otherwise null
            return response.item();
        } catch (DynamoDbException e) {
            throw DynamoDbException.builder()
                .message("Failed to get item: " + e.getMessage())
                .cause(e)
                .build();
        }
    }
```

Beispiel für die Verwendung von Update Expression Order with AWS SDK for Java 2.x.

```
    public static void exampleUsage(DynamoDbClient dynamoDbClient, String
tableName) {
        // Example key
        Map<String, AttributeValue> key = new HashMap<>();
        key.put("ProductId", AttributeValue.builder().s("P12345").build());

        System.out.println("Demonstrating update expression order in DynamoDB");

        try {
            // Example 1: Demonstrate update order effects
            System.out.println("\nExample 1: Demonstrating update order
effects");
            Map<String, Object> orderResults =
demonstrateUpdateOrder(dynamoDbClient, tableName, key);

            if ((boolean) orderResults.get("success")) {
                System.out.println("SET first, then ADD:");
```

```
                System.out.println("  " + orderResults.get("setFirstResponse"));

                System.out.println("ADD first, then SET:");
                System.out.println("  " + orderResults.get("addFirstResponse"));

                System.out.println("SET with multiple attributes:");
                System.out.println("  " + orderResults.get("multiSetResponse"));

                System.out.println("SET with self-reference:");
                System.out.println("  " +
orderResults.get("selfReferenceResponse"));
            } else {
                System.out.println("Error: " + orderResults.get("error"));
            }

            // Example 2: Update with SET first, then REMOVE
            System.out.println("\nExample 2: Update with SET first, then
REMOVE");
            UpdateItemResponse setFirstResponse = updateWithSetFirst(
                dynamoDbClient,
                tableName,
                key,
                "Status",
                AttributeValue.builder().s("Active").build(),
                "OldStatus");

            System.out.println("Updated attributes: " +
setFirstResponse.attributes());

            // Example 3: Update with REMOVE first, then SET
            System.out.println("\nExample 3: Update with REMOVE first, then
SET");
            UpdateItemResponse removeFirstResponse = updateWithRemoveFirst(
                dynamoDbClient,
                tableName,
                key,
                "Status",
                AttributeValue.builder().s("Inactive").build(),
                "OldStatus");

            System.out.println("Updated attributes: " +
removeFirstResponse.attributes());

            // Example 4: Update with all operation types
```

```
            System.out.println("\nExample 4: Update with all operation types");
            UpdateItemResponse allOpsResponse =
updateWithAllOperationTypes(dynamoDbClient, tableName, key);

            System.out.println("Updated attributes: " +
allOpsResponse.attributes());

            // Example 5: Get the current state of the item
            System.out.println("\nExample 5: Current state of the item");
            Map<String, AttributeValue> item = getItem(dynamoDbClient, tableName,
key);

            if (item != null) {
                System.out.println("Item: " + item);
            } else {
                System.out.println("Item not found");
            }

            // Explain update expression order
            System.out.println("\nKey points about update expression order in
DynamoDB:");
            System.out.println("1. Update expressions are processed in this
order: SET, REMOVE, ADD, DELETE");
            System.out.println("2. Within each clause, operations are processed
from left to right");
            System.out.println("3. SET operations use the item state before any
updates in the expression");
            System.out.println("4. When an attribute is referenced multiple
times, the first operation wins");
            System.out.println("5. To reference a new value, split the update
into multiple operations");
            System.out.println("6. The order of clauses in the expression doesn't
change the evaluation order");
            System.out.println("7. For complex updates, consider using multiple
separate update operations");

        } catch (DynamoDbException e) {
            System.err.println("Error: " + e.getMessage());
            e.printStackTrace();
        }
    }
```

- Einzelheiten zur API finden Sie UpdateItemunter AWS SDK for Java 2.x API-Referenz.

JavaScript

SDK für JavaScript (v3)

Demonstrieren Sie die Reihenfolge der Aktualisierungsausdrücke mit AWS SDK für JavaScript.

```
const { DynamoDBClient } = require("@aws-sdk/client-dynamodb");
const {
  DynamoDBDocumentClient,
  UpdateCommand,
  GetCommand,
  PutCommand
} = require("@aws-sdk/lib-dynamodb");

/**
 * Update an item with multiple actions in a single update expression.
 *
 * This function demonstrates how to use multiple actions in a single update
 expression
 * and how DynamoDB processes these actions.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The primary key of the item to update
 * @param {string} updateExpression - The update expression with multiple actions
 * @param {Object} [expressionAttributeNames] - Expression attribute name
 placeholders
 * @param {Object} [expressionAttributeValues] - Expression attribute value
 placeholders
 * @returns {Promise<Object>} - The response from DynamoDB
 */
async function updateWithMultipleActions(
  config,
  tableName,
  key,
  updateExpression,
  expressionAttributeNames,
  expressionAttributeValues
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);
```

```
  // Prepare the update parameters
  const updateParams = {
    TableName: tableName,
    Key: key,
    UpdateExpression: updateExpression,
    ReturnValues: "UPDATED_NEW"
  };

  // Add expression attribute names if provided
  if (expressionAttributeNames) {
    updateParams.ExpressionAttributeNames = expressionAttributeNames;
  }

  // Add expression attribute values if provided
  if (expressionAttributeValues) {
    updateParams.ExpressionAttributeValues = expressionAttributeValues;
  }

  // Execute the update
  const response = await docClient.send(new UpdateCommand(updateParams));

  return response;
}

/**
 * Demonstrate that variables hold copies of existing values before
 modifications.
 *
 * This function creates an item with initial values, then updates it with an
 expression
 * that uses the values of attributes before they are modified in the same
 expression.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The primary key of the item to create and update
 * @returns {Promise<Object>} - A dictionary containing the results of the
 demonstration
 */
async function demonstrateValueCopying(
  config,
  tableName,
  key
) {
```

```
 // Initialize the DynamoDB client
 const client = new DynamoDBClient(config);
 const docClient = DynamoDBDocumentClient.from(client);

 // Step 1: Create an item with initial values
 const initialItem = { ...key, a: 1, b: 2, c: 3 };

 await docClient.send(new PutCommand({
   TableName: tableName,
   Item: initialItem
 }));

 // Step 2: Get the item to verify initial state
 const responseBefore = await docClient.send(new GetCommand({
   TableName: tableName,
   Key: key
 }));

 const itemBefore = responseBefore.Item || {};

 // Step 3: Update the item with an expression that uses values before they are
modified
 // This expression removes 'a', then sets 'b' to the value of 'a', and 'c' to
the value of 'b'
 const updateResponse = await docClient.send(new UpdateCommand({
   TableName: tableName,
   Key: key,
   UpdateExpression: "REMOVE a SET b = a, c = b",
   ReturnValues: "UPDATED_NEW"
 }));

 // Step 4: Get the item to verify final state
 const responseAfter = await docClient.send(new GetCommand({
   TableName: tableName,
   Key: key
 }));

 const itemAfter = responseAfter.Item || {};

 // Return the results
 return {
   initialState: itemBefore,
   updateResponse: updateResponse,
   finalState: itemAfter
```

```
  };
}

/**
 * Demonstrate the order in which different action types are processed.
 *
 * This function creates an item with initial values, then updates it with an
 expression
 * that includes multiple action types (SET, REMOVE, ADD, DELETE) to show the
 order
 * in which they are processed.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The primary key of the item to create and update
 * @returns {Promise<Object>} - A dictionary containing the results of the
 demonstration
 */
async function demonstrateActionOrder(
  config,
  tableName,
  key
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

  // Step 1: Create an item with initial values
  const initialItem = {
    ...key,
    counter: 10,
    set_attr: new Set(["A", "B", "C"]),
    to_remove: "This will be removed",
    to_modify: "Original value"
  };

  await docClient.send(new PutCommand({
    TableName: tableName,
    Item: initialItem
  }));

  // Step 2: Get the item to verify initial state
  const responseBefore = await docClient.send(new GetCommand({
    TableName: tableName,
```

```
    Key: key
  }));

  const itemBefore = responseBefore.Item || {};

  // Step 3: Update the item with multiple action types
  // The actions will be processed in this order: REMOVE, SET, ADD, DELETE
  const updateResponse = await docClient.send(new UpdateCommand({
    TableName: tableName,
    Key: key,
    UpdateExpression: "REMOVE to_remove SET to_modify = :new_value ADD
 counter :increment DELETE set_attr :elements",
    ExpressionAttributeValues: {
      ":new_value": "Updated value",
      ":increment": 5,
      ":elements": new Set(["B"])
    },
    ReturnValues: "UPDATED_NEW"
  }));

  // Step 4: Get the item to verify final state
  const responseAfter = await docClient.send(new GetCommand({
    TableName: tableName,
    Key: key
  }));

  const itemAfter = responseAfter.Item || {};

  // Return the results
  return {
    initialState: itemBefore,
    updateResponse: updateResponse,
    finalState: itemAfter
  };
}

/**
 * Update multiple attributes with a single SET action.
 *
 * This function demonstrates how to update multiple attributes in a single SET
 action,
 * which is more efficient than using multiple separate update operations.
 *
 * @param {Object} config - AWS configuration object
```

```
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The primary key of the item to update
 * @param {Object} attributes - The attributes to update and their new values
 * @returns {Promise<Object>} - The response from DynamoDB
 */
async function updateWithMultipleSetActions(
  config,
  tableName,
  key,
  attributes
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

  // Build the update expression and expression attribute values
  let updateExpression = "SET ";
  const expressionAttributeValues = {};

  // Add each attribute to the update expression
  Object.entries(attributes).forEach(([attrName, attrValue], index) => {
    const valuePlaceholder = `:val${index}`;

    if (index > 0) {
      updateExpression += ", ";
    }
    updateExpression += `${attrName} = ${valuePlaceholder}`;

    expressionAttributeValues[valuePlaceholder] = attrValue;
  });

  // Execute the update
  const response = await docClient.send(new UpdateCommand({
    TableName: tableName,
    Key: key,
    UpdateExpression: updateExpression,
    ExpressionAttributeValues: expressionAttributeValues,
    ReturnValues: "UPDATED_NEW"
  }));

  return response;
}

/**
```

```
 * Update an attribute with a value from another attribute or a default value.
 *
 * This function demonstrates how to use if_not_exists to conditionally copy a
 value
 * from one attribute to another, or use a default value if the source doesn't
 exist.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The primary key of the item to update
 * @param {string} sourceAttribute - The attribute to copy the value from
 * @param {string} targetAttribute - The attribute to update
 * @param {any} defaultValue - The default value to use if the source attribute
 doesn't exist
 * @returns {Promise<Object>} - The response from DynamoDB
 */
async function updateWithConditionalValueCopying(
  config,
  tableName,
  key,
  sourceAttribute,
  targetAttribute,
  defaultValue
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

  // Use if_not_exists to conditionally copy the value
  const response = await docClient.send(new UpdateCommand({
    TableName: tableName,
    Key: key,
    UpdateExpression: `SET ${targetAttribute} =
 if_not_exists(${sourceAttribute}, :default)`,
    ExpressionAttributeValues: {
      ":default": defaultValue
    },
    ReturnValues: "UPDATED_NEW"
  }));

  return response;
}

/**
```

```
 * Demonstrate complex update expressions with multiple operations on the same
 attribute.
 *
 * This function shows how DynamoDB processes multiple operations on the same
 attribute
 * in a single update expression.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The primary key of the item to create and update
 * @returns {Promise<Object>} - A dictionary containing the results of the
 demonstration
 */
async function demonstrateMultipleOperationsOnSameAttribute(
  config,
  tableName,
  key
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

  // Step 1: Create an item with initial values
  const initialItem = {
    ...key,
    counter: 10,
    list_attr: [1, 2, 3],
    map_attr: {
      nested1: "value1",
      nested2: "value2"
    }
  };

  await docClient.send(new PutCommand({
    TableName: tableName,
    Item: initialItem
  }));

  // Step 2: Get the item to verify initial state
  const responseBefore = await docClient.send(new GetCommand({
    TableName: tableName,
    Key: key
  }));
```

```
  const itemBefore = responseBefore.Item || {};

  // Step 3: Update the item with multiple operations on the same attributes
  const updateResponse = await docClient.send(new UpdateCommand({
    TableName: tableName,
    Key: key,
    UpdateExpression: `
      SET counter = counter + :inc1,
          counter = counter + :inc2,
          map_attr.nested1 = :new_val1,
          map_attr.nested3 = :new_val3,
          list_attr[0] = list_attr[1],
          list_attr[1] = list_attr[2]
    `,
    ExpressionAttributeValues: {
      ":inc1": 5,
      ":inc2": 3,
      ":new_val1": "updated_value1",
      ":new_val3": "new_value3"
    },
    ReturnValues: "UPDATED_NEW"
  }));

  // Step 4: Get the item to verify final state
  const responseAfter = await docClient.send(new GetCommand({
    TableName: tableName,
    Key: key
  }));

  const itemAfter = responseAfter.Item || {};

  // Return the results
  return {
    initialState: itemBefore,
    updateResponse: updateResponse,
    finalState: itemAfter
  };
}
```

Beispiel für die Verwendung von Update Expression Order with AWS SDK für JavaScript.

```
/**
```

```
 * Example of how to use update expression order of operations in DynamoDB.
 */
async function exampleUsage() {
  // Example parameters
  const config = { region: "us-west-2" };
  const tableName = "OrderProcessing";

  console.log("Demonstrating update expression order of operations in DynamoDB");

  try {
    // Example 1: Demonstrating value copying in update expressions
    console.log("\nExample 1: Demonstrating value copying in update
 expressions");
    const results1 = await demonstrateValueCopying(
      config,
      tableName,
      { OrderId: "order123" }
    );

    console.log("Initial state:", JSON.stringify(results1.initialState, null,
 2));
    console.log("Update response:", JSON.stringify(results1.updateResponse, null,
 2));
    console.log("Final state:", JSON.stringify(results1.finalState, null, 2));

    console.log("\nExplanation:");
    console.log("1. The initial state had a=1, b=2, c=3");
    console.log("2. The update expression 'REMOVE a SET b = a, c = b' did the
 following:");
    console.log("   - Copied the value of 'a' (which was 1) to be used for 'b'");
    console.log("   - Copied the value of 'b' (which was 2) to be used for 'c'");
    console.log("   - Removed the attribute 'a'");
    console.log("3. The final state has b=1, c=2, and 'a' is removed");
    console.log("4. This demonstrates that DynamoDB uses the values of attributes
 as they were BEFORE any modifications");

    // Example 2: Demonstrating the order of different action types
    console.log("\nExample 2: Demonstrating the order of different action
 types");
    const results2 = await demonstrateActionOrder(
      config,
      tableName,
      { OrderId: "order456" }
    );
```

```
   console.log("Initial state:", JSON.stringify(results2.initialState, null,
2));
   console.log("Update response:", JSON.stringify(results2.updateResponse, null,
2));
   console.log("Final state:", JSON.stringify(results2.finalState, null, 2));

   console.log("\nExplanation:");
   console.log("1. The update expression contained multiple action types:
REMOVE, SET, ADD, DELETE");
   console.log("2. DynamoDB processes these actions in this order: REMOVE, SET,
ADD, DELETE");
   console.log("3. First, 'to_remove' was removed");
   console.log("4. Then, 'to_modify' was set to a new value");
   console.log("5. Next, 'counter' was incremented by 5");
   console.log("6. Finally, 'B' was removed from the set attribute");

   // Example 3: Updating multiple attributes in a single SET action
   console.log("\nExample 3: Updating multiple attributes in a single SET
action");
   const response3 = await updateWithMultipleSetActions(
     config,
     tableName,
     { OrderId: "order789" },
     {
       Status: "Shipped",
       ShippingDate: "2025-05-28",
       TrackingNumber: "1Z999AA10123456784"
     }
   );

   console.log("Multiple attributes updated successfully:",
JSON.stringify(response3.Attributes, null, 2));

   // Example 4: Conditional value copying with if_not_exists
   console.log("\nExample 4: Conditional value copying with if_not_exists");
   const response4 = await updateWithConditionalValueCopying(
     config,
     tableName,
     { OrderId: "order101" },
     "PreferredShippingMethod",
     "ShippingMethod",
     "Standard"
   );
```

```
   console.log("Conditional value copying result:",
JSON.stringify(response4.Attributes, null, 2));

   // Example 5: Multiple operations on the same attribute
   console.log("\nExample 5: Multiple operations on the same attribute");
   const results5 = await demonstrateMultipleOperationsOnSameAttribute(
     config,
     tableName,
     { OrderId: "order202" }
   );

   console.log("Initial state:", JSON.stringify(results5.initialState, null,
2));
   console.log("Update response:", JSON.stringify(results5.updateResponse, null,
2));
   console.log("Final state:", JSON.stringify(results5.finalState, null, 2));

   console.log("\nExplanation:");
   console.log("1. The counter was incremented twice (first by 5, then by 3) for
a total of +8");
   console.log("2. The map attribute had one value updated and a new nested
attribute added");
   console.log("3. The list attribute had values shifted (value at index 1 moved
to index 0, value at index 2 moved to index 1)");
   console.log("4. All operations within the SET action are processed from left
to right");

   // Key points about update expression order of operations
   console.log("\nKey Points About Update Expression Order of Operations:");
   console.log("1. Variables in expressions hold copies of attribute values as
they existed BEFORE any modifications");
   console.log("2. Multiple actions in an update expression are processed in
this order: REMOVE, SET, ADD, DELETE");
   console.log("3. Within each action type, operations are processed from left
to right");
   console.log("4. You can reference the same attribute multiple times in an
expression");
   console.log("5. You can use if_not_exists() to conditionally set values based
on attribute existence");
   console.log("6. Using a single update expression with multiple actions is
more efficient than multiple separate updates");
   console.log("7. The update expression is atomic - either all actions succeed
or none do");
```

```
  } catch (error) {
    console.error("Error:", error);
  }
}
```

- Einzelheiten zur API finden Sie UpdateItemunter AWS SDK für JavaScript API-Referenz.

Python

SDK für Python (Boto3)

Demonstrieren Sie die Reihenfolge der Aktualisierungsausdrücke mit AWS SDK für Python (Boto3).

```
import boto3
import json
from typing import Any, Dict, Optional


def update_with_multiple_actions(
    table_name: str,
    key: Dict[str, Any],
    update_expression: str,
    expression_attribute_names: Optional[Dict[str, str]] = None,
    expression_attribute_values: Optional[Dict[str, Any]] = None,
) -> Dict[str, Any]:
    """
    Update an item with multiple actions in a single update expression.

    This function demonstrates how to use multiple actions in a single update
 expression
    and how DynamoDB processes these actions.

    Args:
        table_name (str): The name of the DynamoDB table.
        key (Dict[str, Any]): The primary key of the item to update.
        update_expression (str): The update expression with multiple actions.
        expression_attribute_names (Optional[Dict[str, str]]): Expression
 attribute name placeholders.
        expression_attribute_values (Optional[Dict[str, Any]]): Expression
 attribute value placeholders.
```

```
    Returns:
        Dict[str, Any]: The response from DynamoDB containing the updated
 attribute values.
    """
    # Initialize the DynamoDB resource
    dynamodb = boto3.resource("dynamodb")
    table = dynamodb.Table(table_name)

    # Prepare the update parameters
    update_params = {
        "Key": key,
        "UpdateExpression": update_expression,
        "ReturnValues": "UPDATED_NEW",
    }

    # Add expression attribute names if provided
    if expression_attribute_names:
        update_params["ExpressionAttributeNames"] = expression_attribute_names

    # Add expression attribute values if provided
    if expression_attribute_values:
        update_params["ExpressionAttributeValues"] = expression_attribute_values

    # Execute the update
    response = table.update_item(**update_params)

    return response


def demonstrate_value_copying(table_name: str, key: Dict[str, Any]) -> Dict[str,
 Any]:
    """
    Demonstrate that variables hold copies of existing values before
 modifications.

    This function creates an item with initial values, then updates it with an
 expression
    that uses the values of attributes before they are modified in the same
 expression.

    Args:
        table_name (str): The name of the DynamoDB table.
        key (Dict[str, Any]): The primary key of the item to create and update.
```

```
    Returns:
        Dict[str, Any]: A dictionary containing the results of the demonstration.
    """
    # Initialize the DynamoDB resource
    dynamodb = boto3.resource("dynamodb")
    table = dynamodb.Table(table_name)

    # Step 1: Create an item with initial values
    initial_item = key.copy()
    initial_item.update({"a": 1, "b": 2, "c": 3})

    table.put_item(Item=initial_item)

    # Step 2: Get the item to verify initial state
    response_before = table.get_item(Key=key)
    item_before = response_before.get("Item", {})

    # Step 3: Update the item with an expression that uses values before they are
 modified
    # This expression removes 'a', then sets 'b' to the value of 'a', and 'c' to
 the value of 'b'
    update_response = table.update_item(
        Key=key, UpdateExpression="REMOVE a SET b = a, c = b",
 ReturnValues="UPDATED_NEW"
    )

    # Step 4: Get the item to verify final state
    response_after = table.get_item(Key=key)
    item_after = response_after.get("Item", {})

    # Return the results
    return {
        "initial_state": item_before,
        "update_response": update_response,
        "final_state": item_after,
    }


def demonstrate_action_order(table_name: str, key: Dict[str, Any]) -> Dict[str,
 Any]:
    """
    Demonstrate the order in which different action types are processed.
```

```
    This function creates an item with initial values, then updates it with an
expression
    that includes multiple action types (SET, REMOVE, ADD, DELETE) to show the
order
    in which they are processed.

    Args:
        table_name (str): The name of the DynamoDB table.
        key (Dict[str, Any]): The primary key of the item to create and update.

    Returns:
        Dict[str, Any]: A dictionary containing the results of the demonstration.
    """
    # Initialize the DynamoDB resource
    dynamodb = boto3.resource("dynamodb")
    table = dynamodb.Table(table_name)

    # Step 1: Create an item with initial values
    initial_item = key.copy()
    initial_item.update(
        {
            "counter": 10,
            "set_attr": set(["A", "B", "C"]),
            "to_remove": "This will be removed",
            "to_modify": "Original value",
        }
    )

    table.put_item(Item=initial_item)

    # Step 2: Get the item to verify initial state
    response_before = table.get_item(Key=key)
    item_before = response_before.get("Item", {})

    # Step 3: Update the item with multiple action types
    # The actions will be processed in this order: REMOVE, SET, ADD, DELETE
    update_response = table.update_item(
        Key=key,
        UpdateExpression="REMOVE to_remove SET to_modify = :new_value ADD
counter :increment DELETE set_attr :elements",
        ExpressionAttributeValues={
            ":new_value": "Updated value",
            ":increment": 5,
            ":elements": set(["B"]),
```

```
        },
        ReturnValues="UPDATED_NEW",
    )

    # Step 4: Get the item to verify final state
    response_after = table.get_item(Key=key)
    item_after = response_after.get("Item", {})

    # Return the results
    return {
        "initial_state": item_before,
        "update_response": update_response,
        "final_state": item_after,
    }


def update_with_multiple_set_actions(
    table_name: str, key: Dict[str, Any], attributes: Dict[str, Any]
) -> Dict[str, Any]:
    """
    Update multiple attributes with a single SET action.

    This function demonstrates how to update multiple attributes in a single SET
 action,
    which is more efficient than using multiple separate update operations.

    Args:
        table_name (str): The name of the DynamoDB table.
        key (Dict[str, Any]): The primary key of the item to update.
        attributes (Dict[str, Any]): The attributes to update and their new
 values.

    Returns:
        Dict[str, Any]: The response from DynamoDB containing the updated
 attribute values.
    """
    # Initialize the DynamoDB resource
    dynamodb = boto3.resource("dynamodb")
    table = dynamodb.Table(table_name)

    # Build the update expression and expression attribute values
    update_expression = "SET "
    expression_attribute_values = {}
```

```
    # Add each attribute to the update expression
    for i, (attr_name, attr_value) in enumerate(attributes.items()):
        value_placeholder = f":val{i}"

        if i > 0:
            update_expression += ", "
        update_expression += f"{attr_name} = {value_placeholder}"

        expression_attribute_values[value_placeholder] = attr_value

    # Execute the update
    response = table.update_item(
        Key=key,
        UpdateExpression=update_expression,
        ExpressionAttributeValues=expression_attribute_values,
        ReturnValues="UPDATED_NEW",
    )

    return response


def update_with_conditional_value_copying(
    table_name: str,
    key: Dict[str, Any],
    source_attribute: str,
    target_attribute: str,
    default_value: Any,
) -> Dict[str, Any]:
    """
    Update an attribute with a value from another attribute or a default value.

    This function demonstrates how to use if_not_exists to conditionally copy a
 value
    from one attribute to another, or use a default value if the source doesn't
 exist.

    Args:
        table_name (str): The name of the DynamoDB table.
        key (Dict[str, Any]): The primary key of the item to update.
        source_attribute (str): The attribute to copy the value from.
        target_attribute (str): The attribute to update.
        default_value (Any): The default value to use if the source attribute
 doesn't exist.
```

```
    Returns:
        Dict[str, Any]: The response from DynamoDB containing the updated
 attribute values.
    """
    # Initialize the DynamoDB resource
    dynamodb = boto3.resource("dynamodb")
    table = dynamodb.Table(table_name)

    # Use if_not_exists to conditionally copy the value
    response = table.update_item(
        Key=key,
        UpdateExpression=f"SET {target_attribute} =
 if_not_exists({source_attribute}, :default)",
        ExpressionAttributeValues={":default": default_value},
        ReturnValues="UPDATED_NEW",
    )

    return response
```

Beispiel für die Verwendung von Update Expression Order with AWS SDK für Python (Boto3).

```
def example_usage():
    """Example of how to use update expression order of operations in
 DynamoDB."""
    # Example parameters
    table_name = "OrderProcessing"
    key = {"OrderId": "order123"}

    print("Example 1: Demonstrating value copying in update expressions")
    try:
        results = demonstrate_value_copying(table_name=table_name, key=key)

        print(f"Initial state: {json.dumps(results['initial_state'],
 default=str)}")
        print(f"Update response: {json.dumps(results['update_response'],
 default=str)}")
        print(f"Final state: {json.dumps(results['final_state'], default=str)}")

        print("\nExplanation:")
        print("1. The initial state had a=1, b=2, c=3")
```

```
        print("2. The update expression 'REMOVE a SET b = a, c = b' did the
following:")
        print("   - Copied the value of 'a' (which was 1) to be used for 'b'")
        print("   - Copied the value of 'b' (which was 2) to be used for 'c'")
        print("   - Removed the attribute 'a'")
        print("3. The final state has b=1, c=2, and 'a' is removed")
        print(
            "4. This demonstrates that DynamoDB uses the values of attributes as
they were BEFORE any modifications"
        )
    except Exception as e:
        print(f"Error demonstrating value copying: {e}")

    print("\nExample 2: Demonstrating the order of different action types")
    try:
        results = demonstrate_action_order(table_name=table_name, key={"OrderId":
"order456"})

        print(f"Initial state: {json.dumps(results['initial_state'],
default=str)}")
        print(f"Update response: {json.dumps(results['update_response'],
default=str)}")
        print(f"Final state: {json.dumps(results['final_state'], default=str)}")

        print("\nExplanation:")
        print("1. The update expression contained multiple action types: REMOVE,
SET, ADD, DELETE")
        print("2. DynamoDB processes these actions in this order: REMOVE, SET,
ADD, DELETE")
        print("3. First, 'to_remove' was removed")
        print("4. Then, 'to_modify' was set to a new value")
        print("5. Next, 'counter' was incremented by 5")
        print("6. Finally, 'B' was removed from the set attribute")
    except Exception as e:
        print(f"Error demonstrating action order: {e}")

    print("\nExample 3: Updating multiple attributes in a single SET action")
    try:
        response = update_with_multiple_set_actions(
            table_name=table_name,
            key={"OrderId": "order789"},
            attributes={
                "Status": "Shipped",
                "ShippingDate": "2025-05-14",
```

```
                "TrackingNumber": "1Z999AA10123456784",
            },
        )

        print(
            f"Multiple attributes updated successfully:
{json.dumps(response.get('Attributes', {}), default=str)}"
        )
    except Exception as e:
        print(f"Error updating multiple attributes: {e}")

    print("\nExample 4: Conditional value copying with if_not_exists")
    try:
        response = update_with_conditional_value_copying(
            table_name=table_name,
            key={"OrderId": "order101"},
            source_attribute="PreferredShippingMethod",
            target_attribute="ShippingMethod",
            default_value="Standard",
        )

        print(
            f"Conditional value copying result:
{json.dumps(response.get('Attributes', {}), default=str)}"
        )
    except Exception as e:
        print(f"Error with conditional value copying: {e}")

    print("\nKey Points About Update Expression Order of Operations:")
    print(
        "1. Variables in expressions hold copies of attribute values as they
existed BEFORE any modifications"
    )
    print(
        "2. Multiple actions in an update expression are processed in this order:
REMOVE, SET, ADD, DELETE"
    )
    print("3. Within each action type, operations are processed from left to
right")
    print("4. You can reference the same attribute multiple times in an
expression")
    print("5. You can use if_not_exists() to conditionally set values based on
attribute existence")
    print(
```

```
        "6. Using a single update expression with multiple actions is more
efficient than multiple separate updates"
    )
    print("7. The update expression is atomic - either all actions succeed or
none do")
```

- Einzelheiten zur API finden Sie UpdateItemin AWS SDK for Python (Boto3) API Reference.

Eine vollständige Liste der AWS SDK-Entwicklerhandbücher und Codebeispiele finden Sie unter. DynamoDB mit einem SDK verwenden AWS Dieses Thema enthält auch Informationen zu den ersten Schritten und Details zu früheren SDK-Versionen.

Aktualisieren Sie eine DynamoDB-Tabelleneinstellung mit warmem Durchsatz mithilfe eines SDK AWS

In den folgenden Codebeispielen wird gezeigt, wie die Warmdurchsatzeinstellung einer Tabelle aktualisiert wird.

Java

SDK für Java 2.x

Aktualisieren Sie mit AWS SDK for Java 2.x die Einstellung für den Warmdurchsatz einer DynamoDB-Tabelle.

```
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.model.GlobalSecondaryIndexUpdate;
import
 software.amazon.awssdk.services.dynamodb.model.UpdateGlobalSecondaryIndexAction;
import software.amazon.awssdk.services.dynamodb.model.UpdateTableRequest;
import software.amazon.awssdk.services.dynamodb.model.WarmThroughput;

    public static WarmThroughput buildWarmThroughput(final Long
 readUnitsPerSecond, final Long writeUnitsPerSecond) {
        return WarmThroughput.builder()
            .readUnitsPerSecond(readUnitsPerSecond)
            .writeUnitsPerSecond(writeUnitsPerSecond)
```

```
            .build();
    }

    /**
     * Updates a DynamoDB table with warm throughput settings for both the table
and a global secondary index.
     *
     * @param ddb The DynamoDB client
     * @param tableName The name of the table to update
     * @param tableReadUnitsPerSecond Read units per second for the table
     * @param tableWriteUnitsPerSecond Write units per second for the table
     * @param globalSecondaryIndexName The name of the global secondary index to
update
     * @param globalSecondaryIndexReadUnitsPerSecond Read units per second for
the GSI
     * @param globalSecondaryIndexWriteUnitsPerSecond Write units per second for
the GSI
     */
    public static void updateDynamoDBTable(
        final DynamoDbClient ddb,
        final String tableName,
        final Long tableReadUnitsPerSecond,
        final Long tableWriteUnitsPerSecond,
        final String globalSecondaryIndexName,
        final Long globalSecondaryIndexReadUnitsPerSecond,
        final Long globalSecondaryIndexWriteUnitsPerSecond) {

        final WarmThroughput tableWarmThroughput =
            buildWarmThroughput(tableReadUnitsPerSecond,
tableWriteUnitsPerSecond);
        final WarmThroughput gsiWarmThroughput =
            buildWarmThroughput(globalSecondaryIndexReadUnitsPerSecond,
globalSecondaryIndexWriteUnitsPerSecond);

        final GlobalSecondaryIndexUpdate globalSecondaryIndexUpdate =
GlobalSecondaryIndexUpdate.builder()
            .update(UpdateGlobalSecondaryIndexAction.builder()
                .indexName(globalSecondaryIndexName)
                .warmThroughput(gsiWarmThroughput)
                .build())
            .build();

        final UpdateTableRequest request = UpdateTableRequest.builder()
            .tableName(tableName)
```

```
            .globalSecondaryIndexUpdates(globalSecondaryIndexUpdate)
            .warmThroughput(tableWarmThroughput)
            .build();

        try {
            ddb.updateTable(request);
        } catch (DynamoDbException e) {
            System.err.println(e.getMessage());
            throw e;
        }

        System.out.println(SUCCESS_MESSAGE);
    }
```

- Einzelheiten zur API finden Sie UpdateTablein der AWS SDK for Java 2.x API-Referenz.

JavaScript

SDK für JavaScript (v3)

Aktualisieren Sie mit AWS SDK für JavaScript die Einstellung für den Warmdurchsatz einer DynamoDB-Tabelle.

```
import { DynamoDBClient, UpdateTableCommand } from "@aws-sdk/client-dynamodb";

export async function updateDynamoDBTableWarmThroughput(
  tableName,
  tableReadUnits,
  tableWriteUnits,
  gsiName,
  gsiReadUnits,
  gsiWriteUnits,
  region = "us-east-1"
) {
  try {
    const ddbClient = new DynamoDBClient({ region: region });

    // Construct the update table request
    const updateTableRequest = {
      TableName: tableName,
      GlobalSecondaryIndexUpdates: [
        {
```

```
            Update: {
                IndexName: gsiName,
                WarmThroughput: {
                    ReadUnitsPerSecond: gsiReadUnits,
                    WriteUnitsPerSecond: gsiWriteUnits,
                },
            },
        },
      ],
      WarmThroughput: {
          ReadUnitsPerSecond: tableReadUnits,
          WriteUnitsPerSecond: tableWriteUnits,
      },
    };

    const command = new UpdateTableCommand(updateTableRequest);
    const response = await ddbClient.send(command);
    console.log(`Table updated successfully! Response:
 ${JSON.stringify(response)}`);
    return response;
  } catch (error) {
    console.error(`Error updating table: ${error}`);
    throw error;
  }
}

// Example usage (commented out for testing)
/*
updateDynamoDBTableWarmThroughput(
  'example-table',
  5, 5,
  'example-index',
  2, 2
);
*/
```

- Einzelheiten zur API finden Sie UpdateTablein der AWS SDK für JavaScript API-Referenz.

Python

SDK für Python (Boto3)

Aktualisieren Sie mit AWS SDK für Python (Boto3) die Einstellung für den Warmdurchsatz einer DynamoDB-Tabelle.

```
from boto3 import client
from botocore.exceptions import ClientError


def update_dynamodb_table_warm_throughput(
    table_name,
    table_read_units,
    table_write_units,
    gsi_name,
    gsi_read_units,
    gsi_write_units,
    region_name="us-east-1",
):
    """
    Updates the warm throughput of a DynamoDB table and a global secondary index.

    :param table_name: The name of the table to update.
    :param table_read_units: The new read units per second for the table's warm
 throughput.
    :param table_write_units: The new write units per second for the table's warm
 throughput.
    :param gsi_name: The name of the global secondary index to update.
    :param gsi_read_units: The new read units per second for the GSI's warm
 throughput.
    :param gsi_write_units: The new write units per second for the GSI's warm
 throughput.
    :param region_name: The AWS Region name to target. defaults to us-east-1
    :return: The response from the update_table operation
    """
    try:
        ddb = client("dynamodb", region_name=region_name)

        # Update the table's warm throughput
        table_warm_throughput = {
            "ReadUnitsPerSecond": table_read_units,
            "WriteUnitsPerSecond": table_write_units,
        }
```

```
        # Update the global secondary index's warm throughput
        gsi_warm_throughput = {
            "ReadUnitsPerSecond": gsi_read_units,
            "WriteUnitsPerSecond": gsi_write_units,
        }

        # Construct the global secondary index update
        global_secondary_index_update = [
            {"Update": {"IndexName": gsi_name, "WarmThroughput":
gsi_warm_throughput}}
        ]

        # Construct the update table request
        update_table_request = {
            "TableName": table_name,
            "GlobalSecondaryIndexUpdates": global_secondary_index_update,
            "WarmThroughput": table_warm_throughput,
        }

        # Update the table
        response = ddb.update_table(**update_table_request)
        print("Table updated successfully!")
        return response  # Make sure to return the response
    except ClientError as e:
        print(f"Error updating table: {e}")
        raise e
```

- Einzelheiten zur API finden Sie UpdateTablein AWS SDK for Python (Boto3) API Reference.

Eine vollständige Liste der AWS SDK-Entwicklerhandbücher und Codebeispiele finden Sie unter. DynamoDB mit einem SDK verwenden AWS Dieses Thema enthält auch Informationen zu den ersten Schritten und Details zu früheren SDK-Versionen.

Aktualisieren Sie ein DynamoDB-Element mit einer TTL mithilfe eines SDK AWS

Die folgenden Codebeispiele veranschaulichen, wie Sie die TTL eines Elements aktualisieren.

Java

SDK für Java 2.x

Aktualisieren Sie die TTL für ein vorhandenes DynamoDB-Element in einer Tabelle.

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.AttributeValue;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.model.ResourceNotFoundException;
import software.amazon.awssdk.services.dynamodb.model.UpdateItemRequest;
import software.amazon.awssdk.services.dynamodb.model.UpdateItemResponse;

import java.util.HashMap;
import java.util.Map;
import java.util.Optional;

    public UpdateItemResponse updateItemWithTTL(
        final String tableName, final String primaryKeyValue, final String
 sortKeyValue) {
        // Get current time in epoch second format
        final long currentTime = System.currentTimeMillis() / 1000;

        // Calculate expiration time 90 days from now in epoch second format
        final long expireDate = currentTime + (DAYS_TO_EXPIRE * SECONDS_PER_DAY);

        // Create the key map for the item to update
        final Map<String, AttributeValue> keyMap = new HashMap<>();
        keyMap.put(PRIMARY_KEY_ATTR,
 AttributeValue.builder().s(primaryKeyValue).build());
        keyMap.put(SORT_KEY_ATTR,
 AttributeValue.builder().s(sortKeyValue).build());

        // Create the expression attribute values
        final Map<String, AttributeValue> expressionAttributeValues = new
 HashMap<>();
        expressionAttributeValues.put(
            ":c",
 AttributeValue.builder().n(String.valueOf(currentTime)).build());
        expressionAttributeValues.put(
            ":e",
 AttributeValue.builder().n(String.valueOf(expireDate)).build());
```

```
        final UpdateItemRequest request = UpdateItemRequest.builder()
            .tableName(tableName)
            .key(keyMap)
            .updateExpression(UPDATE_EXPRESSION)
            .expressionAttributeValues(expressionAttributeValues)
            .build();

        try {
            final UpdateItemResponse response =
 dynamoDbClient.updateItem(request);
            System.out.println(String.format(SUCCESS_MESSAGE, tableName));
            return response;
        } catch (ResourceNotFoundException e) {
            System.err.format(TABLE_NOT_FOUND_ERROR, tableName);
            throw e;
        } catch (DynamoDbException e) {
            System.err.println(e.getMessage());
            throw e;
        }
    }
```

- Einzelheiten zur API finden Sie UpdateItemin der AWS SDK for Java 2.x API-Referenz.

JavaScript

SDK für JavaScript (v3)

```
import { DynamoDBClient, UpdateItemCommand } from "@aws-sdk/client-dynamodb";
import { marshall, unmarshall } from "@aws-sdk/util-dynamodb";

export const updateItem = async (tableName, partitionKey, sortKey, region = 'us-
east-1') => {
    const client = new DynamoDBClient({
        region: region,
        endpoint: `https://dynamodb.${region}.amazonaws.com`
    });

    const currentTime = Math.floor(Date.now() / 1000);
    const expireAt = Math.floor((Date.now() + 90 * 24 * 60 * 60 * 1000) / 1000);

    const params = {
        TableName: tableName,
```

```
        Key: marshall({
            partitionKey: partitionKey,
            sortKey: sortKey
        }),
        UpdateExpression: "SET updatedAt = :c, expireAt = :e",
        ExpressionAttributeValues: marshall({
            ":c": currentTime,
            ":e": expireAt
        }),
    };

    try {
        const data = await client.send(new UpdateItemCommand(params));
        const responseData = unmarshall(data.Attributes);
        console.log("Item updated successfully: %s", responseData);
        return responseData;
    } catch (err) {
        console.error("Error updating item:", err);
        throw err;
    }
}

// Example usage (commented out for testing)
// updateItem('your-table-name', 'your-partition-key-value', 'your-sort-key-
value');
```

- Einzelheiten zur API finden Sie UpdateItemin der AWS SDK für JavaScript API-Referenz.

Python

SDK für Python (Boto3)

```
from datetime import datetime, timedelta

import boto3


def update_dynamodb_item(table_name, region, primary_key, sort_key):
    """
    Update an existing DynamoDB item with a TTL.
    :param table_name: Name of the DynamoDB table
```

```
    :param region: AWS Region of the table - example `us-east-1`
    :param primary_key: one attribute known as the partition key.
    :param sort_key: Also known as a range attribute.
    :return: Void (nothing)
    """
    try:
        # Create the DynamoDB resource.
        dynamodb = boto3.resource("dynamodb", region_name=region)
        table = dynamodb.Table(table_name)

        # Get the current time in epoch second format
        current_time = int(datetime.now().timestamp())

        # Calculate the expireAt time (90 days from now) in epoch second format
        expire_at = int((datetime.now() + timedelta(days=90)).timestamp())

        table.update_item(
            Key={"partitionKey": primary_key, "sortKey": sort_key},
            UpdateExpression="set updatedAt=:c, expireAt=:e",
            ExpressionAttributeValues={":c": current_time, ":e": expire_at},
        )

        print("Item updated successfully.")
    except Exception as e:
        print(f"Error updating item: {e}")


# Replace with your own values
update_dynamodb_item(
    "your-table-name", "us-west-2", "your-partition-key-value", "your-sort-key-
value"
)
```

- Einzelheiten zur API finden Sie UpdateItemin AWS SDK for Python (Boto3) API Reference.

Eine vollständige Liste der AWS SDK-Entwicklerhandbücher und Codebeispiele finden Sie unter. DynamoDB mit einem SDK verwenden AWS Dieses Thema enthält auch Informationen zu den ersten Schritten und Details zu früheren SDK-Versionen.

Aktualisieren Sie DynamoDB-Daten mithilfe von PartiQL UPDATE-Anweisungen mit einem SDK AWS

Im folgenden Codebeispiel wird gezeigt, wie Sie Daten mit PartiQL-UPDATE-Anweisungen aktualisieren.

JavaScript

SDK für JavaScript (v3)

Aktualisieren Sie Elemente in einer DynamoDB-Tabelle mithilfe von PartiQL UPDATE-Anweisungen mit. AWS SDK für JavaScript

```
/**
 * This example demonstrates how to update items in a DynamoDB table using
 PartiQL.
 * It shows different ways to update documents with various index types.
 */
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";
import {
  DynamoDBDocumentClient,
  ExecuteStatementCommand,
  BatchExecuteStatementCommand,
} from "@aws-sdk/lib-dynamodb";

/**
 * Update a single attribute of an item using PartiQL.
 *
 * @param tableName - The name of the DynamoDB table
 * @param partitionKeyName - The name of the partition key attribute
 * @param partitionKeyValue - The value of the partition key
 * @param attributeName - The name of the attribute to update
 * @param attributeValue - The new value for the attribute
 * @returns The response from the ExecuteStatementCommand
 */
export const updateSingleAttribute = async (
  tableName: string,
  partitionKeyName: string,
  partitionKeyValue: string | number,
  attributeName: string,
  attributeValue: any
```

```
) => {
  const client = new DynamoDBClient({});
  const docClient = DynamoDBDocumentClient.from(client);

  const params = {
    Statement: `UPDATE "${tableName}" SET ${attributeName} = ? WHERE
 ${partitionKeyName} = ?`,
    Parameters: [attributeValue, partitionKeyValue],
  };

  try {
    const data = await docClient.send(new ExecuteStatementCommand(params));
    console.log("Item updated successfully");
    return data;
  } catch (err) {
    console.error("Error updating item:", err);
    throw err;
  }
};

/**
 * Update multiple attributes of an item using PartiQL.
 *
 * @param tableName - The name of the DynamoDB table
 * @param partitionKeyName - The name of the partition key attribute
 * @param partitionKeyValue - The value of the partition key
 * @param attributeUpdates - Object containing attribute names and their new
 values
 * @returns The response from the ExecuteStatementCommand
 */
export const updateMultipleAttributes = async (
  tableName: string,
  partitionKeyName: string,
  partitionKeyValue: string | number,
  attributeUpdates: Record<string, any>
) => {
  const client = new DynamoDBClient({});
  const docClient = DynamoDBDocumentClient.from(client);

  // Create SET clause for each attribute
  const setClause = Object.keys(attributeUpdates)
    .map((attr, index) => `${attr} = ?`)
    .join(", ");
```

```
  // Create parameters array with attribute values followed by the partition key
 value
  const parameters = [...Object.values(attributeUpdates), partitionKeyValue];

  const params = {
    Statement: `UPDATE "${tableName}" SET ${setClause} WHERE ${partitionKeyName}
 = ?`,
    Parameters: parameters,
  };

  try {
    const data = await docClient.send(new ExecuteStatementCommand(params));
    console.log("Item updated successfully");
    return data;
  } catch (err) {
    console.error("Error updating item:", err);
    throw err;
  }
};

/**
 * Update an item identified by a composite key (partition key + sort key) using
 PartiQL.
 *
 * @param tableName - The name of the DynamoDB table
 * @param partitionKeyName - The name of the partition key attribute
 * @param partitionKeyValue - The value of the partition key
 * @param sortKeyName - The name of the sort key attribute
 * @param sortKeyValue - The value of the sort key
 * @param attributeName - The name of the attribute to update
 * @param attributeValue - The new value for the attribute
 * @returns The response from the ExecuteStatementCommand
 */
export const updateItemWithCompositeKey = async (
  tableName: string,
  partitionKeyName: string,
  partitionKeyValue: string | number,
  sortKeyName: string,
  sortKeyValue: string | number,
  attributeName: string,
  attributeValue: any
) => {
  const client = new DynamoDBClient({});
  const docClient = DynamoDBDocumentClient.from(client);
```

```
  const params = {
    Statement: `UPDATE "${tableName}" SET ${attributeName} = ? WHERE
 ${partitionKeyName} = ? AND ${sortKeyName} = ?`,
    Parameters: [attributeValue, partitionKeyValue, sortKeyValue],
  };

  try {
    const data = await docClient.send(new ExecuteStatementCommand(params));
    console.log("Item updated successfully");
    return data;
  } catch (err) {
    console.error("Error updating item:", err);
    throw err;
  }
};

/**
 * Update an item with a condition to ensure the update only happens if a
 condition is met.
 *
 * @param tableName - The name of the DynamoDB table
 * @param partitionKeyName - The name of the partition key attribute
 * @param partitionKeyValue - The value of the partition key
 * @param attributeName - The name of the attribute to update
 * @param attributeValue - The new value for the attribute
 * @param conditionAttribute - The attribute to check in the condition
 * @param conditionValue - The value to compare against in the condition
 * @returns The response from the ExecuteStatementCommand
 */
export const updateItemWithCondition = async (
  tableName: string,
  partitionKeyName: string,
  partitionKeyValue: string | number,
  attributeName: string,
  attributeValue: any,
  conditionAttribute: string,
  conditionValue: any
) => {
  const client = new DynamoDBClient({});
  const docClient = DynamoDBDocumentClient.from(client);

  const params = {
```

```
    Statement: `UPDATE "${tableName}" SET ${attributeName} = ? WHERE
 ${partitionKeyName} = ? AND ${conditionAttribute} = ?`,
    Parameters: [attributeValue, partitionKeyValue, conditionValue],
  };

  try {
    const data = await docClient.send(new ExecuteStatementCommand(params));
    console.log("Item updated with condition successfully");
    return data;
  } catch (err) {
    console.error("Error updating item with condition:", err);
    throw err;
  }
};

/**
 * Batch update multiple items using PartiQL.
 *
 * @param tableName - The name of the DynamoDB table
 * @param updates - Array of objects containing key and update information
 * @returns The response from the BatchExecuteStatementCommand
 */
export const batchUpdateItems = async (
  tableName: string,
  updates: Array<{
    partitionKeyName: string;
    partitionKeyValue: string | number;
    attributeName: string;
    attributeValue: any;
  }>
) => {
  const client = new DynamoDBClient({});
  const docClient = DynamoDBDocumentClient.from(client);

  // Create statements for each update
  const statements = updates.map((update) => {
    return {
      Statement: `UPDATE "${tableName}" SET ${update.attributeName} = ? WHERE
 ${update.partitionKeyName} = ?`,
      Parameters: [update.attributeValue, update.partitionKeyValue],
    };
  });

  const params = {
```

```
    Statements: statements,
  };

  try {
    const data = await docClient.send(new BatchExecuteStatementCommand(params));
    console.log("Items batch updated successfully");
    return data;
  } catch (err) {
    console.error("Error batch updating items:", err);
    throw err;
  }
};

/**
 * Example usage showing how to update items with different index types
 */
export const updateExamples = async () => {
  // Update a single attribute using a simple primary key
  await updateSingleAttribute("UsersTable", "userId", "user123", "email",
 "newemail@example.com");

  // Update multiple attributes at once
  await updateMultipleAttributes("UsersTable", "userId", "user123", {
    email: "newemail@example.com",
    name: "John Smith",
    lastLogin: new Date().toISOString(),
  });

  // Update an item with a composite key (partition key + sort key)
  await updateItemWithCompositeKey(
    "OrdersTable",
    "orderId",
    "order456",
    "productId",
    "prod789",
    "quantity",
    5
  );

  // Update with a condition
  await updateItemWithCondition(
    "UsersTable",
    "userId",
    "user123",
```

```
    "userStatus",
    "active",
    "userType",
    "premium"
  );

  // Batch update multiple items
  await batchUpdateItems("UsersTable", [
    {
      partitionKeyName: "userId",
      partitionKeyValue: "user123",
      attributeName: "lastLogin",
      attributeValue: new Date().toISOString(),
    },
    {
      partitionKeyName: "userId",
      partitionKeyValue: "user456",
      attributeName: "lastLogin",
      attributeValue: new Date().toISOString(),
    },
  ]);
};
```

- Weitere API-Informationen finden Sie in den folgenden Themen der AWS SDK für JavaScript -API-Referenz.
  - BatchExecuteStatement
  - ExecuteStatement

Eine vollständige Liste der AWS SDK-Entwicklerhandbücher und Codebeispiele finden Sie unter. DynamoDB mit einem SDK verwenden AWS Dieses Thema enthält auch Informationen zu den ersten Schritten und Details zu früheren SDK-Versionen.

Verwenden von API Gateway zum Aufrufen einer Lambda-Funktion

Die folgenden Codebeispiele zeigen, wie eine von Amazon API Gateway aufgerufene AWS Lambda Funktion erstellt wird.

Java

SDK für Java 2.x

Zeigt, wie eine AWS Lambda Funktion mithilfe der Lambda Java Runtime API erstellt wird. In diesem Beispiel werden verschiedene AWS Dienste aufgerufen, um einen bestimmten Anwendungsfall auszuführen. Dieses Beispiel zeigt, wie man eine Lambda-Funktion erstellt, die von Amazon API Gateway aufgerufen wird und eine Amazon-DynamoDB-Tabelle nach Arbeitsjubiläen durchsucht und Amazon Simple Notification Service (Amazon SNS) verwendet, um eine Textnachricht an Ihre Mitarbeiter zu senden, die ihnen zu ihrem einjährigen Jubiläum gratuliert.

Den vollständigen Quellcode und Anweisungen zur Einrichtung und Ausführung finden Sie im vollständigen Beispiel unter GitHub.

In diesem Beispiel verwendete Dienste

- API Gateway
- DynamoDB
- Lambda
- Amazon SNS

JavaScript

SDK für JavaScript (v3)

Zeigt, wie eine AWS Lambda Funktion mithilfe der JavaScript Lambda-Laufzeit-API erstellt wird. In diesem Beispiel werden verschiedene AWS Dienste aufgerufen, um einen bestimmten Anwendungsfall auszuführen. Dieses Beispiel zeigt, wie man eine Lambda-Funktion erstellt, die von Amazon API Gateway aufgerufen wird und eine Amazon-DynamoDB-Tabelle nach Arbeitsjubiläen durchsucht und Amazon Simple Notification Service (Amazon SNS) verwendet, um eine Textnachricht an Ihre Mitarbeiter zu senden, die ihnen zu ihrem einjährigen Jubiläum gratuliert.

Den vollständigen Quellcode und Anweisungen zur Einrichtung und Ausführung finden Sie im vollständigen Beispiel unter GitHub.

Dieses Beispiel ist auch verfügbar im AWS SDK für JavaScript Entwicklerhandbuch für v3.

In diesem Beispiel verwendete Dienste

- API Gateway
- DynamoDB
- Lambda
- Amazon SNS

Python

SDK für Python (Boto3)

Dieses Beispiel veranschaulicht, wie eine REST-API für Amazon API Gateway erstellt und verwendet wird, die auf eine AWS Lambda -Funktion verweist. Der Lambda-Handler veranschaulicht, wie basierend auf HTTP-Methoden weitergeleitet wird, wie Daten aus der Abfragezeichenfolge, dem Header und dem Text abgerufen werden und wie eine JSON-Antwort zurückgegeben wird.

- Stellen Sie eine Lambda-Funktion bereit.
- REST-API für API Gateway erstellen
- Erstellen Sie eine REST-Ressource, die auf die Lambda-Funktion verweist.
- Erteilen Sie API Gateway die Berechtigung, die Lambda-Funktion aufzurufen.
- Verwenden Sie das Anforderungspaket, um Anforderungen an die REST-API zu senden.
- Bereinigen Sie alle Ressourcen, die während der Demo erstellt wurden.

Dieses Beispiel lässt sich am besten auf ansehen GitHub. Den vollständigen Quellcode und Anweisungen zur Einrichtung und Ausführung finden Sie im vollständigen Beispiel unter GitHub.

In diesem Beispiel verwendete Dienste

- API Gateway
- DynamoDB
- Lambda
- Amazon SNS

Eine vollständige Liste der AWS SDK-Entwicklerhandbücher und Codebeispiele finden Sie unterDynamoDB mit einem SDK verwenden AWS. Dieses Thema enthält auch Informationen zu den ersten Schritten und Details zu früheren SDK-Versionen.

Verwenden von Step Functions, um Lambda-Funktionen aufzurufen

Das folgende Codebeispiel zeigt, wie Sie eine AWS Step Functions Zustandsmaschine erstellen, die nacheinander AWS Lambda Funktionen aufruft.

Java

SDK für Java 2.x

Zeigt, wie Sie mithilfe von AWS Step Functions und einen AWS serverlosen Workflow erstellen. AWS SDK for Java 2.x Jeder Workflow-Schritt wird mithilfe einer AWS Lambda Funktion implementiert.

Den vollständigen Quellcode und Anweisungen zur Einrichtung und Ausführung finden Sie im vollständigen Beispiel unter GitHub.

In diesem Beispiel verwendete Dienste

- DynamoDB
- Lambda
- Amazon SES
- Step Functions

Eine vollständige Liste der AWS SDK-Entwicklerhandbücher und Codebeispiele finden Sie unterDynamoDB mit einem SDK verwenden AWS. Dieses Thema enthält auch Informationen zu den ersten Schritten und Details zu früheren SDK-Versionen.

Verwenden Sie ein Dokumentmodell für DynamoDB mithilfe eines SDK AWS

Das folgende Codebeispiel zeigt, wie Create, Read, Update, Delete (CRUD) und Batch-Operationen mithilfe eines Dokumentmodells für DynamoDB und eines SDK ausgeführt werden. AWS

Weitere Informationen finden Sie unter Dokumentmodell.

.NET

SDK für .NET

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

Führen Sie CRUD-Operationen unter Verwendung eines Dokumentmodells durch.

```
/// <summary>
/// Performs CRUD operations on an Amazon DynamoDB table.
/// </summary>
public class MidlevelItemCRUD
{
    public static async Task Main()
    {
        var tableName = "ProductCatalog";
        var sampleBookId = 555;

        var client = new AmazonDynamoDBClient();
        var productCatalog = LoadTable(client, tableName);

        await CreateBookItem(productCatalog, sampleBookId);
        RetrieveBook(productCatalog, sampleBookId);

        // Couple of sample updates.
        UpdateMultipleAttributes(productCatalog, sampleBookId);
        UpdateBookPriceConditionally(productCatalog, sampleBookId);

        // Delete.
        await DeleteBook(productCatalog, sampleBookId);
    }

    /// <summary>
    /// Loads the contents of a DynamoDB table.
    /// </summary>
    /// <param name="client">An initialized DynamoDB client object.</param>
    /// <param name="tableName">The name of the table to load.</param>
    /// <returns>A DynamoDB table object.</returns>
```

```
        public static Table LoadTable(IAmazonDynamoDB client, string tableName)
        {
            Table productCatalog = Table.LoadTable(client, tableName);
            return productCatalog;
        }

        /// <summary>
        /// Creates an example book item and adds it to the DynamoDB table
        /// ProductCatalog.
        /// </summary>
        /// <param name="productCatalog">A DynamoDB table object.</param>
        /// <param name="sampleBookId">An integer value representing the book's
ID.</param>
        public static async Task CreateBookItem(Table productCatalog, int
sampleBookId)
        {
            Console.WriteLine("\n*** Executing CreateBookItem() ***");
            var book = new Document
            {
                ["Id"] = sampleBookId,
                ["Title"] = "Book " + sampleBookId,
                ["Price"] = 19.99,
                ["ISBN"] = "111-1111111111",
                ["Authors"] = new List<string> { "Author 1", "Author 2", "Author
3" },
                ["PageCount"] = 500,
                ["Dimensions"] = "8.5x11x.5",
                ["InPublication"] = new DynamoDBBool(true),
                ["InStock"] = new DynamoDBBool(false),
                ["QuantityOnHand"] = 0,
            };

            // Adds the book to the ProductCatalog table.
            await productCatalog.PutItemAsync(book);
        }

        /// <summary>
        /// Retrieves an item, a book, from the DynamoDB ProductCatalog table.
        /// </summary>
        /// <param name="productCatalog">A DynamoDB table object.</param>
        /// <param name="sampleBookId">An integer value representing the book's
ID.</param>
        public static async void RetrieveBook(
          Table productCatalog,
```

```
          int sampleBookId)
        {
            Console.WriteLine("\n*** Executing RetrieveBook() ***");

            // Optional configuration.
            var config = new GetItemOperationConfig
            {
                AttributesToGet = new List<string> { "Id", "ISBN", "Title",
"Authors", "Price" },
                ConsistentRead = true,
            };

            Document document = await productCatalog.GetItemAsync(sampleBookId,
config);
            Console.WriteLine("RetrieveBook: Printing book retrieved...");
            PrintDocument(document);
        }

        /// <summary>
        /// Updates multiple attributes for a book and writes the changes to the
        /// DynamoDB table ProductCatalog.
        /// </summary>
        /// <param name="productCatalog">A DynamoDB table object.</param>
        /// <param name="sampleBookId">An integer value representing the book's
ID.</param>
        public static async void UpdateMultipleAttributes(
          Table productCatalog,
          int sampleBookId)
        {
            Console.WriteLine("\nUpdating multiple attributes....");
            int partitionKey = sampleBookId;

            var book = new Document
            {
                ["Id"] = partitionKey,

                // List of attribute updates.
                // The following replaces the existing authors list.
                ["Authors"] = new List<string> { "Author x", "Author y" },
                ["newAttribute"] = "New Value",
                ["ISBN"] = null, // Remove it.
            };

            // Optional parameters.
```

```
            var config = new UpdateItemOperationConfig
            {
                // Gets updated item in response.
                ReturnValues = ReturnValues.AllNewAttributes,
            };

            Document updatedBook = await productCatalog.UpdateItemAsync(book,
config);
            Console.WriteLine("UpdateMultipleAttributes: Printing item after
updates ...");
            PrintDocument(updatedBook);
        }

        /// <summary>
        /// Updates a book item if it meets the specified criteria.
        /// </summary>
        /// <param name="productCatalog">A DynamoDB table object.</param>
        /// <param name="sampleBookId">An integer value representing the book's
ID.</param>
        public static async void UpdateBookPriceConditionally(
          Table productCatalog,
          int sampleBookId)
        {
            Console.WriteLine("\n*** Executing UpdateBookPriceConditionally()
***");

            int partitionKey = sampleBookId;

            var book = new Document
            {
                ["Id"] = partitionKey,
                ["Price"] = 29.99,
            };

            // For conditional price update, creating a condition expression.
            var expr = new Expression
            {
                ExpressionStatement = "Price = :val",
            };
            expr.ExpressionAttributeValues[":val"] = 19.00;

            // Optional parameters.
            var config = new UpdateItemOperationConfig
            {
```

```
                ConditionalExpression = expr,
                ReturnValues = ReturnValues.AllNewAttributes,
            };

            Document updatedBook = await productCatalog.UpdateItemAsync(book,
config);
            Console.WriteLine("UpdateBookPriceConditionally: Printing item whose
price was conditionally updated");
            PrintDocument(updatedBook);
        }

        /// <summary>
        /// Deletes the book with the supplied Id value from the DynamoDB table
        /// ProductCatalog.
        /// </summary>
        /// <param name="productCatalog">A DynamoDB table object.</param>
        /// <param name="sampleBookId">An integer value representing the book's
ID.</param>
        public static async Task DeleteBook(
          Table productCatalog,
          int sampleBookId)
        {
            Console.WriteLine("\n*** Executing DeleteBook() ***");

            // Optional configuration.
            var config = new DeleteItemOperationConfig
            {
                // Returns the deleted item.
                ReturnValues = ReturnValues.AllOldAttributes,
            };
            Document document = await
productCatalog.DeleteItemAsync(sampleBookId, config);
            Console.WriteLine("DeleteBook: Printing deleted just deleted...");

            PrintDocument(document);
        }

        /// <summary>
        /// Prints the information for the supplied DynamoDB document.
        /// </summary>
        /// <param name="updatedDocument">A DynamoDB document object.</param>
        public static void PrintDocument(Document updatedDocument)
        {
            if (updatedDocument is null)
```

```
            {
                return;
            }

            foreach (var attribute in updatedDocument.GetAttributeNames())
            {
                string stringValue = null;
                var value = updatedDocument[attribute];

                if (value is null)
                {
                    continue;
                }

                if (value is Primitive)
                {
                    stringValue = value.AsPrimitive().Value.ToString();
                }
                else if (value is PrimitiveList)
                {
                    stringValue = string.Join(",", (from primitive
                      in value.AsPrimitiveList().Entries
                                                    select
primitive.Value).ToArray());
                }

                Console.WriteLine($"{attribute} - {stringValue}", attribute,
stringValue);
            }
        }
    }
```

Führen Sie Batch-Schreibvorgänge unter Verwendung eines Dokumentmodells durch.

```
    /// <summary>
    /// Shows how to use mid-level Amazon DynamoDB API calls to perform batch
    /// operations.
    /// </summary>
    public class MidLevelBatchWriteItem
    {
```

```
        public static async Task Main()
        {
            IAmazonDynamoDB client = new AmazonDynamoDBClient();

            await SingleTableBatchWrite(client);
            await MultiTableBatchWrite(client);
        }

        /// <summary>
        /// Perform a batch operation on a single DynamoDB table.
        /// </summary>
        /// <param name="client">An initialized DynamoDB object.</param>
        public static async Task SingleTableBatchWrite(IAmazonDynamoDB client)
        {
            Table productCatalog = Table.LoadTable(client, "ProductCatalog");
            var batchWrite = productCatalog.CreateBatchWrite();

            var book1 = new Document
            {
                ["Id"] = 902,
                ["Title"] = "My book1 in batch write using .NET helper classes",
                ["ISBN"] = "902-11-11-1111",
                ["Price"] = 10,
                ["ProductCategory"] = "Book",
                ["Authors"] = new List<string> { "Author 1", "Author 2", "Author
3" },
                ["Dimensions"] = "8.5x11x.5",
                ["InStock"] = new DynamoDBBool(true),
                ["QuantityOnHand"] = new DynamoDBNull(), // Quantity is unknown
at this time.
            };

            batchWrite.AddDocumentToPut(book1);

            // Specify delete item using overload that takes PK.
            batchWrite.AddKeyToDelete(12345);
            Console.WriteLine("Performing batch write in
SingleTableBatchWrite()");
            await batchWrite.ExecuteAsync();
        }

        /// <summary>
        /// Perform a batch operation involving multiple DynamoDB tables.
        /// </summary>
```

```
        /// <param name="client">An initialized DynamoDB client object.</param>
        public static async Task MultiTableBatchWrite(IAmazonDynamoDB client)
        {
            // Specify item to add in the Forum table.
            Table forum = Table.LoadTable(client, "Forum");
            var forumBatchWrite = forum.CreateBatchWrite();

            var forum1 = new Document
            {
                ["Name"] = "Test BatchWrite Forum",
                ["Threads"] = 0,
            };
            forumBatchWrite.AddDocumentToPut(forum1);

            // Specify item to add in the Thread table.
            Table thread = Table.LoadTable(client, "Thread");
            var threadBatchWrite = thread.CreateBatchWrite();

            var thread1 = new Document
            {
                ["ForumName"] = "S3 forum",
                ["Subject"] = "My sample question",
                ["Message"] = "Message text",
                ["KeywordTags"] = new List<string> { "S3", "Bucket" },
            };
            threadBatchWrite.AddDocumentToPut(thread1);

            // Specify item to delete from the Thread table.
            threadBatchWrite.AddKeyToDelete("someForumName", "someSubject");

            // Create multi-table batch.
            var superBatch = new MultiTableDocumentBatchWrite();
            superBatch.AddBatch(forumBatchWrite);
            superBatch.AddBatch(threadBatchWrite);
            Console.WriteLine("Performing batch write in
MultiTableBatchWrite()");

            // Execute the batch.
            await superBatch.ExecuteAsync();
        }
    }
```

Scannen Sie eine Tabelle unter Verwendung eines Dokumentmodells.

```
   /// <summary>
   /// Shows how to use mid-level Amazon DynamoDB API calls to scan a DynamoDB
   /// table for values.
   /// </summary>
   public class MidLevelScanOnly
   {
       public static async Task Main()
       {
           IAmazonDynamoDB client = new AmazonDynamoDBClient();

           Table productCatalogTable = Table.LoadTable(client,
"ProductCatalog");

           await FindProductsWithNegativePrice(productCatalogTable);
           await FindProductsWithNegativePriceWithConfig(productCatalogTable);
       }

       /// <summary>
       /// Retrieves any products that have a negative price in a DynamoDB
table.
       /// </summary>
       /// <param name="productCatalogTable">A DynamoDB table object.</param>
       public static async Task FindProductsWithNegativePrice(
         Table productCatalogTable)
       {
           // Assume there is a price error. So we scan to find items priced <
0.
           var scanFilter = new ScanFilter();
           scanFilter.AddCondition("Price", ScanOperator.LessThan, 0);

           Search search = productCatalogTable.Scan(scanFilter);

           do
           {
               var documentList = await search.GetNextSetAsync();
               Console.WriteLine("\nFindProductsWithNegativePrice:
printing .....");

               foreach (var document in documentList)
               {
                   PrintDocument(document);
```

```
                }
            }
            while (!search.IsDone);
        }

        /// <summary>
        /// Finds any items in the ProductCatalog table using a DynamoDB
        /// configuration object.
        /// </summary>
        /// <param name="productCatalogTable">A DynamoDB table object.</param>
        public static async Task FindProductsWithNegativePriceWithConfig(
          Table productCatalogTable)
        {
            // Assume there is a price error. So we scan to find items priced <
0.
            var scanFilter = new ScanFilter();
            scanFilter.AddCondition("Price", ScanOperator.LessThan, 0);

            var config = new ScanOperationConfig()
            {
                Filter = scanFilter,
                Select = SelectValues.SpecificAttributes,
                AttributesToGet = new List<string> { "Title", "Id" },
            };

            Search search = productCatalogTable.Scan(config);

            do
            {
                var documentList = await search.GetNextSetAsync();
                Console.WriteLine("\nFindProductsWithNegativePriceWithConfig:
printing .....");

                foreach (var document in documentList)
                {
                    PrintDocument(document);
                }
            }
            while (!search.IsDone);
        }

        /// <summary>
        /// Displays the details of the passed DynamoDB document object on the
        /// console.
```

```
        /// </summary>
        /// <param name="document">A DynamoDB document object.</param>
        public static void PrintDocument(Document document)
        {
            Console.WriteLine();
            foreach (var attribute in document.GetAttributeNames())
            {
                string stringValue = null;
                var value = document[attribute];
                if (value is Primitive)
                {
                    stringValue = value.AsPrimitive().Value.ToString();
                }
                else if (value is PrimitiveList)
                {
                    stringValue = string.Join(",", (from primitive
                      in value.AsPrimitiveList().Entries
                                                    select
primitive.Value).ToArray());
                }

                Console.WriteLine($"{attribute} - {stringValue}");
            }
        }
    }
```

Verwenden Sie ein Dokumentmodell, um eine Tabelle abzufragen und zu scannen.

```
    /// <summary>
    /// Shows how to perform mid-level query procedures on an Amazon DynamoDB
    /// table.
    /// </summary>
    public class MidLevelQueryAndScan
    {
        public static async Task Main()
        {
            IAmazonDynamoDB client = new AmazonDynamoDBClient();

            // Query examples.
            Table replyTable = Table.LoadTable(client, "Reply");
```

```
            string forumName = "Amazon DynamoDB";
            string threadSubject = "DynamoDB Thread 2";

            await FindRepliesInLast15Days(replyTable);
            await FindRepliesInLast15DaysWithConfig(replyTable, forumName,
threadSubject);
            await FindRepliesPostedWithinTimePeriod(replyTable, forumName,
threadSubject);

            // Get Example.
            Table productCatalogTable = Table.LoadTable(client,
"ProductCatalog");
            int productId = 101;

            await GetProduct(productCatalogTable, productId);
        }

        /// <summary>
        /// Retrieves information about a product from the DynamoDB table
        /// ProductCatalog based on the product ID and displays the information
        /// on the console.
        /// </summary>
        /// <param name="tableName">The name of the table from which to retrieve
        /// product information.</param>
        /// <param name="productId">The ID of the product to retrieve.</param>
        public static async Task GetProduct(Table tableName, int productId)
        {
            Console.WriteLine("*** Executing GetProduct() ***");
            Document productDocument = await tableName.GetItemAsync(productId);
            if (productDocument != null)
            {
                PrintDocument(productDocument);
            }
            else
            {
                Console.WriteLine("Error: product " + productId + " does not
exist");
            }
        }

        /// <summary>
        /// Retrieves replies from the passed DynamoDB table object.
        /// </summary>
        /// <param name="table">The table we want to query.</param>
```

```
        public static async Task FindRepliesInLast15Days(
          Table table)
        {
            DateTime twoWeeksAgoDate = DateTime.UtcNow - TimeSpan.FromDays(15);
            var filter = new QueryFilter("Id", QueryOperator.Equal, "Id");
            filter.AddCondition("ReplyDateTime", QueryOperator.GreaterThan,
twoWeeksAgoDate);

            // Use Query overloads that take the minimum required query
parameters.
            Search search = table.Query(filter);

            do
            {
                var documentSet = await search.GetNextSetAsync();
                Console.WriteLine("\nFindRepliesInLast15Days:
printing .............");

                foreach (var document in documentSet)
                {
                    PrintDocument(document);
                }
            }
            while (!search.IsDone);
        }

        /// <summary>
        /// Retrieve replies made during a specific time period.
        /// </summary>
        /// <param name="table">The table we want to query.</param>
        /// <param name="forumName">The name of the forum that we're interested
in.</param>
        /// <param name="threadSubject">The subject of the thread, which we are
        /// searching for replies.</param>
        public static async Task FindRepliesPostedWithinTimePeriod(
          Table table,
          string forumName,
          string threadSubject)
        {
            DateTime startDate = DateTime.UtcNow.Subtract(new TimeSpan(21, 0, 0,
0));
            DateTime endDate = DateTime.UtcNow.Subtract(new TimeSpan(1, 0, 0,
0));
```

```
            var filter = new QueryFilter("Id", QueryOperator.Equal, forumName +
"#" + threadSubject);
            filter.AddCondition("ReplyDateTime", QueryOperator.Between,
startDate, endDate);

            var config = new QueryOperationConfig()
            {
                Limit = 2, // 2 items/page.
                Select = SelectValues.SpecificAttributes,
                AttributesToGet = new List<string>
       {
         "Message",
         "ReplyDateTime",
         "PostedBy",
       },
                ConsistentRead = true,
                Filter = filter,
            };

            Search search = table.Query(config);

            do
            {
                var documentList = await search.GetNextSetAsync();
                Console.WriteLine("\nFindRepliesPostedWithinTimePeriod: printing
replies posted within dates: {0} and {1} .....", startDate, endDate);

                foreach (var document in documentList)
                {
                    PrintDocument(document);
                }
            }
            while (!search.IsDone);
       }

       /// <summary>
       /// Perform a query for replies made in the last 15 days using a DynamoDB
       /// QueryOperationConfig object.
       /// </summary>
       /// <param name="table">The table we want to query.</param>
       /// <param name="forumName">The name of the forum that we're interested
in.</param>
       /// <param name="threadName">The bane of the thread that we are searching
       /// for replies.</param>
```

```
        public static async Task FindRepliesInLast15DaysWithConfig(
          Table table,
          string forumName,
          string threadName)
        {
            DateTime twoWeeksAgoDate = DateTime.UtcNow - TimeSpan.FromDays(15);
            var filter = new QueryFilter("Id", QueryOperator.Equal, forumName +
"#" + threadName);
            filter.AddCondition("ReplyDateTime", QueryOperator.GreaterThan,
twoWeeksAgoDate);

            var config = new QueryOperationConfig()
            {
                Filter = filter,

                // Optional parameters.
                Select = SelectValues.SpecificAttributes,
                AttributesToGet = new List<string>
                {
                  "Message",
                  "ReplyDateTime",
                  "PostedBy",
                },
                ConsistentRead = true,
            };

            Search search = table.Query(config);

            do
            {
                var documentSet = await search.GetNextSetAsync();
                Console.WriteLine("\nFindRepliesInLast15DaysWithConfig:
printing .....");

                foreach (var document in documentSet)
                {
                    PrintDocument(document);
                }
            }
            while (!search.IsDone);
        }

        /// <summary>
        /// Displays the contents of the passed DynamoDB document on the console.
```

```
        /// </summary>
        /// <param name="document">A DynamoDB document to display.</param>
        public static void PrintDocument(Document document)
        {
            Console.WriteLine();
            foreach (var attribute in document.GetAttributeNames())
            {
                string stringValue = null;
                var value = document[attribute];

                if (value is Primitive)
                {
                    stringValue = value.AsPrimitive().Value.ToString();
                }
                else if (value is PrimitiveList)
                {
                    stringValue = string.Join(",", (from primitive
                      in value.AsPrimitiveList().Entries
                                                    select
primitive.Value).ToArray());
                }

                Console.WriteLine($"{attribute} - {stringValue}");
            }
        }
    }
```

Eine vollständige Liste der AWS SDK-Entwicklerhandbücher und Codebeispiele finden Sie unterDynamoDB mit einem SDK verwenden AWS. Dieses Thema enthält auch Informationen zu den ersten Schritten und Details zu früheren SDK-Versionen.

Verwenden Sie ein Objektpersistenzmodell auf hoher Ebene für DynamoDB mithilfe eines SDK AWS

Das folgende Codebeispiel zeigt, wie Create, Read, Update, Delete (CRUD) und Batch-Operationen mit einem Objektpersistenzmodell für DynamoDB und einem SDK ausgeführt werden. AWS

Weitere Informationen finden Sie unter Object-Persistence-Modell.

.NET

SDK für .NET

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

Führen Sie CRUD-Operationen unter Verwendung eines übergeordneten Object-Persistence-Modells durch.

```
/// <summary>
/// Shows how to perform high-level CRUD operations on an Amazon DynamoDB
/// table.
/// </summary>
public class HighLevelItemCrud
{
    public static async Task Main()
    {
        var client = new AmazonDynamoDBClient();
        DynamoDBContext context = new DynamoDBContext(client);
        await PerformCRUDOperations(context);
    }

    public static async Task PerformCRUDOperations(IDynamoDBContext context)
    {
        int bookId = 1001; // Some unique value.
        Book myBook = new Book
        {
            Id = bookId,
            Title = "object persistence-AWS SDK for.NET SDK-Book 1001",
            Isbn = "111-1111111001",
            BookAuthors = new List<string> { "Author 1", "Author 2" },
        };

        // Save the book to the ProductCatalog table.
        await context.SaveAsync(myBook);

        // Retrieve the book from the ProductCatalog table.
```

```
            Book bookRetrieved = await context.LoadAsync<Book>(bookId);

            // Update some properties.
            bookRetrieved.Isbn = "222-2222221001";

            // Update existing authors list with the following values.
            bookRetrieved.BookAuthors = new List<string> { " Author 1", "Author
x" };
            await context.SaveAsync(bookRetrieved);

            // Retrieve the updated book. This time, add the optional
            // ConsistentRead parameter using DynamoDBContextConfig object.
            await context.LoadAsync<Book>(bookId, new DynamoDBContextConfig
            {
                ConsistentRead = true,
            });

            // Delete the book.
            await context.DeleteAsync<Book>(bookId);

            // Try to retrieve deleted book. It should return null.
            Book deletedBook = await context.LoadAsync<Book>(bookId, new
DynamoDBContextConfig
            {
                ConsistentRead = true,
            });

            if (deletedBook == null)
            {
                Console.WriteLine("Book is deleted");
            }
        }
    }
```

Führen Sie Batch-Schreibvorgänge unter Verwendung eines übergeordneten Object-Persistence-Modells durch.

```
    /// <summary>
    /// Performs high-level batch write operations to an Amazon DynamoDB table.
    /// This example was written using the AWS SDK for .NET version 3.7 and .NET
```

```
    /// Core 5.0.
    /// </summary>
    public class HighLevelBatchWriteItem
    {
        public static async Task SingleTableBatchWrite(IDynamoDBContext context)
        {
            Book book1 = new Book
            {
                Id = 902,
                InPublication = true,
                Isbn = "902-11-11-1111",
                PageCount = "100",
                Price = 10,
                ProductCategory = "Book",
                Title = "My book3 in batch write",
            };

            Book book2 = new Book
            {
                Id = 903,
                InPublication = true,
                Isbn = "903-11-11-1111",
                PageCount = "200",
                Price = 10,
                ProductCategory = "Book",
                Title = "My book4 in batch write",
            };

            var bookBatch = context.CreateBatchWrite<Book>();
            bookBatch.AddPutItems(new List<Book> { book1, book2 });

            Console.WriteLine("Adding two books to ProductCatalog table.");
            await bookBatch.ExecuteAsync();
        }

        public static async Task MultiTableBatchWrite(IDynamoDBContext context)
        {
            // New Forum item.
            Forum newForum = new Forum
            {
                Name = "Test BatchWrite Forum",
                Threads = 0,
            };
            var forumBatch = context.CreateBatchWrite<Forum>();
```

```
            forumBatch.AddPutItem(newForum);

            // New Thread item.
            Thread newThread = new Thread
            {
                ForumName = "S3 forum",
                Subject = "My sample question",
                KeywordTags = new List<string> { "S3", "Bucket" },
                Message = "Message text",
            };

            DynamoDBOperationConfig config = new DynamoDBOperationConfig();
            config.SkipVersionCheck = true;
            var threadBatch = context.CreateBatchWrite<Thread>(config);
            threadBatch.AddPutItem(newThread);
            threadBatch.AddDeleteKey("some partition key value", "some sort key
value");

            var superBatch = new MultiTableBatchWrite(forumBatch, threadBatch);

            Console.WriteLine("Performing batch write in
MultiTableBatchWrite().");
            await superBatch.ExecuteAsync();
        }

        public static async Task Main()
        {
            AmazonDynamoDBClient client = new AmazonDynamoDBClient();
            DynamoDBContext context = new DynamoDBContext(client);

            await SingleTableBatchWrite(context);
            await MultiTableBatchWrite(context);
        }
    }
```

Weisen Sie einer Tabelle unter Verwendung eines übergeordneten Object-Persistence-Modells beliebige Daten zu.

```
    /// <summary>
    /// Shows how to map arbitrary data to an Amazon DynamoDB table.
```

```
    /// </summary>
    public class HighLevelMappingArbitraryData
    {
        /// <summary>
        /// Creates a book, adds it to the DynamoDB ProductCatalog table,
retrieves
        /// the new book from the table, updates the dimensions and writes the
        /// changed item back to the table.
        /// </summary>
        /// <param name="context">The DynamoDB context object used to write and
        /// read data from the table.</param>
        public static async Task AddRetrieveUpdateBook(IDynamoDBContext context)
        {
            // Create a book.
            DimensionType myBookDimensions = new DimensionType()
            {
                Length = 8M,
                Height = 11M,
                Thickness = 0.5M,
            };

            Book myBook = new Book
            {
                Id = 501,
                Title = "AWS SDK for .NET Object Persistence Model Handling
Arbitrary Data",
                Isbn = "999-9999999999",
                BookAuthors = new List<string> { "Author 1", "Author 2" },
                Dimensions = myBookDimensions,
            };

            // Add the book to the DynamoDB table ProductCatalog.
            await context.SaveAsync(myBook);

            // Retrieve the book.
            Book bookRetrieved = await context.LoadAsync<Book>(501);

            // Update the book dimensions property.
            bookRetrieved.Dimensions.Height += 1;
            bookRetrieved.Dimensions.Length += 1;
            bookRetrieved.Dimensions.Thickness += 0.2M;

            // Write the changed item to the table.
            await context.SaveAsync(bookRetrieved);
```

```
        }

        public static async Task Main()
        {
            var client = new AmazonDynamoDBClient();
            DynamoDBContext context = new DynamoDBContext(client);
            await AddRetrieveUpdateBook(context);
        }
    }
```

Verwenden Sie ein übergeordnetes Object-Persistence-Modell, um eine Tabelle abzufragen und zu scannen.

```
    /// <summary>
    /// Shows how to perform high-level query and scan operations to Amazon
    /// DynamoDB tables.
    /// </summary>
    public class HighLevelQueryAndScan
    {
        public static async Task Main()
        {
            var client = new AmazonDynamoDBClient();

            DynamoDBContext context = new DynamoDBContext(client);

            // Get an item.
            await GetBook(context, 101);

            // Sample forum and thread to test queries.
            string forumName = "Amazon DynamoDB";
            string threadSubject = "DynamoDB Thread 1";

            // Sample queries.
            await FindRepliesInLast15Days(context, forumName, threadSubject);
            await FindRepliesPostedWithinTimePeriod(context, forumName,
threadSubject);

            // Scan table.
            await FindProductsPricedLessThanZero(context);
        }
```

```
        public static async Task GetBook(IDynamoDBContext context, int productId)
        {
            Book bookItem = await context.LoadAsync<Book>(productId);

            Console.WriteLine("\nGetBook: Printing result.....");
            Console.WriteLine($"Title: {bookItem.Title} \n ISBN:{bookItem.Isbn}
\n No. of pages: {bookItem.PageCount}");
        }

        /// <summary>
        /// Queries a DynamoDB table to find replies posted within the last 15
days.
        /// </summary>
        /// <param name="context">The DynamoDB context used to perform the
query.</param>
        /// <param name="forumName">The name of the forum that we're interested
in.</param>
        /// <param name="threadSubject">The thread object containing the query
parameters.</param>
        public static async Task FindRepliesInLast15Days(
          IDynamoDBContext context,
          string forumName,
          string threadSubject)
        {
            string replyId = $"{forumName} #{threadSubject}";
            DateTime twoWeeksAgoDate = DateTime.UtcNow - TimeSpan.FromDays(15);

            List<object> times = new List<object>();
            times.Add(twoWeeksAgoDate);

            List<ScanCondition> scs = new List<ScanCondition>();
            var sc = new ScanCondition("PostedBy", ScanOperator.GreaterThan,
times.ToArray());
            scs.Add(sc);

            var cfg = new DynamoDBOperationConfig
            {
                QueryFilter = scs,
            };

            AsyncSearch<Reply> response = context.QueryAsync<Reply>(replyId,
cfg);
```

```
            IEnumerable<Reply> latestReplies = await
response.GetRemainingAsync();

            Console.WriteLine("\nReplies in last 15 days:");

            foreach (Reply r in latestReplies)
            {

Console.WriteLine($"{r.Id}\t{r.PostedBy}\t{r.Message}\t{r.ReplyDateTime}");
            }
        }

        /// <summary>
        /// Queries for replies posted within a specific time period.
        /// </summary>
        /// <param name="context">The DynamoDB context used to perform the
query.</param>
        /// <param name="forumName">The name of the forum that we're interested
in.</param>
        /// <param name="threadSubject">Information about the subject that we're
        /// interested in.</param>
        public static async Task FindRepliesPostedWithinTimePeriod(
          IDynamoDBContext context,
          string forumName,
          string threadSubject)
        {
            string forumId = forumName + "#" + threadSubject;
            Console.WriteLine("\nReplies posted within time period:");

            DateTime startDate = DateTime.UtcNow - TimeSpan.FromDays(30);
            DateTime endDate = DateTime.UtcNow - TimeSpan.FromDays(1);

            List<object> times = new List<object>();
            times.Add(startDate);
            times.Add(endDate);

            List<ScanCondition> scs = new List<ScanCondition>();
            var sc = new ScanCondition("LastPostedBy", ScanOperator.Between,
times.ToArray());
            scs.Add(sc);

            var cfg = new DynamoDBOperationConfig
            {
                QueryFilter = scs,
```

```
            };

            AsyncSearch<Reply> response = context.QueryAsync<Reply>(forumId,
cfg);
            IEnumerable<Reply> repliesInAPeriod = await
response.GetRemainingAsync();

            foreach (Reply r in repliesInAPeriod)
            {

Console.WriteLine("{r.Id}\t{r.PostedBy}\t{r.Message}\t{r.ReplyDateTime}");
            }
        }

        /// <summary>
        /// Queries the DynamoDB ProductCatalog table for products costing less
        /// than zero.
        /// </summary>
        /// <param name="context">The DynamoDB context object used to perform the
        /// query.</param>
        public static async Task FindProductsPricedLessThanZero(IDynamoDBContext
context)
        {
            int price = 0;

            List<ScanCondition> scs = new List<ScanCondition>();
            var sc1 = new ScanCondition("Price", ScanOperator.LessThan, price);
            var sc2 = new ScanCondition("ProductCategory", ScanOperator.Equal,
"Book");
            scs.Add(sc1);
            scs.Add(sc2);

            AsyncSearch<Book> response = context.ScanAsync<Book>(scs);

            IEnumerable<Book> itemsWithWrongPrice = await
response.GetRemainingAsync();

            Console.WriteLine("\nFindProductsPricedLessThanZero: Printing
result.....");

            foreach (Book r in itemsWithWrongPrice)
            {
                Console.WriteLine($"{r.Id}\t{r.Title}\t{r.Price}\t{r.Isbn}");
            }
```

```
        }
    }
```

Eine vollständige Liste der AWS SDK-Entwicklerhandbücher und Codebeispiele finden Sie unterDynamoDB mit einem SDK verwenden AWS. Dieses Thema enthält auch Informationen zu den ersten Schritten und Details zu früheren SDK-Versionen.

Verwenden Sie atomare Zähleroperationen in DynamoDB mit einem SDK AWS

Die folgenden Codebeispiele zeigen, wie unteilbare Zähleroperationen in DynamoDB verwendet werden.

- Inkrementieren von unteilbaren Zählern mithilfe von ADD- und SET-Operationen
- Sicheres Erhöhen von Zählern, die möglicherweise nicht existieren
- Implementieren einer optimistischen Sperre für Zähleroperationen

Java

SDK für Java 2.x

Demonstrieren Sie atomare Gegenoperationen mit AWS SDK for Java 2.x.

```
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.AttributeValue;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.model.GetItemRequest;
import software.amazon.awssdk.services.dynamodb.model.GetItemResponse;
import software.amazon.awssdk.services.dynamodb.model.ReturnValue;
import software.amazon.awssdk.services.dynamodb.model.UpdateItemRequest;
import software.amazon.awssdk.services.dynamodb.model.UpdateItemResponse;

import java.util.HashMap;
import java.util.Map;

    /**
     * Increments a counter using the ADD operation.
     *
```

```
     * <p>This method demonstrates how to use the ADD operation to atomically
     * increment a counter attribute.
     *
     * @param dynamoDbClient The DynamoDB client
     * @param tableName The name of the DynamoDB table
     * @param key The key of the item to update
     * @param counterName The name of the counter attribute
     * @param incrementValue The value to increment by
     * @return The response from DynamoDB
     * @throws DynamoDbException if an error occurs during the operation
     */
    public static UpdateItemResponse incrementCounterWithAdd(
        DynamoDbClient dynamoDbClient,
        String tableName,
        Map<String, AttributeValue> key,
        String counterName,
        int incrementValue) {

        // Define the update parameters
        UpdateItemRequest request = UpdateItemRequest.builder()
            .tableName(tableName)
            .key(key)
            .updateExpression("ADD #counterName :increment")
            .expressionAttributeNames(Map.of("#counterName", counterName))
            .expressionAttributeValues(Map.of(
                ":increment",

AttributeValue.builder().n(String.valueOf(incrementValue)).build()))
            .returnValues(ReturnValue.UPDATED_NEW)
            .build();

        // Perform the update operation
        return dynamoDbClient.updateItem(request);
    }

    /**
     * Increments a counter using the SET operation.
     *
     * <p>This method demonstrates how to use the SET operation with an
expression
     * to increment a counter attribute.
     *
     * @param dynamoDbClient The DynamoDB client
     * @param tableName The name of the DynamoDB table
```

```
     * @param key The key of the item to update
     * @param counterName The name of the counter attribute
     * @param incrementValue The value to increment by
     * @return The response from DynamoDB
     * @throws DynamoDbException if an error occurs during the operation
     */
    public static UpdateItemResponse incrementCounterWithSet(
        DynamoDbClient dynamoDbClient,
        String tableName,
        Map<String, AttributeValue> key,
        String counterName,
        int incrementValue) {

        // Define the update parameters
        UpdateItemRequest request = UpdateItemRequest.builder()
            .tableName(tableName)
            .key(key)
            .updateExpression("SET #counterName = #counterName + :increment")
            .expressionAttributeNames(Map.of("#counterName", counterName))
            .expressionAttributeValues(Map.of(
                ":increment",

AttributeValue.builder().n(String.valueOf(incrementValue)).build()))
            .returnValues(ReturnValue.UPDATED_NEW)
            .build();

        // Perform the update operation
        return dynamoDbClient.updateItem(request);
    }

    /**
     * Increments a counter safely, handling the case where the counter doesn't
exist yet.
     *
     * <p>This method demonstrates how to use if_not_exists to safely increment a
counter
     * that may not exist yet.
     *
     * @param dynamoDbClient The DynamoDB client
     * @param tableName The name of the DynamoDB table
     * @param key The key of the item to update
     * @param counterName The name of the counter attribute
     * @param incrementValue The value to increment by
     * @return The response from DynamoDB
```

```
     * @throws DynamoDbException if an error occurs during the operation
     */
    public static UpdateItemResponse incrementCounterSafely(
        DynamoDbClient dynamoDbClient,
        String tableName,
        Map<String, AttributeValue> key,
        String counterName,
        int incrementValue) {

        // Define the update parameters
        UpdateItemRequest request = UpdateItemRequest.builder()
            .tableName(tableName)
            .key(key)
            .updateExpression("SET #counterName =
if_not_exists(#counterName, :zero) + :increment")
            .expressionAttributeNames(Map.of("#counterName", counterName))
            .expressionAttributeValues(Map.of(
                ":increment",

AttributeValue.builder().n(String.valueOf(incrementValue)).build(),
                ":zero", AttributeValue.builder().n("0").build()))
            .returnValues(ReturnValue.UPDATED_NEW)
            .build();

        // Perform the update operation
        return dynamoDbClient.updateItem(request);
    }

    /**
     * Decrements a counter safely, ensuring it doesn't go below zero.
     *
     * <p>This method demonstrates how to use a condition expression to safely
     * decrement a counter without going below zero.
     *
     * @param dynamoDbClient The DynamoDB client
     * @param tableName The name of the DynamoDB table
     * @param key The key of the item to update
     * @param counterName The name of the counter attribute
     * @param decrementValue The value to decrement by
     * @return The response from DynamoDB
     * @throws DynamoDbException if an error occurs during the operation or if
the counter would go below zero
     */
    public static UpdateItemResponse decrementCounterSafely(
```

```
        DynamoDbClient dynamoDbClient,
        String tableName,
        Map<String, AttributeValue> key,
        String counterName,
        int decrementValue) {

        // Define the update parameters
        UpdateItemRequest request = UpdateItemRequest.builder()
            .tableName(tableName)
            .key(key)
            .updateExpression("SET #counterName = #counterName - :decrement")
            .conditionExpression("#counterName >= :decrement")
            .expressionAttributeNames(Map.of("#counterName", counterName))
            .expressionAttributeValues(Map.of(
                ":decrement",

AttributeValue.builder().n(String.valueOf(decrementValue)).build()))
            .returnValues(ReturnValue.UPDATED_NEW)
            .build();

        // Perform the update operation
        return dynamoDbClient.updateItem(request);
    }

    /**
     * Compares the ADD and SET approaches for incrementing counters.
     *
     * <p>This method demonstrates the differences between using ADD and SET
     * for incrementing counters in DynamoDB.
     *
     * @param dynamoDbClient The DynamoDB client
     * @param tableName The name of the DynamoDB table
     * @param key The key of the item to update
     * @return Map containing the comparison results
     */
    public static Map<String, Object> compareAddVsSet(
        DynamoDbClient dynamoDbClient, String tableName, Map<String,
AttributeValue> key) {

        Map<String, Object> results = new HashMap<>();

        try {
            // Reset counters to ensure a fair comparison
            UpdateItemRequest resetRequest = UpdateItemRequest.builder()
```

```
                .tableName(tableName)
                .key(key)
                .updateExpression("SET AddCounter = :zero, SetCounter = :zero")
                .expressionAttributeValues(
                    Map.of(":zero", AttributeValue.builder().n("0").build()))
                .build();

            dynamoDbClient.updateItem(resetRequest);

            // Increment with ADD
            long addStartTime = System.nanoTime();
            UpdateItemResponse addResponse =
incrementCounterWithAdd(dynamoDbClient, tableName, key, "AddCounter", 1);
            long addEndTime = System.nanoTime();
            long addDuration = addEndTime - addStartTime;

            // Increment with SET
            long setStartTime = System.nanoTime();
            UpdateItemResponse setResponse =
incrementCounterWithSet(dynamoDbClient, tableName, key, "SetCounter", 1);
            long setEndTime = System.nanoTime();
            long setDuration = setEndTime - setStartTime;

            // Record results
            results.put("addResponse", addResponse);
            results.put("setResponse", setResponse);
            results.put("addDuration", addDuration);
            results.put("setDuration", setDuration);
            results.put("success", true);

        } catch (DynamoDbException e) {
            results.put("success", false);
            results.put("error", e.getMessage());
        }

        return results;
    }

    /**
     * Gets the current value of a counter attribute.
     *
     * <p>Helper method to retrieve the current value of a counter attribute.
     *
     * @param dynamoDbClient The DynamoDB client
```

```
     * @param tableName The name of the DynamoDB table
     * @param key The key of the item to get
     * @param counterName The name of the counter attribute
     * @return The counter value or null if not found
     * @throws DynamoDbException if an error occurs during the operation
     */
    public static Integer getCounterValue(
        DynamoDbClient dynamoDbClient, String tableName, Map<String,
AttributeValue> key, String counterName) {

        // Define the get parameters
        GetItemRequest request = GetItemRequest.builder()
            .tableName(tableName)
            .key(key)
            .projectionExpression(counterName)
            .build();

        // Perform the get operation
        GetItemResponse response = dynamoDbClient.getItem(request);

        // Return the counter value if it exists, otherwise null
        if (response.item() != null && response.item().containsKey(counterName))
{
            return Integer.parseInt(response.item().get(counterName).n());
        }

        return null;
    }
```

Beispiel für die Verwendung atomarer Zähleroperationen mit AWS SDK for Java 2.x.

```
    public static void exampleUsage(DynamoDbClient dynamoDbClient, String
tableName) {
        // Example key
        Map<String, AttributeValue> key = new HashMap<>();
        key.put("ProductId", AttributeValue.builder().s("P12345").build());

        System.out.println("Demonstrating atomic counter operations in
DynamoDB");

        try {
            // Example 1: Increment a counter using ADD
```

```
            System.out.println("\nExample 1: Incrementing a counter using ADD");
            UpdateItemResponse addResponse =
incrementCounterWithAdd(dynamoDbClient, tableName, key, "ViewCount", 1);

            System.out.println("Updated counter: " + addResponse.attributes());

            // Example 2: Increment a counter using SET
            System.out.println("\nExample 2: Incrementing a counter using SET");
            UpdateItemResponse setResponse =
incrementCounterWithSet(dynamoDbClient, tableName, key, "LikeCount", 1);

            System.out.println("Updated counter: " + setResponse.attributes());

            // Example 3: Increment a counter safely
            System.out.println("\nExample 3: Incrementing a counter safely");
            UpdateItemResponse safeResponse =
incrementCounterSafely(dynamoDbClient, tableName, key, "ShareCount", 1);

            System.out.println("Updated counter: " + safeResponse.attributes());

            // Example 4: Decrement a counter safely
            System.out.println("\nExample 4: Decrementing a counter safely");
            try {
                UpdateItemResponse decrementResponse =
                    decrementCounterSafely(dynamoDbClient, tableName, key,
"InventoryCount", 1);

                System.out.println("Updated counter: " +
decrementResponse.attributes());
            } catch (DynamoDbException e) {
                if (e.getMessage().contains("ConditionalCheckFailed")) {
                    System.out.println("Cannot decrement counter below zero");
                } else {
                    throw e;
                }
            }

            // Example 5: Compare ADD vs SET
            System.out.println("\nExample 5: Comparing ADD vs SET");
            Map<String, Object> comparison = compareAddVsSet(dynamoDbClient,
tableName, key);

            if ((boolean) comparison.get("success")) {
```

```
                System.out.println("ADD duration: " +
comparison.get("addDuration") + " ns");
                System.out.println("SET duration: " +
comparison.get("setDuration") + " ns");
                System.out.println("ADD response: " +
comparison.get("addResponse"));
                System.out.println("SET response: " +
comparison.get("setResponse"));
            } else {
                System.out.println("Comparison failed: " +
comparison.get("error"));
            }

            // Explain atomic counter operations
            System.out.println("\nKey points about DynamoDB atomic counter
operations:");
            System.out.println("1. Both ADD and SET can be used for atomic
counters");
            System.out.println("2. ADD is more concise for simple increments");
            System.out.println("3. SET with an expression is more flexible for
complex operations");
            System.out.println("4. Use if_not_exists to handle the case where the
counter doesn't exist yet");
            System.out.println("5. Use condition expressions to prevent counters
from going below zero");
            System.out.println("6. Atomic operations are guaranteed to be
isolated from other writes");
            System.out.println("7. ADD can only be used with number and set data
types");

        } catch (DynamoDbException e) {
            System.err.println("Error: " + e.getMessage());
            e.printStackTrace();
        }
    }
```

- Einzelheiten zur API finden Sie UpdateItemin der AWS SDK for Java 2.x API-Referenz.

JavaScript

SDK für JavaScript (v3)

Demonstrieren Sie atomare Gegenoperationen mit AWS SDK für JavaScript.

```
const { DynamoDBClient } = require("@aws-sdk/client-dynamodb");
const {
  DynamoDBDocumentClient,
  UpdateCommand,
  GetCommand
} = require("@aws-sdk/lib-dynamodb");

/**
 * Increment a counter using the ADD operation.
 *
 * This function demonstrates using the ADD operation for atomic increments.
 * The ADD operation is atomic and is the recommended way to increment counters.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The key of the item to update
 * @param {string} counterName - The name of the counter attribute
 * @param {number} incrementValue - The value to increment by
 * @returns {Promise<Object>} - The response from DynamoDB
 */
async function incrementCounterWithAdd(
  config,
  tableName,
  key,
  counterName,
  incrementValue
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

  // Define the update parameters using ADD
  const params = {
    TableName: tableName,
    Key: key,
    UpdateExpression: `ADD ${counterName} :increment`,
    ExpressionAttributeValues: {
      ":increment": incrementValue
```

```
    },
    ReturnValues: "UPDATED_NEW"
  };

  // Perform the update operation
  const response = await docClient.send(new UpdateCommand(params));

  return response;
}

/**
 * Increment a counter using the SET operation with an expression.
 *
 * This function demonstrates using the SET operation with an expression for
 increments.
 * While this approach works, it's less idiomatic for simple increments than
 using ADD.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The key of the item to update
 * @param {string} counterName - The name of the counter attribute
 * @param {number} incrementValue - The value to increment by
 * @returns {Promise<Object>} - The response from DynamoDB
 */
async function incrementCounterWithSet(
  config,
  tableName,
  key,
  counterName,
  incrementValue
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

  // Define the update parameters using SET with an expression
  const params = {
    TableName: tableName,
    Key: key,
    UpdateExpression: `SET ${counterName} = ${counterName} + :increment`,
    ExpressionAttributeValues: {
      ":increment": incrementValue
    },
```

```
    ReturnValues: "UPDATED_NEW"
  };

  // Perform the update operation
  const response = await docClient.send(new UpdateCommand(params));

  return response;
}

/**
 * Increment a counter safely, handling the case where the counter might not
 exist.
 *
 * This function demonstrates using the if_not_exists function with SET to safely
 * increment a counter that might not exist yet.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The key of the item to update
 * @param {string} counterName - The name of the counter attribute
 * @param {number} incrementValue - The value to increment by
 * @param {number} defaultValue - The default value if the counter doesn't exist
 * @returns {Promise<Object>} - The response from DynamoDB
 */
async function incrementCounterSafely(
  config,
  tableName,
  key,
  counterName,
  incrementValue,
  defaultValue = 0
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

  // Define the update parameters using SET with if_not_exists
  const params = {
    TableName: tableName,
    Key: key,
    UpdateExpression: `SET ${counterName} =
 if_not_exists(${counterName}, :default) + :increment`,
    ExpressionAttributeValues: {
      ":increment": incrementValue,
```

```
      ":default": defaultValue
    },
    ReturnValues: "UPDATED_NEW"
  };

  // Perform the update operation
  const response = await docClient.send(new UpdateCommand(params));

  return response;
}

/**
 * Increment a counter with optimistic locking to prevent race conditions.
 *
 * This function demonstrates using a condition expression to implement
 optimistic
 * locking, which prevents race conditions when multiple processes try to update
 * the same counter.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The key of the item to update
 * @param {string} counterName - The name of the counter attribute
 * @param {number} incrementValue - The value to increment by
 * @param {number} expectedValue - The expected current value of the counter
 * @returns {Promise<Object>} - The response from DynamoDB
 */
async function incrementCounterWithLocking(
  config,
  tableName,
  key,
  counterName,
  incrementValue,
  expectedValue
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

  // Define the update parameters with a condition expression
  const params = {
    TableName: tableName,
    Key: key,
    UpdateExpression: `SET ${counterName} = ${counterName} + :increment`,
```

```
    ConditionExpression: `${counterName} = :expected`,
    ExpressionAttributeValues: {
      ":increment": incrementValue,
      ":expected": expectedValue
    },
    ReturnValues: "UPDATED_NEW"
  };

  try {
    // Perform the update operation
    const response = await docClient.send(new UpdateCommand(params));
    return {
      success: true,
      data: response
    };
  } catch (error) {
    // Check if the error is due to the condition check failing
    if (error.name === "ConditionalCheckFailedException") {
      return {
        success: false,
        error: "Optimistic locking failed: the counter value has changed"
      };
    }
    // Re-throw other errors
    throw error;
  }
}

/**
 * Get the current value of a counter.
 *
 * Helper function to retrieve the current value of a counter attribute.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The key of the item to get
 * @param {string} counterName - The name of the counter attribute
 * @returns {Promise<number|null>} - The current counter value or null if not
 found
 */
async function getCounterValue(
  config,
  tableName,
  key,
```

```
  counterName
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

  // Define the get parameters
  const params = {
    TableName: tableName,
    Key: key
  };

  // Perform the get operation
  const response = await docClient.send(new GetCommand(params));

  // Return the counter value if it exists, otherwise null
  return response.Item && counterName in response.Item
    ? response.Item[counterName]
    : null;
}
```

Beispiel für die Verwendung atomarer Zähleroperationen mit AWS SDK für JavaScript.

```
/**
 * Example of how to use the atomic counter operations.
 */
async function exampleUsage() {
  // Example parameters
  const config = { region: "us-west-2" };
  const tableName = "Products";
  const key = { ProductId: "P12345" };
  const counterName = "ViewCount";
  const incrementValue = 1;

  console.log("Demonstrating different approaches to increment counters in
 DynamoDB");

  try {
    // Example 1: Using ADD operation (recommended for simple increments)
    console.log("\nExample 1: Incrementing counter with ADD operation");
    const response1 = await incrementCounterWithAdd(
      config,
```

```
      tableName,
      key,
      counterName,
      incrementValue
    );

    console.log(`Counter incremented to: ${response1.Attributes[counterName]}`);

    // Example 2: Using SET operation with an expression
    console.log("\nExample 2: Incrementing counter with SET operation");
    const response2 = await incrementCounterWithSet(
      config,
      tableName,
      key,
      counterName,
      incrementValue
    );

    console.log(`Counter incremented to: ${response2.Attributes[counterName]}`);

    // Example 3: Safely incrementing a counter that might not exist
    console.log("\nExample 3: Safely incrementing counter that might not exist");
    const newKey = { ProductId: "P67890" };
    const response3 = await incrementCounterSafely(
      config,
      tableName,
      newKey,
      counterName,
      incrementValue,
      0
    );

    console.log(`Counter initialized and incremented to:
${response3.Attributes[counterName]}`);

    // Example 4: Incrementing with optimistic locking
    console.log("\nExample 4: Incrementing with optimistic locking");

    // First, get the current counter value
    const currentValue = await getCounterValue(config, tableName, key,
counterName);
    console.log(`Current counter value: ${currentValue}`);

    // Then, try to increment with optimistic locking
```

```
    const response4 = await incrementCounterWithLocking(
      config,
      tableName,
      key,
      counterName,
      incrementValue,
      currentValue
    );

    if (response4.success) {
      console.log(`Counter successfully incremented to:
 ${response4.data.Attributes[counterName]}`);
    } else {
      console.log(response4.error);
    }

    // Explain the differences between ADD and SET
    console.log("\nKey differences between ADD and SET for counter operations:");
    console.log("1. ADD is more concise and idiomatic for simple increments");
    console.log("2. SET with expressions is more flexible for complex
 operations");
    console.log("3. Both operations are atomic and safe for concurrent updates");
    console.log("4. SET with if_not_exists is required when the attribute might
 not exist");
    console.log("5. Optimistic locking can be added to either approach for
 additional safety");

  } catch (error) {
    console.error("Error:", error);
  }
}
```

- Einzelheiten zur API finden Sie UpdateItemin der AWS SDK für JavaScript API-Referenz.

Python

SDK für Python (Boto3)

Demonstrieren Sie atomare Zähleroperationen mit AWS SDK für Python (Boto3).

```
import boto3
from botocore.exceptions import ClientError
```

```
from typing import Any, Dict, Union


def increment_counter_with_add(
    table_name: str, key: Dict[str, Any], counter_name: str, increment_value: int
 = 1
) -> Dict[str, Any]:
    """
    Increment a counter attribute using the ADD operation.

    This function demonstrates the atomic ADD operation, which is ideal for
    incrementing counters without the risk of race conditions.

    Args:
        table_name (str): The name of the DynamoDB table.
        key (Dict[str, Any]): The primary key of the item to update.
        counter_name (str): The name of the counter attribute.
        increment_value (int, optional): The value to increment by. Defaults to
 1.

    Returns:
        Dict[str, Any]: The response from DynamoDB containing the updated
 attribute values.
    """
    # Initialize the DynamoDB resource
    dynamodb = boto3.resource("dynamodb")
    table = dynamodb.Table(table_name)

    # Use the ADD operation to atomically increment the counter
    response = table.update_item(
        Key=key,
        UpdateExpression="ADD #counter :increment",
        ExpressionAttributeNames={"#counter": counter_name},
        ExpressionAttributeValues={":increment": increment_value},
        ReturnValues="UPDATED_NEW",
    )

    return response


def increment_counter_with_set(
    table_name: str, key: Dict[str, Any], counter_name: str, increment_value: int
 = 1
) -> Dict[str, Any]:
```

```
    """
    Increment a counter attribute using the SET operation with an expression.

    This function demonstrates using SET with an expression to increment a
 counter.
    While this works, it's generally recommended to use ADD for simple
 increments.

    Args:
        table_name (str): The name of the DynamoDB table.
        key (Dict[str, Any]): The primary key of the item to update.
        counter_name (str): The name of the counter attribute.
        increment_value (int, optional): The value to increment by. Defaults to
 1.

    Returns:
        Dict[str, Any]: The response from DynamoDB containing the updated
 attribute values.
    """
    # Initialize the DynamoDB resource
    dynamodb = boto3.resource("dynamodb")
    table = dynamodb.Table(table_name)

    # Use the SET operation with an expression to increment the counter
    response = table.update_item(
        Key=key,
        UpdateExpression="SET #counter = #counter + :increment",
        ExpressionAttributeNames={"#counter": counter_name},
        ExpressionAttributeValues={":increment": increment_value},
        ReturnValues="UPDATED_NEW",
    )

    return response


def increment_counter_safely(
    table_name: str,
    key: Dict[str, Any],
    counter_name: str,
    increment_value: int = 1,
    initial_value: int = 0,
) -> Dict[str, Any]:
    """
```

```
    Increment a counter attribute safely, handling the case where it might not
 exist.

    This function demonstrates a best practice for incrementing counters by using
    the if_not_exists function to handle the case where the counter doesn't exist
 yet.

    Args:
        table_name (str): The name of the DynamoDB table.
        key (Dict[str, Any]): The primary key of the item to update.
        counter_name (str): The name of the counter attribute.
        increment_value (int, optional): The value to increment by. Defaults to
 1.
        initial_value (int, optional): The initial value if the counter doesn't
 exist. Defaults to 0.

    Returns:
        Dict[str, Any]: The response from DynamoDB containing the updated
 attribute values.
    """
    # Initialize the DynamoDB resource
    dynamodb = boto3.resource("dynamodb")
    table = dynamodb.Table(table_name)

    # Use SET with if_not_exists to safely increment the counter
    response = table.update_item(
        Key=key,
        UpdateExpression="SET #counter = if_not_exists(#counter, :initial)
 + :increment",
        ExpressionAttributeNames={"#counter": counter_name},
        ExpressionAttributeValues={":increment": increment_value, ":initial":
 initial_value},
        ReturnValues="UPDATED_NEW",
    )

    return response


def atomic_conditional_increment(
    table_name: str,
    key: Dict[str, Any],
    counter_name: str,
    condition_attribute: str,
    condition_value: Any,
```

```
    increment_value: int = 1,
) -> Union[Dict[str, Any], None]:
    """
    Atomically increment a counter only if a condition is met.

    This function demonstrates combining atomic counter operations with
    conditional expressions for more complex update scenarios.

    Args:
        table_name (str): The name of the DynamoDB table.
        key (Dict[str, Any]): The primary key of the item to update.
        counter_name (str): The name of the counter attribute.
        condition_attribute (str): The attribute to check in the condition.
        condition_value (Any): The value to compare against.
        increment_value (int, optional): The value to increment by. Defaults to
 1.

    Returns:
        Optional[Dict[str, Any]]: The response from DynamoDB if successful, None
 if condition failed.
    """
    # Initialize the DynamoDB resource
    dynamodb = boto3.resource("dynamodb")
    table = dynamodb.Table(table_name)

    try:
        # Use ADD with a condition expression
        response = table.update_item(
            Key=key,
            UpdateExpression="ADD #counter :increment",
            ConditionExpression="#condition = :value",
            ExpressionAttributeNames={"#counter": counter_name, "#condition":
 condition_attribute},
            ExpressionAttributeValues={":increment": increment_value, ":value":
 condition_value},
            ReturnValues="UPDATED_NEW",
        )
        return response
    except ClientError as e:
        if e.response["Error"]["Code"] == "ConditionalCheckFailedException":
            # Condition was not met
            return None
        else:
            # Other error occurred
```

```
            raise
```

Beispiel für die Verwendung atomarer Zähleroperationen mit AWS SDK für Python (Boto3).

```
def example_usage():
    """Example of how to use the atomic counter operations functions."""
    # Example parameters
    table_name = "GameScores"
    key = {"UserId": "user123", "GameId": "game456"}
    counter_name = "Score"

    print("Example 1: Incrementing a counter with ADD operation")
    try:
        response = increment_counter_with_add(
            table_name=table_name, key=key, counter_name=counter_name,
 increment_value=10
        )
        print(
            f"Counter incremented successfully. New value:
 {response.get('Attributes', {}).get(counter_name)}"
        )
    except Exception as e:
        print(f"Error incrementing counter with ADD: {e}")

    print("\nExample 2: Incrementing a counter with SET operation")
    try:
        response = increment_counter_with_set(
            table_name=table_name, key=key, counter_name=counter_name,
 increment_value=5
        )
        print(
            f"Counter incremented successfully. New value:
 {response.get('Attributes', {}).get(counter_name)}"
        )
    except Exception as e:
        print(f"Error incrementing counter with SET: {e}")

    print("\nExample 3: Safely incrementing a counter that might not exist")
    try:
        new_key = {"UserId": "newuser789", "GameId": "game456"}
```

```
        response = increment_counter_safely(
            table_name=table_name,
            key=new_key,
            counter_name=counter_name,
            increment_value=15,
            initial_value=100,
        )
        print(
            f"Counter safely incremented. New value: {response.get('Attributes',
{}).get(counter_name)}"
        )
    except Exception as e:
        print(f"Error safely incrementing counter: {e}")

    print("\nExample 4: Conditional counter increment")
    try:
        # Fix for mypy: Handle the case where response might be None
        result = atomic_conditional_increment(
            table_name=table_name,
            key=key,
            counter_name="Achievements",
            condition_attribute="Level",
            condition_value=5,
            increment_value=1,
        )

        if result is not None:
            print(
                f"Conditional increment succeeded. New value:
{result.get('Attributes', {}).get('Achievements')}"
            )
        else:
            print("Conditional increment failed because condition was not met.")
        if response:
            print(
                f"Conditional increment succeeded. New value:
{response.get('Attributes', {}).get('Achievements')}"
            )
        else:
            print("Conditional increment failed because condition was not met.")
    except Exception as e:
        print(f"Error with conditional increment: {e}")

    print("\nComparison of ADD vs SET for counter operations:")
```

```
    print("1. ADD is specifically designed for atomic numeric increments and set
operations")
    print("2. SET with an expression can be used for more complex calculations")
    print("3. Both operations are atomic, preventing race conditions")
    print("4. ADD is more concise for simple increments")
    print("5. SET with if_not_exists() is recommended when the attribute might
not exist")
    print("6. For counters, ADD is generally preferred for clarity and
simplicity")
```

- Einzelheiten zur API finden Sie UpdateItemin AWS SDK for Python (Boto3) API Reference.

Eine vollständige Liste der AWS SDK-Entwicklerhandbücher und Codebeispiele finden Sie unter. DynamoDB mit einem SDK verwenden AWS Dieses Thema enthält auch Informationen zu den ersten Schritten und Details zu früheren SDK-Versionen.

Verwenden Sie bedingte Operationen in DynamoDB mit einem SDK AWS

Die folgenden Codebeispiele zeigen, wie bedingte Operationen in DynamoDB verwendet werden.

- Implementieren von bedingten Schreibvorgängen, um das Überschreiben von Daten zu verhindern
- Verwenden von Bedingungsausdrücken, um Geschäftsregeln durchzusetzen
- Ordnungsgemäßer Umgang mit Fehlern bei bedingten Prüfungen

Java

SDK für Java 2.x

Demonstrieren Sie bedingte Operationen mit AWS SDK for Java 2.x.

```
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.AttributeValue;
import
 software.amazon.awssdk.services.dynamodb.model.ConditionalCheckFailedException;
import software.amazon.awssdk.services.dynamodb.model.DeleteItemRequest;
import software.amazon.awssdk.services.dynamodb.model.DeleteItemResponse;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.model.GetItemRequest;
```

```
import software.amazon.awssdk.services.dynamodb.model.GetItemResponse;
import software.amazon.awssdk.services.dynamodb.model.ReturnValue;
import software.amazon.awssdk.services.dynamodb.model.UpdateItemRequest;
import software.amazon.awssdk.services.dynamodb.model.UpdateItemResponse;

import java.util.HashMap;
import java.util.Map;

    /**
     * Performs a conditional update on an item.
     *
     * <p>This method demonstrates how to use a condition expression to update an
 item
     * only if a specific condition is met.
     *
     * @param dynamoDbClient The DynamoDB client
     * @param tableName The name of the DynamoDB table
     * @param key The key of the item to update
     * @param conditionAttribute The attribute to check in the condition
     * @param conditionValue The value to compare against
     * @param updateAttribute The attribute to update
     * @param updateValue The new value to set
     * @return Map containing the operation result and status
     */
    public static Map<String, Object> conditionalUpdate(
        DynamoDbClient dynamoDbClient,
        String tableName,
        Map<String, AttributeValue> key,
        String conditionAttribute,
        AttributeValue conditionValue,
        String updateAttribute,
        AttributeValue updateValue) {

        Map<String, Object> result = new HashMap<>();

        try {
            // Define the update parameters
            UpdateItemRequest request = UpdateItemRequest.builder()
                .tableName(tableName)
                .key(key)
                .updateExpression("SET #updateAttr = :updateVal")
                .conditionExpression("#condAttr = :condVal")
                .expressionAttributeNames(Map.of(
                    "#condAttr", conditionAttribute,
```

```
                    "#updateAttr", updateAttribute))
                .expressionAttributeValues(Map.of(
                    ":condVal", conditionValue,
                    ":updateVal", updateValue))
                .returnValues(ReturnValue.UPDATED_NEW)
                .build();

            // Perform the update operation
            UpdateItemResponse response = dynamoDbClient.updateItem(request);

            // Record success result
            result.put("success", true);
            result.put("message", "Condition was met and update was performed");
            result.put("attributes", response.attributes());

        } catch (ConditionalCheckFailedException e) {
            // Record failure due to condition not being met
            result.put("success", false);
            result.put("message", "Condition was not met, update was not
performed");
            result.put("error", "ConditionalCheckFailedException");

        } catch (DynamoDbException e) {
            // Record failure due to other errors
            result.put("success", false);
            result.put("message", "Error occurred: " + e.getMessage());
            result.put("error", e.getClass().getSimpleName());
        }

        return result;
    }

    /**
     * Performs a conditional delete on an item.
     *
     * <p>This method demonstrates how to use a condition expression to delete an
item
     * only if a specific condition is met.
     *
     * @param dynamoDbClient The DynamoDB client
     * @param tableName The name of the DynamoDB table
     * @param key The key of the item to delete
     * @param conditionAttribute The attribute to check in the condition
     * @param conditionValue The value to compare against
```

```
     * @return Map containing the operation result and status
     */
    public static Map<String, Object> conditionalDelete(
        DynamoDbClient dynamoDbClient,
        String tableName,
        Map<String, AttributeValue> key,
        String conditionAttribute,
        AttributeValue conditionValue) {

        Map<String, Object> result = new HashMap<>();

        try {
            // Define the delete parameters
            DeleteItemRequest request = DeleteItemRequest.builder()
                .tableName(tableName)
                .key(key)
                .conditionExpression("#condAttr = :condVal")
                .expressionAttributeNames(Map.of("#condAttr",
conditionAttribute))
                .expressionAttributeValues(Map.of(":condVal", conditionValue))
                .returnValues(ReturnValue.ALL_OLD)
                .build();

            // Perform the delete operation
            DeleteItemResponse response = dynamoDbClient.deleteItem(request);

            // Record success result
            result.put("success", true);
            result.put("message", "Condition was met and delete was performed");
            result.put("attributes", response.attributes());

        } catch (ConditionalCheckFailedException e) {
            // Record failure due to condition not being met
            result.put("success", false);
            result.put("message", "Condition was not met, delete was not
performed");
            result.put("error", "ConditionalCheckFailedException");

        } catch (DynamoDbException e) {
            // Record failure due to other errors
            result.put("success", false);
            result.put("message", "Error occurred: " + e.getMessage());
            result.put("error", e.getClass().getSimpleName());
        }
```

```
        return result;
    }

    /**
     * Demonstrates optimistic locking using a version attribute.
     *
     * <p>This method shows how to implement optimistic locking by using a
 version
     * attribute that is incremented with each update.
     *
     * @param dynamoDbClient The DynamoDB client
     * @param tableName The name of the DynamoDB table
     * @param key The key of the item to update
     * @param versionAttribute The name of the version attribute
     * @return Map containing the operation result
     */
    public static Map<String, Object> optimisticLockingExample(
        DynamoDbClient dynamoDbClient, String tableName, Map<String,
 AttributeValue> key, String versionAttribute) {

        Map<String, Object> result = new HashMap<>();

        try {
            // Get the current version of the item
            GetItemRequest getRequest = GetItemRequest.builder()
                .tableName(tableName)
                .key(key)
                .projectionExpression(versionAttribute)
                .build();

            GetItemResponse getResponse = dynamoDbClient.getItem(getRequest);

            // Check if the item exists
            if (getResponse.item() == null || !
getResponse.item().containsKey(versionAttribute)) {
                // Item doesn't exist or doesn't have a version attribute
                // Initialize with version 1
                UpdateItemRequest initRequest = UpdateItemRequest.builder()
                    .tableName(tableName)
                    .key(key)
                    .updateExpression("SET #verAttr = :newVer, #dataAttr
 = :data")
```

```
                    .expressionAttributeNames(Map.of("#verAttr",
versionAttribute, "#dataAttr", "Data"))
                    .expressionAttributeValues(Map.of(
                        ":newVer", AttributeValue.builder().n("1").build(),
                        ":data", AttributeValue.builder().s("Initial
data").build()))
                    .returnValues(ReturnValue.UPDATED_NEW)
                    .build();

                UpdateItemResponse initResponse =
dynamoDbClient.updateItem(initRequest);

                result.put("operation", "initialize");
                result.put("success", true);
                result.put("attributes", initResponse.attributes());

                return result;
            }

            // Get the current version number
            int currentVersion =
                Integer.parseInt(getResponse.item().get(versionAttribute).n());
            int newVersion = currentVersion + 1;

            // Update the item with a condition on the version
            UpdateItemRequest updateRequest = UpdateItemRequest.builder()
                .tableName(tableName)
                .key(key)
                .updateExpression("SET #verAttr = :newVer, #dataAttr = :newData")
                .conditionExpression("#verAttr = :curVer")
                .expressionAttributeNames(Map.of("#verAttr", versionAttribute,
"#dataAttr", "Data"))
                .expressionAttributeValues(Map.of(
                    ":curVer",
                        AttributeValue.builder()
                            .n(String.valueOf(currentVersion))
                            .build(),
                    ":newVer",

AttributeValue.builder().n(String.valueOf(newVersion)).build(),
                    ":newData",
                        AttributeValue.builder()
                            .s("Updated data at version " + newVersion)
                            .build()))
```

```
                .returnValues(ReturnValue.UPDATED_NEW)
                .build();

            UpdateItemResponse updateResponse =
dynamoDbClient.updateItem(updateRequest);

            // Record success result
            result.put("operation", "update");
            result.put("success", true);
            result.put("oldVersion", currentVersion);
            result.put("newVersion", newVersion);
            result.put("attributes", updateResponse.attributes());

        } catch (ConditionalCheckFailedException e) {
            // Record failure due to version mismatch
            result.put("operation", "update");
            result.put("success", false);
            result.put("message", "Version mismatch, another process may have
updated the item");
            result.put("error", "ConditionalCheckFailedException");

        } catch (DynamoDbException e) {
            // Record failure due to other errors
            result.put("operation", "update");
            result.put("success", false);
            result.put("message", "Error occurred: " + e.getMessage());
            result.put("error", e.getClass().getSimpleName());
        }

        return result;
    }

    /**
     * Performs a conditional update with multiple conditions.
     *
     * <p>This method demonstrates how to use multiple conditions in a condition
expression.
     *
     * @param dynamoDbClient The DynamoDB client
     * @param tableName The name of the DynamoDB table
     * @param key The key of the item to update
     * @param conditions Map of attribute names to values for conditions
     * @param updateAttribute The attribute to update
     * @param updateValue The new value to set
```

```
     * @return Map containing the operation result and status
     */
    public static Map<String, Object> conditionalUpdateWithMultipleConditions(
        DynamoDbClient dynamoDbClient,
        String tableName,
        Map<String, AttributeValue> key,
        Map<String, AttributeValue> conditions,
        String updateAttribute,
        AttributeValue updateValue) {

        Map<String, Object> result = new HashMap<>();

        try {
            // Build the condition expression and attribute names/values
            StringBuilder conditionExpression = new StringBuilder();
            Map<String, String> expressionAttributeNames = new HashMap<>();
            Map<String, AttributeValue> expressionAttributeValues = new
HashMap<>();

            // Add update attribute
            expressionAttributeNames.put("#updateAttr", updateAttribute);
            expressionAttributeValues.put(":updateVal", updateValue);

            // Add conditions
            int i = 0;
            for (Map.Entry<String, AttributeValue> condition :
conditions.entrySet()) {
                String attrName = condition.getKey();
                AttributeValue attrValue = condition.getValue();

                String nameKey = "#cond" + i;
                String valueKey = ":val" + i;

                expressionAttributeNames.put(nameKey, attrName);
                expressionAttributeValues.put(valueKey, attrValue);

                // Add AND between conditions (except for the first one)
                if (i > 0) {
                    conditionExpression.append(" AND ");
                }

                conditionExpression.append(nameKey).append(" =
").append(valueKey);
                i++;
```

```
            }

            // Define the update parameters
            UpdateItemRequest request = UpdateItemRequest.builder()
                .tableName(tableName)
                .key(key)
                .updateExpression("SET #updateAttr = :updateVal")
                .conditionExpression(conditionExpression.toString())
                .expressionAttributeNames(expressionAttributeNames)
                .expressionAttributeValues(expressionAttributeValues)
                .returnValues(ReturnValue.UPDATED_NEW)
                .build();

            // Perform the update operation
            UpdateItemResponse response = dynamoDbClient.updateItem(request);

            // Record success result
            result.put("success", true);
            result.put("message", "All conditions were met and update was
performed");
            result.put("attributes", response.attributes());

        } catch (ConditionalCheckFailedException e) {
            // Record failure due to condition not being met
            result.put("success", false);
            result.put("message", "One or more conditions were not met, update
was not performed");
            result.put("error", "ConditionalCheckFailedException");

        } catch (DynamoDbException e) {
            // Record failure due to other errors
            result.put("success", false);
            result.put("message", "Error occurred: " + e.getMessage());
            result.put("error", e.getClass().getSimpleName());
        }

        return result;
    }
```

Beispiel für die Verwendung von bedingten Operationen mit AWS SDK for Java 2.x.

```
   public static void exampleUsage(DynamoDbClient dynamoDbClient, String
tableName) {
       // Example key
       Map<String, AttributeValue> key = new HashMap<>();
       key.put("ProductId", AttributeValue.builder().s("P12345").build());

       System.out.println("Demonstrating conditional operations in DynamoDB");

       try {
           // Example 1: Conditional update
           System.out.println("\nExample 1: Conditional update");
           Map<String, Object> updateResult = conditionalUpdate(
               dynamoDbClient,
               tableName,
               key,
               "InStock",
               AttributeValue.builder().bool(true).build(),
               "Status",
               AttributeValue.builder().s("Available").build());

           System.out.println("Update result: " + updateResult.get("message"));
           if ((boolean) updateResult.get("success")) {
               System.out.println("Updated attributes: " +
updateResult.get("attributes"));
           }

           // Example 2: Conditional delete
           System.out.println("\nExample 2: Conditional delete");
           Map<String, Object> deleteResult = conditionalDelete(
               dynamoDbClient,
               tableName,
               key,
               "Status",
               AttributeValue.builder().s("Discontinued").build());

           System.out.println("Delete result: " + deleteResult.get("message"));
           if ((boolean) deleteResult.get("success")) {
               System.out.println("Deleted item: " +
deleteResult.get("attributes"));
           }

           // Example 3: Optimistic locking
           System.out.println("\nExample 3: Optimistic locking");
```

```
            Map<String, Object> lockingResult =
optimisticLockingExample(dynamoDbClient, tableName, key, "Version");

            System.out.println("Optimistic locking result:");
            System.out.println("  Operation: " + lockingResult.get("operation"));
            System.out.println("  Success: " + lockingResult.get("success"));
            if (lockingResult.get("operation").equals("update") && (boolean)
lockingResult.get("success")) {
                System.out.println("  Old version: " +
lockingResult.get("oldVersion"));
                System.out.println("  New version: " +
lockingResult.get("newVersion"));
            }
            System.out.println("  Attributes: " +
lockingResult.get("attributes"));

            // Example 4: Multiple conditions
            System.out.println("\nExample 4: Multiple conditions");
            Map<String, AttributeValue> conditions = new HashMap<>();
            conditions.put("Price",
AttributeValue.builder().n("199.99").build());
            conditions.put("Category",
AttributeValue.builder().s("Electronics").build());

            Map<String, Object> multiConditionResult =
conditionalUpdateWithMultipleConditions(
                dynamoDbClient,
                tableName,
                key,
                conditions,
                "OnSale",
                AttributeValue.builder().bool(true).build());

            System.out.println("Multiple conditions result: " +
multiConditionResult.get("message"));
            if ((boolean) multiConditionResult.get("success")) {
                System.out.println("Updated attributes: " +
multiConditionResult.get("attributes"));
            }

            // Explain conditional operations
            System.out.println("\nKey points about DynamoDB conditional
operations:");
```

```
            System.out.println("1. Conditional operations only succeed if the
 condition is met");
            System.out.println("2. ConditionalCheckFailedException is thrown when
 the condition fails");
            System.out.println("3. No changes are made to the item if the
 condition fails");
            System.out.println("4. Conditions can be used with update, delete,
 and put operations");
            System.out.println("5. Multiple conditions can be combined with AND
 and OR");
            System.out.println("6. Optimistic locking can be implemented using a
 version attribute");
            System.out.println(
                "7. Conditional operations consume the same amount of write
 capacity whether they succeed or fail");

        } catch (DynamoDbException e) {
            System.err.println("Error: " + e.getMessage());
            e.printStackTrace();
        }
    }
```

- Weitere API-Informationen finden Sie in den folgenden Themen der AWS SDK for Java 2.x - API-Referenz.
  - DeleteItem
  - PutItem
  - UpdateItem

JavaScript

SDK für JavaScript (v3)

Demonstrieren Sie bedingte Operationen mit AWS SDK für JavaScript.

```
const { DynamoDBClient } = require("@aws-sdk/client-dynamodb");
const {
  DynamoDBDocumentClient,
  UpdateCommand,
  DeleteCommand,
  GetCommand,
```

```
  PutCommand
} = require("@aws-sdk/lib-dynamodb");

/**
 * Perform a conditional update operation.
 *
 * This function demonstrates how to update an item only if a condition is met.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The key of the item to update
 * @param {string} conditionAttribute - The attribute to check in the condition
 * @param {any} conditionValue - The value to compare against
 * @param {string} updateAttribute - The attribute to update
 * @param {any} updateValue - The new value to set
 * @returns {Promise<Object>} - Result of the operation
 */
async function conditionalUpdate(
  config,
  tableName,
  key,
  conditionAttribute,
  conditionValue,
  updateAttribute,
  updateValue
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

  // Define the update parameters with a condition expression
  const params = {
    TableName: tableName,
    Key: key,
    UpdateExpression: `SET ${updateAttribute} = :value`,
    ConditionExpression: `${conditionAttribute} = :condition`,
    ExpressionAttributeValues: {
      ":value": updateValue,
      ":condition": conditionValue
    },
    ReturnValues: "UPDATED_NEW"
  };

  try {
```

```
    // Perform the update operation
    const response = await docClient.send(new UpdateCommand(params));

    return {
      success: true,
      message: "Condition was met and update was performed",
      updatedAttributes: response.Attributes
    };
  } catch (error) {
    // Check if the error is due to the condition check failing
    if (error.name === "ConditionalCheckFailedException") {
      return {
        success: false,
        message: "Condition was not met, update was not performed",
        error: "ConditionalCheckFailedException"
      };
    }

    // Re-throw other errors
    throw error;
  }
}

/**
 * Perform a conditional delete operation.
 *
 * This function demonstrates how to delete an item only if a condition is met.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The key of the item to delete
 * @param {string} conditionAttribute - The attribute to check in the condition
 * @param {any} conditionValue - The value to compare against
 * @returns {Promise<Object>} - Result of the operation
 */
async function conditionalDelete(
  config,
  tableName,
  key,
  conditionAttribute,
  conditionValue
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
```

```
  const docClient = DynamoDBDocumentClient.from(client);

  // Define the delete parameters with a condition expression
  const params = {
    TableName: tableName,
    Key: key,
    ConditionExpression: `${conditionAttribute} = :condition`,
    ExpressionAttributeValues: {
      ":condition": conditionValue
    },
    ReturnValues: "ALL_OLD"
  };

  try {
    // Perform the delete operation
    const response = await docClient.send(new DeleteCommand(params));

    return {
      success: true,
      message: "Condition was met and item was deleted",
      deletedItem: response.Attributes
    };
  } catch (error) {
    // Check if the error is due to the condition check failing
    if (error.name === "ConditionalCheckFailedException") {
      return {
        success: false,
        message: "Condition was not met, item was not deleted",
        error: "ConditionalCheckFailedException"
      };
    }

    // Re-throw other errors
    throw error;
  }
}

/**
 * Implement optimistic locking with a version number.
 *
 * This function demonstrates how to use a version number for optimistic locking
 * to prevent race conditions when multiple processes update the same item.
 *
 * @param {Object} config - AWS configuration object
```

```
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The key of the item to update
 * @param {Object} updates - The attributes to update
 * @param {number} expectedVersion - The expected current version number
 * @returns {Promise<Object>} - Result of the operation
 */
async function updateWithOptimisticLocking(
  config,
  tableName,
  key,
  updates,
  expectedVersion
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

  // Build the update expression
  const updateExpressions = [];
  const expressionAttributeValues = {
    ":expectedVersion": expectedVersion,
    ":newVersion": expectedVersion + 1
  };

  // Add each update to the expression
  Object.entries(updates).forEach(([attribute, value], index) => {
    updateExpressions.push(`${attribute} = :val${index}`);
    expressionAttributeValues[`:val${index}`] = value;
  });

  // Add the version update
  updateExpressions.push("version = :newVersion");

  // Define the update parameters with a condition expression
  const params = {
    TableName: tableName,
    Key: key,
    UpdateExpression: `SET ${updateExpressions.join(", ")}`,
    ConditionExpression: "version = :expectedVersion",
    ExpressionAttributeValues: expressionAttributeValues,
    ReturnValues: "UPDATED_NEW"
  };

  try {
```

```
    // Perform the update operation
    const response = await docClient.send(new UpdateCommand(params));

    return {
      success: true,
      message: "Update succeeded with optimistic locking",
      newVersion: expectedVersion + 1,
      updatedAttributes: response.Attributes
    };
  } catch (error) {
    // Check if the error is due to the condition check failing
    if (error.name === "ConditionalCheckFailedException") {
      return {
        success: false,
        message: "Optimistic locking failed: the item was modified by another
 process",
        error: "ConditionalCheckFailedException"
      };
    }

    // Re-throw other errors
    throw error;
  }
}

/**
 * Implement a conditional write that creates an item only if it doesn't exist.
 *
 * This function demonstrates how to use attribute_not_exists to create an item
 * only if it doesn't already exist (similar to an "INSERT IF NOT EXISTS"
 operation).
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} item - The item to create
 * @returns {Promise<Object>} - Result of the operation
 */
async function createIfNotExists(
  config,
  tableName,
  item
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
```

```
const docClient = DynamoDBDocumentClient.from(client);

// Extract the primary key attributes
const keyAttributes = Object.keys(item).filter(attr =>
  attr === "id" || attr === "ID" || attr === "Id" ||
  attr.endsWith("Id") || attr.endsWith("ID") ||
  attr.endsWith("Key")
);

if (keyAttributes.length === 0) {
  throw new Error("Could not determine primary key attributes");
}

// Create a condition expression that checks if the item doesn't exist
const conditionExpression = `attribute_not_exists(${keyAttributes[0]})`;

// Define the put parameters with a condition expression
const params = {
  TableName: tableName,
  Item: item,
  ConditionExpression: conditionExpression
};

try {
  // Perform the put operation
  await docClient.send(new PutCommand(params));

  return {
    success: true,
    message: "Item was created because it didn't exist",
    item
  };
} catch (error) {
  // Check if the error is due to the condition check failing
  if (error.name === "ConditionalCheckFailedException") {
    return {
      success: false,
      message: "Item already exists, creation was skipped",
      error: "ConditionalCheckFailedException"
    };
  }

  // Re-throw other errors
  throw error;
```

```
  }
}

/**
 * Get the current value of an item.
 *
 * Helper function to retrieve the current value of an item.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The key of the item to get
 * @returns {Promise<Object|null>} - The item or null if not found
 */
async function getItem(
  config,
  tableName,
  key
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

  // Define the get parameters
  const params = {
    TableName: tableName,
    Key: key
  };

  // Perform the get operation
  const response = await docClient.send(new GetCommand(params));

  // Return the item if it exists, otherwise null
  return response.Item || null;
}
```

Beispiel für die Verwendung von bedingten Operationen mit AWS SDK für JavaScript.

```
/**
 * Example of how to use conditional operations.
 */
async function exampleUsage() {
  // Example parameters
```

```
const config = { region: "us-west-2" };
const tableName = "Products";
const key = { ProductId: "P12345" };

console.log("Demonstrating conditional operations in DynamoDB");

try {
  // Example 1: Conditional update based on attribute value
  console.log("\nExample 1: Conditional update based on attribute value");
  const updateResult = await conditionalUpdate(
    config,
    tableName,
    key,
    "Category",
    "Electronics",
    "Price",
    299.99
  );

  console.log(`Result: ${updateResult.message}`);
  if (updateResult.success) {
    console.log("Updated attributes:", updateResult.updatedAttributes);
  }

  // Example 2: Conditional delete based on attribute value
  console.log("\nExample 2: Conditional delete based on attribute value");
  const deleteResult = await conditionalDelete(
    config,
    tableName,
    key,
    "InStock",
    false
  );

  console.log(`Result: ${deleteResult.message}`);
  if (deleteResult.success) {
    console.log("Deleted item:", deleteResult.deletedItem);
  }

  // Example 3: Optimistic locking with version number
  console.log("\nExample 3: Optimistic locking with version number");

  // First, get the current item to check its version
```

```
    const currentItem = await getItem(config, tableName, { ProductId:
"P67890" });
    const currentVersion = currentItem ? (currentItem.version || 0) : 0;

    console.log(`Current version: ${currentVersion}`);

    // Then, update with optimistic locking
    const lockingResult = await updateWithOptimisticLocking(
      config,
      tableName,
      { ProductId: "P67890" },
      {
        Name: "Updated Product Name",
        Description: "This is an updated description"
      },
      currentVersion
    );

    console.log(`Result: ${lockingResult.message}`);
    if (lockingResult.success) {
      console.log(`New version: ${lockingResult.newVersion}`);
      console.log("Updated attributes:", lockingResult.updatedAttributes);
    }

    // Example 4: Create item only if it doesn't exist
    console.log("\nExample 4: Create item only if it doesn't exist");
    const createResult = await createIfNotExists(
      config,
      tableName,
      {
        ProductId: "P99999",
        Name: "New Product",
        Category: "Accessories",
        Price: 19.99,
        InStock: true
      }
    );

    console.log(`Result: ${createResult.message}`);
    if (createResult.success) {
      console.log("Created item:", createResult.item);
    }

    // Explain conditional operations
```

```
    console.log("\nKey points about conditional operations:");
    console.log("1. Conditional operations only succeed if the condition is
 met");
    console.log("2. ConditionalCheckFailedException indicates the condition
 wasn't met");
    console.log("3. Optimistic locking prevents race conditions in concurrent
 updates");
    console.log("4. attribute_exists and attribute_not_exists are useful for
 checking if attributes are present");
    console.log("5. Conditional operations are atomic - they either succeed
 completely or fail completely");
    console.log("6. You can use any valid comparison operators and functions in
 condition expressions");
    console.log("7. Conditional operations don't consume write capacity if the
 condition fails");

  } catch (error) {
    console.error("Error:", error);
  }
}
```

- Weitere API-Informationen finden Sie in den folgenden Themen der AWS SDK für JavaScript -API-Referenz.
  - DeleteItem
  - PutItem
  - UpdateItem

Python

SDK für Python (Boto3)

Demonstrieren Sie bedingte Operationen mit AWS SDK für Python (Boto3).

```
import boto3
from botocore.exceptions import ClientError
from typing import Any, Dict, Optional, Tuple, Union


def conditional_update(
    table_name: str,
```

```
    key: Dict[str, Any],
    condition_attribute: str,
    condition_value: Any,
    update_attribute: str,
    update_value: Any,
) -> Tuple[bool, Optional[Dict[str, Any]]]:
    """
    Update an item only if a condition is met.

    This function demonstrates how to perform a conditional update operation
    and determine if the condition was met.

    Args:
        table_name (str): The name of the DynamoDB table.
        key (Dict[str, Any]): The primary key of the item to update.
        condition_attribute (str): The attribute to check in the condition.
        condition_value (Any): The value to compare against.
        update_attribute (str): The attribute to update.
        update_value (Any): The new value to set.

    Returns:
        Tuple[bool, Optional[Dict[str, Any]]]: A tuple containing:
            - A boolean indicating if the update succeeded
            - The response from DynamoDB if successful, None otherwise
    """
    # Initialize the DynamoDB resource
    dynamodb = boto3.resource("dynamodb")
    table = dynamodb.Table(table_name)

    try:
        # Perform the conditional update
        response = table.update_item(
            Key=key,
            UpdateExpression="SET #update_attr = :update_val",
            ConditionExpression="#cond_attr = :cond_val",
            ExpressionAttributeNames={
                "#update_attr": update_attribute,
                "#cond_attr": condition_attribute,
            },
            ExpressionAttributeValues={":update_val": update_value, ":cond_val":
 condition_value},
            ReturnValues="UPDATED_NEW",
        )
        # Update succeeded, condition was met
```

```
        return True, response
    except ClientError as e:
        if e.response["Error"]["Code"] == "ConditionalCheckFailedException":
            # Condition was not met
            return False, None
        else:
            # Other error occurred
            raise


def conditional_delete(
    table_name: str, key: Dict[str, Any], condition_attribute: str,
 condition_value: Any
) -> bool:
    """
    Delete an item only if a condition is met.

    This function demonstrates how to perform a conditional delete operation
    and determine if the condition was met.

    Args:
        table_name (str): The name of the DynamoDB table.
        key (Dict[str, Any]): The primary key of the item to delete.
        condition_attribute (str): The attribute to check in the condition.
        condition_value (Any): The value to compare against.

    Returns:
        bool: True if the delete succeeded (condition was met), False otherwise.
    """
    # Initialize the DynamoDB resource
    dynamodb = boto3.resource("dynamodb")
    table = dynamodb.Table(table_name)

    try:
        # Perform the conditional delete
        table.delete_item(
            Key=key,
            ConditionExpression="#attr = :val",
            ExpressionAttributeNames={"#attr": condition_attribute},
            ExpressionAttributeValues={":val": condition_value},
        )
        # Delete succeeded, condition was met
        return True
    except ClientError as e:
```

```
        if e.response["Error"]["Code"] == "ConditionalCheckFailedException":
            # Condition was not met
            return False
        else:
            # Other error occurred
            raise


def optimistic_locking_update(
    table_name: str,
    key: Dict[str, Any],
    version_attribute: str,
    update_attribute: str,
    update_value: Any,
) -> Tuple[bool, Optional[Dict[str, Any]]]:
    """
    Update an item using optimistic locking with a version attribute.

    This function demonstrates how to implement optimistic locking using
    a version attribute that is incremented with each update.

    Args:
        table_name (str): The name of the DynamoDB table.
        key (Dict[str, Any]): The primary key of the item to update.
        version_attribute (str): The name of the version attribute.
        update_attribute (str): The attribute to update.
        update_value (Any): The new value to set.

    Returns:
        Tuple[bool, Optional[Dict[str, Any]]]: A tuple containing:
            - A boolean indicating if the update succeeded
            - The response from DynamoDB if successful, None otherwise
    """
    # Initialize the DynamoDB resource
    dynamodb = boto3.resource("dynamodb")
    table = dynamodb.Table(table_name)

    # First, get the current version
    try:
        response = table.get_item(
            Key=key,
            ProjectionExpression=f"#{version_attribute}",
            ExpressionAttributeNames={f"#{version_attribute}":
 version_attribute},
```

```
        )

        item = response.get("Item", {})
        current_version = item.get(version_attribute, 0)

        # Now, try to update with a condition on the version
        try:
            update_response = table.update_item(
                Key=key,
                UpdateExpression=f"SET #{update_attribute} = :update_val,
 #{version_attribute} = :new_version",
                ConditionExpression=f"#{version_attribute} = :current_version",
                ExpressionAttributeNames={
                    f"#{update_attribute}": update_attribute,
                    f"#{version_attribute}": version_attribute,
                },
                ExpressionAttributeValues={
                    ":update_val": update_value,
                    ":current_version": current_version,
                    ":new_version": current_version + 1,
                },
                ReturnValues="UPDATED_NEW",
            )
            # Update succeeded
            return True, update_response
        except ClientError as e:
            if e.response["Error"]["Code"] == "ConditionalCheckFailedException":
                # Version has changed, optimistic locking failed
                return False, None
            else:
                # Other error occurred
                raise
    except ClientError:
        # Error getting the item
        raise


def conditional_check_and_update(
    table_name: str,
    key: Dict[str, Any],
    check_attribute: str,
    check_value: Any,
    update_attribute: str,
    update_value: Any,
```

```
    create_if_not_exists: bool = False,
) -> Union[Dict[str, Any], None]:
    """
    Check if an attribute has a specific value and update another attribute if it
 does.

    This function demonstrates a more complex conditional update that can also
    create the item if it doesn't exist.

    Args:
        table_name (str): The name of the DynamoDB table.
        key (Dict[str, Any]): The primary key of the item to update.
        check_attribute (str): The attribute to check in the condition.
        check_value (Any): The value to compare against.
        update_attribute (str): The attribute to update.
        update_value (Any): The new value to set.
        create_if_not_exists (bool, optional): Whether to create the item if it
 doesn't exist.

    Returns:
        Union[Dict[str, Any], None]: The response from DynamoDB if successful,
 None otherwise.
    """
    # Initialize the DynamoDB resource
    dynamodb = boto3.resource("dynamodb")
    table = dynamodb.Table(table_name)

    try:
        if create_if_not_exists:
            # Use attribute_not_exists to create the item if it doesn't exist
            condition_expression = "attribute_not_exists(#pk) OR #check_attr
 = :check_val"
            update_expression = "SET #update_attr = :update_val, #check_attr =
 if_not_exists(#check_attr, :check_val)"

            # Get the partition key name from the key dictionary
            pk_name = next(iter(key))

            expression_attribute_names = {
                "#pk": pk_name,
                "#check_attr": check_attribute,
                "#update_attr": update_attribute,
            }
        else:
```

```
            # Only update if the check attribute has the expected value
            condition_expression = "#check_attr = :check_val"
            update_expression = "SET #update_attr = :update_val"

            expression_attribute_names = {
                "#check_attr": check_attribute,
                "#update_attr": update_attribute,
            }

        # Perform the conditional update
        response = table.update_item(
            Key=key,
            UpdateExpression=update_expression,
            ConditionExpression=condition_expression,
            ExpressionAttributeNames=expression_attribute_names,
            ExpressionAttributeValues={":check_val": check_value, ":update_val":
 update_value},
            ReturnValues="UPDATED_NEW",
        )
        return response
    except ClientError as e:
        if e.response["Error"]["Code"] == "ConditionalCheckFailedException":
            # Condition was not met
            return None
        else:
            # Other error occurred
            raise
```

Beispiel für die Verwendung von bedingten Operationen mit AWS SDK für Python (Boto3).

```
def example_usage():
    """Example of how to use the conditional operations functions."""
    # Example parameters
    table_name = "Products"
    key = {"ProductId": "prod123"}

    print("Example 1: Conditional Update")
    try:
        # Update the price only if the current stock is greater than 10
        success, response = conditional_update(
```

```
            table_name=table_name,
            key=key,
            condition_attribute="Stock",
            condition_value=10,
            update_attribute="Price",
            update_value=99.99,
        )

        if success:
            # Fix for mypy: Handle the case where response might be None
            attributes = {} if response is None else response.get("Attributes",
{})
            print(f"Update succeeded! New values: {attributes}")
        else:
            print("Update failed because the condition was not met.")
    except Exception as e:
        print(f"Error during conditional update: {e}")

    print("\nExample 2: Conditional Delete")
    try:
        # Delete the product only if it's discontinued
        success = conditional_delete(
            table_name=table_name,
            key=key,
            condition_attribute="Status",
            condition_value="Discontinued",
        )

        if success:
            print("Delete succeeded! The item was deleted.")
        else:
            print("Delete failed because the condition was not met.")
    except Exception as e:
        print(f"Error during conditional delete: {e}")

    print("\nExample 3: Optimistic Locking")
    try:
        # Update with optimistic locking using a version attribute
        success, response = optimistic_locking_update(
            table_name=table_name,
            key=key,
            version_attribute="Version",
            update_attribute="Description",
            update_value="Updated product description",
```

```
        )

        if success:
            # Fix for mypy: Handle the case where response might be None
            attributes = {} if response is None else response.get("Attributes",
{})
            print(f"Optimistic locking update succeeded! New values:
{attributes}")
        else:
            print("Optimistic locking update failed because the version has
changed.")
    except Exception as e:
        print(f"Error during optimistic locking update: {e}")

    print("\nExample 4: Conditional Check and Update")
    try:
        # Update the featured status if the product is in stock
        response = conditional_check_and_update(
            table_name=table_name,
            key=key,
            check_attribute="InStock",
            check_value=True,
            update_attribute="Featured",
            update_value=True,
            create_if_not_exists=True,
        )

        if response:
            print(
                f"Conditional check and update succeeded! New values:
{response.get('Attributes', {})}"
            )
        else:
            print("Conditional check and update failed because the condition was
not met.")
    except Exception as e:
        print(f"Error during conditional check and update: {e}")

    print("\nUnderstanding Conditional Operations in DynamoDB:")
    print("1. Conditional operations help maintain data integrity")
    print("2. They prevent race conditions in concurrent environments")
    print("3. Failed conditions result in ConditionalCheckFailedException")
    print("4. No DynamoDB capacity is consumed when conditions fail")
    print("5. Optimistic locking is a common pattern using version attributes")
```

```
    print("6. Conditions can be combined with logical operators (AND, OR, NOT)")
    print("7. Conditions can use comparison operators (=, <>, <, <=, >, >=)")
    print(
        "8. attribute_exists() and attribute_not_exists() are useful for checking
attribute presence"
    )
```

- Weitere API-Informationen finden Sie in den folgenden Themen der API-Referenz zum AWS SDK für Python (Boto3).
  - DeleteItem
  - PutItem
  - UpdateItem

Eine vollständige Liste der AWS SDK-Entwicklerhandbücher und Codebeispiele finden Sie unterDynamoDB mit einem SDK verwenden AWS. Dieses Thema enthält auch Informationen zu den ersten Schritten und Details zu früheren SDK-Versionen.

Verwenden Sie Ausdrucksattributnamen in DynamoDB mit einem SDK AWS

Die folgenden Codebeispiele zeigen, wie Ausdrucksattributnamen in DynamoDB verwendet werden.

- Arbeiten mit reservierten Wörtern in DynamoDB-Ausdrücken
- Verwenden von Platzhaltern für Ausdrucksattributnamen
- Behandeln von Sonderzeichen in Attributnamen

Java

SDK für Java 2.x

Demonstrieren Sie die Namen der Ausdrucksattribute mit AWS SDK for Java 2.x.

```
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.AttributeValue;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.model.QueryRequest;
```

```
import software.amazon.awssdk.services.dynamodb.model.QueryResponse;
import software.amazon.awssdk.services.dynamodb.model.ScanRequest;
import software.amazon.awssdk.services.dynamodb.model.ScanResponse;
import software.amazon.awssdk.services.dynamodb.model.UpdateItemRequest;
import software.amazon.awssdk.services.dynamodb.model.UpdateItemResponse;

import java.util.Arrays;
import java.util.HashMap;
import java.util.List;
import java.util.Map;

    /**
     * Updates an attribute that is a reserved word in DynamoDB.
     *
     * <p>This method demonstrates how to use expression attribute names to
 update
     * attributes that are reserved words in DynamoDB.
     *
     * @param dynamoDbClient The DynamoDB client
     * @param tableName The name of the DynamoDB table
     * @param key The key of the item to update
     * @param reservedWordAttribute The reserved word attribute to update
     * @param value The value to set
     * @return The response from DynamoDB
     * @throws DynamoDbException if an error occurs during the operation
     */
    public static UpdateItemResponse updateReservedWordAttribute(
        DynamoDbClient dynamoDbClient,
        String tableName,
        Map<String, AttributeValue> key,
        String reservedWordAttribute,
        AttributeValue value) {

        // Define the update parameters using expression attribute names
        UpdateItemRequest request = UpdateItemRequest.builder()
            .tableName(tableName)
            .key(key)
            .updateExpression("SET #attr = :value")
            .expressionAttributeNames(Map.of("#attr", reservedWordAttribute))
            .expressionAttributeValues(Map.of(":value", value))
            .returnValues("UPDATED_NEW")
            .build();

        // Perform the update operation
```

```
        return dynamoDbClient.updateItem(request);
    }

    /**
     * Updates an attribute that contains special characters.
     *
     * <p>This method demonstrates how to use expression attribute names to
update
     * attributes that contain special characters.
     *
     * @param dynamoDbClient The DynamoDB client
     * @param tableName The name of the DynamoDB table
     * @param key The key of the item to update
     * @param specialCharAttribute The attribute with special characters to
update
     * @param value The value to set
     * @return The response from DynamoDB
     * @throws DynamoDbException if an error occurs during the operation
     */
    public static UpdateItemResponse updateSpecialCharacterAttribute(
        DynamoDbClient dynamoDbClient,
        String tableName,
        Map<String, AttributeValue> key,
        String specialCharAttribute,
        AttributeValue value) {

        // Define the update parameters using expression attribute names
        UpdateItemRequest request = UpdateItemRequest.builder()
            .tableName(tableName)
            .key(key)
            .updateExpression("SET #attr = :value")
            .expressionAttributeNames(Map.of("#attr", specialCharAttribute))
            .expressionAttributeValues(Map.of(":value", value))
            .returnValues("UPDATED_NEW")
            .build();

        // Perform the update operation
        return dynamoDbClient.updateItem(request);
    }

    /**
     * Queries items using an attribute that is a reserved word.
     *
```

```
     * <p>This method demonstrates how to use expression attribute names in a
query
     * when the attribute is a reserved word.
     *
     * @param dynamoDbClient The DynamoDB client
     * @param tableName The name of the DynamoDB table
     * @param partitionKeyName The name of the partition key attribute
     * @param partitionKeyValue The value of the partition key
     * @param reservedWordAttribute The reserved word attribute to filter on
     * @param value The value to compare against
     * @return The response from DynamoDB
     * @throws DynamoDbException if an error occurs during the operation
     */
    public static QueryResponse queryWithReservedWordAttribute(
        DynamoDbClient dynamoDbClient,
        String tableName,
        String partitionKeyName,
        AttributeValue partitionKeyValue,
        String reservedWordAttribute,
        AttributeValue value) {

        // Define the query parameters using expression attribute names
        Map<String, String> expressionAttributeNames = new HashMap<>();
        expressionAttributeNames.put("#pkName", partitionKeyName);
        expressionAttributeNames.put("#attr", reservedWordAttribute);

        Map<String, AttributeValue> expressionAttributeValues = new HashMap<>();
        expressionAttributeValues.put(":pkValue", partitionKeyValue);
        expressionAttributeValues.put(":value", value);

        QueryRequest request = QueryRequest.builder()
            .tableName(tableName)
            .keyConditionExpression("#pkName = :pkValue")
            .filterExpression("#attr = :value")
            .expressionAttributeNames(expressionAttributeNames)
            .expressionAttributeValues(expressionAttributeValues)
            .build();

        // Perform the query operation
        return dynamoDbClient.query(request);
    }

    /**
     * Updates a nested attribute with a path that contains reserved words.
```

```
     *
     * <p>This method demonstrates how to use expression attribute names to
update
     * nested attributes where the path contains reserved words.
     *
     * @param dynamoDbClient The DynamoDB client
     * @param tableName The name of the DynamoDB table
     * @param key The key of the item to update
     * @param attributePath The path to the nested attribute as an array
     * @param value The value to set
     * @return The response from DynamoDB
     * @throws DynamoDbException if an error occurs during the operation
     */
    public static UpdateItemResponse updateNestedReservedWordAttribute(
        DynamoDbClient dynamoDbClient,
        String tableName,
        Map<String, AttributeValue> key,
        List<String> attributePath,
        AttributeValue value) {

        // Create expression attribute names for each part of the path
        Map<String, String> expressionAttributeNames = new HashMap<>();
        for (int i = 0; i < attributePath.size(); i++) {
            expressionAttributeNames.put("#attr" + i, attributePath.get(i));
        }

        // Build the attribute path using the expression attribute names
        StringBuilder attributePathExpression = new StringBuilder();
        for (int i = 0; i < attributePath.size(); i++) {
            if (i > 0) {
                attributePathExpression.append(".");
            }
            attributePathExpression.append("#attr").append(i);
        }

        // Define the update parameters
        UpdateItemRequest request = UpdateItemRequest.builder()
            .tableName(tableName)
            .key(key)
            .updateExpression("SET " + attributePathExpression.toString() + "
= :value")
            .expressionAttributeNames(expressionAttributeNames)
            .expressionAttributeValues(Map.of(":value", value))
            .returnValues("UPDATED_NEW")
```

```
            .build();

        // Perform the update operation
        return dynamoDbClient.updateItem(request);
    }

    /**
     * Scans a table with multiple attribute name placeholders.
     *
     * <p>This method demonstrates how to use multiple expression attribute names
     * in a complex filter expression.
     *
     * @param dynamoDbClient The DynamoDB client
     * @param tableName The name of the DynamoDB table
     * @param filters Object mapping attribute names to filter values
     * @return The response from DynamoDB
     * @throws DynamoDbException if an error occurs during the operation
     */
    public static ScanResponse scanWithMultipleAttributeNames(
        DynamoDbClient dynamoDbClient, String tableName, Map<String,
AttributeValue> filters) {

        // Create expression attribute names and values
        Map<String, String> expressionAttributeNames = new HashMap<>();
        Map<String, AttributeValue> expressionAttributeValues = new HashMap<>();
        StringBuilder filterExpression = new StringBuilder();

        // Build the filter expression
        int index = 0;
        for (Map.Entry<String, AttributeValue> entry : filters.entrySet()) {
            String attrName = entry.getKey();
            AttributeValue attrValue = entry.getValue();

            String nameKey = "#attr" + index;
            String valueKey = ":val" + index;

            expressionAttributeNames.put(nameKey, attrName);
            expressionAttributeValues.put(valueKey, attrValue);

            // Add AND between conditions (except for the first one)
            if (index > 0) {
                filterExpression.append(" AND ");
            }
```

```
            filterExpression.append(nameKey).append(" = ").append(valueKey);
            index++;
        }

        // Define the scan parameters
        ScanRequest request = ScanRequest.builder()
            .tableName(tableName)
            .filterExpression(filterExpression.toString())
            .expressionAttributeNames(expressionAttributeNames)
            .expressionAttributeValues(expressionAttributeValues)
            .build();

        // Perform the scan operation
        return dynamoDbClient.scan(request);
    }
```

Beispiel für die Verwendung von Ausdrucksattributnamen mit AWS SDK for Java 2.x.

```
   public static void exampleUsage(DynamoDbClient dynamoDbClient, String
tableName) {
        // Example key
        Map<String, AttributeValue> key = new HashMap<>();
        key.put("ProductId", AttributeValue.builder().s("P12345").build());

        System.out.println("Demonstrating expression attribute names in
DynamoDB");

        try {
            // Example 1: Update an attribute that is a reserved word
            System.out.println("\nExample 1: Updating an attribute that is a
reserved word");
            UpdateItemResponse response1 = updateReservedWordAttribute(
                dynamoDbClient,
                tableName,
                key,
                "Size", // "SIZE" is a reserved word in DynamoDB
                AttributeValue.builder().s("Large").build());

            System.out.println("Updated attribute: " + response1.attributes());

            // Example 2: Update an attribute with special characters
```

```
            System.out.println("\nExample 2: Updating an attribute with special
characters");
            UpdateItemResponse response2 = updateSpecialCharacterAttribute(
                dynamoDbClient,
                tableName,
                key,
                "Product-Type", // Contains a hyphen, which is a special
character
                AttributeValue.builder().s("Electronics").build());

            System.out.println("Updated attribute: " + response2.attributes());

            // Example 3: Query with a reserved word attribute
            System.out.println("\nExample 3: Querying with a reserved word
attribute");
            QueryResponse response3 = queryWithReservedWordAttribute(
                dynamoDbClient,
                tableName,
                "Category",
                AttributeValue.builder().s("Electronics").build(),
                "Count", // "COUNT" is a reserved word in DynamoDB
                AttributeValue.builder().n("10").build());

            System.out.println("Found " + response3.count() + " items");

            // Example 4: Update a nested attribute with reserved words in the
path
            System.out.println("\nExample 4: Updating a nested attribute with
reserved words in the path");
            UpdateItemResponse response4 = updateNestedReservedWordAttribute(
                dynamoDbClient,
                tableName,
                key,
                Arrays.asList("Dimensions", "Size", "Height"), // "SIZE" is a
reserved word
                AttributeValue.builder().n("30").build());

            System.out.println("Updated nested attribute: " +
response4.attributes());

            // Example 5: Scan with multiple attribute name placeholders
            System.out.println("\nExample 5: Scanning with multiple attribute
name placeholders");
            Map<String, AttributeValue> filters = new HashMap<>();
```

```
            filters.put("Size", AttributeValue.builder().s("Large").build());
            filters.put("Count", AttributeValue.builder().n("10").build());
            filters.put(
                "Product-Type",
 AttributeValue.builder().s("Electronics").build());

            ScanResponse response5 =
 scanWithMultipleAttributeNames(dynamoDbClient, tableName, filters);

            System.out.println("Found " + response5.count() + " items");

            // Show some common reserved words
            System.out.println("\nSome common DynamoDB reserved words:");
            List<String> commonReservedWords = getDynamoDBReservedWords();
            System.out.println(String.join(", ", commonReservedWords));

            // Explain expression attribute names
            System.out.println("\nKey points about expression attribute names:");
            System.out.println("1. Use expression attribute names (#name) for
 reserved words");
            System.out.println("2. Use expression attribute names for attributes
 with special characters");
            System.out.println(
                "3. Special characters include: spaces, hyphens, dots, and other
 non-alphanumeric characters");
            System.out.println("4. Expression attribute names are required for
 nested attributes with reserved words");
            System.out.println("5. You can use multiple expression attribute
 names in a single expression");
            System.out.println("6. Expression attribute names are case-
sensitive");
            System.out.println("7. Expression attribute names are only used in
 expressions, not in the actual data");

        } catch (DynamoDbException e) {
            System.err.println("Error: " + e.getMessage());
            e.printStackTrace();
        }
    }
```

- Weitere API-Informationen finden Sie in den folgenden Themen der AWS SDK for Java 2.x - API-Referenz.

- Query
- UpdateItem

JavaScript

SDK für JavaScript (v3)

Demonstrieren Sie die Namen der Ausdrucksattribute mit AWS SDK für JavaScript.

```
const { DynamoDBClient } = require("@aws-sdk/client-dynamodb");
const {
  DynamoDBDocumentClient,
  UpdateCommand,
  GetCommand,
  QueryCommand,
  ScanCommand
} = require("@aws-sdk/lib-dynamodb");

/**
 * Update an attribute that is a reserved word in DynamoDB.
 *
 * This function demonstrates how to use expression attribute names to update
 * attributes that are reserved words in DynamoDB.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The key of the item to update
 * @param {string} reservedWordAttribute - The reserved word attribute to update
 * @param {any} value - The value to set
 * @returns {Promise<Object>} - The response from DynamoDB
 */
async function updateReservedWordAttribute(
  config,
  tableName,
  key,
  reservedWordAttribute,
  value
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);
```

```
  // Define the update parameters using expression attribute names
  const params = {
    TableName: tableName,
    Key: key,
    UpdateExpression: "SET #attr = :value",
    ExpressionAttributeNames: {
      "#attr": reservedWordAttribute
    },
    ExpressionAttributeValues: {
      ":value": value
    },
    ReturnValues: "UPDATED_NEW"
  };

  // Perform the update operation
  const response = await docClient.send(new UpdateCommand(params));

  return response;
}

/**
 * Update an attribute that contains special characters.
 *
 * This function demonstrates how to use expression attribute names to update
 * attributes that contain special characters.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The key of the item to update
 * @param {string} specialCharAttribute - The attribute with special characters
 to update
 * @param {any} value - The value to set
 * @returns {Promise<Object>} - The response from DynamoDB
 */
async function updateSpecialCharacterAttribute(
  config,
  tableName,
  key,
  specialCharAttribute,
  value
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);
```

```
  // Define the update parameters using expression attribute names
  const params = {
    TableName: tableName,
    Key: key,
    UpdateExpression: "SET #attr = :value",
    ExpressionAttributeNames: {
      "#attr": specialCharAttribute
    },
    ExpressionAttributeValues: {
      ":value": value
    },
    ReturnValues: "UPDATED_NEW"
  };

  // Perform the update operation
  const response = await docClient.send(new UpdateCommand(params));

  return response;
}

/**
 * Query items using an attribute that is a reserved word.
 *
 * This function demonstrates how to use expression attribute names in a query
 * when the attribute is a reserved word.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {string} partitionKeyName - The name of the partition key attribute
 * @param {any} partitionKeyValue - The value of the partition key
 * @param {string} reservedWordAttribute - The reserved word attribute to filter
 on
 * @param {any} value - The value to compare against
 * @returns {Promise<Object>} - The response from DynamoDB
 */
async function queryWithReservedWordAttribute(
  config,
  tableName,
  partitionKeyName,
  partitionKeyValue,
  reservedWordAttribute,
  value
) {
```

```
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

  // Define the query parameters using expression attribute names
  const params = {
    TableName: tableName,
    KeyConditionExpression: "#pkName = :pkValue",
    FilterExpression: "#attr = :value",
    ExpressionAttributeNames: {
      "#pkName": partitionKeyName,
      "#attr": reservedWordAttribute
    },
    ExpressionAttributeValues: {
      ":pkValue": partitionKeyValue,
      ":value": value
    }
  };

  // Perform the query operation
  const response = await docClient.send(new QueryCommand(params));

  return response;
}

/**
 * Update a nested attribute with a path that contains reserved words.
 *
 * This function demonstrates how to use expression attribute names to update
 * nested attributes where the path contains reserved words.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The key of the item to update
 * @param {string[]} attributePath - The path to the nested attribute as an array
 * @param {any} value - The value to set
 * @returns {Promise<Object>} - The response from DynamoDB
 */
async function updateNestedReservedWordAttribute(
  config,
  tableName,
  key,
  attributePath,
  value
```

```
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

  // Create expression attribute names for each part of the path
  const expressionAttributeNames = {};
  for (let i = 0; i < attributePath.length; i++) {
    expressionAttributeNames[`#attr${i}`] = attributePath[i];
  }

  // Build the attribute path using the expression attribute names
  const attributePathExpression = attributePath
    .map((_, i) => `#attr${i}`)
    .join(".");

  // Define the update parameters
  const params = {
    TableName: tableName,
    Key: key,
    UpdateExpression: `SET ${attributePathExpression} = :value`,
    ExpressionAttributeNames: expressionAttributeNames,
    ExpressionAttributeValues: {
      ":value": value
    },
    ReturnValues: "UPDATED_NEW"
  };

  // Perform the update operation
  const response = await docClient.send(new UpdateCommand(params));

  return response;
}

/**
 * Scan a table with multiple attribute name placeholders.
 *
 * This function demonstrates how to use multiple expression attribute names
 * in a complex filter expression.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} filters - Object mapping attribute names to filter values
 * @returns {Promise<Object>} - The response from DynamoDB
```

```
 */
async function scanWithMultipleAttributeNames(
  config,
  tableName,
  filters
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

  // Create expression attribute names and values
  const expressionAttributeNames = {};
  const expressionAttributeValues = {};
  const filterConditions = [];

  // Build the filter expression
  Object.entries(filters).forEach(([attrName, value], index) => {
    const nameKey = `#attr${index}`;
    const valueKey = `:val${index}`;

    expressionAttributeNames[nameKey] = attrName;
    expressionAttributeValues[valueKey] = value;
    filterConditions.push(`${nameKey} = ${valueKey}`);
  });

  // Join the filter conditions with AND
  const filterExpression = filterConditions.join(" AND ");

  // Define the scan parameters
  const params = {
    TableName: tableName,
    FilterExpression: filterExpression,
    ExpressionAttributeNames: expressionAttributeNames,
    ExpressionAttributeValues: expressionAttributeValues
  };

  // Perform the scan operation
  const response = await docClient.send(new ScanCommand(params));

  return response;
}

/**
 * Get the current value of an item.
```

```
 *
 * Helper function to retrieve the current value of an item.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The key of the item to get
 * @returns {Promise<Object|null>} - The item or null if not found
 */
async function getItem(
  config,
  tableName,
  key
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

  // Define the get parameters
  const params = {
    TableName: tableName,
    Key: key
  };

  // Perform the get operation
  const response = await docClient.send(new GetCommand(params));

  // Return the item if it exists, otherwise null
  return response.Item || null;
}
```

Beispiel für die Verwendung von Ausdrucksattributnamen mit AWS SDK für JavaScript.

```
/**
 * Example of how to use expression attribute names.
 */
async function exampleUsage() {
  // Example parameters
  const config = { region: "us-west-2" };
  const tableName = "Products";
  const key = { ProductId: "P12345" };

  console.log("Demonstrating expression attribute names in DynamoDB");
```

```
 try {
   // Example 1: Update an attribute that is a reserved word
   console.log("\nExample 1: Updating an attribute that is a reserved word");
   const response1 = await updateReservedWordAttribute(
     config,
     tableName,
     key,
     "Size", // "SIZE" is a reserved word in DynamoDB
     "Large"
   );

   console.log("Updated attribute:", response1.Attributes);

   // Example 2: Update an attribute with special characters
   console.log("\nExample 2: Updating an attribute with special characters");
   const response2 = await updateSpecialCharacterAttribute(
     config,
     tableName,
     key,
     "Product-Type", // Contains a hyphen, which is a special character
     "Electronics"
   );

   console.log("Updated attribute:", response2.Attributes);

   // Example 3: Query with a reserved word attribute
   console.log("\nExample 3: Querying with a reserved word attribute");
   const response3 = await queryWithReservedWordAttribute(
     config,
     tableName,
     "Category",
     "Electronics",
     "Count", // "COUNT" is a reserved word in DynamoDB
     10
   );

   console.log(`Found ${response3.Items.length} items`);

   // Example 4: Update a nested attribute with reserved words in the path
   console.log("\nExample 4: Updating a nested attribute with reserved words in
the path");
   const response4 = await updateNestedReservedWordAttribute(
     config,
```

```
      tableName,
      key,
      ["Dimensions", "Size", "Height"], // "SIZE" is a reserved word
      30
    );

    console.log("Updated nested attribute:", response4.Attributes);

    // Example 5: Scan with multiple attribute name placeholders
    console.log("\nExample 5: Scanning with multiple attribute name
placeholders");
    const response5 = await scanWithMultipleAttributeNames(
      config,
      tableName,
      {
        "Size": "Large",
        "Count": 10,
        "Product-Type": "Electronics"
      }
    );

    console.log(`Found ${response5.Items.length} items`);

    // Get the final state of the item
    console.log("\nFinal state of the item:");
    const item = await getItem(config, tableName, key);
    console.log(JSON.stringify(item, null, 2));

    // Show some common reserved words
    console.log("\nSome common DynamoDB reserved words:");
    const commonReservedWords = [
      "ABORT", "ABSOLUTE", "ACTION", "ADD", "ALL", "ALTER", "AND", "ANY", "AS",
      "ASC", "BETWEEN", "BY", "CASE", "CAST", "COLUMN", "CONNECT", "COUNT",
      "CREATE", "CURRENT", "DATE", "DELETE", "DESC", "DROP", "ELSE", "EXISTS",
      "FOR", "FROM", "GRANT", "GROUP", "HAVING", "IN", "INDEX", "INSERT", "INTO",
      "IS", "JOIN", "KEY", "LEVEL", "LIKE", "LIMIT", "LOCAL", "MAX", "MIN",
"NAME",
      "NOT", "NULL", "OF", "ON", "OR", "ORDER", "OUTER", "REPLACE", "RETURN",
      "SELECT", "SET", "SIZE", "TABLE", "THEN", "TO", "UPDATE", "USER", "VALUES",
      "VIEW", "WHERE"
    ];
    console.log(commonReservedWords.join(", "));

    // Explain expression attribute names
```

```
    console.log("\nKey points about expression attribute names:");
    console.log("1. Use expression attribute names (#name) for reserved words");
    console.log("2. Use expression attribute names for attributes with special
 characters");
    console.log("3. Special characters include: spaces, hyphens, dots, and other
 non-alphanumeric characters");
    console.log("4. Expression attribute names are required for nested attributes
 with reserved words");
    console.log("5. You can use multiple expression attribute names in a single
 expression");
    console.log("6. Expression attribute names are case-sensitive");
    console.log("7. Expression attribute names are only used in expressions, not
 in the actual data");

  } catch (error) {
    console.error("Error:", error);
  }
}
```

- Weitere API-Informationen finden Sie in den folgenden Themen der AWS SDK für JavaScript -API-Referenz.
  - Query
  - UpdateItem

Python

SDK für Python (Boto3)

Demonstrieren Sie Ausdrucksattributnamen mithilfe von AWS SDK für Python (Boto3).

```
import boto3
from botocore.exceptions import ClientError
from typing import Any, Dict, List


def use_reserved_word_attribute(
    table_name: str, key: Dict[str, Any], reserved_word: str, value: Any
) -> Dict[str, Any]:
    """
    Update an attribute whose name is a DynamoDB reserved word.
```

```
    This function demonstrates how to use expression attribute names to work with
    attributes that have names that are DynamoDB reserved words.

    Args:
        table_name (str): The name of the DynamoDB table.
        key (Dict[str, Any]): The primary key of the item to update.
        reserved_word (str): The reserved word to use as an attribute name.
        value (Any): The value to set for the attribute.

    Returns:
        Dict[str, Any]: The response from DynamoDB containing the updated
 attribute values.
    """
    # Initialize the DynamoDB resource
    dynamodb = boto3.resource("dynamodb")
    table = dynamodb.Table(table_name)

    # Use expression attribute names to handle the reserved word
    response = table.update_item(
        Key=key,
        UpdateExpression="SET #reserved_attr = :value",
        ExpressionAttributeNames={"#reserved_attr": reserved_word},
        ExpressionAttributeValues={":value": value},
        ReturnValues="UPDATED_NEW",
    )

    return response


def use_special_character_attribute(
    table_name: str, key: Dict[str, Any], attribute_with_special_chars: str,
 value: Any
) -> Dict[str, Any]:
    """
    Update an attribute whose name contains special characters.

    This function demonstrates how to use expression attribute names to work with
    attributes that have names containing special characters like spaces, dots,
 or hyphens.

    Args:
        table_name (str): The name of the DynamoDB table.
        key (Dict[str, Any]): The primary key of the item to update.
```

```
        attribute_with_special_chars (str): The attribute name with special
 characters.
        value (Any): The value to set for the attribute.

    Returns:
        Dict[str, Any]: The response from DynamoDB containing the updated
 attribute values.
    """
    # Initialize the DynamoDB resource
    dynamodb = boto3.resource("dynamodb")
    table = dynamodb.Table(table_name)

    # Use expression attribute names to handle special characters
    response = table.update_item(
        Key=key,
        UpdateExpression="SET #special_attr = :value",
        ExpressionAttributeNames={"#special_attr": attribute_with_special_chars},
        ExpressionAttributeValues={":value": value},
        ReturnValues="UPDATED_NEW",
    )

    return response


def query_with_attribute_names(
    table_name: str,
    partition_key_name: str,
    partition_key_value: str,
    filter_attribute_name: str,
    filter_value: Any,
) -> Dict[str, Any]:
    """
    Query a table using expression attribute names for both key and filter
 attributes.

    This function demonstrates how to use expression attribute names in a query
 operation
    for both the key condition expression and filter expression.

    Args:
        table_name (str): The name of the DynamoDB table.
        partition_key_name (str): The name of the partition key attribute.
        partition_key_value (str): The value of the partition key to query.
        filter_attribute_name (str): The name of the attribute to filter on.
```

```
        filter_value (Any): The value to compare against in the filter.

    Returns:
        Dict[str, Any]: The response from DynamoDB containing the query results.
    """
    # Initialize the DynamoDB resource
    dynamodb = boto3.resource("dynamodb")
    table = dynamodb.Table(table_name)

    # Use expression attribute names for both key condition and filter
    response = table.query(
        KeyConditionExpression="#pk = :pk_val",
        FilterExpression="#filter_attr = :filter_val",
        ExpressionAttributeNames={"#pk": partition_key_name, "#filter_attr":
 filter_attribute_name},
        ExpressionAttributeValues={":pk_val": partition_key_value, ":filter_val":
 filter_value},
    )

    return response


def update_nested_attribute_with_dots(
    table_name: str, key: Dict[str, Any], path_with_dots: str, value: Any
) -> Dict[str, Any]:
    """
    Update a nested attribute using a path with dot notation.

    This function demonstrates how to use expression attribute names to work with
    nested attributes specified using dot notation.

    Args:
        table_name (str): The name of the DynamoDB table.
        key (Dict[str, Any]): The primary key of the item to update.
        path_with_dots (str): The path to the nested attribute using dot notation
 (e.g., "a.b.c").
        value (Any): The value to set for the nested attribute.

    Returns:
        Dict[str, Any]: The response from DynamoDB containing the updated
 attribute values.
    """
    # Initialize the DynamoDB resource
    dynamodb = boto3.resource("dynamodb")
```

```
    table = dynamodb.Table(table_name)

    # Split the path into components
    path_parts = path_with_dots.split(".")

    # Build the update expression and attribute names
    update_expression = "SET "
    expression_attribute_names = {}

    # Build the path expression
    path_expression = ""
    for i, part in enumerate(path_parts):
        name_placeholder = f"#attr{i}"
        expression_attribute_names[name_placeholder] = part

        if i == 0:
            path_expression = name_placeholder
        else:
            path_expression += f".{name_placeholder}"

    # Complete the update expression
    update_expression += f"{path_expression} = :value"

    # Execute the update
    response = table.update_item(
        Key=key,
        UpdateExpression=update_expression,
        ExpressionAttributeNames=expression_attribute_names,
        ExpressionAttributeValues={":value": value},
        ReturnValues="UPDATED_NEW",
    )

    return response


def demonstrate_attribute_name_requirements(table_name: str, key: Dict[str, Any])
 -> Dict[str, Any]:
    """
    Demonstrate the requirements and allowed characters for attribute names.

    This function shows examples of valid and invalid attribute names and how to
    handle them using expression attribute names.

    Args:
```

```
        table_name (str): The name of the DynamoDB table.
        key (Dict[str, Any]): The primary key of the item to update.

    Returns:
        Dict[str, Any]: A dictionary containing the results of the demonstration.
    """
    # Initialize the DynamoDB resource
    dynamodb = boto3.resource("dynamodb")
    table = dynamodb.Table(table_name)

    # Examples of attribute names with different characteristics
    examples = {
        "valid_standard": "NormalAttribute",  # Standard attribute name (no
placeholder needed)
        "valid_with_underscore": "Normal_Attribute",  # Underscore is allowed
        "valid_with_number": "Attribute123",  # Numbers are allowed
        "reserved_word": "Timestamp",  # Reserved word (requires placeholder)
        "starts_with_number": "123Attribute",  # Starts with number (valid but
may need placeholder in some contexts)
        "with_space": "Attribute Name",  # Contains space (requires placeholder)
        "with_dot": "Attribute.Name",  # Contains dot (requires placeholder)
        "with_hyphen": "Attribute-Name",  # Contains hyphen (requires
placeholder)
        "with_special_chars": "Attribute#$%",  # Contains special characters
(requires placeholder)
    }

    results = {}

    # Try to update each attribute type
    for example_type, attr_name in examples.items():
        try:
            # For attributes that don't need placeholders, try direct reference
            if example_type in ["valid_standard", "valid_with_underscore",
"valid_with_number"]:
                try:
                    # Try without expression attribute names first
                    response = table.update_item(
                        Key=key,
                        UpdateExpression=f"SET {attr_name} = :value",
                        ExpressionAttributeValues={":value": f"Value for
{attr_name}"},
                        ReturnValues="UPDATED_NEW",
                    )
```

```
                    results[example_type] = {
                        "attribute_name": attr_name,
                        "success": True,
                        "needed_placeholder": False,
                        "response": response,
                    }
                except ClientError:
                    # If direct reference fails, try with placeholder
                    response = table.update_item(
                        Key=key,
                        UpdateExpression="SET #attr = :value",
                        ExpressionAttributeNames={"#attr": attr_name},
                        ExpressionAttributeValues={":value": f"Value for
{attr_name}"},
                        ReturnValues="UPDATED_NEW",
                    )
                    results[example_type] = {
                        "attribute_name": attr_name,
                        "success": True,
                        "needed_placeholder": True,
                        "response": response,
                    }
            else:
                # For attributes that definitely need placeholders
                response = table.update_item(
                    Key=key,
                    UpdateExpression="SET #attr = :value",
                    ExpressionAttributeNames={"#attr": attr_name},
                    ExpressionAttributeValues={":value": f"Value for
{attr_name}"},
                    ReturnValues="UPDATED_NEW",
                )
                results[example_type] = {
                    "attribute_name": attr_name,
                    "success": True,
                    "needed_placeholder": True,
                    "response": response,
                }
        except ClientError as e:
            results[example_type] = {"attribute_name": attr_name, "success":
False, "error": str(e)}

    return results
```

Beispiel für die Verwendung von Ausdrucksattributnamen mit AWS SDK für Python (Boto3).

```
def example_usage():
    """Example of how to use expression attribute names in DynamoDB."""
    # Example parameters
    table_name = "Products"
    key = {"ProductId": "prod123"}

    print("Example 1: Using a reserved word as an attribute name")
    try:
        response = use_reserved_word_attribute(
            table_name=table_name, key=key, reserved_word="Timestamp",
 value="2025-05-14T12:00:00Z"
        )
        print(f"Reserved word attribute updated successfully:
 {response.get('Attributes', {})}")
    except Exception as e:
        print(f"Error updating reserved word attribute: {e}")

    print("\nExample 2: Using an attribute name with special characters")
    try:
        response = use_special_character_attribute(
            table_name=table_name,
            key=key,
            attribute_with_special_chars="Product Info",
            value="Special product information",
        )
        print(f"Special character attribute updated successfully:
 {response.get('Attributes', {})}")
    except Exception as e:
        print(f"Error updating special character attribute: {e}")

    print("\nExample 3: Querying with expression attribute names")
    try:
        response = query_with_attribute_names(
            table_name=table_name,
            partition_key_name="Category",
            partition_key_value="Electronics",
            filter_attribute_name="Price",
            filter_value=500,
```

```
        )
        print(
            f"Query with expression attribute names returned
{len(response.get('Items', []))} items"
        )
    except Exception as e:
        print(f"Error querying with expression attribute names: {e}")

    print("\nExample 4: Updating a nested attribute with dot notation")
    try:
        response = update_nested_attribute_with_dots(
            table_name=table_name,
            key=key,
            path_with_dots="Product.Details.Specifications",
            value={"Weight": "2.5 kg", "Dimensions": "30x20x10 cm"},
        )
        print(f"Nested attribute updated successfully:
{response.get('Attributes', {})}")
    except Exception as e:
        print(f"Error updating nested attribute: {e}")

    print("\nExample 5: Demonstrating attribute name requirements")
    try:
        results = demonstrate_attribute_name_requirements(table_name=table_name,
key=key)

        print("Attribute Name Requirements Results:")
        for example_type, result in results.items():
            if result.get("success", False):
                needed_placeholder = result.get("needed_placeholder", True)
                print(
                    f"  - {example_type}: '{result['attribute_name']}' -
{'Requires' if needed_placeholder else 'Does not require'} placeholder"
                )
            else:
                print(
                    f"  - {example_type}: '{result['attribute_name']}' - Failed:
{result.get('error', 'Unknown error')}"
                )
    except Exception as e:
        print(f"Error demonstrating attribute name requirements: {e}")

    print("\nCommon DynamoDB Reserved Words (sample):")
    reserved_words = get_common_reserved_words()
```

```
    print(", ".join(reserved_words[:20]) + "... (and many more)")

    print("\nWhen to Use Expression Attribute Names:")
    print("1. When the attribute name is a DynamoDB reserved word")
    print("2. When the attribute name contains special characters (spaces, dots,
hyphens)")
    print("3. When the attribute name begins with a number")
    print("4. When working with nested attributes using dot notation")
    print("5. When you need to reference the same attribute multiple times in an
expression")

    print("\nExpression Attribute Name Requirements:")
    print("1. Must begin with a pound sign (#)")
    print("2. After the pound sign, must contain at least one character")
    print("3. Can contain alphanumeric characters and underscore (_)")
    print("4. Are case-sensitive")
    print("5. Must be unique within a single expression")

    print("\nAttribute Name Requirements in DynamoDB:")
    print("1. Can begin with a-z, A-Z, or 0-9")
    print("2. Can contain a-z, A-Z, 0-9, underscore (_), dash (-), and dot (.)")
    print("3. Are case-sensitive")
    print("4. No length restrictions, but practical limits apply")
    print("5. Cannot be a DynamoDB reserved word if used directly in
expressions")
```

- Weitere API-Informationen finden Sie in den folgenden Themen der API-Referenz zum AWS SDK für Python (Boto3).
  - Query
  - UpdateItem

Eine vollständige Liste der AWS SDK-Entwicklerhandbücher und Codebeispiele finden Sie unterDynamoDB mit einem SDK verwenden AWS. Dieses Thema enthält auch Informationen zu den ersten Schritten und Details zu früheren SDK-Versionen.

Verwendung geplanter Ereignisse zum Aufrufen einer Lambda-Funktion

Die folgenden Codebeispiele zeigen, wie eine AWS Lambda Funktion erstellt wird, die durch ein von Amazon EventBridge geplantes Ereignis aufgerufen wird.

Java

SDK für Java 2.x

Zeigt, wie ein von Amazon EventBridge geplantes Ereignis erstellt wird, das eine AWS Lambda Funktion aufruft. Konfigurieren Sie so EventBridge , dass ein Cron-Ausdruck verwendet wird, um zu planen, wann die Lambda-Funktion aufgerufen wird. In diesem Beispiel erstellen Sie eine Lambda-Funktion mithilfe der Lambda-Java-Laufzeit-API. In diesem Beispiel werden verschiedene AWS Dienste aufgerufen, um einen bestimmten Anwendungsfall auszuführen. Dieses Beispiel zeigt, wie man eine App erstellt, die eine mobile Textnachricht an Ihre Mitarbeiter sendet, um ihnen zum einjährigen Jubiläum zu gratulieren.

Den vollständigen Quellcode und Anweisungen zur Einrichtung und Ausführung finden Sie im vollständigen Beispiel unter GitHub.

In diesem Beispiel verwendete Dienste

- CloudWatch Logs
- DynamoDB
- EventBridge
- Lambda
- Amazon SNS

JavaScript

SDK für JavaScript (v3)

Zeigt, wie ein von Amazon EventBridge geplantes Ereignis erstellt wird, das eine AWS Lambda Funktion aufruft. Konfigurieren Sie so EventBridge , dass ein Cron-Ausdruck verwendet wird, um zu planen, wann die Lambda-Funktion aufgerufen wird. In diesem Beispiel erstellen Sie eine Lambda-Funktion mithilfe der JavaScript Lambda-Laufzeit-API. In diesem Beispiel werden verschiedene AWS Dienste aufgerufen, um einen bestimmten Anwendungsfall auszuführen. Dieses Beispiel zeigt, wie man eine App erstellt, die eine mobile Textnachricht an Ihre Mitarbeiter sendet, um ihnen zum einjährigen Jubiläum zu gratulieren.

Den vollständigen Quellcode und Anweisungen zur Einrichtung und Ausführung finden Sie im vollständigen Beispiel unter GitHub.

Dieses Beispiel ist auch verfügbar im AWS SDK für JavaScript Entwicklerhandbuch für v3.

In diesem Beispiel verwendete Dienste

- CloudWatch Logs
- DynamoDB
- EventBridge
- Lambda
- Amazon SNS

Python

SDK für Python (Boto3)

Dieses Beispiel zeigt, wie eine AWS Lambda Funktion als Ziel einer geplanten EventBridge Amazon-Veranstaltung registriert wird. Der Lambda-Handler schreibt eine freundliche Nachricht und die vollständigen Ereignisdaten für den späteren Abruf in Amazon CloudWatch Logs.

- Stellt eine Lambda-Funktion bereit.
- Erzeugt ein EventBridge geplantes Ereignis und macht die Lambda-Funktion zum Ziel.
- Erteilt die Erlaubnis, die EventBridge Lambda-Funktion aufrufen zu lassen.
- Druckt die neuesten Daten aus CloudWatch Logs, um das Ergebnis der geplanten Aufrufe anzuzeigen.
- Bereinigt alle Ressourcen, die während der Demo erstellt wurden.

Dieses Beispiel lässt sich am besten auf ansehen. GitHub Den vollständigen Quellcode und Anweisungen zur Einrichtung und Ausführung finden Sie im vollständigen Beispiel unter GitHub.

In diesem Beispiel verwendete Dienste

- CloudWatch Logs
- DynamoDB
- EventBridge
- Lambda

- Amazon SNS

Eine vollständige Liste der AWS SDK-Entwicklerhandbücher und Codebeispiele finden Sie unterDynamoDB mit einem SDK verwenden AWS. Dieses Thema enthält auch Informationen zu den ersten Schritten und Details zu früheren SDK-Versionen.

Arbeiten Sie mit lokalen sekundären DynamoDB-Indizes unter Verwendung von v2 AWS Command Line Interface

Im folgenden Codebeispiel wird gezeigt, wie Sie Tabellen mit lokalen sekundären Indizes erstellen und abfragen.

- Erstellen einer Tabelle mit einem lokalen sekundären Index (LSI)
- Erstellen Sie eine Tabelle mit mehreren LSIs mit unterschiedlichen Projektionstypen.
- Daten abfragen mit LSIs.

Bash

AWS CLI mit Bash-Skript

Erstellen Sie eine Tabelle mit einem lokalen sekundären Index.

```
# Create a table with a Local Secondary Index
aws dynamodb create-table \
    --table-name CustomerOrders \
    --attribute-definitions \
        AttributeName=CustomerID,AttributeType=S \
        AttributeName=OrderID,AttributeType=S \
        AttributeName=OrderDate,AttributeType=S \
    --key-schema \
        AttributeName=CustomerID,KeyType=HASH \
        AttributeName=OrderID,KeyType=RANGE \
    --local-secondary-indexes \
        "IndexName=OrderDateIndex,\
        KeySchema=[{AttributeName=CustomerID,KeyType=HASH},
{AttributeName=OrderDate,KeyType=RANGE}],\
        Projection={ProjectionType=ALL}" \
    --billing-mode PAY_PER_REQUEST
```

Erstellen Sie eine Tabelle mit mehreren LSIs.

```
# Create a table with multiple Local Secondary Indexes
aws dynamodb create-table \
    --table-name CustomerDetails \
    --attribute-definitions \
        AttributeName=CustomerID,AttributeType=S \
        AttributeName=Name,AttributeType=S \
        AttributeName=Email,AttributeType=S \
        AttributeName=RegistrationDate,AttributeType=S \
    --key-schema \
        AttributeName=CustomerID,KeyType=HASH \
        AttributeName=Name,KeyType=RANGE \
    --local-secondary-indexes \
        "[
            {
                \"IndexName\": \"EmailIndex\",
                \"KeySchema\": [
                    {\"AttributeName\":\"CustomerID\",\"KeyType\":\"HASH\"},
                    {\"AttributeName\":\"Email\",\"KeyType\":\"RANGE\"}
                ],
                \"Projection\": {\"ProjectionType\":\"INCLUDE\",
\"NonKeyAttributes\":[\"Address\",\"Phone\"]}
            },
            {
                \"IndexName\": \"RegistrationIndex\",
                \"KeySchema\": [
                    {\"AttributeName\":\"CustomerID\",\"KeyType\":\"HASH\"},
                    {\"AttributeName\":\"RegistrationDate\",\"KeyType\":\"RANGE
\"}
                ],
                \"Projection\": {\"ProjectionType\":\"KEYS_ONLY\"}
            }
        ]" \
    --billing-mode PAY_PER_REQUEST
```

Daten abfragen mit LSIs.

```
# Query the OrderDateIndex LSI
aws dynamodb query \
    --table-name CustomerOrders \
    --index-name OrderDateIndex \
```

```
    --key-condition-expression "CustomerID = :custId AND OrderDate BETWEEN :date1
 AND :date2" \
    --expression-attribute-values '{
        ":custId": {"S": "C1"},
        ":date1": {"S": "2023-01-01"},
        ":date2": {"S": "2023-02-01"}
    }'

# Query with a filter expression
aws dynamodb query \
    --table-name CustomerOrders \
    --index-name OrderDateIndex \
    --key-condition-expression "CustomerID = :custId" \
    --filter-expression "Amount > :amount" \
    --expression-attribute-values '{
        ":custId": {"S": "C1"},
        ":amount": {"N": "150"}
    }'
```

- Weitere API-Informationen finden Sie in den folgenden Themen der AWS CLI - Befehlsreferenz.
  - CreateTable
  - Query

Eine vollständige Liste der AWS SDK-Entwicklerhandbücher und Codebeispiele finden Sie unterDynamoDB mit einem SDK verwenden AWS. Dieses Thema enthält auch Informationen zu den ersten Schritten und Details zu früheren SDK-Versionen.

Arbeiten Sie mit DynamoDB Streams und Time-to-Live verwenden Sie v2 AWS Command Line Interface

Das folgende Codebeispiel zeigt, wie DynamoDB Streams und Time-to-Live -Funktionen verwaltet werden.

- Erstellen Sie eine Tabelle mit aktiviertem Streams.
- Beschreiben Sie Streams.
- Erstellen Sie eine Lambda-Funktion für die Verarbeitung von Streams.
- Aktivieren Sie TTL für eine Tabelle.

- Fügen Sie Elemente mit TTL-Attributen hinzu.
- Beschreiben Sie TTL-Einstellungen.

Bash

AWS CLI mit Bash-Skript

Erstellen Sie eine Tabelle mit aktiviertem Streams.

```
# Create a table with DynamoDB Streams enabled
aws dynamodb create-table \
    --table-name StreamsDemo \
    --attribute-definitions \
        AttributeName=ID,AttributeType=S \
    --key-schema \
        AttributeName=ID,KeyType=HASH \
    --billing-mode PAY_PER_REQUEST \
    --stream-specification StreamEnabled=true,StreamViewType=NEW_AND_OLD_IMAGES
```

Beschreiben Sie Streams.

```
# Get information about the stream
aws dynamodb describe-table \
    --table-name StreamsDemo \
    --query "Table.StreamSpecification"

# Get the stream ARN
STREAM_ARN=$(aws dynamodb describe-table \
    --table-name StreamsDemo \
    --query "Table.LatestStreamArn" \
    --output text)

echo "Stream ARN: $STREAM_ARN"

# Describe the stream
aws dynamodbstreams describe-stream \
    --stream-arn $STREAM_ARN
```

Erstellen Sie eine Lambda-Funktion für Streams.

```
# Step 1: Create an IAM role for the Lambda function
cat > trust-policy.json << 'EOF'
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "lambda.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
EOF

aws iam create-role \
    --role-name DynamoDBStreamsLambdaRole \
    --assume-role-policy-document file://trust-policy.json

# Step 2: Attach permissions to the role
aws iam attach-role-policy \
    --role-name DynamoDBStreamsLambdaRole \
    --policy-arn arn:aws:iam::aws:policy/service-role/
AWSLambdaDynamoDBExecutionRole

# Step 3: Create a Lambda function (code would be in a separate file)
echo "Lambda function creation would be done separately with appropriate code"

# Step 4: Create an event source mapping
echo "Example command to create event source mapping:"
echo "aws lambda create-event-source-mapping \\"
echo "    --function-name ProcessDynamoDBRecords \\"
echo "    --event-source $STREAM_ARN \\"
echo "    --batch-size 100 \\"
echo "    --starting-position LATEST"
```

Aktivieren Sie TTL für eine Tabelle.

```
# Create a table for TTL demonstration
aws dynamodb create-table \
    --table-name TTLDemo \
```

```
    --attribute-definitions \
        AttributeName=ID,AttributeType=S \
    --key-schema \
        AttributeName=ID,KeyType=HASH \
    --billing-mode PAY_PER_REQUEST

# Wait for table to become active
aws dynamodb wait table-exists --table-name TTLDemo

# Enable TTL on the table
aws dynamodb update-time-to-live \
    --table-name TTLDemo \
    --time-to-live-specification "Enabled=true, AttributeName=ExpirationTime"
```

Fügen Sie Elemente mit TTL-Attributen hinzu.

```
# Calculate expiration time (current time + 1 day in seconds)
EXPIRATION_TIME=$(date -d "+1 day" +%s)

# Add an item with TTL attribute
aws dynamodb put-item \
    --table-name TTLDemo \
    --item '{
        "ID": {"S": "item1"},
        "Data": {"S": "This item will expire in 1 day"},
        "ExpirationTime": {"N": "'$EXPIRATION_TIME'"}
    }'

# Add an item that expires in 1 hour
EXPIRATION_TIME_HOUR=$(date -d "+1 hour" +%s)
aws dynamodb put-item \
    --table-name TTLDemo \
    --item '{
        "ID": {"S": "item2"},
        "Data": {"S": "This item will expire in 1 hour"},
        "ExpirationTime": {"N": "'$EXPIRATION_TIME_HOUR'"}
    }'
```

Beschreiben Sie TTL-Einstellungen.

```
# Describe TTL settings for a table
```

```
aws dynamodb describe-time-to-live \
    --table-name TTLDemo
```

- Weitere API-Informationen finden Sie in den folgenden Themen der AWS CLI - Befehlsreferenz.
  - AttachRolePolicy
  - CreateRole
  - CreateTable
  - DescribeTable
  - DescribeTimeToLive
  - PutItem
  - UpdateTimeToLive

Eine vollständige Liste der AWS SDK-Entwicklerhandbücher und Codebeispiele finden Sie unterDynamoDB mit einem SDK verwenden AWS. Dieses Thema enthält auch Informationen zu den ersten Schritten und Details zu früheren SDK-Versionen.

Arbeiten Sie mit globalen DynamoDB-Tabellen und Multi-Region-Replikation mit Eventual Consistency (MREC) unter Verwendung von v2 AWS Command Line Interface

Das folgende Codebeispiel zeigt, wie globale DynamoDB-Tabellen mit letztendlicher multiregionaler Konsistenz (Multi-Region Eventual Consistency, MREC) verwaltet werden.

- Erstellen Sie eine Tabelle mit multiregionaler Replikation (MREC).
- Fügen Sie Elemente in eine Replikattabelle ein und rufen Sie sie ab.
- Entfernen Sie Replikate one-by-one.
- Führen Sie eine Bereinigung durch Löschen der Tabelle aus.

Bash

AWS CLI mit Bash-Skript

Erstellen Sie eine Tabelle mit multiregionaler Replikation.

```
# Step 1: Create a new table (MusicTable) in US East (Ohio), with DynamoDB
 Streams enabled (NEW_AND_OLD_IMAGES)
aws dynamodb create-table \
    --table-name MusicTable \
    --attribute-definitions \
        AttributeName=Artist,AttributeType=S \
        AttributeName=SongTitle,AttributeType=S \
    --key-schema \
        AttributeName=Artist,KeyType=HASH \
        AttributeName=SongTitle,KeyType=RANGE \
    --billing-mode PAY_PER_REQUEST \
    --stream-specification StreamEnabled=true,StreamViewType=NEW_AND_OLD_IMAGES \
    --region us-east-2

# Step 2: Create an identical MusicTable table in US East (N. Virginia)
aws dynamodb update-table --table-name MusicTable --cli-input-json \
'{
  "ReplicaUpdates":
  [
    {
      "Create": {
        "RegionName": "us-east-1"
      }
    }
  ]
}' \
--region us-east-2

# Step 3: Create a table in Europe (Ireland)
aws dynamodb update-table --table-name MusicTable --cli-input-json \
'{
  "ReplicaUpdates":
  [
    {
      "Create": {
        "RegionName": "eu-west-1"
      }
    }
  ]
}' \
--region us-east-2
```

Beschreiben Sie die multiregionale Tabelle.

```
# Step 4: View the list of replicas created using describe-table
aws dynamodb describe-table \
    --table-name MusicTable \
    --region us-east-2 \
    --query 'Table.
{TableName:TableName,TableStatus:TableStatus,MultiRegionConsistency:MultiRegionConsistenc
{Region:RegionName,Status:ReplicaStatus}}'
```

Fügen Sie Elemente in eine Replikattabelle ein.

```
# Step 5: To verify that replication is working, add a new item to the Music
 table in US East (Ohio)
aws dynamodb put-item \
    --table-name MusicTable \
    --item '{"Artist": {"S":"item_1"},"SongTitle": {"S":"Song Value 1"}}' \
    --region us-east-2
```

Rufen Sie Elemente aus Replikattabellen ab.

```
# Step 6: Wait for a few seconds, and then check to see whether the item has
 been
# successfully replicated to US East (N. Virginia) and Europe (Ireland)
aws dynamodb get-item \
    --table-name MusicTable \
    --key '{"Artist": {"S":"item_1"},"SongTitle": {"S":"Song Value 1"}}' \
    --region us-east-1

aws dynamodb get-item \
    --table-name MusicTable \
    --key '{"Artist": {"S":"item_1"},"SongTitle": {"S":"Song Value 1"}}' \
    --region eu-west-1
```

Entfernen Sie Replikate.

```
# Step 7: Delete the replica table in Europe (Ireland) Region
aws dynamodb update-table --table-name MusicTable --cli-input-json \
```

```
'{
  "ReplicaUpdates":
  [
    {
      "Delete": {
        "RegionName": "eu-west-1"
      }
    }
  ]
}' \
--region us-east-2

# Delete the replica table in US East (N. Virginia) Region
aws dynamodb update-table --table-name MusicTable --cli-input-json \
'{
  "ReplicaUpdates":
  [
    {
      "Delete": {
        "RegionName": "us-east-1"
      }
    }
  ]
}' \
--region us-east-2
```

Führen Sie eine Bereinigung durch Löschen der Tabelle aus.

```
# Clean up: Delete the primary table
aws dynamodb delete-table --table-name MusicTable --region us-east-2

echo "Global table demonstration complete."
```

- Weitere API-Informationen finden Sie in den folgenden Themen der AWS CLI - Befehlsreferenz.
  - CreateTable
  - DeleteTable
  - DescribeTable
  - GetItem

- PutItem
- UpdateTable

Eine vollständige Liste der AWS SDK-Entwicklerhandbücher und Codebeispiele finden Sie unterDynamoDB mit einem SDK verwenden AWS. Dieses Thema enthält auch Informationen zu den ersten Schritten und Details zu früheren SDK-Versionen.

Arbeiten Sie mit DynamoDB-Ressourcen-Tagging mithilfe von v2 AWS Command Line Interface

Im folgenden Codebeispiel wird gezeigt, wie Sie Tags für DynamoDB-Ressourcen verwalten.

- Erstellen Sie eine Tabelle mit Tags.
- Listen Sie Tags für eine Ressource auf.
- Fügen Sie Tags zu einer Ressource hinzu.
- Entfernen Sie Tags aus einer Ressource.
- Filtern Sie Tabellen nach Tags.

Bash

AWS CLI mit Bash-Skript

Erstellen Sie eine Tabelle mit Tags.

```
# Create a table with tags
aws dynamodb create-table \
    --table-name TaggedTable \
    --attribute-definitions \
        AttributeName=ID,AttributeType=S \
    --key-schema \
        AttributeName=ID,KeyType=HASH \
    --billing-mode PAY_PER_REQUEST \
    --tags \
        Key=Environment,Value=Production \
        Key=Project,Value=Analytics \
        Key=Owner,Value=DataTeam
```

Listen Sie Tags für eine Ressource auf.

```
# Get the table ARN
TABLE_ARN=$(aws dynamodb describe-table \
    --table-name TaggedTable \
    --query "Table.TableArn" \
    --output text)

# List tags for the table
aws dynamodb list-tags-of-resource \
    --resource-arn $TABLE_ARN
```

Fügen Sie Tags zu einer Ressource hinzu.

```
# Add tags to an existing table
aws dynamodb tag-resource \
    --resource-arn $TABLE_ARN \
    --tags \
        Key=CostCenter,Value=12345 \
        Key=BackupSchedule,Value=Daily
```

Entfernen Sie Tags aus einer Ressource.

```
# Remove tags from a table
aws dynamodb untag-resource \
    --resource-arn $TABLE_ARN \
    --tag-keys Owner BackupSchedule
```

Filtern Sie Tabellen nach Tags.

```
# Create another table with different tags
aws dynamodb create-table \
    --table-name AnotherTaggedTable \
    --attribute-definitions \
        AttributeName=ID,AttributeType=S \
    --key-schema \
        AttributeName=ID,KeyType=HASH \
    --billing-mode PAY_PER_REQUEST \
```

```
    --tags \
        Key=Environment,Value=Development \
        Key=Project,Value=Testing

# Wait for table to become active
aws dynamodb wait table-exists --table-name AnotherTaggedTable

# List all tables
echo "All tables:"
aws dynamodb list-tables

# Get ARNs for all tables
echo -e "\nFiltering tables by Environment=Production tag:"
TABLE_ARNS=$(aws dynamodb list-tables --query "TableNames[*]" --output text |
 xargs -I {} aws dynamodb describe-table --table-name {} --query "Table.TableArn"
 --output text)

# Find tables with specific tag
for ARN in $TABLE_ARNS; do
    TABLE_NAME=$(echo $ARN | awk -F/ '{print $2}')
    TAGS=$(aws dynamodb list-tags-of-resource --resource-arn $ARN --query "Tags[?
Key=='Environment' && Value=='Production']" --output text)
    if [ ! -z "$TAGS" ]; then
        echo "Table with Production tag: $TABLE_NAME"
    fi
done
```

- Weitere API-Informationen finden Sie in den folgenden Themen der AWS CLI - Befehlsreferenz.
  - CreateTable
  - ListTagsOfResource
  - TagResource
  - UntagResource

Eine vollständige Liste der AWS SDK-Entwicklerhandbücher und Codebeispiele finden Sie unterDynamoDB mit einem SDK verwenden AWS. Dieses Thema enthält auch Informationen zu den ersten Schritten und Details zu früheren SDK-Versionen.

Arbeiten Sie mit DynamoDB-Tabellenverschlüsselung mit Version 2 AWS Command Line Interface

Im folgenden Codebeispiel wird gezeigt, wie Sie Verschlüsselungsoptionen für DynamoDB-Tabellen verwalten.

- Erstellen Sie eine Tabelle mit Standardverschlüsselung.
- Erstellen Sie eine Tabelle mit einem kundenverwalteten CMK.
- Aktualisieren Sie die Einstellungen für die Tabellenverschlüsselung.
- Beschreiben Sie die Tabellenverschlüsselung.

Bash

AWS CLI mit Bash-Skript

Erstellen Sie eine Tabelle mit Standardverschlüsselung.

```
# Create a table with default encryption (AWS owned key)
aws dynamodb create-table \
    --table-name CustomerData \
    --attribute-definitions \
        AttributeName=CustomerID,AttributeType=S \
    --key-schema \
        AttributeName=CustomerID,KeyType=HASH \
    --billing-mode PAY_PER_REQUEST \
    --sse-specification Enabled=true,SSEType=KMS
```

Erstellen Sie eine Tabelle mit einem kundenverwalteten CMK.

```
# Step 1: Create a customer managed key in KMS
aws kms create-key \
    --description "Key for DynamoDB table encryption" \
    --key-usage ENCRYPT_DECRYPT \
    --customer-master-key-spec SYMMETRIC_DEFAULT

# Store the key ID for later use
KEY_ID=$(aws kms list-keys --query "Keys[?contains(KeyArn, 'Key for
 DynamoDB')].KeyId" --output text)
```

```
# Step 2: Create a table with the customer managed key
aws dynamodb create-table \
    --table-name SensitiveData \
    --attribute-definitions \
        AttributeName=RecordID,AttributeType=S \
    --key-schema \
        AttributeName=RecordID,KeyType=HASH \
    --billing-mode PAY_PER_REQUEST \
    --sse-specification Enabled=true,SSEType=KMS,KMSMasterKeyId=$KEY_ID
```

Aktualisieren Sie die Tabellenverschlüsselung.

```
# Update a table to use a different KMS key
aws dynamodb update-table \
    --table-name CustomerData \
    --sse-specification Enabled=true,SSEType=KMS,KMSMasterKeyId=$KEY_ID
```

Beschreiben Sie die Tabellenverschlüsselung.

```
# Describe the table to see encryption settings
aws dynamodb describe-table \
    --table-name CustomerData \
    --query "Table.SSEDescription"
```

- Weitere API-Informationen finden Sie in den folgenden Themen der AWS CLI - Befehlsreferenz.
  - CreateKey
  - CreateTable
  - DescribeTable
  - UpdateTable

Eine vollständige Liste der AWS SDK-Entwicklerhandbücher und Codebeispiele finden Sie unterDynamoDB mit einem SDK verwenden AWS. Dieses Thema enthält auch Informationen zu den ersten Schritten und Details zu früheren SDK-Versionen.

Serverless-Beispiele für DynamoDB

Die folgenden Codebeispiele zeigen, wie DynamoDB mit verwendet wird. AWS SDKs

Beispiele

- Aufrufen einer Lambda-Funktion über einen DynamoDB-Auslöser
- Melden von Batch-Elementfehlern für Lambda-Funktionen mit einem DynamoDB-Auslöser

Aufrufen einer Lambda-Funktion über einen DynamoDB-Auslöser

Die folgenden Codebeispiele veranschaulichen, wie eine Lambda-Funktion implementiert wird, die ein durch den Empfang von Datensätzen aus einem DynamoDB-Stream ausgelöstes Ereignis empfängt. Die Funktion ruft die DynamoDB-Nutzdaten ab und protokolliert den Inhalt des Datensatzes.

.NET

SDK für .NET

> **Note**
>
> Es gibt noch mehr dazu GitHub. Das vollständige Beispiel sowie eine Anleitung zum Einrichten und Ausführen finden Sie im Repository mit Serverless-Beispielen.

Nutzen eines DynamoDB-Ereignisses mit Lambda unter Verwendung von .NET.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
using System.Text.Json;
using System.Text;
using Amazon.Lambda.Core;
using Amazon.Lambda.DynamoDBEvents;

// Assembly attribute to enable the Lambda function's JSON input to be converted
 into a .NET class.
[assembly:
 LambdaSerializer(typeof(Amazon.Lambda.Serialization.SystemTextJson.DefaultLambdaJsonSeri

namespace AWSLambda_DDB;
```

```
public class Function
{
    public void FunctionHandler(DynamoDBEvent dynamoEvent, ILambdaContext
 context)
    {
        context.Logger.LogInformation($"Beginning to process
 {dynamoEvent.Records.Count} records...");

        foreach (var record in dynamoEvent.Records)
        {
            context.Logger.LogInformation($"Event ID: {record.EventID}");
            context.Logger.LogInformation($"Event Name: {record.EventName}");

            context.Logger.LogInformation(JsonSerializer.Serialize(record));
        }

        context.Logger.LogInformation("Stream processing complete.");
    }
}
```

Go

SDK für Go V2

> **Note**
>
> Es gibt noch mehr GitHub. Das vollständige Beispiel sowie eine Anleitung zum Einrichten und Ausführen finden Sie im Repository mit Serverless-Beispielen.

Nutzen eines DynamoDB-Ereignisses mit Lambda unter Verwendung von Go.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
package main

import (
 "context"
 "github.com/aws/aws-lambda-go/lambda"
 "github.com/aws/aws-lambda-go/events"
```

```
 "fmt"
)

func HandleRequest(ctx context.Context, event events.DynamoDBEvent) (*string,
 error) {
 if len(event.Records) == 0 {
  return nil, fmt.Errorf("received empty event")
 }

 for _, record := range event.Records {
   LogDynamoDBRecord(record)
 }

 message := fmt.Sprintf("Records processed: %d", len(event.Records))
 return &message, nil
}

func main() {
 lambda.Start(HandleRequest)
}

func LogDynamoDBRecord(record events.DynamoDBEventRecord){
 fmt.Println(record.EventID)
 fmt.Println(record.EventName)
 fmt.Printf("%+v\n", record.Change)
}
```

Java

SDK für Java 2.x

> **Note**
>
> Es gibt noch mehr GitHub. Das vollständige Beispiel sowie eine Anleitung zum Einrichten und Ausführen finden Sie im Repository mit Serverless-Beispielen.

Nutzen eines DynamoDB-Ereignisses mit Lambda unter Verwendung von Java.

```
import com.amazonaws.services.lambda.runtime.Context;
import com.amazonaws.services.lambda.runtime.RequestHandler;
import com.amazonaws.services.lambda.runtime.events.DynamodbEvent;
```

```
import
 com.amazonaws.services.lambda.runtime.events.DynamodbEvent.DynamodbStreamRecord;
import com.google.gson.Gson;
import com.google.gson.GsonBuilder;

public class example implements RequestHandler<DynamodbEvent, Void> {

    private static final Gson GSON = new
 GsonBuilder().setPrettyPrinting().create();

    @Override
    public Void handleRequest(DynamodbEvent event, Context context) {
        System.out.println(GSON.toJson(event));
        event.getRecords().forEach(this::logDynamoDBRecord);
        return null;
    }

    private void logDynamoDBRecord(DynamodbStreamRecord record) {
        System.out.println(record.getEventID());
        System.out.println(record.getEventName());
        System.out.println("DynamoDB Record: " +
 GSON.toJson(record.getDynamodb()));
    }
}
```

JavaScript

SDK für JavaScript (v3)

> **Note**
>
> Es gibt noch mehr dazu GitHub. Das vollständige Beispiel sowie eine Anleitung zum Einrichten und Ausführen finden Sie im Repository mit Serverless-Beispielen.

Konsumieren eines DynamoDB-Ereignisses mit Lambda unter Verwendung. JavaScript

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
exports.handler = async (event, context) => {
    console.log(JSON.stringify(event, null, 2));
    event.Records.forEach(record => {
```

```
        logDynamoDBRecord(record);
    });
};

const logDynamoDBRecord = (record) => {
    console.log(record.eventID);
    console.log(record.eventName);
    console.log(`DynamoDB Record: ${JSON.stringify(record.dynamodb)}`);
};
```

Konsumieren eines DynamoDB-Ereignisses mit Lambda unter Verwendung. TypeScript

```
export const handler = async (event, context) => {
    console.log(JSON.stringify(event, null, 2));
    event.Records.forEach(record => {
        logDynamoDBRecord(record);
    });
}
const logDynamoDBRecord = (record) => {
    console.log(record.eventID);
    console.log(record.eventName);
    console.log(`DynamoDB Record: ${JSON.stringify(record.dynamodb)}`);
};
```

PHP

SDK für PHP

> **Note**
>
> Es gibt noch mehr dazu. GitHub Das vollständige Beispiel sowie eine Anleitung zum Einrichten und Ausführen finden Sie im Repository mit Serverless-Beispielen.

Nutzen eines DynamoDB-Ereignisses mit Lambda unter Verwendung von PHP.

```
<?php

# using bref/bref and bref/logger for simplicity
```

```
use Bref\Context\Context;
use Bref\Event\DynamoDb\DynamoDbEvent;
use Bref\Event\DynamoDb\DynamoDbHandler;
use Bref\Logger\StderrLogger;

require __DIR__ . '/vendor/autoload.php';

class Handler extends DynamoDbHandler
{
    private StderrLogger $logger;

    public function __construct(StderrLogger $logger)
    {
        $this->logger = $logger;
    }

    /**
     * @throws JsonException
     * @throws \Bref\Event\InvalidLambdaEvent
     */
    public function handleDynamoDb(DynamoDbEvent $event, Context $context): void
    {
        $this->logger->info("Processing DynamoDb table items");
        $records = $event->getRecords();

        foreach ($records as $record) {
            $eventName = $record->getEventName();
            $keys = $record->getKeys();
            $old = $record->getOldImage();
            $new = $record->getNewImage();

            $this->logger->info("Event Name:".$eventName."\n");
            $this->logger->info("Keys:". json_encode($keys)."\n");
            $this->logger->info("Old Image:". json_encode($old)."\n");
            $this->logger->info("New Image:". json_encode($new));

            // TODO: Do interesting work based on the new data

            // Any exception thrown will be logged and the invocation will be
 marked as failed
        }

        $totalRecords = count($records);
        $this->logger->info("Successfully processed $totalRecords items");
```

```
    }
}

$logger = new StderrLogger();
return new Handler($logger);
```

Python

SDK für Python (Boto3)

> **Note**
>
> Es gibt noch mehr GitHub. Das vollständige Beispiel sowie eine Anleitung zum Einrichten und Ausführen finden Sie im Repository mit Serverless-Beispielen.

Nutzen eines DynamoDB-Ereignisses mit Lambda unter Verwendung von Python.

```
import json

def lambda_handler(event, context):
    print(json.dumps(event, indent=2))

    for record in event['Records']:
        log_dynamodb_record(record)

def log_dynamodb_record(record):
    print(record['eventID'])
    print(record['eventName'])
    print(f"DynamoDB Record: {json.dumps(record['dynamodb'])}")
```

Ruby

SDK für Ruby

> **Note**
>
> Es gibt noch mehr GitHub. Das vollständige Beispiel sowie eine Anleitung zum Einrichten und Ausführen finden Sie im Repository mit Serverless-Beispielen.

Nutzen eines DynamoDB-Ereignisses mit Lambda unter Verwendung von Ruby.

```
def lambda_handler(event:, context:)
    return 'received empty event' if event['Records'].empty?

    event['Records'].each do |record|
      log_dynamodb_record(record)
    end

    "Records processed: #{event['Records'].length}"
  end

  def log_dynamodb_record(record)
    puts record['eventID']
    puts record['eventName']
    puts "DynamoDB Record: #{JSON.generate(record['dynamodb'])}"
  end
```

Rust

SDK für Rust

> **Note**
>
> Es gibt noch mehr dazu GitHub. Das vollständige Beispiel sowie eine Anleitung zum Einrichten und Ausführen finden Sie im Repository mit Serverless-Beispielen.

Nutzen eines DynamoDB-Ereignisses mit Lambda unter Verwendung von Rust.

```
use lambda_runtime::{service_fn, tracing, Error, LambdaEvent};
use aws_lambda_events::{
    event::dynamodb::{Event, EventRecord},
   };


// Built with the following dependencies:
//lambda_runtime = "0.11.1"
//serde_json = "1.0"
//tokio = { version = "1", features = ["macros"] }
//tracing = { version = "0.1", features = ["log"] }
//tracing-subscriber = { version = "0.3", default-features = false, features =
 ["fmt"] }
//aws_lambda_events = "0.15.0"

async fn function_handler(event: LambdaEvent<Event>) ->Result<(), Error> {

    let records = &event.payload.records;
    tracing::info!("event payload: {:?}",records);
    if records.is_empty() {
        tracing::info!("No records found. Exiting.");
        return Ok(());
    }

    for record in records{
        log_dynamo_dbrecord(record);
    }

    tracing::info!("Dynamo db records processed");

    // Prepare the response
    Ok(())

}

fn log_dynamo_dbrecord(record: &EventRecord)-> Result<(), Error>{
    tracing::info!("EventId: {}", record.event_id);
    tracing::info!("EventName: {}", record.event_name);
    tracing::info!("DynamoDB Record: {:?}", record.change );
    Ok(())
```

```
}

#[tokio::main]
async fn main() -> Result<(), Error> {
    tracing_subscriber::fmt()
    .with_max_level(tracing::Level::INFO)
    .with_target(false)
    .without_time()
    .init();

    let func = service_fn(function_handler);
    lambda_runtime::run(func).await?;
    Ok(())

}
```

Eine vollständige Liste der AWS SDK-Entwicklerhandbücher und Codebeispiele finden Sie unterDynamoDB mit einem SDK verwenden AWS. Dieses Thema enthält auch Informationen zu den ersten Schritten und Details zu früheren SDK-Versionen.

Melden von Batch-Elementfehlern für Lambda-Funktionen mit einem DynamoDB-Auslöser

Die folgenden Codebeispiele zeigen, wie eine teilweise Batch-Antwort für Lambda-Funktionen implementiert wird, die Ereignisse von einem DynamoDB-Stream empfangen. Die Funktion meldet die Batch-Elementfehler in der Antwort und signalisiert Lambda, diese Nachrichten später erneut zu versuchen.

.NET

SDK für .NET

> **Note**
>
> Es gibt noch mehr dazu GitHub. Das vollständige Beispiel sowie eine Anleitung zum Einrichten und Ausführen finden Sie im Repository mit Serverless-Beispielen.

Melden von DynamoDB-Batchelementfehlern mit Lambda unter Verwendung von .NET.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
using System.Text.Json;
using System.Text;
using Amazon.Lambda.Core;
using Amazon.Lambda.DynamoDBEvents;

// Assembly attribute to enable the Lambda function's JSON input to be converted
 into a .NET class.
[assembly:
 LambdaSerializer(typeof(Amazon.Lambda.Serialization.SystemTextJson.DefaultLambdaJsonSeri

namespace AWSLambda_DDB;

public class Function
{
    public StreamsEventResponse FunctionHandler(DynamoDBEvent dynamoEvent,
 ILambdaContext context)

    {
        context.Logger.LogInformation($"Beginning to process
 {dynamoEvent.Records.Count} records...");
        List<StreamsEventResponse.BatchItemFailure> batchItemFailures = new
 List<StreamsEventResponse.BatchItemFailure>();
        StreamsEventResponse streamsEventResponse = new StreamsEventResponse();

        foreach (var record in dynamoEvent.Records)
        {
            try
            {
                var sequenceNumber = record.Dynamodb.SequenceNumber;
                context.Logger.LogInformation(sequenceNumber);
            }
            catch (Exception ex)
            {
                context.Logger.LogError(ex.Message);
                batchItemFailures.Add(new StreamsEventResponse.BatchItemFailure()
 { ItemIdentifier = record.Dynamodb.SequenceNumber });
            }
        }

        if (batchItemFailures.Count > 0)
        {
```

```
            streamsEventResponse.BatchItemFailures = batchItemFailures;
        }

        context.Logger.LogInformation("Stream processing complete.");
        return streamsEventResponse;
    }
}
```

Go

SDK für Go V2

> **Note**
>
> Es gibt noch mehr GitHub. Das vollständige Beispiel sowie eine Anleitung zum Einrichten und Ausführen finden Sie im Repository mit Serverless-Beispielen.

Melden von DynamoDB-Batchelementfehlern mit Lambda unter Verwendung von Go.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
package main

import (
 "context"
 "github.com/aws/aws-lambda-go/events"
 "github.com/aws/aws-lambda-go/lambda"
)

type BatchItemFailure struct {
 ItemIdentifier string `json:"ItemIdentifier"`
}

type BatchResult struct {
 BatchItemFailures []BatchItemFailure `json:"BatchItemFailures"`
}

func HandleRequest(ctx context.Context, event events.DynamoDBEvent)
 (*BatchResult, error) {
 var batchItemFailures []BatchItemFailure
 curRecordSequenceNumber := ""
```

```
 for _, record := range event.Records {
  // Process your record
  curRecordSequenceNumber = record.Change.SequenceNumber
 }

 if curRecordSequenceNumber != "" {
  batchItemFailures = append(batchItemFailures, BatchItemFailure{ItemIdentifier:
 curRecordSequenceNumber})
 }

 batchResult := BatchResult{
  BatchItemFailures: batchItemFailures,
 }

 return &batchResult, nil
}

func main() {
 lambda.Start(HandleRequest)
}
```

Java

SDK für Java 2.x

> **Note**
>
> Es gibt noch mehr GitHub. Das vollständige Beispiel sowie eine Anleitung zum Einrichten und Ausführen finden Sie im Repository mit Serverless-Beispielen.

Melden von DynamoDB-Batchelementfehlern mit Lambda unter Verwendung von Java.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
import com.amazonaws.services.lambda.runtime.Context;
import com.amazonaws.services.lambda.runtime.RequestHandler;
import com.amazonaws.services.lambda.runtime.events.DynamodbEvent;
import com.amazonaws.services.lambda.runtime.events.StreamsEventResponse;
import com.amazonaws.services.lambda.runtime.events.models.dynamodb.StreamRecord;
```

```
import java.util.ArrayList;
import java.util.List;

public class ProcessDynamodbRecords implements RequestHandler<DynamodbEvent,
 StreamsEventResponse> {

    @Override
    public StreamsEventResponse handleRequest(DynamodbEvent input, Context
 context) {

        List<StreamsEventResponse.BatchItemFailure> batchItemFailures = new
 ArrayList<>();
        String curRecordSequenceNumber = "";

        for (DynamodbEvent.DynamodbStreamRecord dynamodbStreamRecord :
 input.getRecords()) {
          try {
                //Process your record
                StreamRecord dynamodbRecord = dynamodbStreamRecord.getDynamodb();
                curRecordSequenceNumber = dynamodbRecord.getSequenceNumber();

            } catch (Exception e) {
                /* Since we are working with streams, we can return the failed
 item immediately.
                   Lambda will immediately begin to retry processing from this
 failed item onwards. */
                batchItemFailures.add(new
 StreamsEventResponse.BatchItemFailure(curRecordSequenceNumber));
                return new StreamsEventResponse(batchItemFailures);
            }
        }

       return new StreamsEventResponse();
    }
}
```

JavaScript

SDK für JavaScript (v3)

> **Note**
>
> Es gibt noch mehr dazu GitHub. Das vollständige Beispiel sowie eine Anleitung zum Einrichten und Ausführen finden Sie im Repository mit Serverless-Beispielen.

Melden von DynamoDB-Batchelementfehlern mit Lambda unter Verwendung von. JavaScript

```
export const handler = async (event) => {
  const records = event.Records;
  let curRecordSequenceNumber = "";

  for (const record of records) {
    try {
      // Process your record
      curRecordSequenceNumber = record.dynamodb.SequenceNumber;
    } catch (e) {
      // Return failed record's sequence number
      return { batchItemFailures: [{ itemIdentifier:
 curRecordSequenceNumber }] };
    }
  }

  return { batchItemFailures: [] };
};
```

Melden von DynamoDB-Batchelementfehlern mit Lambda unter Verwendung von. TypeScript

```
import {
  DynamoDBBatchResponse,
  DynamoDBBatchItemFailure,
  DynamoDBStreamEvent,
} from "aws-lambda";

export const handler = async (
  event: DynamoDBStreamEvent
): Promise<DynamoDBBatchResponse> => {
```

```
  const batchItemFailures: DynamoDBBatchItemFailure[] = [];
  let curRecordSequenceNumber;

  for (const record of event.Records) {
    curRecordSequenceNumber = record.dynamodb?.SequenceNumber;

    if (curRecordSequenceNumber) {
      batchItemFailures.push({
        itemIdentifier: curRecordSequenceNumber,
      });
    }
  }

  return { batchItemFailures: batchItemFailures };
};
```

PHP

SDK für PHP

> **Note**
>
> Es gibt noch mehr dazu. GitHub Das vollständige Beispiel sowie eine Anleitung zum Einrichten und Ausführen finden Sie im Repository mit Serverless-Beispielen.

Melden von DynamoDB-Batchelementfehlern mit Lambda unter Verwendung von PHP.

```
<?php

# using bref/bref and bref/logger for simplicity

use Bref\Context\Context;
use Bref\Event\DynamoDb\DynamoDbEvent;
use Bref\Event\Handler as StdHandler;
use Bref\Logger\StderrLogger;

require __DIR__ . '/vendor/autoload.php';

class Handler implements StdHandler
{
```

```
    private StderrLogger $logger;
    public function __construct(StderrLogger $logger)
    {
        $this->logger = $logger;
    }

    /**
     * @throws JsonException
     * @throws \Bref\Event\InvalidLambdaEvent
     */
    public function handle(mixed $event, Context $context): array
    {
        $dynamoDbEvent = new DynamoDbEvent($event);
        $this->logger->info("Processing records");

        $records = $dynamoDbEvent->getRecords();
        $failedRecords = [];
        foreach ($records as $record) {
            try {
                $data = $record->getData();
                $this->logger->info(json_encode($data));
                // TODO: Do interesting work based on the new data
            } catch (Exception $e) {
                $this->logger->error($e->getMessage());
                // failed processing the record
                $failedRecords[] = $record->getSequenceNumber();
            }
        }
        $totalRecords = count($records);
        $this->logger->info("Successfully processed $totalRecords records");

        // change format for the response
        $failures = array_map(
            fn(string $sequenceNumber) => ['itemIdentifier' => $sequenceNumber],
            $failedRecords
        );

        return [
            'batchItemFailures' => $failures
        ];
    }
}

$logger = new StderrLogger();
```

```
return new Handler($logger);
```

Python

SDK für Python (Boto3)

> **Note**
>
> Es gibt noch mehr GitHub. Das vollständige Beispiel sowie eine Anleitung zum Einrichten und Ausführen finden Sie im Repository mit Serverless-Beispielen.

Melden von DynamoDB-Batchelementfehlern mit Lambda unter Verwendung von Python.

```
# Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
# SPDX-License-Identifier: Apache-2.0
def handler(event, context):
    records = event.get("Records")
    curRecordSequenceNumber = ""

    for record in records:
        try:
            # Process your record
            curRecordSequenceNumber = record["dynamodb"]["SequenceNumber"]
        except Exception as e:
            # Return failed record's sequence number
            return {"batchItemFailures":[{"itemIdentifier":
 curRecordSequenceNumber}]}

    return {"batchItemFailures":[]}
```

Ruby

SDK für Ruby

> **Note**
>
> Es gibt noch mehr GitHub. Das vollständige Beispiel sowie eine Anleitung zum Einrichten und Ausführen finden Sie im Repository mit Serverless-Beispielen.

Melden von DynamoDB-Batchelementfehlern mit Lambda unter Verwendung von Ruby.

```
def lambda_handler(event:, context:)
    records = event["Records"]
    cur_record_sequence_number = ""

    records.each do |record|
      begin
        # Process your record
        cur_record_sequence_number = record["dynamodb"]["SequenceNumber"]
      rescue StandardError => e
        # Return failed record's sequence number
        return {"batchItemFailures" => [{"itemIdentifier" =>
 cur_record_sequence_number}]}
      end
    end

    {"batchItemFailures" => []}
  end
```

Rust

SDK für Rust

> **Note**
>
> Es gibt noch mehr dazu GitHub. Das vollständige Beispiel sowie eine Anleitung zum Einrichten und Ausführen finden Sie im Repository mit Serverless-Beispielen.

Melden von DynamoDB-Batchelementfehlern mit Lambda unter Verwendung von Rust.

```
use aws_lambda_events::{
    event::dynamodb::{Event, EventRecord, StreamRecord},
    streams::{DynamoDbBatchItemFailure, DynamoDbEventResponse},
};
use lambda_runtime::{run, service_fn, Error, LambdaEvent};

/// Process the stream record
fn process_record(record: &EventRecord) -> Result<(), Error> {
    let stream_record: &StreamRecord = &record.change;

    // process your stream record here...
    tracing::info!("Data: {:?}", stream_record);

    Ok(())
}

/// Main Lambda handler here...
async fn function_handler(event: LambdaEvent<Event>) ->
 Result<DynamoDbEventResponse, Error> {
    let mut response = DynamoDbEventResponse {
        batch_item_failures: vec![],
    };

    let records = &event.payload.records;

    if records.is_empty() {
        tracing::info!("No records found. Exiting.");
        return Ok(response);
    }

    for record in records {
        tracing::info!("EventId: {}", record.event_id);

        // Couldn't find a sequence number
        if record.change.sequence_number.is_none() {
            response.batch_item_failures.push(DynamoDbBatchItemFailure {
                item_identifier: Some("".to_string()),
            });
            return Ok(response);
        }

        // Process your record here...
```

```
        if process_record(record).is_err() {
            response.batch_item_failures.push(DynamoDbBatchItemFailure {
                item_identifier: record.change.sequence_number.clone(),
            });
            /* Since we are working with streams, we can return the failed item
 immediately.
            Lambda will immediately begin to retry processing from this failed
 item onwards. */
            return Ok(response);
        }
    }

    tracing::info!("Successfully processed {} record(s)", records.len());

    Ok(response)
}

#[tokio::main]
async fn main() -> Result<(), Error> {
    tracing_subscriber::fmt()
        .with_max_level(tracing::Level::INFO)
        // disable printing the name of the module in every log line.
        .with_target(false)
        // disabling time is handy because CloudWatch will add the ingestion
 time.
        .without_time()
        .init();

    run(service_fn(function_handler)).await
}
```

Eine vollständige Liste der AWS SDK-Entwicklerhandbücher und Codebeispiele finden Sie unterDynamoDB mit einem SDK verwenden AWS. Dieses Thema enthält auch Informationen zu den ersten Schritten und Details zu früheren SDK-Versionen.

AWS Community-Beiträge für DynamoDB

AWS Community-Beiträge sind Beispiele, die von mehreren Teams erstellt wurden und von ihnen verwaltet werden AWS. Um Feedback zu geben, verwenden Sie den Mechanismus, der in den verlinkten Repositorys bereitgestellt wird.

Beispiele

- Erstellen und Testen einer Serverless-Anwendung

Erstellen und Testen einer Serverless-Anwendung

Die folgenden Codebeispiele zeigen, wie eine Serverless-Anwendung mithilfe von API Gateway mit Lambda und DynamoDB erstellt und getestet wird.

.NET

SDK für .NET

Es wird gezeigt, wie eine Serverless-Anwendung, bestehend aus einem API Gateway mit Lambda und DynamoDB, mithilfe des .NET SDK erstellt und getestet wird.

Den vollständigen Quellcode und Anweisungen zur Einrichtung und Ausführung finden Sie im vollständigen Beispiel unter GitHub.

In diesem Beispiel verwendete Dienste

- API Gateway
- DynamoDB
- Lambda

Go

SDK für Go V2

Es wird gezeigt, wie eine Serverless-Anwendung, bestehend aus einem API Gateway mit Lambda und DynamoDB, mithilfe des Go SDK erstellt und getestet wird.

Den vollständigen Quellcode und Anweisungen zur Einrichtung und Ausführung finden Sie im vollständigen Beispiel unter GitHub.

In diesem Beispiel verwendete Dienste

- API Gateway
- DynamoDB
- Lambda

Java

SDK für Java 2.x

Es wird gezeigt, wie eine Serverless-Anwendung, bestehend aus einem API Gateway mit Lambda und DynamoDB, mithilfe des Java SDK erstellt und getestet wird.

Den vollständigen Quellcode und Anweisungen zur Einrichtung und Ausführung finden Sie im vollständigen Beispiel unter GitHub.

In diesem Beispiel verwendete Dienste

- API Gateway
- DynamoDB
- Lambda

Rust

SDK für Rust

Es wird gezeigt, wie eine Serverless-Anwendung, bestehend aus einem API Gateway mit Lambda und DynamoDB, mithilfe des Rust SDK erstellt und getestet wird.

Den vollständigen Quellcode und Anweisungen zur Einrichtung und Ausführung finden Sie im vollständigen Beispiel unter GitHub.

In diesem Beispiel verwendete Dienste

- API Gateway
- DynamoDB
- Lambda

Eine vollständige Liste der AWS SDK-Entwicklerhandbücher und Codebeispiele finden Sie unterDynamoDB mit einem SDK verwenden AWS. Dieses Thema enthält auch Informationen zu den ersten Schritten und Details zu früheren SDK-Versionen.

Sicherheit und Compliance in Amazon DynamoDB

Cloud-Sicherheit hat höchste AWS Priorität. Als AWS Kunde profitieren Sie von einer Rechenzentrums- und Netzwerkarchitektur, die darauf ausgelegt sind, die Anforderungen der sicherheitssensibelsten Unternehmen zu erfüllen.

Sicherheit ist eine gemeinsame Verantwortung von Ihnen AWS und Ihnen. Das Modell der geteilten Verantwortung beschreibt dies als Sicherheit der Cloud und Sicherheit in der Cloud:

- Sicherheit der Cloud — AWS ist verantwortlich für den Schutz der Infrastruktur, die AWS Dienste in der AWS Cloud ausführt. AWS bietet Ihnen auch Dienste, die Sie sicher nutzen können. Die Wirksamkeit unserer Sicherheitsfunktionen wird regelmäßig von externen Prüfern im Rahmen des AWS -Compliance-Programms getestet und überprüft. Weitere Informationen zu den Compliance-Programmen für DynamoDB finden Sie unter Compliance-Programm für abgedeckte AWS -Services.
- Sicherheit in der Cloud — Ihre Verantwortung richtet sich nach dem AWS Dienst, den Sie nutzen. In Ihre Verantwortung fallen außerdem weitere Faktoren, wie z. B. die Vertraulichkeit der Daten, die Anforderungen Ihrer Organisation sowie geltende Gesetze und Vorschriften.

Diese Dokumentation beschreibt, wie Sie das Modell der übergreifenden Verantwortlichkeit bei der Verwendung von DynamoDB anwenden können. Die folgenden Themen veranschaulichen, wie Sie DynamoDB zur Erfüllung Ihrer Sicherheits- und Compliance-Ziele konfigurieren können. Sie erfahren auch, wie Sie andere AWS Dienste verwenden können, die Ihnen bei der Überwachung und Sicherung Ihrer DynamoDB-Ressourcen helfen können.

Topics

- AWS verwaltete Richtlinien für Amazon DynamoDB
- Verwenden ressourcenbasierter Richtlinien für DynamoDB
- Verwenden der attributbasierten Zugriffskontrolle mit DynamoDB
- Datenschutz in DynamoDB
- AWS Identity and Access Management (IAM) und DynamoDB
- Compliance-Validierung nach Branche für DynamoDB
- Ausfallsicherheit und Notfallwiederherstellung in Amazon DynamoDB
- Infrastruktursicherheit in Amazon DynamoDB

- AWS PrivateLink für DynamoDB
- Konfigurations- und Schwachstellenanalyse in Amazon DynamoDB
- Bewährte Methoden für die Sicherheit für Amazon DynamoDB

AWS verwaltete Richtlinien für Amazon DynamoDB

DynamoDB verwendet AWS verwaltete Richtlinien, um eine Reihe von Berechtigungen zu definieren, die der Dienst benötigt, um bestimmte Aktionen auszuführen. DynamoDB verwaltet und aktualisiert seine AWS verwalteten Richtlinien. Sie können die Berechtigungen in AWS verwalteten Richtlinien nicht ändern. Weitere Informationen zu AWS verwalteten Richtlinien finden Sie im IAM-Benutzerhandbuch unter AWS Verwaltete Richtlinien.

DynamoDB kann gelegentlich zusätzliche Berechtigungen zu einer AWS verwalteten Richtlinie hinzufügen, um neue Funktionen zu unterstützen. Diese Art von Update betrifft alle Identitäten (Benutzer, Gruppen und Rollen), an welche die Richtlinie angehängt ist. Eine AWS verwaltete Richtlinie wird höchstwahrscheinlich aktualisiert, wenn eine neue Funktion eingeführt wird oder wenn neue Operationen verfügbar werden. DynamoDB entfernt keine Berechtigungen aus einer AWS verwalteten Richtlinie, sodass durch Richtlinienaktualisierungen Ihre bestehenden Berechtigungen nicht beeinträchtigt werden. Eine vollständige Liste der AWS verwalteten Richtlinien finden Sie unter AWS Verwaltete Richtlinien.

AWS verwaltete Richtlinie: Dynamo DBReplication ServiceRolePolicy

Sie können die `DynamoDBReplicationServiceRolePolicy`-Richtlinie Ihren IAM-Entitäten nicht anfügen. Diese Richtlinie ist an eine serviceverknüpfte Rolle angehängt, die DynamoDB die Durchführung von Aktionen in Ihrem Namen ermöglicht. Weitere Informationen finden Sie unter Verwenden von IAM mit globalen Tabellen.

Diese Richtlinie gewährt Berechtigungen, die es der serviceverknüpften Rolle ermöglichen, eine Datenreplikation zwischen Replikaten globaler Tabellen durchzuführen. Sie gewährt auch Berechtigungen zur Verwaltung von Replikaten globaler Tabellen in Ihrem Namen.

Details zu Berechtigungen

Diese Richtlinie gewährt Berechtigungen für folgende Aktionen:

- `dynamodb` – Ausführung einer Datenreplikation und Verwaltung von Tabellenreplikaten

- `application-autoscaling` – Abruf und Verwaltung von Auto-Scaling-Einstellungen für Tabellen
- `account` – Abruf des Regionsstatus, um die Zugänglichkeit von Replikaten zu bewerten
- `iam` – Erstellung der serviceverknüpften Rolle für das Auto Scaling für Anwendungen, falls die betreffende serviceverknüpfte Rolle nicht bereits vorhanden ist

Die Definition dieser verwalteten Richtlinie finden Sie hier.

AWS verwaltete Richtlinie: Access_v2 AmazonDynamo DBFull

Die eingeschränkte `AmazonDynamoDBFullAccess_v2`-Richtlinie gewährt Benutzern bestimmte Zugriffsberechtigungen. Sie können die `AmazonDynamoDBFullAccess_v2`-Richtlinie an Ihre IAM-Identitäten anfügen. Diese Richtlinie gewährt Administratorzugriff auf Amazon DynamoDB DynamoDB-Ressourcen und gewährt einer IAM-Identität (z. B. einem Benutzer, einer Gruppe oder einer Rolle) Zugriff auf die, in die DynamoDB integriert ist AWS-Services , um alle DynamoDB-Funktionen nutzen zu können. Die Verwendung dieser Richtlinie ermöglicht den Zugriff auf alle DynamoDB-Funktionen, die in der AWS-Managementkonsole verfügbar sind.

Details zu Berechtigungen

Diese Richtlinie umfasst die folgenden Berechtigungen:

- `Amazon DynamoDB`
- `DynamoDB Accelerator`
- `AWS KMS`
- `AWS -Ressourcengruppen Tagging`
- `Lambda`
- `Application Auto Scaling`
- `CloudWatch`
- `Amazon Kinesis`
- `Amazon EC2`
- `IAM`

Informationen zur Formatierung der Richtlinie finden Sie unter Access_v2. `JSON` AmazonDynamo DBFull

AWS verwaltete Richtlinie: AmazonDynamo DBRead OnlyAccess

Sie können die `AmazonDynamoDBReadOnlyAccess`-Richtlinie an Ihre IAM-Identitäten anfügen.

Diese Richtlinie gewährt schreibgeschützten Zugriff auf Amazon DynamoDB.

Details zu Berechtigungen

Diese Richtlinie umfasst die folgenden Berechtigungen:

- `Amazon DynamoDB` – Bietet schreibgeschützten Zugriff auf Amazon DynamoDB.
- `Amazon DynamoDB Accelerator (DAX)` – Bietet schreibgeschützten Zugriff auf Amazon DynamoDB Accelerator (DAX).
- `Application Auto Scaling` – Ermöglicht Prinzipalen, Konfigurationen Application Auto Scaling anzuzeigen. Dies ist erforderlich, damit Benutzer Richtlinien für die automatische Skalierung anzeigen können, die an eine Tabelle angehängt sind.
- `CloudWatch`— Ermöglicht Prinzipalen das Anzeigen von Metrikdaten und Alarmen, die in CloudWatch konfiguriert sind. Dies ist erforderlich, damit Benutzer die Größe der Fakturierungstabelle und die CloudWatch Alarme, die für eine Tabelle konfiguriert wurden, einsehen können.
- `AWS Data Pipeline`— Ermöglicht Prinzipalen das Anzeigen AWS Data Pipeline und Zuordnen von Objekten.
- `Amazon EC2`— Ermöglicht Prinzipalen das Anzeigen von Amazon EC2 VPCs, Subnetzen und Sicherheitsgruppen.
- `IAM` – Ermöglicht es Prinzipalen, IAM-Rollen anzuzeigen.
- `AWS KMS`— Ermöglicht Prinzipalen das Anzeigen von Schlüsseln, die in konfiguriert sind. AWS KMS Dies ist erforderlich, damit Benutzer sehen können AWS KMS keys , was sie in ihrem Konto erstellt und verwaltet haben.
- `Amazon SNS` – Ermöglicht es Prinzipalen, Amazon-SNS-Themen und -Abonnements nach Thema aufzulisten.
- `AWS -Ressourcengruppen` – Ermöglicht es Prinzipalen, Ressourcengruppen und ihre Abfragen anzuzeigen.
- `AWS -Ressourcengruppen Tagging` – Ermöglicht es Prinzipalen, alle markierten oder zuvor markierten Ressourcen in einer Region aufzulisten.
- `Kinesis` – Ermöglicht es Prinzipalen, Beschreibungen für Kinesis-Datenströme anzuzeigen.

- `Amazon CloudWatch Contributor Insights` – Ermöglicht es Prinzipale, Zeitreihendaten anzuzeigen, die von Contributor-Insights-Regeln erfasst wurden.

Informationen zum `JSON` Format der Richtlinie finden Sie unter AmazonDynamoDBReadOnlyAccess.

DynamoDB-Updates für verwaltete Richtlinien AWS

Diese Tabelle zeigt Aktualisierungen der AWS Zugriffsverwaltungsrichtlinien für DynamoDB.

Änderungen	Beschreibung	Änderungsdatum
`AmazonDynamoDBFullAccess` – als veraltet gekennzeichnet	Diese Richtlinie wurde durch eine eingeschränkte Richtlinie namens `AmazonDynamoDBFullAccess_v2` ersetzt. Seit April 2025 können Sie die `AmazonDynamoDBFullAccess` - Richtlinie keinen neuen Benutzern, Gruppen oder Rollen mehr anfügen. Weitere Informationen finden Sie unter AWS verwaltete Richtlinie: Access_v2 AmazonDynamo DBFull.	28. April 2025
`AmazonDynamoDBReadOnlyAccess` - Aktualisierung einer bestehenden Richtlinie	`AmazonDynamoDBReadOnlyAccess` – Die Berechtigungen `dynamodb:GetAbacStatus` und `dynamodb:UpdateAbacStatus` wurden hinzugefügt. Mit diesen Berechtigungen können Sie den ABAC-Status anzeigen und ABAC für Sie AWS-Konto in der aktuellen Region aktivieren.	18. November 2024
`AmazonDynamoDBReadOnlyAccess` - Aktualisierung	`AmazonDynamoDBReadOnlyAccess` hat die Berechtigung `dynamodb:GetResourcePolicy` hinzugefügt. Diese	20. März 2024

Änderungen	Beschreibung	Änderungsdatum
einer bestehend en Richtlinie	Berechtigung ermöglicht den Zugriff auf Richtlinien, die auf Leseresso urcen basieren und die DynamoDB-Ressourcen zugeordnet sind.	
`DynamoDBR eplicatio nServiceR olePolicy` - Aktualisierung einer bestehend en Richtlinie	`DynamoDBReplicatio nServiceRolePolicy` hat die Berechtigung `dynamodb: GetResourcePolicy` hinzugefü gt. Diese Berechtigung ermöglich t es der serviceverknüpften Rolle, ressourcenbasierte Richtlinien zu lesen, die DynamoDB-Ressourcen zugeordnet sind.	15. Dezember 2023
`DynamoDBR eplicatio nServiceR olePolicy` - Aktualisierung einer bestehend en Richtlinie	`DynamoDBReplicatio nServiceRolePolicy` hat die Berechtigung `account:L istRegions` hinzugefügt. Diese Berechtigung ermöglicht es der serviceverknüpften Rolle, die Zugänglichkeit von Replikaten zu bewerten.	10. Mai 2023
`DynamoDBR eplicatio nServiceR olePolicy` zur Liste der verwaltet en Richtlinien hinzugefügt	Es wurden Informationen über die verwaltete Richtlinie `DynamoDBR eplicationServiceR olePolicy` hinzugefügt, die von der serviceverknüpften Rolle der globalen DynamoDB-Tabellen verwendet wird.	10. Mai 2023

Änderungen	Beschreibung	Änderungsdatum
Globale DynamoDB-Tabellen haben mit der Verfolgung von Änderungen begonnen	DynamoDB Global Tables begann, Änderungen für seine AWS verwalteten Richtlinien nachzuverfolgen.	10. Mai 2023

Verwenden ressourcenbasierter Richtlinien für DynamoDB

DynamoDB unterstützt ressourcenbasierte Richtlinien für Tabellen, Indizes und Streams. Mit ressourcenbasierten Richtlinien können Sie Zugriffsberechtigungen definieren, indem Sie angeben, wer Zugriff auf jede Ressource hat und welche Aktionen für jede Ressource ausgeführt werden dürfen.

Sie können eine ressourcenbasierte Richtlinie an DynamoDB-Ressourcen wie eine Tabelle oder einen Stream anfügen. In dieser Richtlinie geben Sie Berechtigungen für Identity and Access Management (IAM)-Prinzipale an, die bestimmte Aktionen an diesen DynamoDB-Ressourcen ausführen können. Beispielsweise enthält die Richtlinie, die Sie an eine Tabelle anfügen, Berechtigungen für die Tabelle und ihre Indizes. Daher können ressourcenbasierte Richtlinien zur Vereinfachung der Zugriffskontrolle für Ihre DynamoDB-Tabellen, -Indizes und -Streams beitragen, indem Sie Berechtigungen auf Ressourcenebene definieren. Eine Richtlinie, die Sie an eine DynamoDB-Ressource anhängen können, darf maximal 20 KB groß sein.

Ein wesentlicher Vorteil der Verwendung ressourcenbasierter Richtlinien ist die Vereinfachung der kontoübergreifenden Zugriffskontrolle zur Bereitstellung eines kontoübergreifenden Zugriffs auf IAM-Prinzipale in verschiedenen AWS-Konten. Weitere Informationen finden Sie unter Ressourcenbasierte Richtlinie für kontoübergreifenden Zugriff.

Ressourcenbasierte Richtlinien unterstützen außerdem Integrationen in den IAM Access Analyzer-Analysator für externen Zugriff und Block Public Access (BPA)-Funktionen. IAM Access Analyzer meldet den kontoübergreifenden Zugriff auf externe Entitäten, die in ressourcenbasierten Richtlinien angegeben sind. Das Tool bietet außerdem Transparenz, sodass Sie Ihre Berechtigungen verfeinern und dem Prinzip der geringsten Berechtigung entsprechen können. BPA hilft Ihnen, den öffentlichen Zugriff auf Ihre DynamoDB-Tabellen, -Indizes und -Streams zu verhindern, und wird in den Workflows zur Erstellung und Änderung ressourcenbasierter Richtlinien automatisch aktiviert.

Themen

- Erstellen Sie eine Tabelle mit einer ressourcenbasierten Richtlinie
- Anhängen einer Richtlinie an eine vorhandene DynamoDB-Tabelle
- Anhängen einer ressourcenbasierten Richtlinie an einen DynamoDB-Stream
- Entfernen einer ressourcenbasierten Richtlinie aus einer DynamoDB-Tabelle
- Kontoübergreifender Zugriff mit ressourcenbasierten Richtlinien in DynamoDB
- Blockieren des öffentlichen Zugriffs mit ressourcenbasierten Richtlinien in DynamoDB
- DynamoDB API-Operationen, die von ressourcenbasierten Richtlinien unterstützt werden
- Autorisierung mit identitätsbasierten IAM-Richtlinien und ressourcenbasierten DynamoDB-Richtlinien
- Beispiele für ressourcenbasierte DynamoDB-Richtlinien
- Überlegungen zu ressourcenbasierten DynamoDB-Richtlinien
- Bewährte Methoden für ressourcenbasierte DynamoDB-Richtlinien

Erstellen Sie eine Tabelle mit einer ressourcenbasierten Richtlinie

Sie können beim Erstellen einer Tabelle mithilfe der DynamoDB-Konsole, der CreateTableAPI, des AWS SDK oder einer Vorlage eine ressourcenbasierte Richtlinie hinzufügen. AWS CLI CloudFormation

AWS CLI

Im folgenden Beispiel wird eine Tabelle erstellt, die mit dem Befehl benannt *`MusicCollection`* wird. `create-table` AWS CLI Dieser Befehl enthält auch den Parameter `resource-policy`, mit dem der Tabelle eine ressourcenbasierte Richtlinie hinzugefügt wird. Diese Richtlinie ermöglicht es *`John`* dem Benutzer RestoreTableToPointInTime, die PutItemAPI-Aktionen GetItem, und für die Tabelle auszuführen.

Denken Sie daran, den *`italicized`* Text durch Ihre ressourcenspezifischen Informationen zu ersetzen.

```
aws dynamodb create-table \
    --table-name MusicCollection \
    --attribute-definitions AttributeName=Artist,AttributeType=S
 AttributeName=SongTitle,AttributeType=S \
```

```
    --key-schema AttributeName=Artist,KeyType=HASH
 AttributeName=SongTitle,KeyType=RANGE \
    --provisioned-throughput ReadCapacityUnits=5,WriteCapacityUnits=5 \
    --resource-policy \
        "{
            \"Version\": \"2012-10-17\",
            \"Statement\": [
              {
                    \"Effect\": \"Allow\",
                    \"Principal\": {
                        \"AWS\": \"arn:aws:iam::123456789012:user/John\"
                    },
                    \"Action\": [
                        \"dynamodb:RestoreTableToPointInTime\",
                        \"dynamodb:GetItem\",
                        \"dynamodb:DescribeTable\"
                    ],
                    \"Resource\": \"arn:aws:dynamodb:us-
west-2:123456789012:table/MusicCollection\"
                }
            ]
        }"
```

AWS-Managementkonsole

1. Melden Sie sich bei der an AWS-Managementkonsole und öffnen Sie die DynamoDB-Konsole unter. https://console.aws.amazon.com/dynamodb/
2. Wählen Sie auf dem Dashboard Tabelle erstellen aus.
3. Geben Sie im Feld Tabellendetails den Tabellennamen, den Partitionsschlüssel und den Sortierschlüssel ein.
4. Wählen Sie unter Tabelleneinstellungen die Option Einstellungen anpassen aus.
5. (Optional) Geben Sie Ihre Optionen für Tabellenklasse, Kapazitätsrechner, Lese-/Schreibkapazitätseinstellungen, Sekundäre Indizes, Verschlüsselung im Ruhezustand und Löschschutz an.
6. Fügen Sie unter Ressourcenbasierte Richtlinie eine Richtlinie hinzu, um die Zugriffsberechtigungen für die Tabelle und ihre Indizes zu definieren. In dieser Richtlinie geben Sie an, wer Zugriff auf jede Ressource hat und welche Aktionen für jede Ressource ausgeführt werden dürfen. Um eine Richtlinie hinzuzufügen, gehen Sie wie folgt vor:

- Geben oder fügen Sie ein JSON-Richtliniendokument ein. Einzelheiten zur Sprache der IAM-Richtlinien finden Sie unter Erstellen von Richtlinien mit dem JSON-Editor im IAM-Benutzerhandbuch.

> **Tip**
>
> Beispiele für ressourcenbasierte Richtlinien finden Sie unter Beispiele für Richtlinien im Entwicklerhandbuch für Amazon DynamoDB.

- Wählen Sie Neue Anweisung hinzufügen aus, um eine neue Anweisung hinzuzufügen, und geben Sie die Informationen in die bereitgestellten Felder ein. Wiederholen Sie diesen Vorgang für so viele Anweisungen, wie Sie hinzufügen möchten.

> **Important**
>
> Beheben Sie mögliche Sicherheitswarnungen, Fehler und Vorschläge, bevor Sie die Richtlinie speichern.

Das folgende Beispiel für eine IAM-Richtlinie ermöglicht es dem Benutzer*John*, die PutItemAPI-Aktionen RestoreTableToPointInTimeGetItem, und für die Tabelle auszuführen. *MusicCollection*

Denken Sie daran, den *italicized* Text durch Ihre ressourcenspezifischen Informationen zu ersetzen.

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "AWS": "arn:aws:iam::123456789012:user/username"
      },
      "Action": [
        "dynamodb:RestoreTableToPointInTime",
```

```
        "dynamodb:GetItem",
        "dynamodb:PutItem"
      ],
      "Resource": "arn:aws:dynamodb:us-
east-1:123456789012:table/MusicCollection"
    }
  ]
}
```

7. (Optional) Wählen Sie unten rechts Preview external access (Vorschau des externen Zugriffs) aus, um eine Vorschau anzuzeigen, wie sich Ihre neue Richtlinie auf den öffentlichen und kontoübergreifenden Zugriff auf Ihre Ressource auswirkt. Bevor Sie Ihre Richtlinie speichern, können Sie überprüfen, ob sie neue IAM-Access-Analyzer-Ergebnisse einführt oder vorhandene Ergebnisse löst. Wenn Sie keinen aktiven Analyzer sehen, wählen Sie Go to Access Analyzer (Zu Access Analyzer wechseln) aus, um einen Account Analyzer in IAM Access Analyzer zu erstellen. Weitere Informationen finden Sie unter Vorschau des Zugriffs.
8. Wählen Sie Create table (Tabelle erstellen) aus.

AWS CloudFormation Vorlage

Using the AWS::DynamoDB::Table resource

Die folgende CloudFormation Vorlage erstellt mithilfe der Ressource AWS: :DynamoDB: :Table eine Tabelle mit einem Stream. Diese Vorlage enthält außerdem ressourcenbasierte Richtlinien, die sowohl der Tabelle als auch dem Stream angehängt sind.

```
{
    "AWSTemplateFormatVersion": "2010-09-09",
    "Resources": {
        "MusicCollectionTable": {
            "Type": "AWS::DynamoDB::Table",
            "Properties": {
                "AttributeDefinitions": [
                    {
                        "AttributeName": "Artist",
                        "AttributeType": "S"
                    }
                ],
                "KeySchema": [
                    {
                        "AttributeName": "Artist",
```

```
                "KeyType": "HASH"
            }
        ],
        "BillingMode": "PROVISIONED",
        "ProvisionedThroughput": {
            "ReadCapacityUnits": 5,
            "WriteCapacityUnits": 5
        },
        "StreamSpecification": {
          "StreamViewType": "OLD_IMAGE",
          "ResourcePolicy": {
            "PolicyDocument": {
              "Version": "2012-10-17",
              "Statement": [
                {
                    "Principal": {
                        "AWS": "arn:aws:iam::111122223333:user/John"
                    },
                    "Effect": "Allow",
                    "Action": [
                        "dynamodb:GetRecords",
                        "dynamodb:GetShardIterator",
                        "dynamodb:DescribeStream"
                    ],
                    "Resource": "*"
                }
              ]
            }
          }
        },
        "TableName": "MusicCollection",
        "ResourcePolicy": {
            "PolicyDocument": {
                "Version": "2012-10-17",
                "Statement": [
                    {
                        "Principal": {
                            "AWS": [
                                "arn:aws:iam::111122223333:user/John"
                            ]
                        },
                        "Effect": "Allow",
                        "Action": "dynamodb:GetItem",
                        "Resource": "*"
```

```
                            }
                        ]
                    }
                }
            }

        }
    }
}
```

Using the AWS::DynamoDB::GlobalTable resource

Die folgende CloudFormation Vorlage erstellt eine Tabelle mit der AWS: :DynamoDB:: GlobalTable -Ressource und fügt der Tabelle und ihrem Stream eine ressourcenbasierte Richtlinie hinzu.

```
{
    "AWSTemplateFormatVersion": "2010-09-09",
    "Resources": {
        "GlobalMusicCollection": {
            "Type": "AWS::DynamoDB::GlobalTable",
            "Properties": {
                "TableName": "MusicCollection",
                "AttributeDefinitions": [{
                    "AttributeName": "Artist",
                    "AttributeType": "S"
                }],
                "KeySchema": [{
                    "AttributeName": "Artist",
                    "KeyType": "HASH"
                }],
                "BillingMode": "PAY_PER_REQUEST",
                "StreamSpecification": {
                    "StreamViewType": "NEW_AND_OLD_IMAGES"
                },
                "Replicas": [
                    {
                        "Region": "us-east-1",
                        "ResourcePolicy": {
                            "PolicyDocument": {
                                "Version": "2012-10-17",
                                "Statement": [{
                                    "Principal": {
```

```
                                        "AWS": [
                                            "arn:aws:iam::111122223333:user/John"
                                        ]
                                    },
                                    "Effect": "Allow",
                                    "Action": "dynamodb:GetItem",
                                    "Resource": "*"
                                }]
                            }
                        },
                        "ReplicaStreamSpecification": {
                            "ResourcePolicy": {
                                "PolicyDocument": {
                                    "Version": "2012-10-17",
                                    "Statement": [{
                                        "Principal": {
                                            "AWS":
 "arn:aws:iam::111122223333:user/John"
                                        },
                                        "Effect": "Allow",
                                        "Action": [
                                            "dynamodb:GetRecords",
                                            "dynamodb:GetShardIterator",
                                            "dynamodb:DescribeStream"
                                        ],
                                        "Resource": "*"
                                    }]
                                }
                            }
                        }
                    }
                ]
            }
        }
    }
}
```

Anhängen einer Richtlinie an eine vorhandene DynamoDB-Tabelle

Sie können eine ressourcenbasierte Richtlinie an eine bestehende Tabelle anhängen oder eine vorhandene Richtlinie ändern, indem Sie die DynamoDB-Konsole, die PutResourcePolicyAPI AWS CLI, das AWS SDK oder eine Vorlage verwenden.CloudFormation

AWS CLI Beispiel zum Anhängen einer neuen Richtlinie

Im folgenden Beispiel für eine IAM-Richtlinie wird der `put-resource-policy` AWS CLI Befehl verwendet, um eine ressourcenbasierte Richtlinie an eine vorhandene Tabelle anzuhängen. In diesem Beispiel kann der Benutzer *John* die UpdateTableAPI-Aktionen GetItem, PutItemUpdateItem, und für eine vorhandene Tabelle mit dem Namen ausführen. *MusicCollection*

Denken Sie daran, den *italicized* Text durch Ihre ressourcenspezifischen Informationen zu ersetzen.

```
aws dynamodb put-resource-policy \
    --resource-arn arn:aws:dynamodb:us-west-2:123456789012:table/MusicCollection \
    --policy \
        "{
            \"Version\": \"2012-10-17\",
            \"Statement\": [
              {
                    \"Effect\": \"Allow\",
                    \"Principal\": {
                        \"AWS\": \"arn:aws:iam::111122223333:user/John\"
                    },
                    \"Action\": [
                        \"dynamodb:GetItem\",
                        \"dynamodb:PutItem\",
                        \"dynamodb:UpdateItem\",
                        \"dynamodb:UpdateTable\"
                    ],
                    \"Resource\": \"arn:aws:dynamodb:us-
west-2:123456789012:table/MusicCollection\"
                }
            ]
        }"
```

AWS CLI Beispiel für die bedingte Aktualisierung einer bestehenden Richtlinie

Um eine vorhandene ressourcenbasierte Richtlinie einer Tabelle bedingt zu aktualisieren, können Sie den optionalen Parameter `expected-revision-id` verwenden. Im folgenden Beispiel wird die Richtlinie nur aktualisiert, wenn sie in DynamoDB vorhanden ist und ihre aktuelle Revisions-ID mit dem angegebenen `expected-revision-id`-Parameter übereinstimmt.

```
aws dynamodb put-resource-policy \
    --resource-arn arn:aws:dynamodb:us-west-2:123456789012:table/MusicCollection \
```

```
    --expected-revision-id 1709841168699 \
    --policy \
        "{
            \"Version\": \"2012-10-17\",
            \"Statement\": [
              {
                    \"Effect\": \"Allow\",
                    \"Principal\": {
                        \"AWS\": \"arn:aws:iam::111122223333:user/John\"
                    },
                    \"Action\": [
                        \"dynamodb:GetItem\",
                        \"dynamodb:UpdateItem\",
                        \"dynamodb:UpdateTable\"
                    ],
                    \"Resource\": \"arn:aws:dynamodb:us-
west-2:123456789012:table/MusicCollection\"
                }
            ]
        }"
```

AWS-Managementkonsole

1. Melden Sie sich bei der an AWS-Managementkonsole und öffnen Sie die DynamoDB-Konsole unter. https://console.aws.amazon.com/dynamodb/
2. Wählen Sie im Dashboard eine vorhandene Tabelle aus.
3. Navigieren Sie zur Registerkarte Berechtigungen und wählen Sie Tabellenrichtlinie erstellen aus.
4. Fügen Sie im Editor für ressourcenbasierte Richtlinien die Richtlinie hinzu, die Sie anhängen möchten, und wählen Sie Richtlinie erstellen aus.

   Das folgende Beispiel für eine IAM-Richtlinie ermöglicht es dem Benutzer *John* GetItem, die UpdateTableAPI-Aktionen, PutItemUpdateItem, und für eine bestehende Tabelle mit dem Namen auszuführen. *MusicCollection*

   Denken Sie daran, den *italicized* Text durch Ihre ressourcenspezifischen Informationen zu ersetzen.

   JSON

   ```
   {
   ```

```
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "AWS": "arn:aws:iam::111122223333:user/username"
      },
      "Action": [
        "dynamodb:GetItem",
        "dynamodb:PutItem",
        "dynamodb:UpdateItem",
        "dynamodb:UpdateTable"
      ],
      "Resource": "arn:aws:dynamodb:us-
east-1:123456789012:table/MusicCollection"
    }
  ]
}
```

AWS SDK for Java 2.x

Im folgenden IAM-Richtlinienbeispiel wird die Methode `putResourcePolicy` verwendet, um eine ressourcenbasierte Richtlinie an eine vorhandene Tabelle anzuhängen. Diese Richtlinie ermöglicht es einem Benutzer, die GetItemAPI-Aktion für eine vorhandene Tabelle auszuführen.

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.model.PutResourcePolicyRequest;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * Get started with the AWS SDK for Java 2.x
 */
public class PutResourcePolicy {

    public static void main(String[] args) {
        final String usage = """
```

```
                Usage:
                    <tableArn> <allowedAWSPrincipal>

                Where:
                    tableArn - The Amazon DynamoDB table ARN to attach the policy to.
For example, arn:aws:dynamodb:us-west-2:123456789012:table/MusicCollection.
                    allowed AWS Principal - Allowed AWS principal ARN that the example
policy will give access to. For example, arn:aws:iam::123456789012:user/John.
                """;

        if (args.length != 2) {
            System.out.println(usage);
            System.exit(1);
        }

        String tableArn = args[0];
        String allowedAWSPrincipal = args[1];
        System.out.println("Attaching a resource-based policy to the Amazon DynamoDB
table with ARN " +
                tableArn);
        Region region = Region.US_WEST_2;
        DynamoDbClient ddb = DynamoDbClient.builder()
                .region(region)
                .build();

        String result = putResourcePolicy(ddb, tableArn, allowedAWSPrincipal);
        System.out.println("Revision ID for the attached policy is " + result);
        ddb.close();
    }

    public static String putResourcePolicy(DynamoDbClient ddb, String tableArn, String
allowedAWSPrincipal) {
        String policy = generatePolicy(tableArn, allowedAWSPrincipal);
        PutResourcePolicyRequest request = PutResourcePolicyRequest.builder()
                .policy(policy)
                .resourceArn(tableArn)
                .build();

        try {
            return ddb.putResourcePolicy(request).revisionId();
        } catch (DynamoDbException e) {
            System.err.println(e.getMessage());
            System.exit(1);
```

```
        }

        return "";
    }

    private static String generatePolicy(String tableArn, String allowedAWSPrincipal) {
        return "{\n" +
                "    \"Version\": \"2012-10-17\",\n" +,
                "    \"Statement\": [\n" +
                "        {\n" +
                "            \"Effect\": \"Allow\",\n" +
                "            \"Principal\": {\"AWS\":\"" + allowedAWSPrincipal + "\"},
\n" +
                "            \"Action\": [\n" +
                "                \"dynamodb:GetItem\"\n" +
                "            ],\n" +
                "            \"Resource\": \"" + tableArn + "\"\n" +
                "        }\n" +
                "    ]\n" +
                "}";
    }
}
```

Anhängen einer ressourcenbasierten Richtlinie an einen DynamoDB-Stream

Sie können eine ressourcenbasierte Richtlinie an den Stream einer vorhandenen Tabelle anhängen oder eine vorhandene Richtlinie ändern, indem Sie die DynamoDB-Konsole, die PutResourcePolicyAPI AWS CLI, das AWS SDK oder eine Vorlage verwenden.CloudFormation

> **Note**
>
> Sie können eine Richtlinie nicht an einen Stream anhängen, während Sie ihn mit dem oder erstellen. CreateTableUpdateTable APIs Sie können eine Richtlinie jedoch ändern oder löschen, nachdem eine Tabelle gelöscht wurde. Sie können außerdem die Richtlinie eines deaktivierten Streams ändern oder löschen.

AWS CLI

Im folgenden Beispiel für eine IAM-Richtlinie wird der `put-resource-policy` AWS CLI Befehl verwendet, um eine ressourcenbasierte Richtlinie an den Stream einer Tabelle mit dem Namen anzuhängen. *`MusicCollection`* In diesem Beispiel kann der Benutzer *`John`* die DescribeStreamAPI-Aktionen GetRecordsGetShardIterator, und im Stream ausführen.

Denken Sie daran, den *`italicized`* Text durch Ihre ressourcenspezifischen Informationen zu ersetzen.

```
aws dynamodb put-resource-policy \
    --resource-arn arn:aws:dynamodb:us-west-2:123456789012:table/MusicCollection/
stream/2024-02-12T18:57:26.492 \
    --policy \
        "{
            \"Version\": \"2012-10-17\",
            \"Statement\": [
              {
                    \"Effect\": \"Allow\",
                    \"Principal\": {
                        \"AWS\": \"arn:aws:iam::111122223333:user/John\"
                    },
                    \"Action\": [
                        \"dynamodb:GetRecords\",
                        \"dynamodb:GetShardIterator\",
                        \"dynamodb:DescribeStream\"
                    ],
                    \"Resource\": \"arn:aws:dynamodb:us-
west-2:123456789012:table/MusicCollection/stream/2024-02-12T18:57:26.492\"
                }
            ]
        }"
```

AWS-Managementkonsole

1. Melden Sie sich bei der an AWS-Managementkonsole und öffnen Sie die DynamoDB-Konsole unter. https://console.aws.amazon.com/dynamodb/
2. Wählen Sie im Dashboard der DynamoDB-Konsole die Option Tabellen und dann eine vorhandene Tabelle aus.

   Vergewissern Sie sich, dass in der ausgewählten Tabelle Streams aktiviert sind. Hinweise zum Aktivieren von Streams für eine Tabelle finden Sie unter Aktivieren eines Streams.

3. Wählen Sie die Registerkarte Berechtigungen.
4. Wählen Sie unter Ressourcenbasierte Richtlinie für aktiven Stream die Option Stream-Richtlinie erstellen aus.
5. Fügen Sie im Editor Ressourcenbasierte Richtlinie eine Richtlinie hinzu, um die Zugriffsberechtigungen für den Stream zu definieren. In dieser Richtlinie geben Sie an, wer Zugriff auf jede Ressource hat und welche Aktionen für jede Ressource ausgeführt werden dürfen. Um eine Richtlinie hinzuzufügen, gehen Sie wie folgt vor:

   - Geben oder fügen Sie ein JSON-Richtliniendokument ein. Einzelheiten zur Sprache der IAM-Richtlinien finden Sie unter Erstellen von Richtlinien mit dem JSON-Editor im IAM-Benutzerhandbuch.

     > **Tip**
     >
     > Beispiele für ressourcenbasierte Richtlinien finden Sie unter Beispiele für Richtlinien im Entwicklerhandbuch für Amazon DynamoDB.

   - Wählen Sie Neue Anweisung hinzufügen aus, um eine neue Anweisung hinzuzufügen, und geben Sie die Informationen in die bereitgestellten Felder ein. Wiederholen Sie diesen Vorgang für so viele Anweisungen, wie Sie hinzufügen möchten.

     > **Important**
     >
     > Beheben Sie mögliche Sicherheitswarnungen, Fehler und Vorschläge, bevor Sie die Richtlinie speichern.

6. (Optional) Wählen Sie unten rechts Preview external access (Vorschau des externen Zugriffs) aus, um eine Vorschau anzuzeigen, wie sich Ihre neue Richtlinie auf den öffentlichen und kontoübergreifenden Zugriff auf Ihre Ressource auswirkt. Bevor Sie Ihre Richtlinie speichern, können Sie überprüfen, ob sie neue IAM-Access-Analyzer-Ergebnisse einführt oder vorhandene Ergebnisse löst. Wenn Sie keinen aktiven Analyzer sehen, wählen Sie Go to Access Analyzer (Zu Access Analyzer wechseln) aus, um einen Account Analyzer in IAM Access Analyzer zu erstellen. Weitere Informationen finden Sie unter Vorschau des Zugriffs.
7. Wählen Sie Richtlinie erstellen aus.

Im folgenden Beispiel für eine IAM-Richtlinie wird eine ressourcenbasierte Richtlinie an den Stream einer Tabelle mit dem Namen angehängt. *MusicCollection* In diesem Beispiel kann der Benutzer *John* die DescribeStreamAPI-Aktionen GetRecordsGetShardIterator, und im Stream ausführen.

Denken Sie daran, den *italicized* Text durch Ihre ressourcenspezifischen Informationen zu ersetzen.

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "AWS": "arn:aws:iam::111122223333:user/username"
      },
      "Action": [
        "dynamodb:GetRecords",
        "dynamodb:GetShardIterator",
        "dynamodb:DescribeStream"
      ],
      "Resource": [
        "arn:aws:dynamodb:us-east-1:123456789012:table/MusicCollection/
stream/2024-02-12T18:57:26.492"
      ]
    }
  ]
}
```

Entfernen einer ressourcenbasierten Richtlinie aus einer DynamoDB-Tabelle

Sie können eine ressourcenbasierte Richtlinie aus einer vorhandenen Tabelle löschen, indem Sie die DynamoDB-Konsole, die DeleteResourcePolicyAPI AWS CLI, das AWS SDK oder eine Vorlage verwenden. CloudFormation

AWS CLI

Im folgenden Beispiel wird der `delete-resource-policy` AWS CLI Befehl verwendet, um eine ressourcenbasierte Richtlinie aus einer Tabelle mit dem Namen zu entfernen. *`MusicCollection`*

Denken Sie daran, den *`italicized`* Text durch Ihre ressourcenspezifischen Informationen zu ersetzen.

```
aws dynamodb delete-resource-policy \
    --resource-arn arn:aws:dynamodb:us-west-2:123456789012:table/MusicCollection
```

AWS-Managementkonsole

1. Melden Sie sich bei der an AWS-Managementkonsole und öffnen Sie die DynamoDB-Konsole unter. https://console.aws.amazon.com/dynamodb/
2. Wählen Sie im Dashboard der DynamoDB-Konsole die Option Tabellen und dann eine vorhandene Tabelle aus.
3. Wählen Sie Berechtigungen.
4. Wählen Sie im Dropdown-Menü Richtlinie verwalten die Option Richtlinie löschen aus.
5. Geben Sie im Dialogfeld Ressourcenbasierte Richtlinie für Tabelle löschen **`confirm`** ein, um Löschvorgang zu bestätigen.
6. Wählen Sie Löschen aus.

Kontoübergreifender Zugriff mit ressourcenbasierten Richtlinien in DynamoDB

Mithilfe einer ressourcenbasierten Richtlinie können Sie einen kontoübergreifenden Zugriff auf Ressourcen gewähren, die in verschiedenen AWS-Konten verfügbar sind. Jeder kontenübergreifende Zugriff, der gemäß den ressourcenbasierten Richtlinien zulässig ist, wird anhand der Ergebnisse des externen Zugriffs von IAM Access Analyzer gemeldet, wenn Sie einen Analyzer in derselben Ressource haben. AWS-Region IAM Access Analyzer führt Richtlinienprüfungen durch, um Ihre Richtlinie anhand der IAM-Richtliniengrammatik und der bewährten Methoden zu validieren. Diese Prüfungen generieren Ergebnisse und bieten umsetzbare Empfehlungen, die Sie beim Erstellen von Richtlinien unterstützen, die funktionsfähig sind und den bewährten Methoden für Sicherheit entsprechen. Sie können die aktiven Ergebnisse von IAM Access Analyzer auf der Registerkarte Berechtigungen der DynamoDB-Konsole anzeigen.

Weitere Informationen zum Validieren von Richtlinien mit IAM Access Analyzer finden Sie unter Richtlinienvalidierung mit IAM Access Analyzer im IAM-Benutzerhandbuch. Eine Liste der Warnungen, Fehler und Vorschläge, die von IAM Access Analyzer zurückgegeben werden, finden Sie unter IAM-Access-Analyzer-Richtlinienprüfungsreferenz.

Gehen Sie wie folgt vor, um einem Benutzer A in Konto A die GetItemErlaubnis zu erteilen, auf eine Tabelle B in Konto B zuzugreifen:

1. Hängen Sie eine ressourcenbasierte Richtlinie an Tabelle B an, die Benutzer A die Berechtigung zum Ausführen der Aktion `GetItem` erteilt.
2. Hängen Sie eine ressourcenbasierte Richtlinie an Benutzer A an, die diesem die Berechtigung zum Ausführen der Aktion `GetItem` in Tabelle B erteilt.

Mit der Option Vorschau des externen Zugriffs in der DynamoDB-Konsole können Sie in einer Vorschau sehen, wie sich Ihre neue Richtlinie auf den öffentlichen und den kontoübergreifenden Zugriff auf Ihre Ressource auswirkt. Bevor Sie Ihre Richtlinie speichern, können Sie überprüfen, ob sie neue IAM-Access-Analyzer-Ergebnisse einführt oder vorhandene Ergebnisse löst. Wenn Sie keinen aktiven Analyzer sehen, wählen Sie Go to Access Analyzer (Zu Access Analyzer wechseln) aus, um einen Account Analyzer in IAM Access Analyzer zu erstellen. Weitere Informationen finden Sie unter Vorschau des Zugriffs.

Der Tabellennamenparameter in der DynamoDB-Datenebene und der Steuerungsebene APIs akzeptiert den vollständigen Amazon-Ressourcennamen (ARN) der Tabelle, um kontenübergreifende Operationen zu unterstützen. Wenn Sie nur den Parameter für den Tabellennamen anstelle eines vollständigen ARN angeben, wird die API-Operation für die Tabelle in dem Konto ausgeführt, zu dem der Anforderer gehört. Ein Beispiel einer Richtlinie, die den kontoübergreifenden Zugriff verwendet, finden Sie unter Ressourcenbasierte Richtlinie für kontoübergreifenden Zugriff.

Das Konto des Ressourceneigentümers wird auch dann belastet, wenn ein Prinzipal aus einem anderen Konto einen Lese- oder Schreibvorgang in der DynamoDB-Tabelle im Konto des Eigentümers ausführt. Wenn für die Tabelle Durchsatz bereitgestellt wurde, bestimmt die Summe aller Anfragen von den Eigentümerkonten und den Anforderern in anderen Konten, ob die Anfrage gedrosselt (wenn Autoscaling deaktiviert ist) oder skaliert wird, wenn Autoscaling aktiviert ist. up/down

Die Anfragen werden sowohl in den CloudTrail Protokollen des Besitzers als auch des Anfordererkontos protokolliert, sodass jedes der beiden Konten nachverfolgen kann, welches Konto auf welche Daten zugegriffen hat.

Teilen Sie den Zugriff mit kontoübergreifenden AWS Lambda-Funktionen

Lambda-Funktionen in Konto A

1. Gehen Sie zur IAM-Konsole, um eine IAM-Rolle zu erstellen, die als Lambda-Ausführungsrolle für Ihre AWS Lambda-Funktion in Konto A verwendet wird. Fügen Sie die verwaltete IAM-Richtlinie `AWSLambdaDynamoDBExecutionRole` hinzu, die über die erforderlichen DynamoDB Streams- und Lambda-Aufrufberechtigungen verfügt. Diese Richtlinie gewährt auch Zugriff auf alle potenziellen DynamoDB Streams Streams-Ressourcen, auf die Sie möglicherweise in Konto A Zugriff haben.
2. Erstellen Sie in der Lambda-Konsole eine AWS Lambda-Funktion zur Verarbeitung von Datensätzen in einem DynamoDB-Stream und wählen Sie während der Einrichtung für die Ausführungsrolle die Rolle aus, die Sie im vorherigen Schritt erstellt haben.
3. Stellen Sie dem Eigentümer von Konto B der DynamoDB-Streams die Lambda-Funktionsausführungsrolle zur Verfügung, um die ressourcenbasierte Richtlinie für den kontoübergreifenden Lesezugriff zu konfigurieren.
4. Schließen Sie die Einrichtung der Lambda-Funktion ab.

DynamoDB-Stream in Konto B

1. Rufen Sie die kontoübergreifende Lambda-Ausführungsrolle von Konto A ab, die die Lambda-Funktion aufruft.
2. Wählen Sie in der Amazon DynamoDB DynamoDB-Konsole in Konto B die Tabelle für den kontoübergreifenden Lambda-Trigger aus. Suchen Sie auf der Registerkarte Exporte und Streams Ihren DynamoDB-Stream-ARN. Stellen Sie sicher, dass der DynamoDB-Stream-Status On lautet, und notieren Sie sich den vollständigen Stream-ARN, da Sie ihn für die Ressourcenrichtlinie benötigen.
3. Klicken Sie auf der Registerkarte „Berechtigungen“ auf die Schaltfläche „Stream-Richtlinie erstellen“, um den visuellen Richtlinien-Editor zu starten. Klicken Sie auf die Schaltfläche Neue Erklärung hinzufügen oder bearbeiten Sie die Richtlinie, falls bereits eine vorhanden ist.
4. Erstellen Sie eine Richtlinie, die die Lambda-Ausführungsrolle in Konto A als Principal angibt, und gewähren Sie die erforderlichen DynamoDB-Stream-Aktionen. Stellen Sie sicher, dass Sie die Aktionen`dynamodb:DescribeStream`,`dynamodb:GetRecords`, `dynamodb:GetShardIterator` und angeben. `dynamodb:ListShards` Weitere Informationen zu Beispielressourcenrichtlinien für DynamoDB Streams finden Sie unter Beispiele für ressourcenbasierte DynamoDB-Richtlinien.

> **Note**
>
> Für den kontenübergreifenden Zugriff auf die Kontrollebene gilt ein niedrigerer APIs Grenzwert für Transaktionen pro Sekunde (TPS) von 500 Anfragen.

Blockieren des öffentlichen Zugriffs mit ressourcenbasierten Richtlinien in DynamoDB

Block Public Access (BPA) ist eine Funktion zum Identifizieren und Verhindern des Anhängens von ressourcenbasierten Richtlinien, die öffentlichen Zugriff auf Ihre DynamoDB-Tabellen, -Indizes oder -Streams in Ihren Amazon Web Services (AWS)-Konten gewähren. Mit BPA können Sie den öffentlichen Zugriff auf Ihre DynamoDB-Ressourcen verhindern. BPA führt während der Erstellung oder Änderung einer ressourcenbasierten Richtlinie Prüfungen durch und hilft Ihnen, Ihre Sicherheitslage mit DynamoDB zu verbessern.

BPA analysiert mittels Automated Reasoning den durch Ihre ressourcenbasierte Richtlinie gewährten Zugriff und warnt Sie, wenn solche Berechtigungen zum Zeitpunkt der Verwaltung einer ressourcenbasierten Richtlinie gefunden werden. Bei der Analyse wird der Zugriff über alle ressourcenbasierten Richtlinienanweisungen, Aktionen und die in Ihren Richtlinien verwendeten Bedingungsschlüssel überprüft.

> **Important**
>
> BPA trägt zum Schutz Ihrer Ressourcen bei, indem verhindert wird, dass öffentlicher Zugriff über die ressourcenbasierten Richtlinien gewährt wird, die Ihren DynamoDB-Ressourcen wie Tabellen, Indizes und Streams direkt angehängt sind. Überprüfen Sie zusätzlich zur Aktivierung von BPA sorgfältig die folgenden Richtlinien, um sicherzustellen, dass sie keinen öffentlichen Zugriff gewähren:
>
> - Identitätsbasierte Richtlinien, die mit zugehörigen AWS Principals verknüpft sind (z. B. IAM-Rollen)
> - Ressourcenbasierte Richtlinien, die mit zugehörigen AWS Ressourcen verknüpft sind (z. B. (KMS-) Schlüssel) AWS Key Management Service

Sie müssen sicherstellen, dass der Prinzipal keinen *-Eintrag enthält oder dass keiner der angegebenen Bedingungsschlüssel den Zugriff der Prinzipale auf die Ressource einschränkt. Wenn

die ressourcenbasierte Richtlinie öffentlichen Zugriff auf Ihre Tabelle, Indizes oder Stream-Across gewährt, blockiert DynamoDB Sie daran AWS-Konten, die Richtlinie zu erstellen oder zu ändern, bis die Spezifikation in der Richtlinie korrigiert und als nicht öffentlich eingestuft wurde.

Sie können eine Richtlinie als nicht öffentlich festlegen, indem Sie einen oder mehrere Prinzipale im `Principal`-Block angeben. Im folgenden Beispiel für eine ressourcenbasierte Richtlinie wird der öffentliche Zugriff blockiert, indem zwei Prinzipale angegeben werden.

```
{
  "Effect": "Allow",
  "Principal": {
    "AWS": [
      "123456789012",
      "111122223333"
    ]
  },
  "Action": "dynamodb:*",
  "Resource": "*"
}
```

Richtlinien, die den Zugriff durch die Angabe bestimmter Bedingungsschlüssel einschränken, gelten ebenfalls nicht als öffentlich. Neben der Auswertung des in der ressourcenbasierten Richtlinie angegebenen Prinzipals werden die folgenden vertrauenswürdigen Bedingungsschlüssel verwendet, um die Auswertung einer ressourcenbasierten Richtlinie für den nicht öffentlichen Zugriff abzuschließen:

- `aws:PrincipalAccount`
- `aws:PrincipalArn`
- `aws:PrincipalOrgID`
- `aws:PrincipalOrgPaths`
- `aws:SourceAccount`
- `aws:SourceArn`
- `aws:SourceVpc`
- `aws:SourceVpce`
- `aws:UserId`
- `aws:PrincipalServiceName`
- `aws:PrincipalServiceNamesList`

- aws:PrincipalIsAWSService
- aws:Ec2InstanceSourceVpc
- aws:SourceOrgID
- aws:SourceOrgPaths

Damit eine ressourcenbasierte Richtlinie nicht öffentlich ist, dürfen die Werte für den Amazon-Ressourcennamen (ARN) und Zeichenfolgenschlüssel außerdem keine Platzhalter oder Variablen enthalten. Wenn Ihre ressourcenbasierte Richtlinie den aws:PrincipalIsAWSService-Schlüssel verwendet, müssen Sie sicherstellen, dass Sie den Schlüsselwert auf „true“ gesetzt haben.

Die folgende Richtlinie grenzt den Zugriff auf den Benutzer John im angegebenen Konto ein. Aufgrund dieser Bedingung ist der Principal eingeschränkt und wird als nicht öffentlich betrachtet.

```
{
  "Effect": "Allow",
  "Principal": {
    "AWS": "*"
  },
  "Action": "dynamodb:*",
  "Resource": "*",
  "Condition": {
    "StringEquals": {
      "aws:PrincipalArn": "arn:aws:iam::123456789012:user/John"
    }
  }
}
```

Das folgende Beispiel für eine nicht öffentliche ressourcenbasierte Richtlinie schränkt sourceVPC auf die Verwendung des StringEquals-Operators ein.

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "AWS": "*"
```

```
      },
      "Action": "dynamodb:*",
      "Resource": "arn:aws:dynamodb:us-east-1:123456789012:table/
MusicCollection",
      "Condition": {
        "StringEquals": {
          "aws:SourceVpc": [
            "vpc-91237329"
          ]
        }
      }
    }
  ]
}
```

DynamoDB API-Operationen, die von ressourcenbasierten Richtlinien unterstützt werden

In diesem Thema werden die API-Operationen aufgeführt, die von ressourcenbasierten Richtlinien unterstützt werden. Für den kontoübergreifenden Zugriff können Sie jedoch nur einen bestimmten Satz von DynamoDB APIs über ressourcenbasierte Richtlinien verwenden. Sie können ressourcenbasierte Richtlinien nicht an Ressourcentypen wie Backups und Importe anhängen. Die IAM-Aktionen, die dem APIs Betrieb mit diesen Ressourcentypen entsprechen, sind von den unterstützten IAM-Aktionen in ressourcenbasierten Richtlinien ausgeschlossen. Weil Tabellenadministratoren interne Tabelleneinstellungen, wie z. B. UpdateTimeToLiveund, innerhalb desselben Kontos konfigurieren APIs DisableKinesisStreamingDestination, keinen kontenübergreifenden Zugriff über ressourcenbasierte Richtlinien unterstützen.

Die DynamoDB-Daten- und Steuerungsebene APIs , die den kontenübergreifenden Zugriff unterstützen, unterstützen auch das Überladen von Tabellennamen, sodass Sie den Tabellen-ARN anstelle des Tabellennamens angeben können. Sie können den Tabellen-ARN im `TableName` Parameter dieser angeben APIs. Nicht alle APIs unterstützen jedoch den kontoübergreifenden Zugriff.

Themen

- Datenebenen-API-Operationen
- PartiQL-API-Operationen
- Steuerebenen-API-Operationen

- API-Operationen für globale Tabellen in Version 2019.11.21 (aktuell)
- API-Operationen für globale Tabellen in Version 2017.11.29 (veraltet)
- Tags-API-Operation
- API-Operationen für Backup und Wiederherstellungen
- Kontinuierliche API-Operationen Backup/Restore (PITR)
- API-Operationen für Contributor Insights
- Export-API-Operationen
- Import-API-Operationen
- API-Operationen für Amazon Kinesis Data Streams
- API-Operationen für ressourcenbasierte Richtlinien
- Time-to-Live API-Operationen
- Sonstige API-Operationen
- API-Operationen in DynamoDB Streams

Datenebenen-API-Operationen

In der folgenden Tabelle ist die Unterstützung auf API-Ebene aufgeführt, die durch Datenebenen-API-Operationen für ressourcenbasierte Richtlinien und den kontoübergreifenden Zugriff bereitgestellt wird.

Datenebene - Tables/indexes APIs	Unterstützung für ressourcenbasierte Richtlinien	Kontoübergreifende Unterstützung
DeleteItem	Ja	Ja
GetItem	Ja	Ja
PutItem	Ja	Ja
Query	Ja	Ja
Scan	Ja	Ja
UpdateItem	Ja	Ja
TransactGetItems	Ja	Ja

Datenebene - Tables/indexes APIs	Unterstützung für ressourcenbasierte Richtlinien	Kontoübergreifende Unterstützung
TransactWriteItems	Ja	Ja
BatchGetItem	Ja	Ja
BatchWriteItem	Ja	Ja

PartiQL-API-Operationen

In der folgenden Tabelle ist die Unterstützung auf API-Ebene aufgeführt, die durch PartiQL-API-Operationen für ressourcenbasierte Richtlinien und den kontoübergreifenden Zugriff bereitgestellt wird.

PartiQL APIs	Unterstützung für ressourcenbasierte Richtlinien	Kontoübergreifende Unterstützung
BatchExecuteStatement	Ja	Nein
ExecuteStatement	Ja	Nein
ExecuteTransaction	Ja	Nein

Steuerebenen-API-Operationen

In der folgenden Tabelle ist die Unterstützung auf API-Ebene aufgeführt, die durch Steuerebenen-API-Operationen für ressourcenbasierte Richtlinien und den kontoübergreifenden Zugriff bereitgestellt wird.

Steuerungsebene - Tabellen APIs	Unterstützung für ressourcenbasierte Richtlinien	Kontoübergreifende Unterstützung
CreateTable	Nein	Nein
DeleteTable	Ja	Ja

Steuerungsebene - Tabellen APIs	Unterstützung für ressourcenbasierte Richtlinien	Kontoübergreifende Unterstützung
DescribeTable	Ja	Ja
UpdateTable	Ja	Ja

API-Operationen für globale Tabellen in Version 2019.11.21 (aktuell)

In der folgenden Tabelle ist die Unterstützung auf API-Ebene aufgeführt, die durch API-Operationen für globale Tabellen in Version 2019.11.21 (aktuell) für ressourcenbasierte Richtlinien und den kontoübergreifenden Zugriff bereitgestellt wird.

Version 2019.11.21 (Aktuell) globale Tabellen APIs	Unterstützung für ressourcenbasierte Richtlinien	Kontoübergreifende Unterstützung
DescribeTableReplicaAutoScaling	Ja	Nein
UpdateTableReplicaAutoScaling	Ja	Nein

API-Operationen für globale Tabellen in Version 2017.11.29 (veraltet)

In der folgenden Tabelle ist die Unterstützung auf API-Ebene aufgeführt, die durch API-Operationen für globale Tabellen in Version 2017.11.29 (veraltet) für ressourcenbasierte Richtlinien und den kontoübergreifenden Zugriff bereitgestellt wird.

Version 2017.11.29 (Legacy) globale Tabellen APIs	Unterstützung für ressourcenbasierte Richtlinien	Kontoübergreifende Unterstützung
CreateGlobalTable	Nein	Nein
DescribeGlobalTable	Nein	Nein
DescribeGlobalTableSettings	Nein	Nein

Version 2017.11.29 (Legacy) globale Tabellen APIs	Unterstützung für ressourcenbasierte Richtlinien	Kontoübergreifende Unterstützung
ListGlobalTables	Nein	Nein
UpdateGlobalTable	Nein	Nein
UpdateGlobalTableSettings	Nein	Nein

Tags-API-Operation

In der folgenden Tabelle ist die Unterstützung auf API-Ebene aufgeführt, die durch API-Operationen im Zusammenhang mit Tags für ressourcenbasierte Richtlinien und den kontoübergreifenden Zugriff bereitgestellt wird.

Schlagworte APIs	Unterstützung für ressourcenbasierte Richtlinien	Kontoübergreifende Unterstützung
ListTagsOfResource	Ja	Ja
TagResource	Ja	Ja
UntagResource	Ja	Ja

API-Operationen für Backup und Wiederherstellungen

In der folgenden Tabelle ist die Unterstützung auf API-Ebene aufgeführt, die durch API-Operationen im Zusammenhang mit Backups und Wiederherstellungen für ressourcenbasierte Richtlinien und den kontoübergreifenden Zugriff bereitgestellt wird.

Backup und Wiederherstellen APIs	Unterstützung für ressourcenbasierte Richtlinien	Kontoübergreifende Unterstützung
CreateBackup	Ja	Nein
DescribeBackup	Nein	Nein

Backup und Wiederherstellen APIs	Unterstützung für ressourcenbasierte Richtlinien	Kontoübergreifende Unterstützung
DeleteBackup	Nein	Nein
RestoreTableFromBackup	Nein	Nein

Kontinuierliche API-Operationen Backup/Restore (PITR)

In der folgenden Tabelle ist die Unterstützung auf API-Ebene aufgeführt, die durch API-Operationen im Zusammenhang mit Continuous Backup/Restore (PITR) für ressourcenbasierte Richtlinien und kontenübergreifenden Zugriff bereitgestellt wird.

Kontinuierlich (PITR) Backup/Restore APIs	Unterstützung für ressourcenbasierte Richtlinien	Kontoübergreifende Unterstützung
DescribeContinuousBackups	Ja	Nein
RestoreTableToPointInTime	Ja	Nein
UpdateContinuousBackups	Ja	Nein

API-Operationen für Contributor Insights

In der folgenden Tabelle ist die Unterstützung auf API-Ebene aufgeführt, die durch API-Operationen im Zusammenhang mit Continuous Backup/Restore (PITR) für ressourcenbasierte Richtlinien und kontenübergreifenden Zugriff bereitgestellt wird.

Einblicke von Mitwirkenden APIs	Unterstützung für ressourcenbasierte Richtlinien	Kontoübergreifende Unterstützung
DescribeContributorInsights	Ja	Nein
ListContributorInsights	Nein	Nein
UpdateContributorInsights	Ja	Nein

Export-API-Operationen

In der folgenden Tabelle ist die Unterstützung auf API-Ebene aufgeführt, die durch Export-API-Operationen für ressourcenbasierte Richtlinien und den kontoübergreifenden Zugriff bereitgestellt wird.

Exportieren APIs	Unterstützung für ressourcenbasierte Richtlinien	Kontoübergreifende Unterstützung
DescribeExport	Nein	Nein
ExportTableToPointInTime	Ja	Nein
ListExports	Nein	Nein

Import-API-Operationen

In der folgenden Tabelle ist die Unterstützung auf API-Ebene aufgeführt, die durch Import-API-Operationen für ressourcenbasierte Richtlinien und den kontoübergreifenden Zugriff bereitgestellt wird.

Importieren APIs	Unterstützung für ressourcenbasierte Richtlinien	Kontoübergreifende Unterstützung
DescribeImport	Nein	Nein
ImportTable	Nein	Nein
ListImports	Nein	Nein

API-Operationen für Amazon Kinesis Data Streams

In der folgenden Tabelle ist die Unterstützung auf API-Ebene aufgeführt, die durch API-Operationen für Kinesis Data Streams für ressourcenbasierte Richtlinien und den kontoübergreifenden Zugriff bereitgestellt wird.

Kinesis APIs	Unterstützung für ressourcenbasierte Richtlinien	Kontoübergreifende Unterstützung
DescribeKinesisStreamingDestination	Ja	Nein
DisableKinesisStreamingDestination	Ja	Nein
EnableKinesisStreamingDestination	Ja	Nein
UpdateKinesisStreamingDestination	Ja	Nein

API-Operationen für ressourcenbasierte Richtlinien

In der folgenden Tabelle ist die Unterstützung auf API-Ebene aufgeführt, die durch API-Operationen im Zusammenhang mit ressourcenbasierten Richtlinien für ressourcenbasierte Richtlinien und den kontoübergreifenden Zugriff bereitgestellt wird.

Ressourcenbasierte Richtlinie APIs	Unterstützung für ressourcenbasierte Richtlinien	Kontoübergreifende Unterstützung
GetResourcePolicy	Ja	Nein
PutResourcePolicy	Ja	Nein
DeleteResourcePolicy	Ja	Nein

Time-to-Live API-Operationen

In der folgenden Tabelle ist die Unterstützung auf API-Ebene aufgeführt, die durch Time-to-Live (TTL)-API-Operationen für ressourcenbasierte Richtlinien und den kontoübergreifenden Zugriff bereitgestellt wird.

TTL APIs	Unterstützung für ressourcenbasierte Richtlinien	Kontoübergreifende Unterstützung
DescribeTimeToLive	Ja	Nein
UpdateTimeToLive	Ja	Nein

Sonstige API-Operationen

In der folgenden Tabelle ist die Unterstützung auf API-Ebene aufgeführt, die durch sonstige verschiedene API-Operationen für ressourcenbasierte Richtlinien und den kontoübergreifenden Zugriff bereitgestellt wird.

Andere APIs	Unterstützung für ressourcenbasierte Richtlinien	Kontoübergreifende Unterstützung
DescribeLimits	Nein	Nein
DescribeEndpoints	Nein	Nein
ListBackups	Nein	Nein
ListTables	Nein	Nein

API-Operationen in DynamoDB Streams

In der folgenden Tabelle ist die Unterstützung von DynamoDB Streams auf API-Ebene APIs für ressourcenbasierte Richtlinien und kontoübergreifenden Zugriff aufgeführt.

DynamoDB Streams APIs	Unterstützung für ressourcenbasierte Richtlinien	Kontoübergreifende Unterstützung
DescribeStream	Ja	Ja
GetRecords	Ja	Ja
GetShardIterator	Ja	Ja

DynamoDB Streams APIs	Unterstützung für ressourcenbasierte Richtlinien	Kontoübergreifende Unterstützung
ListStreams	Nein	Nein

Autorisierung mit identitätsbasierten IAM-Richtlinien und ressourcenbasierten DynamoDB-Richtlinien

Identitätsbasierte Richtlinien sind an eine Identität wie IAM-Benutzer, -Benutzergruppen und -Rollen angehängt. Diese IAM-Richtliniendokumente steuern, welche Aktionen eine Identität für welche Ressourcen und unter welchen Bedingungen ausführen kann. Identitätsbasierte Richtlinien können verwaltet oder Inline sein.

Ressourcenbasierte Richtlinien sind IAM-Richtliniendokumente, die Sie an eine Ressource wie eine DynamoDB-Tabelle anhängen. Diese Richtlinien erteilen dem angegebenen Auftraggeber die Berechtigung zum Ausführen bestimmter Aktionen für diese Ressource und definiert, unter welchen Bedingungen dies gilt. Beispielsweise umfasst die ressourcenbasierte Richtlinie für eine DynamoDB-Tabelle auch den Index, der der Tabelle zugeordnet ist. Ressourcenbasierte Richtlinien sind Inline-Richtlinien. Es gibt keine verwalteten ressourcenbasierten Richtlinien.

Weitere Informationen zu diesen Richtlinien finden Sie unter Identitätsbasierte Richtlinien und ressourcenbasierte Richtlinien im IAM-Benutzerhandbuch.

Wenn der IAM-Prinzipal aus demselben Konto stammt wie der Ressourceneigentümer, reicht eine ressourcenbasierte Richtlinie aus, um Zugriffsberechtigungen für die Ressource festzulegen. Sie können sich weiterhin für eine identitätsbasierte IAM-Richtlinie und eine ressourcenbasierte Richtlinie entscheiden. Für den kontoübergreifenden Zugriff müssen Sie den Zugriff sowohl in den Identitäts- als auch den Ressourcenrichtlinien explizit zulassen, wie unter Kontoübergreifender Zugriff mit ressourcenbasierten Richtlinien in DynamoDB beschrieben. Wenn Sie beide Arten von Richtlinien verwenden, wird eine Richtlinie wie unter Ermitteln, ob eine Anforderung innerhalb eines Kontos zugelassen oder verweigert wird beschrieben ausgewertet.

> **Important**
>
> Wenn eine identitätsbasierte Richtlinie bedingungslosen Zugriff auf eine DynamoDB-Tabelle gewährt (z. B. ohne Bedingungen), `dynamodb:GetItem` schränkt eine ressourcenbasierte Richtlinie, die den Zugriff mit aktivierten Bedingungen erlaubt, diesen Zugriff nicht ein.

`dynamodb:Attributes` Die bedingungslose Zulassung der identitätsbasierten Richtlinie hat Vorrang, und die Bedingungen der ressourcenbasierten Richtlinie gelten nicht als Einschränkungen. Um den Zugriff auf bestimmte Attribute einzuschränken, verwenden Sie eine explizite `Deny` Anweisung, anstatt sich ausschließlich auf bedingte `Allow` Anweisungen in der ressourcenbasierten Richtlinie zu verlassen.

Beispiele für ressourcenbasierte DynamoDB-Richtlinien

Wenn Sie im Feld `Resource` einer ressourcenbasierten Richtlinie einen ARN angeben, wird die Richtlinie nur wirksam, wenn der angegebene ARN mit dem ARN der DynamoDB-Ressource übereinstimmt, an die er angehängt ist.

> **Note**
>
> Denken Sie daran, den *`italicized`* Text durch Ihre ressourcenspezifischen Informationen zu ersetzen.

Ressourcenbasierte Richtlinie für eine Tabelle

Die folgende ressourcenbasierte Richtlinie, die an eine DynamoDB-Tabelle mit dem Namen angehängt ist*`MusicCollection`*, erteilt den IAM-Benutzern *`John`* die *`Jane`* Berechtigung, Aktionen auf der Ressource auszuführen GetItem. BatchGetItem*`MusicCollection`*

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Sid": "1111",
        "Effect": "Allow",
        "Principal": {
          "AWS": [
            "arn:aws:iam::111122223333:user/username",
            "arn:aws:iam::111122223333:user/Jane"
          ]
```

```
        },
        "Action": [
          "dynamodb:GetItem",
          "dynamodb:BatchGetItem"
        ],
        "Resource": [
          "arn:aws:dynamodb:us-east-1:123456789012:table/MusicCollection"
        ]
    }
  ]
}
```

Ressourcenbasierte Richtlinie für einen Stream

Die folgende ressourcenbasierte Richtlinie, die an einen DynamoDB-Stream mit dem Namen angehängt ist, `2024-02-12T18:57:26.492` gibt den IAM-Benutzern die *Jane* Berechtigung GetRecords,, GetShardIterator, *John* und DescribeStreamAPI-Aktionen auf der Ressource durchzuführen. `2024-02-12T18:57:26.492`

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Sid": "1111",
      "Effect": "Allow",
      "Principal": {
        "AWS": [
          "arn:aws:iam::111122223333:user/username",
          "arn:aws:iam::111122223333:user/Jane"
        ]
      },
      "Action": [
        "dynamodb:DescribeStream",
        "dynamodb:GetRecords",
        "dynamodb:GetShardIterator"
      ],
      "Resource": [
```

```
        "arn:aws:dynamodb:us-east-1:123456789012:table/MusicCollection/
stream/2024-02-12T18:57:26.492"
      ]
    }
  ]
}
```

Ressourcenbasierte Richtlinie für den Zugriff zur Ausführung aller Aktionen für bestimmte Ressourcen

Damit ein Benutzer alle Aktionen für eine Tabelle und alle zugehörigen Indizes mit einer Tabelle ausführen kann, können Sie einen Platzhalter (*) verwenden, um die mit der Tabelle verknüpften Aktionen und Ressourcen darzustellen. Die Verwendung eines Platzhalterzeichens für die Ressourcen ermöglicht dem Benutzer den Zugriff auf die DynamoDB-Tabelle und alle zugehörigen Indizes, einschließlich der Indizes, die noch nicht erstellt wurden. Die folgende Richtlinie gibt dem Benutzer beispielsweise die *John* Erlaubnis, alle Aktionen an der *MusicCollection* Tabelle und all ihren Indizes durchzuführen, einschließlich aller Indizes, die in future erstellt werden.

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Sid": "1111",
      "Effect": "Allow",
      "Principal": {
        "AWS": "arn:aws:iam::111122223333:user/role-name"
      },
      "Action": "dynamodb:*",
      "Resource": [
        "arn:aws:dynamodb:us-east-1:123456789012:table/MusicCollection",
        "arn:aws:dynamodb:us-east-1:123456789012:table/MusicCollection/
index/index-name"
      ]
    }
  ]
}
```

Ressourcenbasierte Richtlinie für kontoübergreifenden Zugriff

Sie können Berechtigungen für eine kontoübergreifende IAM-Identität für den Zugriff auf DynamoDB-Ressourcen angeben. Beispielsweise benötigen Sie möglicherweise einen Benutzer mit einem vertrauenswürdigen Konto, um Lesezugriff auf den Inhalt Ihrer Tabelle zu erhalten, unter der Bedingung, dass er nur auf bestimmte Elemente und bestimmte Attribute in diesen Elementen zugreift. Die folgende Richtlinie ermöglicht Benutzern *John* über eine vertrauenswürdige AWS-Konto ID *111111111111* den Zugriff auf Daten aus einer Tabelle im Konto *123456789012* mithilfe der GetItemAPI. Die Richtlinie stellt sicher, dass der Benutzer nur auf Elemente mit einem Primärschlüssel zugreifen kann *Jane* und dass der Benutzer nur die Attribute `Artist` und`SongTitle`, aber keine anderen Attribute abrufen kann.

> **Important**
>
> Wenn Sie die `SPECIFIC_ATTRIBUTES`-Bedingung nicht angeben, werden Ihnen alle Attribute für die zurückgegebenen Elemente angezeigt.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "CrossAccountTablePolicy",
            "Effect": "Allow",
            "Principal": {
                "AWS": "arn:aws:iam::111111111111:user/John"
            },
            "Action": "dynamodb:GetItem",
            "Resource": [
                "arn:aws:dynamodb:us-west-2:123456789012:table/MusicCollection"
            ],
            "Condition": {
                "ForAllValues:StringEquals": {
                    "dynamodb:LeadingKeys": "Jane",
                    "dynamodb:Attributes": [
```

```
                    "Artist",
                    "SongTitle"
                ]
            },
            "StringEquals": {
                "dynamodb:Select": "SPECIFIC_ATTRIBUTES"
            }
        }
    }
  ]
}
```

Zusätzlich zur vorherigen ressourcenbasierten Richtlinie muss die dem Benutzer zugeordnete identitätsbasierte Richtlinie *John* auch die `GetItem` API-Aktion ermöglichen, damit der kontoübergreifende Zugriff funktioniert. Im Folgenden finden Sie ein Beispiel für eine identitätsbasierte Richtlinie, die Sie dem Benutzer zuordnen müssen. *John*

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "CrossAccountIdentityBasedPolicy",
            "Effect": "Allow",
            "Action": [
                "dynamodb:GetItem"
            ],
            "Resource": [
                "arn:aws:dynamodb:us-east-1:123456789012:table/MusicCollection"
            ],
            "Condition": {
                "ForAllValues:StringEquals": {
                    "dynamodb:LeadingKeys": "Jane",
                    "dynamodb:Attributes": [
                        "Artist",
                        "SongTitle"
                    ]
                },
                "StringEquals": {
                    "dynamodb:Select": "SPECIFIC_ATTRIBUTES"
```

```
                }
            }
        }
    ]
}
```

Der Benutzer John kann eine `GetItem` Anfrage stellen, indem er den Tabellen-ARN im `table-name` Parameter für den Zugriff auf die Tabelle *`MusicCollection`* im Konto angibt*`123456789012`*.

```
aws dynamodb get-item \
    --table-name arn:aws:dynamodb:us-west-2:123456789012:table/MusicCollection \
    --key '{"Artist": {"S": "Jane"}' \
    --projection-expression 'Artist, SongTitle' \
    --return-consumed-capacity TOTAL
```

Ressourcenbasierte Richtlinie mit Bedingungen für die IP-Adresse

Sie können eine Bedingung anwenden, um Quell-IP-Adressen, virtuelle private Clouds (VPCs) und VPC-Endpunkte (VPCE) einzuschränken. Sie können Berechtigungen auf der Grundlage der Quelladressen der ursprünglichen Anforderung angeben. Angenommen, Sie möchten einem Benutzer nur dann Zugriff auf DynamoDB-Ressourcen gewähren, wenn von einer bestimmten IP-Quelle aus darauf zugegriffen wird, z. B. von einem Unternehmens-VPN-Endpunkt. Geben Sie diese IP-Adressen in der `Condition`-Anweisung an.

Das folgende Beispiel ermöglicht dem Benutzer den *`John`* Zugriff auf jede DynamoDB-Ressource, wenn die Quelle IPs und ist. `54.240.143.0/24` `2001:DB8:1234:5678::/64`

JSON

```
{
  "Id":"PolicyId2",
  "Version":"2012-10-17",
  "Statement":[
    {
      "Sid":"AllowIPmix",
      "Effect":"Allow",
      "Principal": {
        "AWS": "arn:aws:iam::111111111111:user/username"
      },
      "Action":"dynamodb:*",
```

```
      "Resource":"*",
      "Condition": {
        "IpAddress": {
          "aws:SourceIp": [
            "54.240.143.0/24",
            "2001:DB8:1234:5678::/64"
          ]
        }
      }
    }
  ]
}
```

Sie können auch jeglichen Zugriff auf DynamoDB-Ressourcen verweigern, es sei denn, die Quelle ist beispielsweise ein bestimmter VPC-Endpunkt. *vpce-1a2b3c4d*

> **Important**
>
> Wenn Sie DAX mit DynamoDB-Tabellen verwenden, die IP-basierte Ressourcenrichtlinien in IPv6 reinen Umgebungen haben, müssen Sie zusätzliche Zugriffsregeln konfigurieren. Wenn Ihre Ressourcenrichtlinie den Zugriff auf den IPv4 Adressraum `0.0.0.0/0` in Tabellen einschränkt, müssen Sie den Zugriff für die mit Ihrem DAX-Cluster verknüpfte IAM-Rolle zulassen. Fügen Sie Ihrer Richtlinie eine `ArnNotEquals` Bedingung hinzu, um sicherzustellen, dass DAX den Zugriff auf Ihre DynamoDB-Tabellen beibehält. Weitere Informationen finden Sie unter DAX und. IPv6

JSON

```
{
  "Id":"PolicyId",
  "Version":"2012-10-17",
  "Statement": [
    {
      "Sid": "AccessToSpecificVPCEOnly",
      "Principal": "*",
      "Action": "dynamodb:*",
      "Effect": "Deny",
      "Resource": "*",
      "Condition": {
```

```
        "StringNotEquals":{
          "aws:sourceVpce":"vpce-1a2b3c4d"
        }
      }
    }
  ]
}
```

Ressourcenbasierte Richtlinie mit einer IAM-Rolle

Sie können auch eine IAM-Servicerolle in der ressourcenbasierten Richtlinie angeben. IAM-Entitäten, die diese Rolle übernehmen, sind an die für die Rolle angegebenen zulässigen Aktionen und an die spezifische Gruppe von Ressourcen innerhalb der ressourcenbasierten Richtlinie gebunden.

Im folgenden Beispiel kann eine IAM-Entität alle DynamoDB-Aktionen für die *MusicCollection* und *MusicCollection* DynamoDB-Ressourcen ausführen.

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Sid": "1111",
      "Effect": "Allow",
      "Principal": { "AWS": "arn:aws:iam::111122223333:role/role-name" },
      "Action": "dynamodb:*",
      "Resource": [
        "arn:aws:dynamodb:us-east-1:123456789012:table/MusicCollection",
        "arn:aws:dynamodb:us-east-1:123456789012:table/MusicCollection/*"
      ]
    }
  ]
}
```

Überlegungen zu ressourcenbasierten DynamoDB-Richtlinien

Wenn Sie ressourcenbasierte Richtlinien für Ihre DynamoDB-Ressourcen definieren, berücksichtigen Sie die folgenden Überlegungen:

Allgemeine Überlegungen

- Ein ressourcenbasiertes Richtliniendokument kann maximal 20 KB groß sein. Leerzeichen werden in DynamoDB beim Berechnen der Richtliniengröße in Bezug auf diesen Limit nicht mitgezählt.
- Nachfolgende Aktualisierungen einer Richtlinie für eine bestimmte Ressource werden nach einer erfolgreichen Aktualisierung der Richtlinie für dieselbe Ressource 15 Sekunden lang blockiert.
- Derzeit können Sie nur eine ressourcenbasierte Richtlinie an vorhandene Streams anhängen. Sie können einem Stream während der Erstellung keine Richtlinie zuordnen.

Überlegungen zu globalen Tabellen

- Ressourcenbasierte Richtlinien werden für Replikate der Version 2017.11.29 (veraltet) für eine globale Tabelle nicht unterstützt.
- Wenn innerhalb einer ressourcenbasierten Richtlinie die Aktion für eine serviceverknüpfte DynamoDB-Rolle (SLR) zum Replizieren von Daten für eine globale Tabelle verweigert wird, schlägt das Hinzufügen oder Löschen eines Replikats mit einem Fehler fehl.
- Die AWS: :DynamoDB:: GlobalTable -Ressource unterstützt nicht das Erstellen eines Replikats und das Hinzufügen einer ressourcenbasierten Richtlinie zu diesem Replikat in demselben Stack-Update in anderen Regionen als der Region, in der Sie das Stack-Update bereitstellen.

Kontoübergreifende Überlegungen

- Der kontenübergreifende Zugriff mithilfe ressourcenbasierter Richtlinien unterstützt keine verschlüsselten Tabellen mit AWS verwalteten Schlüsseln, da Sie keinen kontoübergreifenden Zugriff auf die verwaltete KMS-Richtlinie gewähren können. AWS

CloudFormation Überlegungen

- Ressourcenbasierte Richtlinien bieten keine Unterstützung für die Erkennung von Abweichungen. Wenn Sie eine ressourcenbasierte Richtlinie außerhalb der AWS CloudFormation Stack-Vorlage aktualisieren, müssen Sie den CloudFormation Stack mit den Änderungen aktualisieren.
- Ressourcenbasierte Richtlinien unterstützen keine Out-of-Band-Änderungen. Wenn Sie eine Richtlinie außerhalb der CloudFormation Vorlage hinzufügen, aktualisieren oder löschen, wird die Änderung nicht überschrieben, sofern in der Vorlage keine Änderungen an der Richtlinie vorgenommen wurden.

Nehmen wir beispielsweise an, dass Ihre Vorlage eine ressourcenbasierte Richtlinie enthält, die Sie später außerhalb der Vorlage aktualisieren. Wenn Sie keine Änderungen an der Richtlinie in der Vorlage vornehmen, wird die aktualisierte Richtlinie in DynamoDB nicht mit der Richtlinie in der Vorlage synchronisiert.

Nehmen wir umgekehrt an, dass Ihre Vorlage keine ressourcenbasierte Richtlinie enthält, Sie aber eine Richtlinie außerhalb der Vorlage hinzufügen. Diese Richtlinie wird nicht aus DynamoDB entfernt, solange Sie sie nicht zur Vorlage hinzufügen. Wenn Sie der Vorlage eine Richtlinie hinzufügen und den Stack aktualisieren, wird die vorhandene Richtlinie in DynamoDB aktualisiert, sodass sie mit der in der Vorlage definierten Richtlinie übereinstimmt.

Bewährte Methoden für ressourcenbasierte DynamoDB-Richtlinien

In diesem Thema werden die bewährten Methoden für die Definition von Zugriffsberechtigungen für Ihre DynamoDB-Ressourcen und die für diese Ressourcen zulässigen Aktionen beschrieben.

Vereinfachen der Zugriffskontrolle auf DynamoDB-Ressourcen

Wenn die AWS Identity and Access Management Prinzipale, die Zugriff auf eine DynamoDB-Ressource benötigen, zu demselben gehören AWS-Konto wie der Ressourcenbesitzer, ist keine identitätsbasierte IAM-Richtlinie für jeden Prinzipal erforderlich. Eine ressourcenbasierte Richtlinie, die an die angegebenen Ressourcen angehängt ist, reicht aus. Diese Art der Konfiguration vereinfacht die Zugriffskontrolle.

Schützen Ihrer DynamoDB-Ressourcen mit ressourcenbasierten Richtlinien

Erstellen Sie für alle DynamoDB-Tabellen und -Streams ressourcenbasierte Richtlinien, um die Zugriffskontrolle für diese Ressourcen durchzusetzen. Mit ressourcenbasierten Richtlinien können Sie Berechtigungen auf Ressourcenebene zentralisieren, die Zugriffskontrolle für DynamoDB-Tabellen, -Indizes und -Streams vereinfachen sowie den Verwaltungsaufwand reduzieren. Wenn keine ressourcenbasierte Richtlinie für eine Tabelle oder einen Stream angegeben ist, wird der Zugriff auf die Tabelle oder den Stream implizit verweigert, es sei denn, identitätsbasierte Richtlinien, die den IAM-Prinzipalen zugeordnet sind, lassen den Zugriff zu.

Gewähren Sie die geringsten Berechtigungen

Wenn Sie Berechtigungen mit ressourcenbasierten Richtlinien für DynamoDB-Ressourcen festlegen, gewähren Sie nur die Berechtigungen, die zum Ausführen einer Aktion erforderlich sind. Sie tun dies,

indem Sie die Aktionen definieren, die für bestimmte Ressourcen unter bestimmten Bedingungen durchgeführt werden können, auch bekannt als die geringsten Berechtigungen. Sie können mit umfassenden Berechtigungen beginnen, während Sie die für Ihren Workload oder Anwendungsfall erforderlichen Berechtigungen untersuchen. Mit zunehmender Reife Ihres Anwendungsfalls können Sie daran arbeiten, die Berechtigungen zu reduzieren, die Sie gewähren, um auf die geringsten Berechtigungen hinzuarbeiten.

Analysieren der kontoübergreifenden Zugriffsaktivität zum Generieren von Richtlinien mit der geringsten Berechtigung

IAM Access Analyzer meldet den kontoübergreifenden Zugriff auf externe Entitäten, die in ressourcenbasierten Richtlinien angegeben sind, und bietet Transparenz, damit Sie Ihre Berechtigungen eingrenzen und das Prinzip der geringsten Berechtigung einhalten können. Weitere Informationen zur Richtlinienerstellung finden Sie unter IAM Access Analyzer Richtlinienerstellung.

Verwenden von IAM Access Analyzer zum Generieren von Richtlinien mit der geringsten Berechtigung

Um nur die zum Ausführen einer Aufgabe erforderlichen Berechtigungen zu erteilen, können Sie Richtlinien auf der Grundlage Ihrer in AWS CloudTrail protokollierten Zugriffsaktivitäten erstellen. IAM Access Analyzer analysiert die Services und Aktionen, die Ihre Richtlinien verwenden.

Verwenden der attributbasierten Zugriffskontrolle mit DynamoDB

Die attributebasierte Zugriffskontrolle (ABAC) ist eine Autorisierungsstrategie, die Zugriffsberechtigungen auf der Grundlage von Tag-Bedingungen in Ihren identitätsbasierten Richtlinien oder anderen AWS Richtlinien, wie ressourcenbasierten Richtlinien und IAM-Richtlinien der Organisation, definiert. Sie können Tags an DynamoDB-Tabellen anhängen, die dann anhand der Tag-basierten Bedingungen ausgewertet werden. Die mit einer Tabelle verknüpften Indizes übernehmen die Tags, die Sie der Tabelle hinzufügen. Sie können für jede DynamoDB-Tabelle bis zu 50 Tags hinzufügen. Die maximale Größe, die für alle Tags in einer Tabelle unterstützt wird, beträgt 10 KB. Weitere Informationen zum Taggen von DynamoDB-Ressourcen und zu Tagging-Einschränkungen finden Sie unter Tagging von Ressourcen in DynamoDB and the section called “Markierungseinschränkungen in DynamoDB”.

Weitere Informationen zur Verwendung von Tags zur Steuerung des Zugriffs auf AWS Ressourcen finden Sie in den folgenden Themen im IAM-Benutzerhandbuch:

- Wofür ist ABAC AWS
- Steuern des Zugriffs auf AWS Ressourcen mithilfe von Tags

Mit ABAC können Sie unterschiedliche Zugriffsebenen für Ihre Teams und Anwendungen erzwingen, um mit weniger Richtlinien Aktionen für DynamoDB-Tabellen auszuführen. Sie können ein Tag im Bedingungselement einer IAM-Richtlinie angeben, um den Zugriff auf Ihre DynamoDB-Tabellen oder -Indizes zu kontrollieren. Diese Bedingungen bestimmen die Zugriffsebene, die ein IAM-Prinzipal, ein Benutzer oder eine Rolle auf DynamoDB-Tabellen und -Indizes erhält. Wenn ein IAM-Prinzipal eine Zugriffsanforderung an DynamoDB stellt, werden die Tags der Ressource und Identität anhand der Tag-Bedingungen in der IAM-Richtlinie ausgewertet. Danach wird die Richtlinie nur wirksam, wenn die Tag-Bedingungen erfüllt sind. Auf diese Weise können Sie eine IAM-Richtlinie erstellen, die effektiv eine der folgenden Aussagen enthält:

- Erlaube dem Benutzer, nur die Ressourcen zu verwalten, die über ein Tag mit dem Schlüssel *X* und dem Wert *Y* verfügen.
- Verweigere allen Benutzern den Zugriff auf Ressourcen, die mit dem Schlüssel *X* getaggt sind.

Sie können beispielsweise eine Richtlinie erstellen, die es Benutzern ermöglicht, eine Tabelle nur zu aktualisieren, wenn sie über das Tag-Schlüssel-Wert-Paar `"environment": "staging"` verfügt. Sie können den ResourceTag Bedingungsschlüssel aws: verwenden, um den Zugriff auf eine Tabelle auf der Grundlage der Tags, die an diese Tabelle angehängt sind, zu erlauben oder zu verweigern.

Sie können attributebasierte Bedingungen bei der Erstellung der Richtlinie oder zu einem späteren Zeitpunkt mithilfe von AWS API AWS-Managementkonsole, AWS Command Line Interface (AWS CLI), AWS SDK oder angeben. AWS CloudFormation

Im folgenden Beispiel ist die UpdateItemAktion für eine Tabelle mit dem Namen zulässig, `MusicTable` wenn sie einen Tag-Schlüssel mit dem Namen `environment` und dem Wert enthält. `production`

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
```

```
      "Action": [
        "dynamodb:UpdateItem"
      ],
      "Resource": "arn:aws:dynamodb:*:*:table/MusicTable",
      "Condition": {
        "StringEquals": {
          "aws:ResourceTag/environment": "production"
        }
      }
    }
  ]
}
```

Themen

- Weshalb sollte ich ABAC verwenden?
- Bedingungsschlüssel für die Implementierung von ABAC mit DynamoDB
- Überlegungen zur Verwendung von ABAC mit DynamoDB
- Aktivieren von ABAC in DynamoDB
- Verwenden von ABAC mit DynamoDB-Tabellen und -Indizes
- Beispiele für die Verwendung von ABAC mit DynamoDB-Tabellen und -Indizes
- Behebung häufiger ABAC-Fehler für DynamoDB-Tabellen und -Indizes

Weshalb sollte ich ABAC verwenden?

- Einfachere Richtlinienverwaltung: Sie verwenden weniger Richtlinien, da Sie keine verschiedenen Richtlinien erstellen müssen, um die Zugriffsebene für jeden IAM-Prinzipal zu definieren.
- Skalierbare Zugriffskontrolle: Die Skalierung der Zugriffskontrolle ist mit ABAC einfacher, da Sie Ihre Richtlinien nicht aktualisieren müssen, wenn Sie neue DynamoDB-Ressourcen erstellen. Sie können Tags verwenden, um den Zugriff auf IAM-Prinzipale zu autorisieren, die Tags enthalten, die den Tags der Ressource entsprechen. Sie können neue IAM-Prinzipale oder DynamoDB-Ressourcen einbinden und entsprechende Tags anwenden, um automatisch die erforderlichen Berechtigungen zu gewähren, ohne Änderungen an den Richtlinien vornehmen zu müssen.
- Detaillierte Berechtigungsverwaltung: Als bewährte Methode sollten Sie beim Erstellen von Richtlinien die geringste Berechtigung gewähren. Mit ABAC können Sie Tags für den IAM-Prinzipal erstellen und verwenden, um ihm Zugriff auf bestimmte Aktionen und Ressourcen zu gewährten, die den Tags für den IAM-Prinzipal entsprechen.

- Ausrichtung am Unternehmensverzeichnis: Sie können Tags vorhandenen Mitarbeiterattributen aus Ihrem Unternehmensverzeichnis zuordnen, um Ihre Zugriffskontrollrichtlinien an Ihre Organisationsstruktur anzupassen.

Bedingungsschlüssel für die Implementierung von ABAC mit DynamoDB

Sie können die folgenden Bedingungsschlüssel in Ihren AWS Richtlinien verwenden, um die Zugriffsebene auf Ihre DynamoDB-Tabellen und -Indizes zu steuern:

- aws: ResourceTag /tag-key: Steuert den Zugriff basierend darauf, ob das Tag-Schlüssel-Wert-Paar in einer DynamoDB-Tabelle oder einem DynamoDB-Index mit dem Tag-Schlüssel und -Wert in einer Richtlinie übereinstimmt oder nicht. Dieser Bedingungsschlüssel ist für alle relevant, die mit einer vorhandenen Tabelle oder einem APIs vorhandenen Index arbeiten.

  Die `dynamodb:ResourceTag`-Bedingungen werden so ausgewertet, als hätten Sie einer Ressource keine Tags angehängt.
- aws: RequestTag /tag-key: Ermöglicht den Vergleich des Tag-Schlüssel-Wert-Paars, das in der Anfrage übergeben wurde, mit dem Tag-Paar, das Sie in der Richtlinie angeben. Dieser Bedingungsschlüssel ist relevant für diejenigen APIs , die Tags als Teil der Nutzlast der Anfrage enthalten. Dazu APIs gehören CreateTableund. TagResource
- aws:TagKeys: Vergleicht die Tag-Schlüssel in einer Anfrage mit den Schlüsseln, die Sie in der Richtlinie angeben. Dieser Bedingungsschlüssel ist relevant für diejenigen APIs , die Tags als Teil der Nutzlast der Anfrage enthalten. Dazu APIs gehören `CreateTableTagResource`, und`UntagResource`.

Überlegungen zur Verwendung von ABAC mit DynamoDB

Wenn Sie ABAC mit DynamoDB-Tabellen oder -Indizes verwenden, berücksichtigen Sie die folgenden Überlegungen:

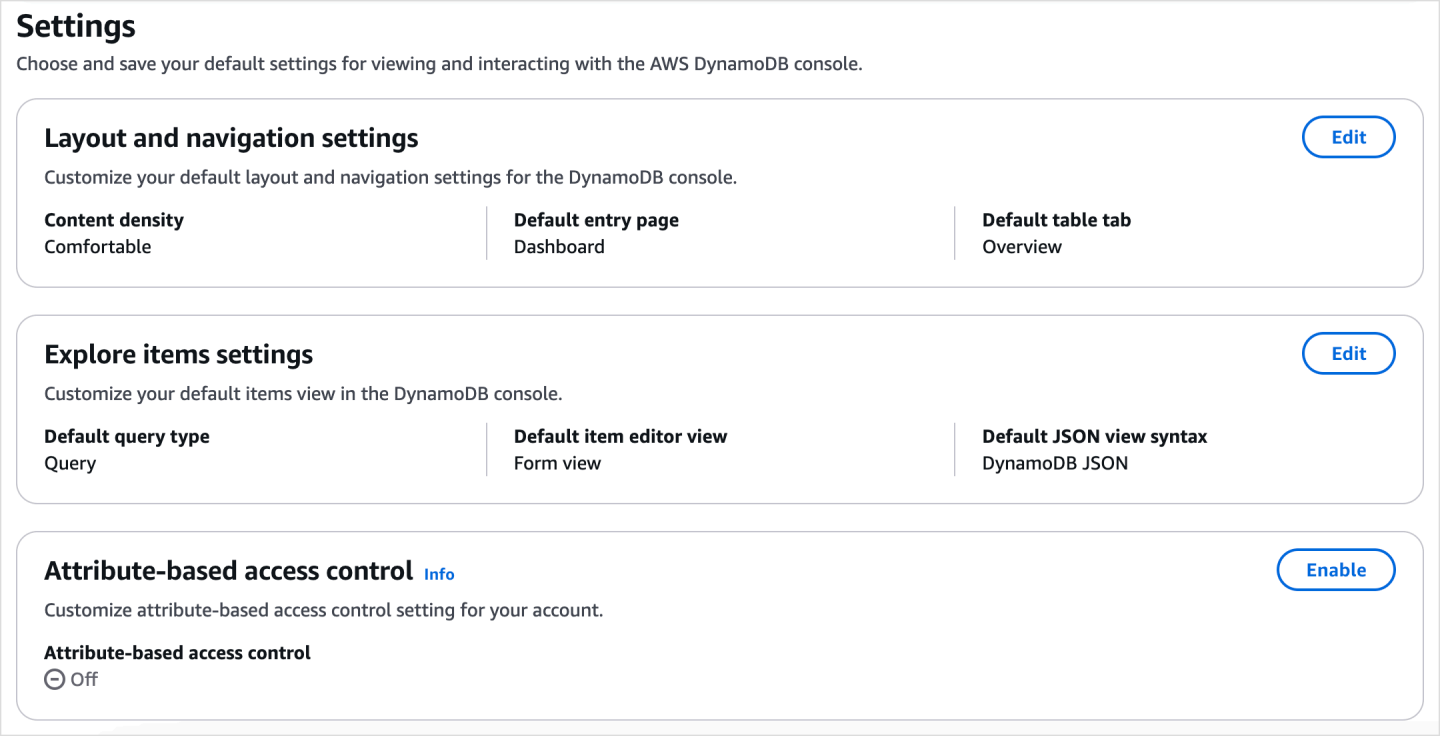
- Tagging und ABAC werden für DynamoDB Streams nicht unterstützt.
- Tagging und ABAC werden für DynamoDB-Backups nicht unterstützt. Um ABAC mit Backups zu verwenden, empfehlen wir die Verwendung von AWS Backup.
- Tags werden in wiederhergestellten Tabellen nicht beibehalten. Sie müssen den wiederhergestellten Tabellen Tags hinzufügen, bevor Sie Tag-basierte Bedingungen in Ihren Richtlinien verwenden können.

Aktivieren von ABAC in DynamoDB

In den meisten AWS-Konten Fällen ist ABAC standardmäßig aktiviert. Mithilfe der DynamoDB-Konsole können Sie überprüfen, ob ABAC für Ihr Konto aktiviert ist. Stellen Sie dazu sicher, dass Sie die DynamoDB-Konsole mit einer Rolle öffnen, die über die dynamodb: -Berechtigung verfügt. GetAbacStatus Öffnen Sie dann die Seite Einstellungen der DynamoDB-Konsole.

Wenn Sie die Registerkarte Attributbasierte Zugriffskontrolle nicht sehen oder wenn auf der Registerkarte der Status An angezeigt wird, ist ABAC für Ihr Konto aktiviert. Wenn auf der Registerkarte Attributbasierte Zugangskontrolle jedoch wie in der folgenden Abbildung gezeigt der Status Aus angezeigt wird, ist ABAC für Ihr Konto nicht aktiviert.

Attributbasierte Zugriffskontrolle – nicht aktiviert



ABAC ist nicht aktiviert, weshalb tagbasierte Bedingungen, die in ihren identitätsbasierten Richtlinien oder anderen Richtlinien angegeben sind, noch geprüft werden müssen. AWS-Konten Wenn ABAC für Ihr Konto nicht aktiviert ist, werden die Tag-basierten Bedingungen in Ihren Richtlinien, die auf DynamoDB-Tabellen oder -Indizes angewendet werden sollen, so ausgewertet, als wären keine Tags für Ihre Ressourcen oder API-Anforderungen vorhanden. Wenn ABAC für Ihr Konto aktiviert ist, werden die Tag-basierten Bedingungen in den Richtlinien Ihres Kontos unter Berücksichtigung der mit Ihren Tabellen oder API-Anforderungen verknüpften Tags ausgewertet.

Um ABAC für Ihr Konto zu aktivieren, empfehlen wir, zunächst Ihre Richtlinien wie im Abschnitt Richtlinienprüfung beschrieben zu überprüfen. Schließen Sie dann die erforderlichen Berechtigungen für ABAC in Ihre IAM-Richtlinie ein. Führen Sie abschließend die unter Aktivieren von ABAC in der Konsole beschriebenen Schritte aus, um ABAC für Ihr Konto in der aktuellen Region zu aktivieren. Nachdem Sie ABAC aktiviert haben, können Sie die Option innerhalb der nächsten sieben Kalendertage nach der Aktivierung wieder deaktivieren.

Themen

- Überprüfen Ihrer Richtlinien vor der Aktivierung von ABAC
- IAM-Berechtigungen, die für die Aktivierung von ABAC erforderlich sind
- Aktivieren von ABAC in der Konsole

Überprüfen Ihrer Richtlinien vor der Aktivierung von ABAC

Bevor Sie ABAC für Ihr Konto aktivieren, prüfen Sie Ihre Richtlinien, um sicherzustellen, dass die Tag-basierten Bedingungen, die möglicherweise in den Richtlinien Ihres Kontos enthalten sind, wie vorgesehen eingerichtet sind. Durch die Prüfung Ihrer Richtlinien können Sie Überraschungen durch Autorisierungsänderungen in Ihren DynamoDB-Workflows vermeiden, nachdem ABAC aktiviert wurde. Beispiele für die Verwendung von attributbasierten Bedingungen mit Tags sowie das Vorher-Nachher-Verhalten der ABAC-Implementierung finden Sie unter Beispiele für die Verwendung von ABAC mit DynamoDB-Tabellen und -Indizes.

IAM-Berechtigungen, die für die Aktivierung von ABAC erforderlich sind

Sie benötigen die `dynamodb:UpdateAbacStatus`-Berechtigung, um ABAC für Ihr Konto in der aktuellen Region zu aktivieren. Um zu überprüfen, ob ABAC für Ihr Konto aktiviert ist, benötigen Sie außerdem die `dynamodb:GetAbacStatus`-Berechtigung. Mit dieser Berechtigung können Sie den ABAC-Status für ein Konto in einer beliebigen Region anzeigen. Sie benötigen diese Berechtigungen zusätzlich zu den Berechtigungen, die für den Zugriff auf die DynamoDB-Konsole erforderlich sind.

Die folgende IAM-Richtlinie gewährt die Berechtigung, ABAC zu aktivieren und ihren Status für ein Konto in der aktuellen Region anzuzeigen.

```
{
"version": "2012-10-17",        &TCX5-2025-waiver;
    "Statement": [
        {
            "Effect": "Allow",
```

```
            "Action": [
                "dynamodb:UpdateAbacStatus",
                "dynamodb:GetAbacStatus"
             ],
            "Resource": "*"
        }
    ]
}
```

Aktivieren von ABAC in der Konsole

1. Melden Sie sich bei der an AWS-Managementkonsole und öffnen Sie die DynamoDB-Konsole unter. https://console.aws.amazon.com/dynamodb/
2. Wählen Sie im Navigationsbereich die Region, für die Sie ABAC aktivieren möchten.
3. Wählen Sie im linken Navigationsbereich die Option Einstellungen aus.
4. Führen Sie auf der Seite Settings (Einstellungen) die folgenden Schritte aus:

    a. Wählen Sie auf der Registerkarte Attributbasierte Zugriffskontrolle die Option Aktivieren aus.

    b. Wählen im Feld Einstellung für die attributbasierte Zugriffskontrolle bestätigen die Option Aktivieren aus, um Ihre Auswahl zu bestätigen.

    Dadurch wird ABAC für die aktuelle Region aktiviert und auf der Registerkarte Attributbasierte Zugriffskontrolle wird der Status Ein angezeigt.

    Wenn Sie ABAC nach der Aktivierung in der Konsole wieder deaktivieren möchten, können Sie die Deaktivierung innerhalb der nächsten sieben Kalendertage nach der Aktivierung durchführen. Wählen Sie dafür Deaktivieren auf der Registerkarte Attributbasierte Zugriffskontrolle auf der Seite Einstellungen aus.

    > **Note**
    >
    > Das Aktualisieren des ABAC-Status ist ein asynchroner Vorgang. Wenn die Tags in Ihren Richtlinien nicht sofort ausgewertet werden, müssen Sie möglicherweise einige Zeit warten, da die Änderungen letztendlich konsistent angewendet werden.

Verwenden von ABAC mit DynamoDB-Tabellen und -Indizes

Die folgenden Schritte zeigen, wie Sie Berechtigungen mithilfe von ABAC einrichten. In diesem Beispielszenario fügen Sie einer DynamoDB-Tabelle Tags hinzu und erstellen eine IAM-Rolle mit einer Richtlinie, die Tag-basierte Bedingungen beinhaltet. Anschließend testen Sie die zulässigen Berechtigungen für die DynamoDB-Tabelle, indem Sie die Tag-Bedingungen erfüllen.

Themen

- Schritt 1: Hinzufügen von Tags zu einer DynamoDB-Tabelle
- Schritt 2: Erstellen einer IAM-Rolle mit einer Richtlinie, die Tag-basierte Bedingungen enthält
- Schritt 3: Testen der Berechtigungen

Schritt 1: Hinzufügen von Tags zu einer DynamoDB-Tabelle

Sie können Tags zu neuen oder vorhandenen DynamoDB-Tabellen hinzufügen, indem Sie die AWS API AWS-Managementkonsole, AWS Command Line Interface (AWS CLI), das AWS SDK oder verwenden. AWS CloudFormation Der folgende CLI-Befehl tag-resource fügt beispielsweise einer Tabelle namens `MusicTable` ein Tag hinzu.

```
aws dynamodb tag-resource –resource-arn arn:aws:dynamodb:us-east-1:123456789012:table/
MusicTable –tags Key=environment,Value=staging
```

Schritt 2: Erstellen einer IAM-Rolle mit einer Richtlinie, die Tag-basierte Bedingungen enthält

Erstellen Sie eine IAM-Richtlinie mithilfe des ResourceTagBedingungsschlüssels aws: /tag-key, um das in der IAM-Richtlinie angegebene Tag-Schlüssel-Wert-Paar mit dem Schlüssel-Wert-Paar zu vergleichen, das an die Tabelle angehängt ist. Die folgende Beispielrichtlinie ermöglicht es Benutzern, Elemente in Tabellen abzulegen oder zu aktualisieren, wenn diese Tabellen das Tag-Schlüssel-Wert-Paar `"environment": "staging"` enthalten. Wenn eine Tabelle das angegebene Tag-Schlüssel-Wert-Paar nicht enthält, werden diese Aktionen verweigert.

JSON

```
{
    "Version":"2012-10-17",
```

```
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "dynamodb:PutItem",
                "dynamodb:UpdateItem"
            ],
            "Resource": "arn:aws:dynamodb:*:*:table/*",
            "Condition": {
                "StringEquals": {
                    "aws:ResourceTag/environment": "staging"
                }
            }
        }
    ]
}
```

Schritt 3: Testen der Berechtigungen

1. Hängen Sie die IAM-Richtlinie einem Testbenutzer oder einer Testrolle in Ihrem AWS-Konto zu. Stellen Sie sicher, dass der von Ihnen verwendete IAM-Prinzipal nicht bereits über eine andere Richtlinie Zugriff auf die DynamoDB-Tabelle hat.
2. Achten Sie darauf, dass Ihre DynamoDB-Tabelle den `"environment"`-Tag-Schlüssel mit dem Wert `"staging"` enthält.
3. Führen Sie die Aktionen `dynamodb:PutItem` und `dynamodb:UpdateItem` für die mit Tags versehene Tabelle aus. Diese Aktionen sollten erfolgreich sein, wenn das `"environment": "staging"`-Tag-Schlüssel-Wert-Paar vorhanden ist.

   Wenn Sie diese Aktionen für eine Tabelle ausführen, die nicht über das `"environment": "staging"`-Tag-Schlüssel-Wert-Paar verfügt, schlägt Ihre Anforderung mit einem `AccessDeniedException`-Fehler fehl.

Sie können sich auch die anderen im folgenden Abschnitt beschriebenen Beispielanwendungsfälle ansehen, um ABAC zu implementieren und weitere Tests durchzuführen.

Beispiele für die Verwendung von ABAC mit DynamoDB-Tabellen und -Indizes

Die folgenden Beispiele zeigen einige Anwendungsfälle zur Implementierung von attributbasierten Bedingungen mithilfe von Tags.

Themen

- Beispiel 1: Eine Aktion mit aws zulassen: ResourceTag
- Beispiel 2: Eine Aktion mit aws zulassen: RequestTag
- Beispiel 3: Eine Aktion mit aws ablehnen: TagKeys

Beispiel 1: Eine Aktion mit aws zulassen: ResourceTag

Mithilfe des `aws:ResourceTag/tag-key`-Bedingungsschlüssels können Sie das Tag-Schlüssel-Wert-Paar, das in einer IAM-Richtlinie angegeben ist, mit dem Schlüssel-Wert-Paar vergleichen, das in einer DynamoDB-Tabelle angehängt ist. Sie können beispielsweise eine bestimmte Aktion zulassen, z. B. PutItemwenn die Tag-Bedingungen in einer IAM-Richtlinie und einer Tabelle übereinstimmen. Führen Sie dazu die folgenden Schritte aus:

Using the AWS CLI

1. Erstellen Sie eine -Tabelle. Im folgenden Beispiel wird der AWS CLI Befehl create-table verwendet, um eine Tabelle mit dem Namen zu erstellen. `myMusicTable`

   ```
   aws dynamodb create-table \
     --table-name myMusicTable \
     --attribute-definitions AttributeName=id,AttributeType=S \
     --key-schema AttributeName=id,KeyType=HASH \
     --provisioned-throughput ReadCapacityUnits=5,WriteCapacityUnits=5 \
     --region us-east-1
   ```

2. Fügen Sie dieser Tabelle ein Tag hinzu. Im folgenden Beispiel für den AWS CLI Befehl tag-resource wird das Schlüssel-Wert-Paar für das Tag hinzugefügt. `Title: ProductManager myMusicTable`

   ```
   aws dynamodb tag-resource --region us-east-1 --resource-arn arn:aws:dynamodb:us-east-1:123456789012:table/myMusicTable --tags Key=Title,Value=ProductManager
   ```

3. Erstellen Sie eine Inline-Richtlinie und fügen Sie sie einer Rolle hinzu, an die die AmazonDynamoDBReadOnlyAccess AWS verwaltete Richtlinie angehängt ist, wie im folgenden Beispiel gezeigt.

   JSON

   ```
   {
     "Version":"2012-10-17",
     "Statement": [
       {
         "Effect": "Allow",
         "Action": "dynamodb:PutItem",
         "Resource": "arn:aws:dynamodb:*:*:table/*",
         "Condition": {
           "StringEquals": {
             "aws:ResourceTag/Title": "ProductManager"
           }
         }
       }
     ]
   }
   ```

   Mit dieser Richtlinie kann die Aktion `PutItem` für die Tabelle ausgeführt werden, wenn der Tag-Schlüssel und -Wert, die der Tabelle angehängt sind, mit den in der Richtlinie angegebenen Tags übereinstimmen.

4. Nehmen Sie die Rolle mit den in Schritt 3 beschriebenen Richtlinien an.

5. Verwenden Sie den AWS CLI Befehl put-item, um ein Element in den zu platzieren. `myMusicTable`

   ```
   aws dynamodb put-item \
       --table-name myMusicTable --region us-east-1 \
       --item '{
           "id": {"S": "2023"},
           "title": {"S": "Happy Day"},
           "info": {"M": {
               "rating": {"N": "9"},
               "Artists": {"L": [{"S": "Acme Band"}, {"S": "No One You Know"}]},
               "release_date": {"S": "2023-07-21"}
           }}
   ```

```
    }'
```

6. Scannen Sie die Tabelle, um zu überprüfen, ob das Element der Tabelle hinzugefügt wurde.

```
aws dynamodb scan --table-name myMusicTable  --region us-east-1
```

Using the AWS SDK for Java 2.x

1. Erstellen Sie eine -Tabelle. Das folgende Beispiel verwendet die CreateTableAPI, um eine Tabelle mit dem Namen zu erstellen. `myMusicTable`

```
DynamoDbClient dynamoDB = DynamoDbClient.builder().region(region).build();
CreateTableRequest createTableRequest = CreateTableRequest.builder()
    .attributeDefinitions(
        Arrays.asList(
            AttributeDefinition.builder()
            .attributeName("id")
            .attributeType(ScalarAttributeType.S)
            .build()
        )
    )
    .keySchema(
        Arrays.asList(
            KeySchemaElement.builder()
            .attributeName("id")
            .keyType(KeyType.HASH)
            .build()
        )
    )
    .provisionedThroughput(ProvisionedThroughput.builder()
        .readCapacityUnits(5L)
        .writeCapacityUnits(5L)
        .build()
    )
    .tableName("myMusicTable")
    .build();

CreateTableResponse createTableResponse =
 dynamoDB.createTable(createTableRequest);
String tableArn = createTableResponse.tableDescription().tableArn();
String tableName = createTableResponse.tableDescription().tableName();
```

2. Fügen Sie dieser Tabelle ein Tag hinzu. Die TagResourceAPI im folgenden Beispiel fügt dem Tag das Schlüssel-Wert-Paar `Title: ProductManager` hinzu. `myMusicTable`

```
TagResourceRequest tagResourceRequest = TagResourceRequest.builder()
    .resourceArn(tableArn)
    .tags(
        Arrays.asList(
            Tag.builder()
            .key("Title")
            .value("ProductManager")
            .build()
        )
    )
    .build();
dynamoDB.tagResource(tagResourceRequest);
```

3. Erstellen Sie eine Inline-Richtlinie und fügen Sie sie einer Rolle hinzu, an die die AmazonDynamoDBReadOnlyAccess AWS verwaltete Richtlinie angehängt ist, wie im folgenden Beispiel gezeigt.

   JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "dynamodb:PutItem",
      "Resource": "arn:aws:dynamodb:*:*:table/*",
      "Condition": {
        "StringEquals": {
          "aws:ResourceTag/Title": "ProductManager"
        }
      }
    }
  ]
}
```

Mit dieser Richtlinie kann die Aktion `PutItem` für die Tabelle ausgeführt werden, wenn der Tag-Schlüssel und -Wert, die der Tabelle angehängt sind, mit den in der Richtlinie angegebenen Tags übereinstimmen.

4. Nehmen Sie die Rolle mit den in Schritt 3 beschriebenen Richtlinien an.
5. Verwenden Sie die PutItemAPI, um ein Element in den zu platzierenmy`MusicTable`.

```
HashMap<String, AttributeValue> info = new HashMap<>();
info.put("rating", AttributeValue.builder().s("9").build());
info.put("artists", AttributeValue.builder().ss(List.of("Acme Band","No One You
 Know").build());
info.put("release_date", AttributeValue.builder().s("2023-07-21").build());

HashMap<String, AttributeValue> itemValues = new HashMap<>();
itemValues.put("id", AttributeValue.builder().s("2023").build());
itemValues.put("title", AttributeValue.builder().s("Happy Day").build());
itemValues.put("info", AttributeValue.builder().m(info).build());


PutItemRequest putItemRequest = PutItemRequest.builder()
                .tableName(tableName)
                .item(itemValues)
                .build();
dynamoDB.putItem(putItemRequest);
```

6. Scannen Sie die Tabelle, um zu überprüfen, ob das Element der Tabelle hinzugefügt wurde.

```
ScanRequest scanRequest = ScanRequest.builder()
                .tableName(tableName)
                .build();

ScanResponse scanResponse = dynamoDB.scan(scanRequest);
```

Using the AWS SDK für Python (Boto3)

1. Erstellen Sie eine -Tabelle. Das folgende Beispiel verwendet die CreateTableAPI, um eine Tabelle mit dem Namen zu erstellenmy`MusicTable`.

```
create_table_response = ddb_client.create_table(
    AttributeDefinitions=[
```

```
        {
            'AttributeName': 'id',
            'AttributeType': 'S'
        },
    ],
    TableName='myMusicTable',
    KeySchema=[
        {
            'AttributeName': 'id',
            'KeyType': 'HASH'
        },
    ],
        ProvisionedThroughput={
        'ReadCapacityUnits': 5,
        'WriteCapacityUnits': 5
    },
)

table_arn = create_table_response['TableDescription']['TableArn']
```

2. Fügen Sie dieser Tabelle ein Tag hinzu. Die TagResourceAPI im folgenden Beispiel fügt dem Tag das Schlüssel-Wert-Paar `Title: ProductManager` hinzu. `myMusicTable`

```
tag_resouce_response = ddb_client.tag_resource(
    ResourceArn=table_arn,
    Tags=[
        {
            'Key': 'Title',
            'Value': 'ProductManager'
        },
    ]
)
```

3. Erstellen Sie eine Inline-Richtlinie und fügen Sie sie einer Rolle hinzu, an die die AmazonDynamoDBReadOnlyAccess AWS verwaltete Richtlinie angehängt ist, wie im folgenden Beispiel gezeigt.

   JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
```

```
        {
        "Effect": "Allow",
        "Action": "dynamodb:PutItem",
        "Resource": "arn:aws:dynamodb:*:*:table/*",
        "Condition": {
            "StringEquals": {
            "aws:ResourceTag/Title": "ProductManager"
            }
        }
        }
    ]
    }
```

Mit dieser Richtlinie kann die Aktion `PutItem` für die Tabelle ausgeführt werden, wenn der Tag-Schlüssel und -Wert, die der Tabelle angehängt sind, mit den in der Richtlinie angegebenen Tags übereinstimmen.

4. Nehmen Sie die Rolle mit den in Schritt 3 beschriebenen Richtlinien an.
5. Verwenden Sie die PutItemAPI, um ein Element in den zu platzierenmy`MusicTable`.

```
put_item_response = client.put_item(
    TableName = 'myMusicTable'
    Item = {
        'id': '2023',
        'title': 'Happy Day',
        'info': {
            'rating': '9',
            'artists': ['Acme Band','No One You Know'],
            'release_date': '2023-07-21'
        }
    }
)
```

6. Scannen Sie die Tabelle, um zu überprüfen, ob das Element der Tabelle hinzugefügt wurde.

```
scan_response = client.scan(
    TableName='myMusicTable'
)
```

Ohne ABAC

Wenn ABAC für Sie nicht aktiviert ist AWS-Konto, stimmen die Tag-Bedingungen in der IAM-Richtlinie und der DynamoDB-Tabelle nicht überein. Folglich gibt die `PutItem`-Aktion aufgrund der Auswirkung der `AmazonDynamoDBReadOnlyAccess`-Richtlinie einen `AccessDeniedException`-Fehler zurück.

```
An error occurred (AccessDeniedException) when calling the PutItem operation:
 User: arn:aws:sts::123456789012:assumed-role/DynamoDBReadOnlyAccess/Alice is
 not authorized to perform: dynamodb:PutItem on resource: arn:aws:dynamodb:us-
east-1:123456789012:table/myMusicTable because no identity-based policy allows the
 dynamodb:PutItem action.
```

Mit ABAC

Wenn ABAC für Sie aktiviert ist AWS-Konto, wird die `put-item` Aktion erfolgreich abgeschlossen und ein neues Element zu Ihrer Tabelle hinzugefügt. Das liegt daran, dass die Inline-Richtlinie in der Tabelle die `PutItem`-Aktion zulässt, wenn die Tag-Bedingungen in der IAM-Richtlinie und der Tabelle übereinstimmen.

Beispiel 2: Eine Aktion mit aws zulassen: RequestTag

Mithilfe des RequestTagBedingungsschlüssels aws: /tag-key können Sie das Tag-Schlüssel-Wert-Paar, das in Ihrer Anfrage übergeben wurde, mit dem Tag-Paar vergleichen, das in der IAM-Richtlinie angegeben ist. Sie können beispielsweise eine bestimmte Aktion wie `CreateTable` mithilfe von `aws:RequestTag` zulassen, wenn die Tag-Bedingungen nicht übereinstimmen. Führen Sie dazu die folgenden Schritte aus:

Using the AWS CLI

1. Erstellen Sie eine Inline-Richtlinie und fügen Sie sie einer Rolle hinzu, an die die AmazonDynamoDBReadOnlyAccess AWS verwaltete Richtlinie angehängt ist, wie im folgenden Beispiel gezeigt.

   JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
```

```
                "dynamodb:CreateTable",
                "dynamodb:TagResource"
            ],
            "Resource": "arn:aws:dynamodb:*:*:table/*",
            "Condition": {
                "StringEquals": {
                    "aws:RequestTag/Owner": "John"
                }
            }
        }
    ]
}
```

2. Erstellen Sie eine Tabelle, die das Tag-Schlüssel-Wert-Paar von `"Owner": "John"` enthält.

```
aws dynamodb create-table \
--attribute-definitions AttributeName=ID,AttributeType=S \
--key-schema AttributeName=ID,KeyType=HASH  \
--provisioned-throughput ReadCapacityUnits=1000,WriteCapacityUnits=500 \
--region us-east-1 \
--tags Key=Owner,Value=John \
--table-name myMusicTable
```

Using the AWS SDK für Python (Boto3)

1. Erstellen Sie eine Inline-Richtlinie und fügen Sie sie einer Rolle hinzu, an die die AmazonDynamoDBReadOnlyAccess AWS verwaltete Richtlinie angehängt ist, wie im folgenden Beispiel gezeigt.

   JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "dynamodb:CreateTable",
                "dynamodb:TagResource"
            ],
            "Resource": "arn:aws:dynamodb:*:*:table/*",
```

```
            "Condition": {
                "StringEquals": {
                    "aws:RequestTag/Owner": "John"
                }
            }
        }
    ]
}
```

2. Erstellen Sie eine Tabelle, die das Tag-Schlüssel-Wert-Paar von `"Owner": "John"` enthält.

```
ddb_client = boto3.client('dynamodb')

create_table_response = ddb_client.create_table(
    AttributeDefinitions=[
        {
            'AttributeName': 'id',
            'AttributeType': 'S'
        },
    ],
    TableName='myMusicTable',
    KeySchema=[
        {
            'AttributeName': 'id',
            'KeyType': 'HASH'
        },
    ],
        ProvisionedThroughput={
        'ReadCapacityUnits': 1000,
        'WriteCapacityUnits': 500
    },
    Tags=[
        {
            'Key': 'Owner',
            'Value': 'John'
        },
    ],
)
```

Ohne ABAC

Wenn ABAC für Sie nicht aktiviert ist AWS-Konto, stimmen die Tag-Bedingungen in der Inline-Richtlinie und der DynamoDB-Tabelle nicht überein. Folglich schlägt die `CreateTable`-Anforderung fehl und Ihre Tabelle wird nicht erstellt.

```
An error occurred (AccessDeniedException) when calling the CreateTable operation:
 User: arn:aws:sts::123456789012:assumed-role/Admin/John is not authorized to perform:
 dynamodb:CreateTable on resource: arn:aws:dynamodb:us-east-1:123456789012:table/
myMusicTable because no identity-based policy allows the dynamodb:CreateTable action.
```

Mit ABAC

Wenn ABAC für Sie aktiviert ist AWS-Konto, wird Ihre Anfrage zur Tabellenerstellung erfolgreich abgeschlossen. Da das Tag-Schlüssel-Wert-Paar von `"Owner": "John"` in der `CreateTable`-Anforderung vorhanden ist, ermöglicht die Inline-Richtlinie dem Benutzer `John`, die Aktion `CreateTable` auszuführen.

Beispiel 3: Eine Aktion mit aws ablehnen: TagKeys

Mithilfe des TagKeys Bedingungsschlüssels aws: können Sie die Tag-Schlüssel in einer Anfrage mit den Schlüsseln vergleichen, die in der IAM-Richtlinie angegeben sind. Sie können beispielsweise eine bestimmte Aktion wie `CreateTable` mithilfe von `aws:TagKeys` verweigern, wenn ein bestimmter Tag-Schlüssel nicht in der Anforderung vorhanden ist. Führen Sie dazu die folgenden Schritte aus:

Using the AWS CLI

1. Fügen Sie eine vom Kunden verwaltete Richtlinie zu einer Rolle hinzu, der die AWS verwaltete AmazonDynamoDBFullAccess-Richtlinie zugeordnet ist, wie im folgenden Beispiel gezeigt.

   JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Deny",
            "Action": [
                "dynamodb:CreateTable",
                "dynamodb:TagResource"
```

```
            ],
            "Resource": "arn:aws:dynamodb:*:*:table/*",
            "Condition": {
                "Null": {
                    "aws:TagKeys": "false"
                },
                "ForAllValues:StringNotEquals": {
                    "aws:TagKeys": "CostCenter"
                }
            }
        }
    ]
}
```

2. Nehmen Sie die Rolle an, der die Richtlinie angehängt wurde, und erstellen Sie eine Tabelle mit dem Tag-Schlüssel `Title`.

```
aws dynamodb create-table \
--attribute-definitions AttributeName=ID,AttributeType=S \
--key-schema AttributeName=ID,KeyType=HASH  \
--provisioned-throughput ReadCapacityUnits=1000,WriteCapacityUnits=500 \
--region us-east-1 \
--tags Key=Title,Value=ProductManager \
--table-name myMusicTable
```

Using the AWS SDK für Python (Boto3)

1. Fügen Sie einer Rolle, der die verwaltete AmazonDynamoDBFullAccess-Richtlinie zugewiesen ist, eine vom Kunden AWS verwaltete Richtlinie hinzu, wie im folgenden Beispiel gezeigt.

   JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Deny",
            "Action": [
                "dynamodb:CreateTable",
                "dynamodb:TagResource"
```

```
            ],
            "Resource": "arn:aws:dynamodb:*:*:table/*",
            "Condition": {
                "Null": {
                    "aws:TagKeys": "false"
                },
                "ForAllValues:StringNotEquals": {
                    "aws:TagKeys": "CostCenter"
                }
            }
        }
    ]
}
```

2. Nehmen Sie die Rolle an, der die Richtlinie angehängt wurde, und erstellen Sie eine Tabelle mit dem Tag-Schlüssel `Title`.

```
ddb_client = boto3.client('dynamodb')

create_table_response = ddb_client.create_table(
    AttributeDefinitions=[
        {
            'AttributeName': 'id',
            'AttributeType': 'S'
        },
    ],
    TableName='myMusicTable',
    KeySchema=[
        {
            'AttributeName': 'id',
            'KeyType': 'HASH'
        },
    ],
        ProvisionedThroughput={
        'ReadCapacityUnits': 1000,
        'WriteCapacityUnits': 500
    },
    Tags=[
        {
            'Key': 'Title',
            'Value': 'ProductManager'
        },
    ],
```

```
)
```

Ohne ABAC

Wenn ABAC für Sie nicht aktiviert ist AWS-Konto, sendet DynamoDB die Tag-Schlüssel im `create-table` Befehl nicht an IAM. Die `Null`-Bedingung stellt sicher, dass die Bedingung als `false` ausgewertet wird, wenn in der Anforderung keine Tags vorhanden sind. Da die `Deny`-Richtlinie nicht übereinstimmt, wird der Befehl `create-table` erfolgreich ausgeführt.

Mit ABAC

Wenn ABAC für Sie aktiviert ist AWS-Konto, werden die im `create-table` Befehl übergebenen Tag-Schlüssel an IAM übergeben. Der Tag-Schlüssel `Title` wird anhand des bedingungsbasierten Tag-Schlüssels `CostCenter` ausgewertet, der in der `Deny`-Richtlinie enthalten ist. Der Tag-Schlüssel `Title` stimmt aufgrund des `StringNotEquals`-Operators nicht mit dem in der `Deny`-Richtlinie vorhandenen Tag-Schlüssel überein. Daher schlägt die `CreateTable`-Aktion fehl und Ihre Tabelle wird nicht erstellt. Wenn Sie den Befehl `create-table` ausführen, wird ein`AccessDeniedException`-Fehler zurückgegeben.

```
An error occurred (AccessDeniedException) when calling the CreateTable operation:
 User: arn:aws:sts::123456789012:assumed-role/DynamoFullAccessRole/ProductManager is
 not authorized to perform: dynamodb:CreateTable on resource: arn:aws:dynamodb:us-
east-1:123456789012:table/myMusicTable with an explicit deny in an identity-based
 policy.
```

Behebung häufiger ABAC-Fehler für DynamoDB-Tabellen und -Indizes

Dieses Thema enthält Hinweise zur Fehlerbehebung bei häufigen Fehlern und Problemen, die bei der Implementierung von ABAC in DynamoDB-Tabellen oder -Indizes auftreten können.

Servicespezifische Bedingungsschlüssel in Richtlinien führen zu einem Fehler

Servicespezifische Bedingungsschlüssel werden nicht als gültige Bedingungsschlüssel betrachtet. Wenn Sie solche Schlüssel in Ihren Richtlinien verwendet haben, führen diese zu einem Fehler. Um dieses Problem zu beheben, müssen Sie die servicespezifischen Bedingungsschlüssel durch einen geeigneten Bedingungsschlüssel für die Implementierung von ABAC in DynamoDB ersetzen.

Nehmen wir beispielsweise an, Sie haben den `dynamodb:ResourceTag` Bedingungsschlüssel in einer Inline-Richtlinie verwendet, die die Anforderung ausführt. PutItem Stellen Sie sich vor, dass die Anforderung mit einem `AccessDeniedException`-Fehler fehlschlägt. Das folgende Beispiel zeigt die fehlerhafte Inline-Richtlinie mit dem `dynamodb:ResourceTag`-Bedingungsschlüssel.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "dynamodb:PutItem"
            ],
            "Resource": "arn:aws:dynamodb:*:*:table/*",
            "Condition": {
                "StringEquals": {
                    "dynamodb:ResourceTag/Owner": "John"
                }
            }
        }
    ]
}
```

Um dieses Problem zu beheben, ersetzen Sie den `dynamodb:ResourceTag`-Bedingungsschlüssel durch `aws:ResourceTag`, wie im folgenden Beispiel gezeigt.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "dynamodb:PutItem"
            ],
            "Resource": "arn:aws:dynamodb:*:*:table/*",
            "Condition": {
```

```
                "StringEquals": {
                    "aws:ResourceTag/Owner": "John"
                }
            }
        }
    ]
}
```

ABAC kann nicht deaktiviert werden

Wenn ABAC für Ihr Konto aktiviert wurde Support, können Sie sich nicht über die DynamoDB-Konsole von ABAC abmelden. Um sich abzumelden, wenden Sie sich an den Support.

Sie können sich nur dann selbst von ABAC abmelden, wenn Folgendes zutrifft:

- Sie haben die Self-Service-Methode verwendet, um sich über die DynamoDB-Konsole anzumelden.
- Sie melden sich innerhalb von sieben Kalendertagen nach der Anmeldung ab.

Datenschutz in DynamoDB

Amazon DynamoDB bietet eine sehr robuste Speicherinfrastruktur, die für geschäftskritische und primäre Speicheranwendungen entwickelt wurde. Daten werden redundant auf mehreren Geräten in verschiedenen Anlagen einer Amazon-DynamoDB-Region gespeichert.

DynamoDB schützt im Ruhezustand gespeicherte Benutzerdaten sowie Daten, die zwischen lokalen Clients und DynamoDB sowie zwischen DynamoDB und anderen Ressourcen innerhalb derselben Region übertragen werden. AWS AWS

Themen

- Ruhende DynamoDB-Verschlüsselung
- Sichern von DynamoDB-Verbindungen mithilfe von VPC-Endpunkten und IAM-Richtlinien

Ruhende DynamoDB-Verschlüsselung

Alle Benutzerdaten, die in Amazon DynamoDB gespeichert sind, werden im Ruhezustand vollständig verschlüsselt. Die DynamoDB-Verschlüsselung im Ruhezustand bietet erhöhte Sicherheit, indem alle Ruhedaten mit Verschlüsselungsschlüsseln verschlüsselt werden, die in AWS Key

Management Service (AWS KMS) gespeichert sind. Diese Funktionalität trägt zur Verringerung des Betriebsaufwands und der Komplexität bei, die mit dem Schutz sensibler Daten einhergeht. Mit der Verschlüsselung ruhender Daten können Sie sicherheitsrelevante Anwendungen erstellen, die eine strenge Einhaltung der Verschlüsselungsvorschriften und der gesetzlichen Bestimmungen erfordern.

Die DynamoDB-Verschlüsselung im Ruhezustand bietet eine zusätzliche Datenschutzebene, indem Ihre Daten stets in einer verschlüsselten Tabelle gesichert werden – einschließlich des Primärschlüssels, lokaler und globaler sekundärer Indizes, Datenströme, globaler Tabellen, Backups und DynamoDB-Accelerator-(DAX)-Clustern –, wenn die Daten auf dauerhaften Datenträgern gespeichert werden. Richtlinien der Organisation, Vorschriften der Branche oder Behörde und Compliance-Anforderungen schreiben oft die Verschlüsselung ruhender Daten vor, um den Datenschutz Ihrer Anwendungen zu erhöhen. Weitere Informationen zur Verschlüsselung für Datenbankanwendungen finden Sie unter AWS Database Encryption SDK.

Encryption at Rest ist in die Verwaltung der Verschlüsselungsschlüssel integriert, die AWS KMS zur Verschlüsselung Ihrer Tabellen verwendet werden. Weitere Informationen zu Schlüsseltypen und -status finden Sie unter AWS Key Management Service -Konzepte im AWS Key Management Service -Entwicklerhandbuch.

Wenn Sie eine neue Tabelle erstellen, können Sie einen der folgenden AWS KMS key Typen zum Verschlüsseln Ihrer Tabelle auswählen. Sie können jederzeit zwischen diesen Schlüsseltypen wechseln.

- AWS-eigener Schlüssel — Standardverschlüsselungstyp. Der Schlüssel befindet sich im Besitz von DynamoDB (kein Aufpreis).
- Von AWS verwalteter Schlüssel — Der Schlüssel ist in Ihrem Konto gespeichert und wird von verwaltet AWS KMS (es AWS KMS fallen Gebühren an).
- Vom Kunden verwalteter Schlüssel – Der Schlüssel wird in Ihrem Konto gespeichert und von Ihnen erstellt, besessen und verwaltet. Sie haben die volle Kontrolle über den KMS-Schlüssel (es AWS KMS fallen Gebühren an).

Weitere Informationen zu Schlüsseltypen finden Sie unter Kundenschlüssel und AWS Schlüssel.

> **Note**
>
> - Beim Erstellen eines neuen DAX-Clusters mit aktivierter Verschlüsselung im Ruhezustand wird ein Von AWS verwalteter Schlüssel verwendet, um Daten im Ruhezustand im Cluster zu verschlüsseln.

- Wenn Ihre Tabelle einen Sortierschlüssel hat, werden einige der Sortierschlüssel, die Bereichsgrenzen markieren, in Klartext in den Metadaten der Tabelle gespeichert.

Wenn Sie auf eine verschlüsselte Tabelle zugreifen, entschlüsselt DynamoDB die Tabellendaten transparent. Sie müssen keinen Code oder Anwendungen ändern, um verschlüsselte Tabellen zu verwenden oder zu verwalten. DynamoDB liefert weiterhin dieselbe Latenz im einstelligen Millisekundenbereich, die Sie gewohnt sind, und alle DynamoDB Abfragen funktionieren nahtlos für Ihre verschlüsselten Daten.

Sie können einen Verschlüsselungsschlüssel angeben, wenn Sie eine neue Tabelle erstellen oder die Verschlüsselungsschlüssel für eine bestehende Tabelle mithilfe der AWS-Managementkonsole, AWS Command Line Interface (AWS CLI) oder der Amazon DynamoDB DynamoDB-API austauschen. Um zu erfahren wie dies geht, vgl. Verwalten von verschlüsselten Tabellen in DynamoDB.

Die Verschlüsselung im Ruhezustand mithilfe von AWS-eigener Schlüssel wird ohne zusätzliche Kosten angeboten. Es AWS KMS fallen jedoch Gebühren für einen Von AWS verwalteter Schlüssel und für einen vom Kunden verwalteten Schlüssel an. Weitere Informationen zur Preisgestaltung finden Sie unter AWS KMS – Preise.

DynamoDB-Verschlüsselung im Ruhezustand ist in allen AWS Regionen verfügbar, einschließlich der Regionen AWS China (Peking) und AWS China (Ningxia) sowie der Regionen AWS GovCloud (USA). Weitere Informationen erhalten Sie unter Verschlüsselung im Ruhezustand mit DynamoDB: Funktionsweise und Nutzungshinweise zur Verschlüsselung ruhender Daten in DynamoDB.

Verschlüsselung im Ruhezustand mit DynamoDB: Funktionsweise

Die Amazon-DynamoDB-Verschlüsselung verschlüsselt Ihre Daten mit 256-bit-Advanced-Encryption-Standard (AES-256), um Ihre Daten vor unautorisiertem Zugriff auf den zugrunde liegenden Speicher zu schützen.

Encryption at Rest ist in AWS Key Management Service (AWS KMS) integriert, um die Verschlüsselungsschlüssel zu verwalten, die zur Verschlüsselung Ihrer Tabellen verwendet werden.

> **Note**
>
> Im Mai 2022 AWS KMS wurde der Rotationsplan Von AWS verwaltete Schlüssel von allen drei Jahren (etwa 1.095 Tage) auf jedes Jahr (ungefähr 365 Tage) geändert.
> Neue Von AWS verwaltete Schlüssel werden automatisch ein Jahr nach ihrer Erstellung und danach ungefähr jedes Jahr rotiert.

Bestehende Von AWS verwaltete Schlüssel werden automatisch ein Jahr nach ihrer letzten Rotation und danach jedes Jahr gewechselt.

AWS-eigene Schlüssel

AWS-eigene Schlüssel sind nicht in Ihrem AWS Konto gespeichert. Sie sind Teil einer Sammlung von KMS-Schlüsseln, die AWS Eigentümer sind und für die Verwendung in mehreren AWS Konten verwaltet werden. AWS Dienste, die Sie AWS-eigene Schlüssel zum Schutz Ihrer Daten verwenden können. AWS-eigene Schlüssel die von DynamoDB verwendet werden, werden jedes Jahr rotiert (ungefähr 365 Tage).

Sie können ihre Verwendung nicht einsehen, verwalten AWS-eigene Schlüssel, verwenden oder überwachen. Sie müssen jedoch keine Maßnahmen ergreifen oder Programme zum Schutz der Schlüssel ändern, die zur Verschlüsselung Ihrer Daten verwendet werden.

Ihnen wird weder eine monatliche Gebühr noch eine Nutzungsgebühr für die Nutzung von berechnet AWS-eigene Schlüssel, und sie werden auch nicht auf die AWS KMS Kontingente für Ihr Konto angerechnet.

Von AWS verwaltete Schlüssel

Von AWS verwaltete Schlüssel sind KMS-Schlüssel in Ihrem Konto, die in Ihrem Namen von einem integrierten AWS Dienst erstellt, verwaltet und verwendet werden. AWS KMS Sie können die Von AWS verwaltete Schlüssel in Ihrem Konto anzeigen, ihre Schlüsselrichtlinien abrufen und ihre Verwendung in AWS CloudTrail -Protokollen prüfen. Sie haben jedoch nicht die Möglichkeit, diese KMS-Schlüssel zu verwalten oder ihre Berechtigungen zu ändern.

Encryption at Rest integriert sich automatisch in die Verwaltung von AWS KMS Von AWS verwaltete Schlüssel for DynamoDB (`aws/dynamodb`), die zum Verschlüsseln Ihrer Tabellen verwendet werden. Wenn beim Erstellen Ihrer verschlüsselten DynamoDB-Tabelle Von AWS verwalteter Schlüssel kein vorhanden ist, AWS KMS wird automatisch ein neuer Schlüssel für Sie erstellt. Dieser Schlüssel wird für verschlüsselte Tabellen verwendet, die in future erstellt werden. AWS KMS kombiniert sichere, hochverfügbare Hardware und Software, um ein für die Cloud skaliertes Schlüsselverwaltungssystem bereitzustellen.

Weitere Informationen zur Verwaltung der Von AWS verwalteter Schlüssel Berechtigungen von finden Sie unter Autorisieren der Verwendung von Von AWS verwalteter Schlüssel im AWS Key Management Service Entwicklerhandbuch.

Kundenseitig verwaltete Schlüssel

Von Kunden verwaltete Schlüssel sind KMS-Schlüssel in Ihrem AWS Konto, die Sie erstellen, besitzen und verwalten. Sie haben die volle Kontrolle über diese KMS-Schlüssel, einschließlich der Festlegung und Pflege ihrer wichtigsten Richtlinien, IAM-Richtlinien und Unterstützungen. Aktivieren und Deaktivieren; Drehen ihres kryptographischen Materials; Hinzufügen von Tags; Erstellen von Aliasen, die auf sie verweisen; und sie zum Löschen planen. Weitere Informationen zum Verwalten von Berechtigungen eines kundenseitig verwalteten Schlüssels finden Sie unter Kundenseitig verwaltete Schlüssel.

Wenn Sie einen kundenverwalteten Schlüssel als Verschlüsselungsschlüssel auf Tabellenebene angeben, werden die DynamoDB-Tabelle, die lokalen und globalen sekundären Indizes und Streams mit demselben kundenverwalteten Schlüssel verschlüsselt. Bei Bedarf werden Backups mit dem Verschlüsselungsschlüssel auf Tabellenebene verschlüsselt, der zum Zeitpunkt der Erstellung der Backup angegeben wird. Beim Aktualisieren des Verschlüsselungsschlüssels auf Tabellenebene wird der Verschlüsselungsschlüssel nicht geändert, der vorhandenen On-Demand-Backups zugeordnet ist.

Durch Festlegen des Status des kundenverwalteten Schlüssels auf Deaktivierung oder Planung für das Löschen wird verhindert, dass alle Benutzer und der DynamoDB-Service Daten verschlüsseln oder entschlüsseln und Lese- und Schreibvorgänge für die Tabelle ausführen können. DynamoDB muss Zugriff auf Ihren Verschlüsselungsschlüssel haben, um sicherzustellen, dass Sie weiterhin auf Ihre Tabelle zugreifen können und Datenverluste zu verhindern.

Wenn Sie den kundenverwalteten Schlüssel deaktivieren oder planen ihn zu Löschen, wird Ihr Tabellenstatus Unzugänglich. Um sicherzustellen, dass Sie die Arbeit mit der Tabelle fortsetzen können, müssen Sie DynamoDB Zugriff auf den angegebenen Verschlüsselungsschlüssel innerhalb von sieben Tagen gewähren. Sobald der Dienst erkennt, dass der Zugriff auf den Verschlüsselungsschlüssel nicht möglich ist, sendet DynamoDB eine E-Mail-Benachrichtigung, um Sie zu benachrichtigen.

> **Note**
>
> - Wenn auf Ihren kundenverwalteten Schlüssel länger als sieben Tage nicht mehr vom DynamoDB-Service zugegriffen werden kann, wird die Tabelle archiviert und es kann nicht mehr darauf zugegriffen werden. DynamoDB erstellt eine On-Demand-Backup Ihrer Tabelle, und es wird Ihnen in Rechnung gestellt. Sie können diese On-Demand-Backup verwenden, um Ihre Daten in einer neuen Tabelle wiederherzustellen. Um die

Wiederherstellung zu initiieren, muss der letzte kundenverwaltete Schlüssel in der Tabelle aktiviert sein, und DynamoDB muss Zugriff darauf haben.

- Wenn Ihr kundenverwalteter Schlüssel, der zum Verschlüsseln eines globalen Tabellenreplikats verwendet wurde, nicht zugänglich ist, entfernt DynamoDB dieses Replikat aus der Replikationsgruppe. Das Replikat wird nicht gelöscht, und die Replikation wird 20 Stunden nach der Erkennung des kundenverwalteten Schlüssels als nicht zugänglich angehalten.

Weitere Informationen finden Sie unter Aktivieren von Schlüsseln und Löschen von Schlüsseln.

Hinweise zur Verwendung Von AWS verwaltete Schlüssel

Amazon DynamoDB kann Ihre Tabellendaten nur lesen, wenn es Zugriff auf den in Ihrem AWS KMS Konto gespeicherten KMS-Schlüssel hat. DynamoDB verwendet die Envelope-Verschlüsselung und die Schlüsselhierarchie, um Daten zu verschlüsseln. Ihr AWS KMS Verschlüsselungsschlüssel wird verwendet, um den Stammschlüssel dieser Schlüsselhierarchie zu verschlüsseln. Weitere Informationen zur Envelope-Verschlüsselung finden Sie im AWS Key Management Service - Entwicklerhandbuch.

DynamoDB ruft nicht AWS KMS für jeden DynamoDB-Vorgang auf. Der Schlüssel wird einmal alle 5 Minuten pro Aufrufer mit aktivem Datenverkehr aktualisiert.

Stellen Sie sicher, dass das SDK für die Wiederverwendung von Verbindungen konfiguriert ist. Andernfalls treten Latenzen auf, da DynamoDB für jeden DynamoDB-Vorgang neue AWS KMS Cacheeinträge neu einrichten muss. Darüber hinaus müssen Sie möglicherweise mit höheren Kosten rechnen. AWS KMS CloudTrail Um dies beispielsweise mit dem Node.js SDK zu tun, können Sie einen neuen HTTPS-Agent mit aktivierter `keepAlive` erstellen. Weitere Informationen finden Sie unter Konfigurieren von keepAlive in Node.js im AWS SDK für JavaScript -Entwicklerhandbuch.

Nutzungshinweise zur Verschlüsselung ruhender Daten in DynamoDB

Beachten Sie Folgendes, wenn Sie die Verschlüsselung ruhender Daten in Amazon DynamoDB verwenden.

Alle Tabellendaten werden verschlüsselt

Die serverseitige Verschlüsselung ruhender Daten ist für alle DynamoDB-Tabellendaten aktiviert und kann nicht deaktiviert werden. Sie können nicht nur eine Teilmenge von Elementen in einer Tabelle verschlüsseln.

Die Verschlüsselung ruhender Daten verschlüsselt Daten nur, während sie auf persistenten Speichermedien statisch (ruhende Daten) sind. Wenn die Datensicherheit für in Übertragung befindliche oder genutzte Daten wichtig ist, müssen Sie ggf. zusätzliche Maßnahmen ergreifen:

- Daten während der Übertragung: Alle Ihre Daten in DynamoDB werden während der Übertragung verschlüsselt. Standardmäßig verwenden Mitteilungen an und von DynamoDB das HTTPS-Protokoll, das den Netzwerkverkehr mittels Secure-Sockets-Layer-(SSL)-/Transport-Layer-Security-(TLS)-Verschlüsselung schützt.
- Genutzte Daten: Schützen Sie Daten durch Client-seitige Verschlüsselung, bevor sie an DynamoDB gesendet werden. Weitere Informationen finden Sie unter Client- und serverseitige Verschlüsselung im Entwicklerhandbuch für Amazon DynamoDB Encryption Client.

Sie können Streams mit verschlüsselten Tabellen verwenden. DynamoDB-Streams werden immer mit einem Verschlüsselungsschlüssel auf Tabellenebene verschlüsselt. Weitere Informationen finden Sie unter Ändern Sie die Datenerfassung für DynamoDB Streams.

DynamoDB-Backups werden verschlüsselt. Für die Tabelle, die aus einem Backup wiederhergestellt wird, ist die Verschlüsselung ebenfalls aktiviert. Sie können den AWS-eigener Schlüssel Von AWS verwalteter Schlüssel, oder vom Kunden verwalteten Schlüssel verwenden, um Ihre Backup-Daten zu verschlüsseln. Weitere Informationen finden Sie unter Backup und Wiederherstellung für DynamoDB.

Lokale Sekundärindizes und globale Sekundärindizes werden mit dem gleichen Schlüssel wie die Basistabelle verschlüsselt.

Verschlüsselungstypen

> **Note**
>
> Vom Kunden verwaltete Schlüssel werden in der globalen Tabelle Version 2017 nicht unterstützt. Wenn Sie einen vom Kunden verwalteten Schlüssel in einer globalen DynamoDB-Tabelle verwenden möchten, müssen Sie die Tabelle auf Version 2019 aktualisieren und dann aktivieren.

Bei der handelt es sich um den Verschlüsselungstyp AWS-Managementkonsole, `KMS` wenn Sie den Von AWS verwalteter Schlüssel oder den vom Kunden verwalteten Schlüssel zum Verschlüsseln Ihrer Daten verwenden. Der Verschlüsselungstyp lautet `DEFAULT`, wenn Sie den AWS-eigener Schlüssel verwenden. In der Amazon DynamoDB DynamoDB-API ist der Verschlüsselungstyp der

Fall, KMS wenn Sie den Von AWS verwalteter Schlüssel oder den vom Kunden verwalteten Schlüssel verwenden. Wenn kein Verschlüsselungstyp angegeben wird, werden Ihre Daten mit dem AWS-eigener Schlüssel verschlüsselt. Sie können jederzeit zwischen dem AWS-eigener Schlüssel Von AWS verwalteter Schlüssel, und dem vom Kunden verwalteten Schlüssel wechseln. Sie können die Konsole, die AWS Command Line Interface (AWS CLI) oder die Amazon DynamoDB DynamoDB-API verwenden, um die Verschlüsselungsschlüssel zu wechseln.

Beachten Sie die folgenden Einschränkungen bei Verwendung kundenverwalteter Schlüssel:

- Sie können einen kundenverwalteten Schlüssel nicht mit den DynamoDB-Accelerator (DAX)-Clustern verwenden. Weitere Informationen finden Sie unter DAX-Verschlüsselung im Ruhezustand.
- Sie können einen kundenverwalteten Schlüssel nutzen, um Tabellen zu verschlüsseln, die Transaktionen verwenden. Um jedoch die Dauerhaftigkeit für die Weitergabe von Transaktionen zu gewährleisten, wird eine Kopie der Transaktionsanforderung vom Dienst vorübergehend gespeichert und mit einem AWS-eigener Schlüssel verschlüsselt. Festgeschriebene Daten in Ihren Tabellen und sekundären Indizes werden im Ruhezustand immer mit Ihrem kundenverwalteten Schlüssel verschlüsselt.
- Sie können einen kundenverwalteten Schlüssel nutzen, um Tabellen zu verschlüsseln, die Contributor Insights verwenden. Daten, an die übertragen werden, sind jedoch mit einem Amazon CloudWatch verschlüsselt. AWS-eigener Schlüssel
- Achten Sie beim Umstieg auf einen neuen, vom Kunden verwalteten Schlüssel darauf, dass der ursprüngliche Schlüssel aktiviert bleibt, bis der Vorgang abgeschlossen ist. AWS Sie benötigen weiterhin den Originalschlüssel, um die Daten zu entschlüsseln, bevor Sie sie mit dem neuen Schlüssel verschlüsseln. Der Vorgang ist abgeschlossen, wenn der SSEDescription Status der Tabelle AKTIVIERT ist und der KMSMaster KeyArn des neuen vom Kunden verwalteten Schlüssels angezeigt wird. Zu diesem Zeitpunkt kann der ursprüngliche Schlüssel deaktiviert oder zum Löschen eingeplant werden.
- Sobald der neue kundenverwaltete Schlüssel angezeigt wird, werden die Tabelle und alle neuen On-Demand-Backups mit dem neuen Schlüssel verschlüsselt.
- Alle vorhandenen On-Demand-Backups bleiben mit dem kundenverwalteten Schlüssel verschlüsselt, der beim Erstellen dieser Backups verwendet wurde. Sie benötigen denselben Schlüssel, um diese Backups wiederherzustellen. Sie können den Schlüssel für den Zeitraum identifizieren, in dem jedes Backup erstellt wurde, indem Sie die DescribeBackup API verwenden, um die Backups einzusehen SSEDescription.

- Wenn Sie den kundenverwalteten Schlüssel deaktivieren oder zum Löschen vorsehen, gilt für alle Daten in DynamoDB Streams weiterhin eine 24-stündige Lebensdauer. Nicht abgerufene Daten, die älter als 24 Stunden sind, können jederzeit entfernt werden.
- Wenn Sie den kundenverwalteten Schlüssel deaktivieren oder zum Löschen planen, werden Time-to-Live (TTL)-Löschungen für 30 Minuten fortgesetzt. Diese TTL-Löschungen werden weiterhin an DynamoDB Streams ausgegeben und unterliegen dem Standardintervall. trimming/retention

  Weitere Informationen finden Sie unter Aktivieren von Schlüsseln und Löschen von Schlüsseln.

Verwenden von KMS-Schlüsseln und Datenschlüsseln

Die DynamoDB-Funktion zur Verschlüsselung im Ruhezustand verwendet eine AWS KMS key und eine Hierarchie von Datenschlüsseln, um Ihre Tabellendaten zu schützen. DynamoDB verwendet die gleiche Schlüsselhierarchie zum Schutz von DynamoDB-Streams, globalen Tabellen und Backups, wenn sie auf beständige Medien geschrieben werden.

Wir empfehlen Ihnen, Ihre Verschlüsselungsstrategie zu planen, bevor Sie Ihre Tabelle in DynamoDB implementieren. Wenn Sie sensible oder vertrauliche Daten in DynamoDB speichern, sollten Sie erwägen, eine clientseitige Verschlüsselung in Ihren Plan aufzunehmen. Auf diese Weise können Sie Daten so nah wie möglich an ihrem Ursprung verschlüsseln und den Schutz der Daten während ihres gesamten Lebenszyklus gewährleisten. Weitere Informationen finden Sie in der Dokumentation zum DynamoDB-Verschlüsselungsclient.

AWS KMS key

Die Verschlüsselung im Ruhezustand schützt Ihre DynamoDB-Tabellen mit einem AWS KMS key. Standardmäßig verwendet DynamoDB einen AWS-eigener Schlüssel, einen Mehrmandanten-Verschlüsselungsschlüssel, der in einem DynamoDB-Servicekonto erstellt und verwaltet wird. Sie können Ihre DynamoDB-Tabellen jedoch mit einem vom Kunden verwalteten Schlüssel für DynamoDB (`aws/dynamodb`) in Ihrem AWS-Konto verschlüsseln. Sie können für jede Tabelle einen anderen KMS-Schlüssel auswählen. Der KMS-Schlüssel, den Sie für eine Tabelle auswählen, wird auch zur Verschlüsselung der lokalen und globalen sekundären Indizes, Streams und Backups verwendet.

Sie wählen den KMS-Schlüssel für eine Tabelle aus, wenn Sie die Tabelle erstellen oder aktualisieren. Sie können den KMS-Schlüssel für eine Tabelle jederzeit ändern, entweder in der DynamoDB-Konsole oder mithilfe des UpdateTableVorgangs. Der Vorgang des Schlüsselwechsels ist nahtlos und erfolgt ohne Ausfallzeiten oder Beeinträchtigung des Service.

> **Important**
>
> DynamoDB unterstützt nur symmetrische KMS-Schlüssel. Sie können keinen asymmetrischen KMS-Schlüssel verwenden, um Ihre DynamoDB-Tabellen zu verschlüsseln.

Verwenden Sie einen kundenverwalteten KMS-Schlüssel, um die folgenden Funktionen zu erhalten:

- Sie erstellen und verwalten den KMS-Schlüssel, einschließlich der Einstellung der Schlüsselrichtlinien, IAM-Richtlinien und Erteilungen, um den Zugriff auf den KMS-Schlüssel zu steuern. Sie können den KMS-Schlüssel aktivieren und deaktivieren, die automatische Schlüsseldrehung aktivieren und deaktivieren und den KMS-Schlüssel löschen, wenn er nicht mehr verwendet wird.
- Sie können einen kundenverwalteten Schlüssel mit importiertem Schlüsselmaterial oder einen kundenverwalteten Schlüssel in einem benutzerdefinierten Schlüsselspeicher verwenden, den Sie besitzen und verwalten.
- Sie können die Verschlüsselung und Entschlüsselung Ihrer DynamoDB-Tabelle überprüfen, indem Sie die DynamoDB-API-Aufrufe in Logs untersuchen. AWS KMSAWS CloudTrail

Verwenden Sie die, Von AWS verwalteter Schlüssel wenn Sie eine der folgenden Funktionen benötigen:

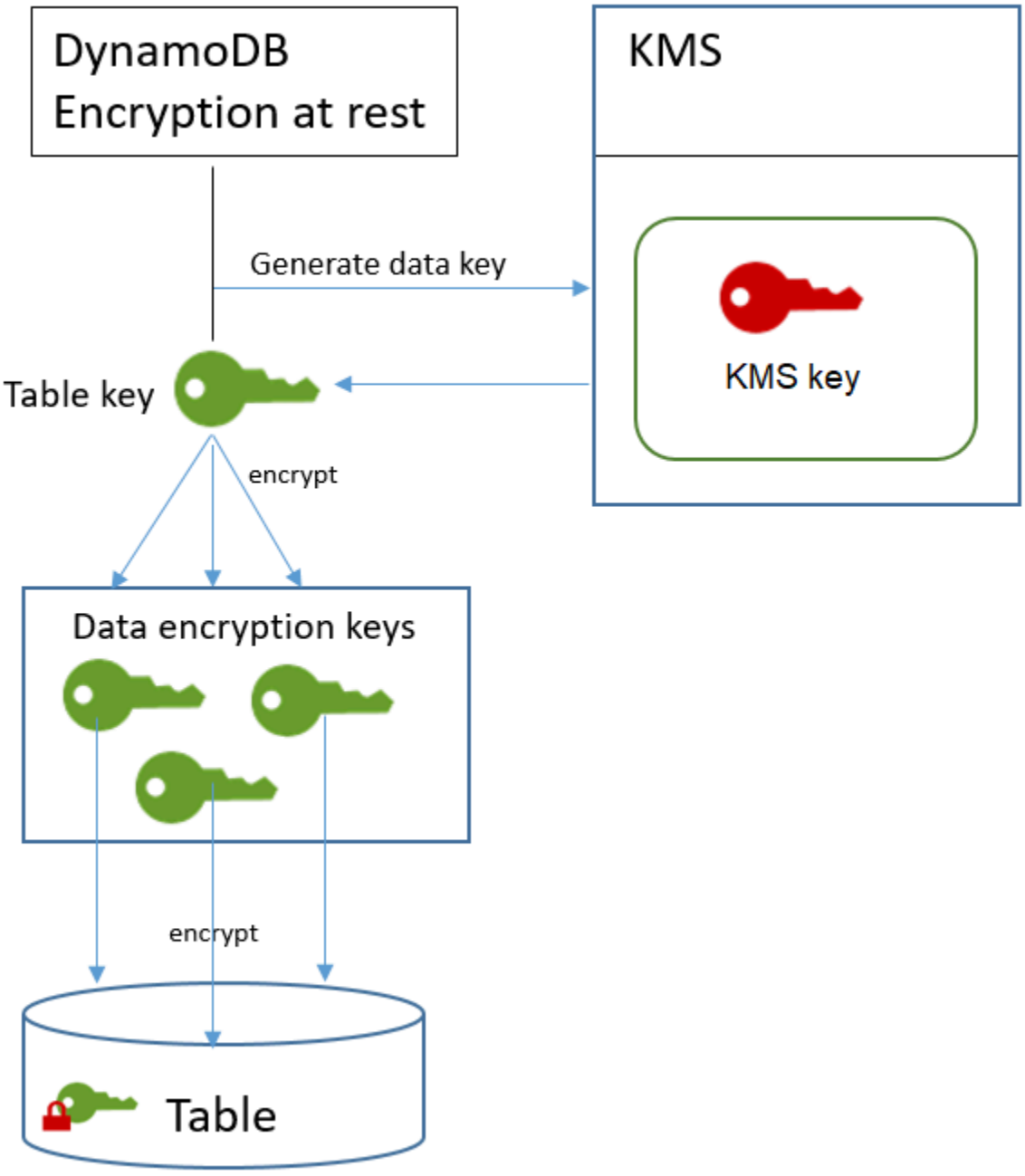
- Sie können den KMS-Schlüssel anzeigen und seine Schlüsselrichtlinie anzeigen. (Sie können die Schlüsselrichtlinie nicht ändern.)
- Sie können die Verschlüsselung und Entschlüsselung Ihrer DynamoDB-Tabelle überprüfen, indem Sie die DynamoDB-API-Aufrufe in Logs untersuchen. AWS KMSAWS CloudTrail

Sie ist jedoch kostenlos und ihre Nutzung AWS-eigener Schlüssel wird nicht auf Ressourcen- oder Anforderungskontingente angerechnet AWS KMS . Vom Kunden verwaltete Schlüssel, für die für jeden API-Aufruf eine Von AWS verwaltete Schlüssel Gebühr anfällt. Für diese KMS-Schlüssel gelten AWS KMS Kontingente.

Tabellenschlüssel

DynamoDB verwendet den KMS-Schlüssel, der als Tabellenschlüssel bezeichnet wird, zum Generieren und Verschlüsseln eines eindeutigen Datenschlüssels für die Tabelle. Der Tabellenschlüssel bleibt für die Lebensdauer der verschlüsselten Tabelle bestehen.

Der Tabellenschlüssel wird als Schlüssel-Verschlüsselungsschlüssel verwendet. DynamoDB verwendet diesen Tabellenschlüssel zum Schutz von Daten-Verschlüsselungsschlüsseln, die zum Verschlüsseln der Tabellendaten verwendet werden. DynamoDB generiert einen eindeutigen Datenverschlüsselungsschlüssel für jede zugrunde liegende Struktur in einer Tabelle, mehrere Tabellenelemente können jedoch vom gleichen Datenverschlüsselungsschlüssel geschützt werden.



Wenn Sie zum ersten Mal auf eine verschlüsselte Tabelle zugreifen, sendet DynamoDB eine Anfrage an, den KMS-Schlüssel AWS KMS zum Entschlüsseln des Tabellenschlüssels zu verwenden. Anschließend wird der Klartext-Tabellenschlüssel zum Entschlüsseln der Datenverschlüsselungsschlüssel verwendet und die Klartext-Datenverschlüsselungsschlüssel werden zum Entschlüsseln der Tabellendaten verwendet.

DynamoDB speichert und verwendet den Tabellenschlüssel und die Datenverschlüsselungsschlüssel außerhalb von. AWS KMS Alle Schlüssel werden mit Advanced Encryption Standard (AES)-Verschlüsselung und 256-Bit-Verschlüsselungsschlüsseln geschützt. Anschließend werden die verschlüsselten Schlüssel zusammen mit den verschlüsselten Daten gespeichert, damit sie bei Bedarf für die Entschlüsselung der Tabellendaten verfügbar sind.

Wenn Sie den KMS-Schlüssel für Ihre Tabelle ändern, generiert DynamoDB einen neuen Tabellenschlüssel. Anschließend wird der neue Tabellenschlüssel verwendet, um die Datenverschlüsselungsschlüssel neu zu verschlüsseln.

Tabellenschlüssel-Caching

Um zu vermeiden, dass AWS KMS jede DynamoDB-Operation aufgerufen wird, speichert DynamoDB die Klartext-Tabellenschlüssel für jeden Aufrufer im Speicher zwischen. Wenn DynamoDB nach fünf Minuten Inaktivität eine Anfrage für den zwischengespeicherten Tabellenschlüssel erhält, sendet es eine neue Anfrage an, um den Tabellenschlüssel AWS KMS zu entschlüsseln. Dieser Aufruf erfasst alle Änderungen, die seit der letzten Anforderung zur Entschlüsselung des Tabellenschlüssels an den Zugriffsrichtlinien des KMS-Schlüssels in AWS KMS oder AWS Identity and Access Management (IAM) vorgenommen wurden.

Autorisieren der Nutzung Ihres KMS-Schlüssels

Wenn Sie einen kundenverwalteten Schlüssel oder den Von AWS verwalteter Schlüssel-verwalteten Schlüssel in Ihrem Konto zum Schutz Ihrer DynamoDB-Tabelle verwenden, müssen die Richtlinien für diesen KMS-Schlüssel DynamoDB zu seiner Verwendung in Ihrem Namen berechtigen. Der Autorisierungskontext auf dem Von AWS verwalteter Schlüssel für DynamoDB umfasst seine wichtigsten Richtlinien und Berechtigungen, mit denen die Berechtigungen zu seiner Verwendung delegiert werden.

Sie haben die volle Kontrolle über die Richtlinien und Erteilungen für einen kundenverwalteten Schlüssel. Nachdem der Von AWS verwalteter Schlüssel in Ihrem Konto ist, können Sie die seine Richtlinien und Erteilungen anzeigen. Da er jedoch von verwaltet wird AWS, können Sie die Richtlinien nicht ändern.

DynamoDB benötigt keine zusätzliche Autorisierung, um den Standard zum Schutz der DynamoDB-Tabellen in Ihrem AWS-eigener Schlüsselzu verwenden. AWS-Konto

Themen

- Wichtige Richtlinie für ein Von AWS verwalteter Schlüssel
- Schlüsselrichtlinie für einen kundenverwalteten Schlüssel
- Verwenden von Erteilungen zum Autorisieren von DynamoDB

Wichtige Richtlinie für ein Von AWS verwalteter Schlüssel

Wenn DynamoDB den Von AWS verwalteter Schlüssel für DynamoDB (`aws/dynamodb`) in kryptografischen Operationen verwendet, geschieht dies im Auftrag des Benutzers, der auf die DynamoDB-Ressource zugreift. Die wichtigste Richtlinie für Von AWS verwalteter Schlüssel gibt allen Benutzern des Kontos die Erlaubnis, das Von AWS verwalteter Schlüssel für bestimmte Operationen zu verwenden. Die Berechtigung wird aber nur gewährt, wenn DynamoDB die Anforderung für den Benutzer ausgibt. Die ViaService Bedingung in der Schlüsselrichtlinie erlaubt es keinem Benutzer, die zu verwenden, Von AWS verwalteter Schlüssel es sei denn, die Anforderung stammt vom DynamoDB-Dienst.

Diese wichtige Richtlinie wurde, wie alle Richtlinien Von AWS verwaltete Schlüssel, von festgelegt. AWS Sie können sie nicht ändern, aber jederzeit anzeigen. Details dazu finden Sie unter Anzeigen einer Schlüsselrichtlinie.

Die Richtlinienanweisungen in der Schlüsselrichtlinie haben folgende Wirkungen:

- Erlauben Sie Benutzern im Konto, Von AWS verwalteter Schlüssel for DynamoDB in kryptografischen Vorgängen zu verwenden, wenn die Anforderung von DynamoDB in ihrem Namen kommt. Die Richtlinie erlaubt Benutzern auch das Erstellen von Erteilungen für den KMS-Schlüssel.
- Ermöglicht es autorisierten IAM-Identitäten im Konto, die Eigenschaften des Von AWS verwalteter Schlüssel für DynamoDB anzuzeigen und die Erteilung zu widerrufen, die DynamoDB die Verwendung des KMS-Schlüssels erlaubt. DynamoDB verwendet Erteilungen für laufende Wartungsoperationen.
- Ermöglicht DynamoDB, schreibgeschützte Operationen durchzuführen, um die Von AWS verwalteter Schlüssel für DynamoDB in Ihrem Konto zu finden.

JSON

```
{
  "Version":"2012-10-17",
  "Id" : "auto-dynamodb-1",
  "Statement" : [ {
    "Sid" : "Allow access through Amazon DynamoDB for all principals in the
 account that are authorized to use Amazon DynamoDB",
    "Effect" : "Allow",
    "Principal" : {
      "AWS" : "*"
    },
    "Action" : [ "kms:Encrypt", "kms:Decrypt", "kms:ReEncrypt*",
 "kms:GenerateDataKey*", "kms:CreateGrant", "kms:DescribeKey" ],
    "Resource" : "*",
    "Condition" : {
      "StringEquals" : {
        "kms:CallerAccount" : "111122223333",
        "kms:ViaService" : "dynamodb.us-west-2.amazonaws.com"
      }
    }
  }, {
    "Sid" : "Allow direct access to key metadata to the account",
    "Effect" : "Allow",
    "Principal" : {
      "AWS" : "arn:aws:iam::111122223333:root"
    },
    "Action" : [ "kms:Describe*", "kms:Get*", "kms:List*", "kms:RevokeGrant" ],
    "Resource" : "*"
  }, {
    "Sid" : "Allow DynamoDB Service with service principal name
 dynamodb.amazonaws.com to describe the key directly",
    "Effect" : "Allow",
    "Principal" : {
      "Service" : "dynamodb.amazonaws.com"
    },
    "Action" : [ "kms:Describe*", "kms:Get*", "kms:List*" ],
    "Resource" : "*"
  } ]
}
```

Schlüsselrichtlinie für einen kundenverwalteten Schlüssel

Wenn Sie einen kundenverwalteten KMS-Schlüssel zum Schutz einer DynamoDB-Tabelle auswählen, erhält DynamoDB die Berechtigung, den KMS-Schlüssel im Namen des Prinzipals zu verwenden, der die Auswahl trifft. Dieser Prinzipal, ein Benutzer oder eine Rolle, muss über die Berechtigungen für den KMS-Schlüssel verfügen, die DynamoDB benötigt. Sie können diese Berechtigungen in einer Schlüsselrichtlinie, einer IAM-Richtlinie oder einer Erteilung bereitstellen.

DynamoDB erfordert mindestens die folgenden Berechtigungen für einen kundenverwalteten Schlüssel:

- kms:Encrypt
- kms:Decrypt
- kms: * (für und) ReEncrypt kms: ReEncryptFrom kms: ReEncryptTo
- kms: GenerateDataKey * (für kms: GenerateDataKey und kms: GenerateDataKeyWithoutPlaintext)
- km: DescribeKey
- km: CreateGrant

Beispielsweise bietet die folgende Beispiel-Schlüsselrichtlinie nur die erforderlichen Berechtigungen. Die Richtlinie hat folgende Auswirkungen:

- Erlaubt DynamoDB die Verwendung des KMS-Schlüssels in kryptografischen Operationen und das Erstellen von Erteilungen, jedoch nur, wenn es im Auftrag von Prinzipalen im Konto handelt, die über die Berechtigung zur Verwendung von DynamoDB verfügen. Wenn die in der Richtlinienanweisung angegebenen Prinzipale nicht zur Verwendung von DynamoDB berechtigt sind, schlägt der Aufruf selbst dann fehl, wenn er vom DynamoDB-Service stammt.
- Der ViaService Bedingungsschlüssel kms: erlaubt die Berechtigungen nur, wenn die Anforderung von DynamoDB im Namen der in der Richtlinienerklärung aufgeführten Prinzipale kommt. Diese Prinzipale können diese Operationen nicht direkt aufrufen. Beachten Sie, dass der `kms:ViaService`-Wert, `dynamodb.*.amazonaws.com`, in der Region-Position ein Sternchen (*) hat. DynamoDB benötigt die Berechtigung, unabhängig von einem bestimmten Objekt zu sein, AWS-Region damit es regionsübergreifende Aufrufe zur Unterstützung globaler DynamoDB-Tabellen durchführen kann.
- Gewährt den KMS-Schlüsseladministratoren (Benutzer, die die `db-team`-Rolle annehmen können) schreibgeschützten Zugriff auf den KMS-Schlüssel und die Berechtigung, Erteilungen zu widerrufen, einschließlich der Erteilungen, die DynamoDB zum Schutz der Tabelle benötigt.

Bevor Sie eine Beispielschlüsselrichtlinie verwenden, ersetzen Sie die Beispielprinzipale durch tatsächliche Prinzipale aus Ihrer. AWS-Konto

JSON

```
{
  "Id": "key-policy-dynamodb",
  "Version":"2012-10-17",
  "Statement": [
    {
      "Sid" : "Allow access through Amazon DynamoDB for all principals in the
 account that are authorized to use Amazon DynamoDB",
      "Effect": "Allow",
      "Principal": {"AWS": "arn:aws:iam::111122223333:user/db-lead"},
      "Action": [
        "kms:Encrypt",
        "kms:Decrypt",
        "kms:ReEncrypt*",
        "kms:GenerateDataKey*",
        "kms:DescribeKey",
        "kms:CreateGrant"
      ],
      "Resource": "*",
      "Condition": {
         "StringLike": {
           "kms:ViaService" : "dynamodb.*.amazonaws.com"
         }
      }
    },
    {
      "Sid":  "Allow administrators to view the KMS key and revoke grants",
      "Effect": "Allow",
      "Principal": {
        "AWS": "arn:aws:iam::111122223333:role/db-team"
       },
      "Action": [
        "kms:Describe*",
        "kms:Get*",
        "kms:List*",
        "kms:RevokeGrant"
      ],
      "Resource": "*"
    }
```

```
  ]
}
```

Verwenden von Erteilungen zum Autorisieren von DynamoDB

Zusätzlich zu den Schlüsselrichtlinien verwendet DynamoDB Erteilungen, um Berechtigungen für einen kundenverwalteten KMS-Schlüssel oder den Von AWS verwalteter Schlüssel -verwalteten Schlüssel für DynamoDB festzulegen (`aws/dynamodb`). Verwenden Sie den Vorgang, um die Zuweisungen für einen KMS-Schlüssel in Ihrem Konto anzuzeigen. ListGrants DynamoDB benötigt keine Erteilungen oder zusätzliche Berechtigungen, um die AWS-eigener Schlüssel zum Schutz Ihrer Tabellen zu verwenden.

DynamoDB verwendet die Berechtigungen aus der Erteilung zur Ausführung von Hintergrundsystemwartung und kontinuierlichen Datenschutzaufgaben. Außerdem werden Erteilungen verwenden, um Tabellenschlüssel zu generieren.

Jede Erteilung ist tabellenspezifisch. Wenn in Ihrem Konto mehrere Tabellen mit demselben KMS-Schlüssel verschlüsselt sind, ist eine Erteilung für jede Art von Tabelle vorhanden. Die Gewährung wird durch den DynamoDB-Verschlüsselungskontext eingeschränkt, der den Tabellennamen und die AWS-Konto ID umfasst, und sie beinhaltet die Berechtigung, die Grant zurückzuziehen, wenn sie nicht mehr benötigt wird.

Zum Erstellen von Erteilungen muss DynamoDB zum Aufrufen von `CreateGrant` im Auftrag des Benutzers berechtigt sein, der die verschlüsselte Tabelle erstellt hat. Denn Von AWS verwaltete Schlüssel DynamoDB erhält die `kms:CreateGrant` Erlaubnis von der Schlüsselrichtlinie, die es Kontonutzern erlaubt, den KMS-Schlüssel nur dann CreateGrantaufzurufen, wenn DynamoDB die Anfrage im Namen eines autorisierten Benutzers stellt.

Die Schlüsselrichtlinie kann es dem Konto auch erlauben, die Erteilung für den KMS-Schlüssel zu widerrufen. Wenn Sie jedoch die Erteilung für eine aktive verschlüsselte Tabelle widerrufen, kann DynamoDB die Tabelle nicht mehr schützen und pflegen.

DynamoDB-Verschlüsselungsclient

Ein Verschlüsselungskontext ist eine Gruppe von Schlüssel/Wert-Paaren mit willkürlichen, nicht geheimen Daten. Wenn Sie einen Verschlüsselungskontext in eine Datenverschlüsselungsanforderung aufnehmen, wird der Verschlüsselungskontext AWS KMS kryptografisch an die verschlüsselten Daten gebunden. Zur Entschlüsselung der Daten müssen Sie denselben Verschlüsselungskontext übergeben.

DynamoDB verwendet bei allen AWS KMS kryptografischen Vorgängen denselben Verschlüsselungskontext. Wenn Sie einen kundenverwalteten Schlüssel oder einen Von AWS verwalteter Schlüssel-verwalteten Schlüssel zum Schutz Ihrer DynamoDB-Tabelle verwenden, können Sie anhand des Verschlüsselungskontexts die Verwendung des KMS-Schlüssels in Prüfungs-Datensätzen und -Protokollen identifizieren. Es erscheint auch im Klartext in Protokollen wie AWS CloudTrailAmazon CloudWatch Logs.

Der Verschlüsselungskontext kann auch als Bedingung für die Autorisierung in Richtlinien und Erteilungen verwendet werden. DynamoDB verwendet den Verschlüsselungskontext, um die Berechtigungen einzuschränken, die den Zugriff auf den vom Kunden verwalteten Schlüssel oder Von AWS verwalteter Schlüssel in Ihrem Konto und Ihrer Region ermöglichen.

In seinen Anfragen an AWS KMS verwendet DynamoDB einen Verschlüsselungskontext mit zwei Schlüssel-Wert-Paaren.

```
"encryptionContextSubset": {
    "aws:dynamodb:tableName": "Books"
    "aws:dynamodb:subscriberId": "111122223333"
}
```

- Tabelle: Das erste Schlüssel-Wert-Paar gibt die Tabelle an, die DynamoDB verschlüsselt. Der Schlüssel lautet `aws:dynamodb:tableName`. Der Wert ist der Name der Tabelle.

  ```
  "aws:dynamodb:tableName": "<table-name>"
  ```

  Beispiel:

  ```
  "aws:dynamodb:tableName": "Books"
  ```

- Konto – Das zweite Schlüssel-Wert-Paar identifiziert das AWS-Konto. Der Schlüssel lautet `aws:dynamodb:subscriberId`. Der Wert ist die Konto-ID.

  ```
  "aws:dynamodb:subscriberId": "<account-id>"
  ```

  Beispiel:

  ```
  "aws:dynamodb:subscriberId": "111122223333"
  ```

Überwachen der DynamoDB-Interaktion mit AWS KMS

Wenn Sie einen vom Kunden verwalteten Schlüssel oder einen Von AWS verwalteter Schlüsselzum Schutz Ihrer DynamoDB-Tabellen verwenden, können Sie mithilfe von AWS CloudTrail Protokollen die Anfragen verfolgen, an die DynamoDB in Ihrem Namen sendet. AWS KMS

Die Anforderungen `GenerateDataKey`, `Decrypt` und `CreateGrant` werden in diesem Abschnitt beschrieben. Darüber hinaus ermittelt DynamoDB mithilfe eines DescribeKeyVorgangs, ob der von Ihnen ausgewählte KMS-Schlüssel im Konto und in der Region vorhanden ist. Außerdem wird ein RetireGrantVorgang verwendet, um einen Zuschuss zu entfernen, wenn Sie eine Tabelle löschen.

GenerateDataKey

Wenn Sie die Verschlüsselung im Ruhezustand für eine Tabelle aktivieren, erstellt DynamoDB einen eindeutigen Tabellenschlüssel. Es sendet eine GenerateDataKeyAnfrage an AWS KMS , in der der KMS-Schlüssel für die Tabelle angegeben wird.

Das Ereignis, das die `GenerateDataKey`-Operation aufzeichnet, ähnelt dem folgenden Beispielereignis. Der Benutzer ist das DynamoDB-Servicekonto. Die Parameter enthalten den Amazon-Ressourcennamen (ARN) des KMS-Schlüssels, einen Schlüsselbezeichner, der einen 256-Bit-Schlüssel benötigt, und den Verschlüsselungskontext, der die Tabelle und das AWS-Konto identifiziert.

```
{
    "eventVersion": "1.05",
    "userIdentity": {
        "type": "AWSService",
        "invokedBy": "dynamodb.amazonaws.com"
    },
    "eventTime": "2018-02-14T00:15:17Z",
    "eventSource": "kms.amazonaws.com",
    "eventName": "GenerateDataKey",
    "awsRegion": "us-west-2",
    "sourceIPAddress": "dynamodb.amazonaws.com",
    "userAgent": "dynamodb.amazonaws.com",
    "requestParameters": {
        "encryptionContext": {
            "aws:dynamodb:tableName": "Services",
            "aws:dynamodb:subscriberId": "111122223333"
        },
        "keySpec": "AES_256",
        "keyId": "1234abcd-12ab-34cd-56ef-1234567890ab"
```

```
    },
    "responseElements": null,
    "requestID": "229386c1-111c-11e8-9e21-c11ed5a52190",
    "eventID": "e3c436e9-ebca-494e-9457-8123a1f5e979",
    "readOnly": true,
    "resources": [
        {
            "ARN": "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab",
            "accountId": "111122223333",
            "type": "AWS::KMS::Key"
        }
    ],
    "eventType": "AwsApiCall",
    "recipientAccountId": "111122223333",
    "sharedEventID": "bf915fa6-6ceb-4659-8912-e36b69846aad"
}
```


Decrypt

Wenn Sie auf eine verschlüsselte DynamoDB-Tabelle zugreifen, muss DynamoDB den Tabellenschlüssel entschlüsseln, damit die in der Hierarchie tiefer angeordneten Schlüssel entschlüsselt werden können. Anschließend werden die Daten in der Tabelle entschlüsselt. Entschlüsseln des Tabellenschlüssels DynamoDB sendet eine Decrypt-Anfrage an AWS KMS , die den KMS-Schlüssel für die Tabelle angibt.

Das Ereignis, das die `Decrypt`-Operation aufzeichnet, ähnelt dem folgenden Beispielereignis. Der Benutzer ist Ihr Hauptbenutzer, der auf AWS-Konto die Tabelle zugreift. Zu den Parametern gehören der verschlüsselte Tabellenschlüssel (als Chiffretext-Blob) und der Verschlüsselungskontext, der die Tabelle und den identifiziert. AWS-Konto AWS KMS leitet die ID des KMS-Schlüssels aus dem Chiffretext ab.


```
{
    "eventVersion": "1.05",
    "userIdentity": {
        "type": "AssumedRole",
        "principalId": "AROAIGDTESTANDEXAMPLE:user01",
        "arn": "arn:aws:sts::111122223333:assumed-role/Admin/user01",
        "accountId": "111122223333",
        "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
        "sessionContext": {
            "attributes": {
```

```
                "mfaAuthenticated": "false",
                "creationDate": "2018-02-14T16:42:15Z"
            },
            "sessionIssuer": {
                "type": "Role",
                "principalId": "AROAIGDT3HGFQZX4RY6RU",
                "arn": "arn:aws:iam::111122223333:role/Admin",
                "accountId": "111122223333",
                "userName": "Admin"
            }
        },
        "invokedBy": "dynamodb.amazonaws.com"
    },
    "eventTime": "2018-02-14T16:42:39Z",
    "eventSource": "kms.amazonaws.com",
    "eventName": "Decrypt",
    "awsRegion": "us-west-2",
    "sourceIPAddress": "dynamodb.amazonaws.com",
    "userAgent": "dynamodb.amazonaws.com",
    "requestParameters":
    {
        "encryptionContext":
        {
            "aws:dynamodb:tableName": "Books",
            "aws:dynamodb:subscriberId": "111122223333"
        }
    },
    "responseElements": null,
    "requestID": "11cab293-11a6-11e8-8386-13160d3e5db5",
    "eventID": "b7d16574-e887-4b5b-a064-bf92f8ec9ad3",
    "readOnly": true,
    "resources": [
        {
            "ARN": "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab",
            "accountId": "111122223333",
            "type": "AWS::KMS::Key"
        }
    ],
    "eventType": "AwsApiCall",
    "recipientAccountId": "111122223333"
}
```

CreateGrant

Wenn Sie einen kundenverwalteten Schlüssel oder einen Von AWS verwalteter Schlüssel zum Schutz Ihrer DynamoDB-Tabelle verwenden, verwendet DynamoDB Erteilungen, um dem Service die Ausführung von Aufgaben für kontinuierlichen Datenschutz, Wartung und Haltbarkeit zu erlauben. Diese Erteilungen sind für AWS-eigener Schlüssel nicht erforderlich.

Die von DynamoDB erstellten Erteilungen sind tabellenspezifisch. Der Principal in der CreateGrantAnfrage ist der Benutzer, der die Tabelle erstellt hat.

Das Ereignis, das die `CreateGrant`-Operation aufzeichnet, ähnelt dem folgenden Beispielereignis. Die Parameter enthalten den Amazon-Ressourcennamen (ARN) des KMS-Schlüssels für die Tabelle, den Empfänger-Prinzipal und den ausscheidenden Prinzipal (der DynamoDB-Service) sowie die Operationen, für die die Erteilung gilt. Enthalten ist auch eine Beschränkung, die für alle Verschlüsselungsoperationen den angegebenen Verschlüsselungskontext voraussetzt.

```
{
    "eventVersion": "1.05",
    "userIdentity":
    {
        "type": "AssumedRole",
        "principalId": "AROAIGDTESTANDEXAMPLE:user01",
        "arn": "arn:aws:sts::111122223333:assumed-role/Admin/user01",
        "accountId": "111122223333",
        "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
        "sessionContext": {
            "attributes": {
                "mfaAuthenticated": "false",
                "creationDate": "2018-02-14T00:12:02Z"
            },
            "sessionIssuer": {
                "type": "Role",
                "principalId": "AROAIGDTESTANDEXAMPLE",
                "arn": "arn:aws:iam::111122223333:role/Admin",
                "accountId": "111122223333",
                "userName": "Admin"
            }
        },
        "invokedBy": "dynamodb.amazonaws.com"
    },
    "eventTime": "2018-02-14T00:15:15Z",
```

```
    "eventSource": "kms.amazonaws.com",
    "eventName": "CreateGrant",
    "awsRegion": "us-west-2",
    "sourceIPAddress": "dynamodb.amazonaws.com",
    "userAgent": "dynamodb.amazonaws.com",
    "requestParameters": {
        "keyId": "1234abcd-12ab-34cd-56ef-1234567890ab",
        "retiringPrincipal": "dynamodb.us-west-2.amazonaws.com",
        "constraints": {
            "encryptionContextSubset": {
                "aws:dynamodb:tableName": "Books",
                "aws:dynamodb:subscriberId": "111122223333"
            }
        },
        "granteePrincipal": "dynamodb.us-west-2.amazonaws.com",
        "operations": [
            "DescribeKey",
            "GenerateDataKey",
            "Decrypt",
            "Encrypt",
            "ReEncryptFrom",
            "ReEncryptTo",
            "RetireGrant"
        ]
    },
    "responseElements": {
        "grantId":
 "5c5cd4a3d68e65e77795f5ccc2516dff057308172b0cd107c85b5215c6e48bde"
    },
    "requestID": "2192b82a-111c-11e8-a528-f398979205d8",
    "eventID": "a03d65c3-9fee-4111-9816-8bf96b73df01",
    "readOnly": false,
    "resources": [
        {
            "ARN": "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab",
            "accountId": "111122223333",
            "type": "AWS::KMS::Key"
        }
    ],
    "eventType": "AwsApiCall",
    "recipientAccountId": "111122223333"
}
```

Verwalten von verschlüsselten Tabellen in DynamoDB

Sie können das AWS-Managementkonsole oder das AWS Command Line Interface (AWS CLI) verwenden, um den Verschlüsselungsschlüssel für neue Tabellen anzugeben und die Verschlüsselungsschlüssel für bestehende Tabellen in Amazon DynamoDB zu aktualisieren.

Themen

- Angeben des Verschlüsselungsschlüssels für eine neue Tabelle
- Aktualisieren eines Verschlüsselungsschlüssels

Angeben des Verschlüsselungsschlüssels für eine neue Tabelle

Führen Sie die folgenden Schritte aus, um den Verschlüsselungsschlüssel für eine neue Tabelle mithilfe der Amazon-DynamoDB-Konsole oder AWS CLI.

Erstellen einer verschlüsselten Tabelle (Konsole)

1. Melden Sie sich bei der an AWS-Managementkonsole und öffnen Sie die DynamoDB-Konsole unter. https://console.aws.amazon.com/dynamodb/
2. Klicken Sie im Navigationsbereich auf der linken Seite der Konsole auf Tabellen.
3. Wählen Sie Create Table (Tabelle erstellen) aus. Geben Sie für Table name (Tabellenname) den Namen **`Music`** ein. Geben Sie als Primärschlüssel **`Artist`** und als Sortierschlüssel **`SongTitle`** ein, beide als Zeichenfolgen.
4. Stellen sie in Einstellungen sicher, dass Einstellungen anpassen ausgewählt ist.

   > **Note**
   >
   > Wenn „Standardeinstellungen verwenden“ ausgewählt ist, werden Tabellen im Ruhezustand ohne zusätzliche AWS-eigener Schlüssel Kosten verschlüsselt.

5. Wählen Sie unter Verschlüsselung im Ruhezustand einen Verschlüsselungstyp - AWS-eigener Schlüssel Von AWS verwalteter Schlüssel, oder einen vom Kunden verwalteten Schlüssel aus.

   - Gehört Amazon DynamoDB. AWS eigener Schlüssel, der speziell DynamoDB gehört und von DynamoDB verwaltet wird. Für die Verwendung dieses Schlüssels wird Ihnen keine zusätzliche Gebühr berechnet.

- AWS verwalteter Schlüssel. Schlüssel-Alias: `aws/dynamodb` Der Schlüssel ist in Ihrem Konto gespeichert und wird von AWS Key Management Service (AWS KMS) verwaltet. AWS KMS Es fallen Gebühren an.
- In Ihrem Konto gespeichert, in Ihrem Besitz und von Ihnen verwaltet. Vom Kunden verwalteter Schlüssel. Der Schlüssel ist in Ihrem Konto gespeichert und wird von AWS Key Management Service (AWS KMS) verwaltet. AWS KMS Es fallen Gebühren an.

> **Note**
>
> Wenn Sie Ihren eigenen Schlüssel besitzen und verwalten möchten, stellen Sie sicher, dass die KMS-Schlüsselrichtlinie ordnungsgemäß festgelegt ist. Weitere Informationen sowie Beispiele finden Sie unter Schlüsselrichtlinie für einen kundenverwalteten Schlüssel.

6. Wählen Sie Create table (Tabelle erstellen) , um die verschlüsselte Tabelle zu erstellen. Um den Verschlüsselungstyp zu bestimmen, wählen Sie die Tabellendetails auf der Registerkarte Übersicht und überprüfen Sie den Abschnitt Weitere Details.

Erstellen einer verschlüsselten Tabelle (AWS CLI)

Verwenden Sie die AWS CLI , um eine Tabelle mit dem Standard- AWS-eigener Schlüssel Von AWS verwalteter Schlüssel, dem oder einem vom Kunden verwalteten Schlüssel für Amazon DynamoDB zu erstellen.

Um eine verschlüsselte Tabelle mit dem Standard zu erstellen AWS-eigener Schlüssel

- Erstellen Sie die verschlüsselte `Music`-Tabelle folgendermaßen.

```
aws dynamodb create-table \
  --table-name Music \
  --attribute-definitions \
      AttributeName=Artist,AttributeType=S \
      AttributeName=SongTitle,AttributeType=S \
  --key-schema \
      AttributeName=Artist,KeyType=HASH \
      AttributeName=SongTitle,KeyType=RANGE \
  --provisioned-throughput \
      ReadCapacityUnits=10,WriteCapacityUnits=5
```

> **Note**
>
> Diese Tabelle ist jetzt mit dem Standard AWS-eigener Schlüssel im DynamoDB-Dienstkonto verschlüsselt.

Um eine verschlüsselte Tabelle mit dem Von AWS verwalteter Schlüssel für DynamoDB zu erstellen

- Erstellen Sie die verschlüsselte `Music`-Tabelle folgendermaßen.

```
aws dynamodb create-table \
  --table-name Music \
  --attribute-definitions \
      AttributeName=Artist,AttributeType=S \
      AttributeName=SongTitle,AttributeType=S \
  --key-schema \
      AttributeName=Artist,KeyType=HASH \
      AttributeName=SongTitle,KeyType=RANGE \
  --provisioned-throughput \
      ReadCapacityUnits=10,WriteCapacityUnits=5 \
  --sse-specification Enabled=true,SSEType=KMS
```

  Der `SSEDescription`-Status der Tabellenbeschreibung ist auf `ENABLED` festgelegt und der `SSEType` ist `KMS`.

```
"SSEDescription": {
  "SSEType": "KMS",
  "Status": "ENABLED",
  "KMSMasterKeyArn": "arn:aws:kms:us-east-1:123456789012:key/abcd1234-abcd-1234-
a123-ab1234a1b234",
}
```

So erstellen Sie eine verschlüsselte Tabelle mit einem kundenverwalteten Schlüssel für DynamoDB

- Erstellen Sie die verschlüsselte `Music`-Tabelle folgendermaßen.

```
aws dynamodb create-table \
  --table-name Music \
  --attribute-definitions \
```

```
      AttributeName=Artist,AttributeType=S \
      AttributeName=SongTitle,AttributeType=S \
  --key-schema \
      AttributeName=Artist,KeyType=HASH \
      AttributeName=SongTitle,KeyType=RANGE \
  --provisioned-throughput \
      ReadCapacityUnits=10,WriteCapacityUnits=5 \
  --sse-specification Enabled=true,SSEType=KMS,KMSMasterKeyId=abcd1234-abcd-1234-
a123-ab1234a1b234
```

Der `SSEDescription`-Status der Tabellenbeschreibung ist auf `ENABLED` festgelegt und der `SSEType` ist `KMS`.

```
"SSEDescription": {
  "SSEType": "KMS",
  "Status": "ENABLED",
  "KMSMasterKeyArn": "arn:aws:kms:us-east-1:123456789012:key/abcd1234-abcd-1234-
a123-ab1234a1b234",
}
```

Aktualisieren eines Verschlüsselungsschlüssels

Sie können auch die DynamoDB-Konsole oder die verwenden AWS CLI , um die Verschlüsselungsschlüssel einer vorhandenen Tabelle zwischen einem AWS-eigener Schlüssel Von AWS verwalteter Schlüssel, und einem vom Kunden verwalteten Schlüssel jederzeit zu aktualisieren.

Aktualisieren eines Verschlüsselungsschlüssels (Konsole)

1. Melden Sie sich bei der an AWS-Managementkonsole und öffnen Sie die DynamoDB-Konsole unter. https://console.aws.amazon.com/dynamodb/
2. Klicken Sie im Navigationsbereich auf der linken Seite der Konsole auf Tabellen.
3. Wählen Sie die Tabelle, die Sie aktualisieren möchten.
4. Wählen Sie Aktionen aus dem Dropdown-Menü und wählen Sie dann die Option Aktualisieren von Einstellungen.
5. Wechseln Sie zur Registerkarte Zusätzliche Einstellungen.
6. Wählen Sie unter Verschlüsselung Verwalten der Verschlüsselung aus.
7. Wählen Sie einen Verschlüsselungstyp aus:

- Im Besitz von Amazon DynamoDB. Der AWS KMS Schlüssel gehört DynamoDB und wird von DynamoDB verwaltet. Für die Verwendung dieses Schlüssels wird Ihnen keine zusätzliche Gebühr berechnet.
- AWS verwalteter Schlüssel Schlüsselalias:. `aws/dynamodb` Der Schlüssel ist in Ihrem Konto gespeichert und wird von verwaltet AWS Key Management Service. (AWS KMS). AWS KMS Gebühren fallen an.
- In Ihrem Konto gespeichert, in Ihrem Besitz und von Ihnen verwaltet. Der Schlüssel ist in Ihrem Konto gespeichert und wird von verwaltet AWS Key Management Service. (AWS KMS). AWS KMS Gebühren fallen an.

> **Note**
>
> Wenn Sie Ihren eigenen Schlüssel besitzen und verwalten möchten, stellen Sie sicher, dass die KMS-Schlüsselrichtlinie ordnungsgemäß festgelegt ist. Weitere Informationen finden Sie unter Schlüsselrichtlinie für einen kundenverwalteten Schlüssel.

Wählen Sie dann Save (Speichern) um die verschlüsselte Tabelle zu aktualisieren. Um den Verschlüsselungstyp zu bestimmen, überprüfen Sie die Tabellendetails auf der Registerkarte Overview (Übersicht).

Aktualisieren eines Verschlüsselungsschlüssels (AWS CLI)

Das folgende Beispiel zeigt das Aktualisieren einer verschlüsselten Tabelle mit der AWS CLI.

Um eine verschlüsselte Tabelle mit dem Standard zu aktualisieren AWS-eigener Schlüssel

- Aktualisieren Sie die verschlüsselte `Music`-Tabelle wie im folgenden Beispiel.

```
aws dynamodb update-table \
  --table-name Music \
  --sse-specification Enabled=false
```

> **Note**
>
> Diese Tabelle ist jetzt mit dem Standard AWS-eigener Schlüssel im DynamoDB-Dienstkonto verschlüsselt.

Um eine verschlüsselte Tabelle mit dem Von AWS verwalteter Schlüssel für DynamoDB zu aktualisieren

- Aktualisieren Sie die verschlüsselte `Music`-Tabelle wie im folgenden Beispiel.

```
aws dynamodb update-table \
  --table-name Music \
  --sse-specification Enabled=true
```

Der `SSEDescription`-Status der Tabellenbeschreibung ist auf `ENABLED` festgelegt und der `SSEType` ist `KMS`.

```
"SSEDescription": {
  "SSEType": "KMS",
  "Status": "ENABLED",
  "KMSMasterKeyArn": "arn:aws:kms:us-east-1:123456789012:key/abcd1234-abcd-1234-
a123-ab1234a1b234",
}
```

Um eine verschlüsselte Tabelle mit einem kundenverwalteten Schlüssel für DynamoDB zu aktualisieren

- Aktualisieren Sie die verschlüsselte `Music`-Tabelle wie im folgenden Beispiel.

```
aws dynamodb update-table \
  --table-name Music \
  --sse-specification Enabled=true,SSEType=KMS,KMSMasterKeyId=abcd1234-abcd-1234-
a123-ab1234a1b234
```

Der `SSEDescription`-Status der Tabellenbeschreibung ist auf `ENABLED` festgelegt und der `SSEType` ist `KMS`.

```
"SSEDescription": {
  "SSEType": "KMS",
  "Status": "ENABLED",
  "KMSMasterKeyArn": "arn:aws:kms:us-east-1:123456789012:key/abcd1234-abcd-1234-
a123-ab1234a1b234",
}
```

Sichern von DynamoDB-Verbindungen mithilfe von VPC-Endpunkten und IAM-Richtlinien

Verbindungen sind sowohl zwischen Amazon DynamoDB und lokalen Anwendungen als auch zwischen DynamoDB und anderen AWS Ressourcen innerhalb derselben Region geschützt. AWS

Erforderliche Richtlinie für Endpunkte

Amazon DynamoDB bietet eine DescribeEndpoints-API, mit der Sie regionale Endpunktinformationen auflisten können. Bei Anforderungen an die öffentlichen DynamoDB-Endpunkte reagiert die API unabhängig von der konfigurierten DynamoDB-IAM-Richtlinie, auch wenn es in der IAM- oder VPC-Endpunktrichtlinie ein explizites oder implizites Verweigern gibt. Dies liegt daran, dass DynamoDB die Autorisierung für die `DescribeEndpoints`-API absichtlich überspringt.

Bei Anfragen von einem VPC-Endpunkt müssen sowohl die IAM- als auch die Endpunktrichtlinien der Virtual Private Cloud (VPC) den `DescribeEndpoints`-API-Aufruf für die anfordernden Prinzipale vom Identity and Access Management (IAM) mithilfe der IAM-`dynamodb:DescribeEndpoints`-Aktion autorisieren. Andernfalls wird der Zugriff auf die `DescribeEndpoints`-API verweigert.

Im Folgenden finden Sie ein Beispiel für eine Endpunktrichtlinie.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "AWS": "arn:aws:iam::111122223333:root"
            },
```

```
            "Action": "dynamodb:DescribeEndpoints",
            "Resource": "*"
        }
    ]
}
```

Datenverkehr zwischen Service und On-Premises-Clients und -Anwendungen

Sie haben zwei Verbindungsoptionen zwischen Ihrem privaten Netzwerk und: AWS

- Eine AWS Site-to-Site VPN Verbindung. Weitere Informationen finden Sie unter Was ist AWS Site-to-Site VPN? im AWS Site-to-Site VPN -Benutzerhandbuch.
- Eine Direct Connect Verbindung. Weitere Informationen finden Sie unter Was ist Direct Connect? im Direct Connect -Benutzerhandbuch.

Der Zugriff auf DynamoDB über das Netzwerk erfolgt über AWS Published. APIs Clients müssen Transport Layer Security (TLS) 1.2 unterstützen. Wir empfehlen TLS 1.3. Clients müssen außerdem Cipher Suites mit PFS (Perfect Forward Secrecy) wie DHE (Ephemeral Diffie-Hellman) oder ECDHE (Elliptic Curve Ephemeral Diffie-Hellman) unterstützen. Die meisten modernen Systemen wie Java 7 und höher unterstützen diese Modi. Außerdem müssen Sie die Anfragen mit einer Zugriffsschlüssel-ID und einem geheimen Zugriffsschlüssel signieren, die einem IAM-Prinzipal zugeordnet sind. Sie können auch AWS -Security-Token-Service (STS) verwenden um temporäre Sicherheitsanmeldeinformationen zu generieren.

Verkehr zwischen AWS Ressourcen in derselben Region

Ein Amazon-Virtual-Private-Cloud-Endpunkt (Amazon VPC) für DynamoDB ist eine logische Entität innerhalb einer VPC, die nur Die Verbindung zu DynamoDB zulässt. Die Amazon VPC leitet Anforderungen an DynamoDB weiter und leitet Antworten an die VPC zurück. Weitere Informationen finden Sie unter VPC-Endpunkte im Amazon-VPC-Benutzerhandbuch. Dieser Abschnitt enthält Beispiele für Richtlinien, die für die Steuerung des Zugriffs auf VPC-Endpunkte verwendet werden können. Sehen Sie Verwenden von IAM-Richtlinien zum Steuern des Zugriffs auf DynamoDB.

> **Note**
>
> Amazon VPC-Endpunkte sind nicht über AWS Site-to-Site VPN oder zugänglich. Direct Connect

AWS Identity and Access Management (IAM) und DynamoDB

AWS Identity and Access Management ist ein AWS Dienst, der einem Administrator hilft, den Zugriff auf Ressourcen sicher zu kontrollieren. AWS Administratoren steuern, wer authentifiziert (angemeldet) und autorisiert (im Besitz von Berechtigungen) ist, Amazon-DynamoDB- und DynamoDB-Accelerator-Ressourcen zu nutzen. Sie können IAM zum Verwalten von Zugriffsberechtigungen und Implementieren von Sicherheitsrichtlinien für Amazon DynamoDB und DynamoDB Accelerator verwenden. IAM ist ein AWS Dienst, den Sie ohne zusätzliche Kosten nutzen können.

Themen

- Identity and Access Management für Amazon DynamoDB
- Verwenden von IAM-Richtlinienbedingungen für die differenzierte Zugriffskontrolle

Identity and Access Management für Amazon DynamoDB

AWS Identity and Access Management (IAM) hilft einem Administrator AWS-Service , den Zugriff auf Ressourcen sicher zu AWS kontrollieren. IAM-Administratoren steuern, wer für die Nutzung von DynamoDB-Ressourcen authentifiziert (angemeldet) und autorisiert (mit Berechtigungen ausgestattet) werden kann. IAM ist ein Programm AWS-Service , das Sie ohne zusätzliche Kosten nutzen können.

Themen

- Zielgruppe
- Authentifizierung mit Identitäten
- Verwalten des Zugriffs mit Richtlinien
- Funktionsweise von Amazon DynamoDB mit IAM
- Beispiele für identitätsbasierte Richtlinien für Amazon DynamoDB
- Fehlerbehebung für Amazon-DynamoDB-Identität und -Zugriff
- IAM-Richtlinie zum Verhindern des Erwerbs von reservierter DynamoDB-Kapazität

Zielgruppe

Wie Sie AWS Identity and Access Management (IAM) verwenden, hängt von Ihrer Rolle ab:

- Servicebenutzer – Fordern Sie von Ihrem Administrator Berechtigungen an, wenn Sie nicht auf Features zugreifen können (siehe Fehlerbehebung für Amazon-DynamoDB-Identität und -Zugriff).
- Serviceadministrator – Bestimmen Sie den Benutzerzugriff und stellen Sie Berechtigungsanfragen (siehe Funktionsweise von Amazon DynamoDB mit IAM).
- IAM-Administrator – Schreiben Sie Richtlinien zur Zugriffsverwaltung (siehe Beispiele für identitätsbasierte Richtlinien für Amazon DynamoDB).

Authentifizierung mit Identitäten

Authentifizierung ist die Art und Weise, wie Sie sich AWS mit Ihren Identitätsdaten anmelden. Sie müssen sich als IAM-Benutzer authentifizieren oder eine IAM-Rolle annehmen. Root-Benutzer des AWS-Kontos

Sie können sich als föderierte Identität anmelden, indem Sie Anmeldeinformationen aus einer Identitätsquelle wie AWS IAM Identity Center (IAM Identity Center), Single Sign-On-Authentifizierung oder Anmeldeinformationen verwenden. Google/Facebook Weitere Informationen zum Anmelden finden Sie unter So melden Sie sich bei Ihrem AWS-Konto an im Benutzerhandbuch für AWS-Anmeldung .

AWS Bietet für den programmatischen Zugriff ein SDK und eine CLI zum kryptografischen Signieren von Anfragen. Weitere Informationen finden Sie unter AWS Signature Version 4 for API requests im IAM-Benutzerhandbuch.

AWS-Konto Root-Benutzer

Wenn Sie einen erstellen AWS-Konto, beginnen Sie mit einer Anmeldeidentität, dem sogenannten AWS-Konto Root-Benutzer, der vollständigen Zugriff auf alle AWS-Services Ressourcen hat. Wir raten ausdrücklich davon ab, den Root-Benutzer für Alltagsaufgaben zu verwenden. Eine Liste der Aufgaben, für die Sie sich als Root-Benutzer anmelden müssen, finden Sie unter Tasks that require root user credentials im IAM-Benutzerhandbuch.

Verbundidentität

Es hat sich bewährt, dass menschliche Benutzer für den Zugriff AWS-Services mithilfe temporärer Anmeldeinformationen einen Verbund mit einem Identitätsanbieter verwenden müssen.

Eine föderierte Identität ist ein Benutzer aus Ihrem Unternehmensverzeichnis, Ihrem Directory Service Web-Identitätsanbieter oder der AWS-Services mithilfe von Anmeldeinformationen aus einer Identitätsquelle zugreift. Verbundene Identitäten übernehmen Rollen, die temporäre Anmeldeinformationen bereitstellen.

Für die zentrale Zugriffsverwaltung empfehlen wir AWS IAM Identity Center. Weitere Informationen finden Sie unter Was ist IAM Identity Center? im AWS IAM Identity Center -Benutzerhandbuch.

IAM-Benutzer und -Gruppen

Ein IAM-Benutzer ist eine Identität mit bestimmten Berechtigungen für eine einzelne Person oder Anwendung. Wir empfehlen die Verwendung temporärer Anmeldeinformationen anstelle von IAM-Benutzern mit langfristigen Anmeldeinformationen. Weitere Informationen finden Sie im IAM-Benutzerhandbuch unter Erfordern, dass menschliche Benutzer den Verbund mit einem Identitätsanbieter verwenden müssen, um AWS mithilfe temporärer Anmeldeinformationen darauf zugreifen zu können.

Eine IAM-Gruppe spezifiziert eine Sammlung von IAM-Benutzern und erleichtert die Verwaltung von Berechtigungen für große Gruppen von Benutzern. Weitere Informationen finden Sie unter Anwendungsfälle für IAM-Benutzer im IAM-Benutzerhandbuch.

IAM-Rollen

Eine IAM-Rolle ist eine Identität mit spezifischen Berechtigungen, die temporäre Anmeldeinformationen bereitstellt. Sie können eine Rolle übernehmen, indem Sie von einer Benutzer- zu einer IAM-Rolle (Konsole) wechseln AWS CLI oder einen AWS API-Vorgang aufrufen. Weitere Informationen finden Sie unter Methoden, um eine Rolle zu übernehmen im IAM-Benutzerhandbuch.

IAM-Rollen sind nützlich für den Verbundbenutzer-Zugriff, temporäre IAM-Benutzerberechtigungen, kontoübergreifenden Zugriff, serviceübergreifenden Zugriff und Anwendungen, die auf Amazon EC2 laufen. Weitere Informationen finden Sie unter Kontoübergreifender Ressourcenzugriff in IAM im IAM-Benutzerhandbuch.

Verwalten des Zugriffs mit Richtlinien

Sie kontrollieren den Zugriff, AWS indem Sie Richtlinien erstellen und diese an AWS Identitäten oder Ressourcen anhängen. Eine Richtlinie definiert Berechtigungen, wenn sie mit einer Identität oder Ressource verknüpft sind. AWS bewertet diese Richtlinien, wenn ein Principal eine Anfrage stellt. Die meisten Richtlinien werden AWS als JSON-Dokumente gespeichert. Weitere Informationen zu JSON-Richtliniendokumenten finden Sie unter Übersicht über JSON-Richtlinien im IAM-Benutzerhandbuch.

Mit Hilfe von Richtlinien legen Administratoren fest, wer Zugriff auf was hat, indem sie definieren, welches Prinzipal welche Aktionen auf welchen Ressourcenund unter welchen Bedingungendurchführen darf.

Standardmäßig haben Benutzer, Gruppen und Rollen keine Berechtigungen. Ein IAM-Administrator erstellt IAM-Richtlinien und fügt sie zu Rollen hinzu, die die Benutzer dann übernehmen können. IAM-Richtlinien definieren Berechtigungen unabhängig von der Methode, die zur Ausführung der Operation verwendet wird.

Identitätsbasierte Richtlinien

Identitätsbasierte Richtlinien sind JSON-Berechtigungsrichtliniendokumente, die Sie einer Identität (Benutzer, Gruppe oder Rolle) anfügen können. Diese Richtlinien steuern, welche Aktionen Identitäten für welche Ressourcen und unter welchen Bedingungen ausführen können. Informationen zum Erstellen identitätsbasierter Richtlinien finden Sie unter Definieren benutzerdefinierter IAM-Berechtigungen mit vom Kunden verwalteten Richtlinien im IAM-Benutzerhandbuch.

Identitätsbasierte Richtlinien können Inline-Richtlinien (direkt in eine einzelne Identität eingebettet) oder verwaltete Richtlinien (eigenständige Richtlinien, die mit mehreren Identitäten verbunden sind) sein. Informationen dazu, wie Sie zwischen verwalteten und Inline-Richtlinien wählen, finden Sie unter Choose between managed policies and inline policies im IAM-Benutzerhandbuch.

Ressourcenbasierte Richtlinien

Ressourcenbasierte Richtlinien sind JSON-Richtliniendokumente, die Sie an eine Ressource anfügen. Beispiele hierfür sind Vertrauensrichtlinien für IAM-Rollen und Amazon S3-Bucket-Richtlinien. In Services, die ressourcenbasierte Richtlinien unterstützen, können Service-Administratoren sie verwenden, um den Zugriff auf eine bestimmte Ressource zu steuern. Sie müssen in einer ressourcenbasierten Richtlinie einen Prinzipal angeben.

Ressourcenbasierte Richtlinien sind Richtlinien innerhalb dieses Diensts. Sie können AWS verwaltete Richtlinien von IAM nicht in einer ressourcenbasierten Richtlinie verwenden.

Weitere Richtlinientypen

AWS unterstützt zusätzliche Richtlinientypen, mit denen die maximalen Berechtigungen festgelegt werden können, die durch gängigere Richtlinientypen gewährt werden:

- Berechtigungsgrenzen – Eine Berechtigungsgrenze legt die maximalen Berechtigungen fest, die eine identitätsbasierte Richtlinie einer IAM-Entität erteilen kann. Weitere Informationen finden Sie unter Berechtigungsgrenzen für IAM-Entitäten im -IAM-Benutzerhandbuch.

- Richtlinien zur Dienstkontrolle (SCPs) — Geben Sie die maximalen Berechtigungen für eine Organisation oder Organisationseinheit in an AWS Organizations. Weitere Informationen finden Sie unter Service-Kontrollrichtlinien im AWS Organizations -Benutzerhandbuch.
- Richtlinien zur Ressourcenkontrolle (RCPs) — Legen Sie die maximal verfügbaren Berechtigungen für Ressourcen in Ihren Konten fest. Weitere Informationen finden Sie im AWS Organizations Benutzerhandbuch unter Richtlinien zur Ressourcenkontrolle (RCPs).
- Sitzungsrichtlinien – Sitzungsrichtlinien sind erweiterte Richtlinien, die als Parameter übergeben werden, wenn Sie eine temporäre Sitzung für eine Rolle oder einen Verbundbenutzer erstellen. Weitere Informationen finden Sie unter Sitzungsrichtlinien im IAM-Benutzerhandbuch.

Mehrere Richtlinientypen

Wenn für eine Anfrage mehrere Arten von Richtlinien gelten, sind die sich daraus ergebenden Berechtigungen schwieriger zu verstehen. Informationen darüber, wie AWS bestimmt wird, ob eine Anfrage zulässig ist, wenn mehrere Richtlinientypen betroffen sind, finden Sie unter Bewertungslogik für Richtlinien im IAM-Benutzerhandbuch.

Funktionsweise von Amazon DynamoDB mit IAM

Bevor Sie IAM zum Verwalten des Zugriffs auf DynamoDB verwenden, informieren Sie sich, welche IAM-Funktionen Sie mit DynamoDB verwenden können.

IAM-Feature	DynamoDB-Unterstützung
Identitätsbasierte Richtlinien	Ja
Ressourcenbasierte Richtlinien	Ja
Richtlinienaktionen	Ja
Richtlinienressourcen	Ja
Bedingungsschlüssel für die Richtlinie	Ja
ACLs	Nein
ABAC (Tags in Richtlinien)	Ja

IAM-Feature	DynamoDB-Unterstützung
Temporäre Anmeldeinformationen	Ja
Prinzipalberechtigungen	Ja
Servicerollen	Ja
Service-verknüpfte Rollen	Ja

Einen allgemeinen Überblick darüber, wie DynamoDB und andere AWS Dienste mit den meisten IAM-Funktionen funktionieren, finden Sie im AWS IAM-Benutzerhandbuch unter Dienste, die mit IAM funktionieren.

Identitätsbasierte Richtlinien für DynamoDB

Unterstützt Richtlinien auf Identitätsbasis: Ja

Identitätsbasierte Richtlinien sind JSON-Berechtigungsrichtliniendokumente, die Sie einer Identität anfügen können, wie z. B. IAM-Benutzern, -Benutzergruppen oder -Rollen. Diese Richtlinien steuern, welche Aktionen die Benutzer und Rollen für welche Ressourcen und unter welchen Bedingungen ausführen können. Informationen zum Erstellen identitätsbasierter Richtlinien finden Sie unter Definieren benutzerdefinierter IAM-Berechtigungen mit vom Kunden verwalteten Richtlinien im IAM-Benutzerhandbuch.

Mit identitätsbasierten IAM-Richtlinien können Sie angeben, welche Aktionen und Ressourcen zugelassen oder abgelehnt werden. Darüber hinaus können Sie die Bedingungen festlegen, unter denen Aktionen zugelassen oder abgelehnt werden. Informationen zu sämtlichen Elementen, die Sie in einer JSON-Richtlinie verwenden, finden Sie in der IAM-Referenz für JSON-Richtlinienelemente im IAM-Benutzerhandbuch.

Beispiele für identitätsbasierte Richtlinien für DynamoDB

Beispiele für identitätsbasierte DynamoDB-Richtlinien finden Sie unter Beispiele für identitätsbasierte Richtlinien für Amazon DynamoDB.

Ressourcenbasierte Richtlinien in DynamoDB

Unterstützt ressourcenbasierte Richtlinien: Ja

Ressourcenbasierte Richtlinien sind JSON-Richtliniendokumente, die Sie an eine Ressource anfügen. Beispiele für ressourcenbasierte Richtlinien sind IAM-Rollen-Vertrauensrichtlinien und Amazon-S3-Bucket-Richtlinien. In Services, die ressourcenbasierte Richtlinien unterstützen, können Service-Administratoren sie verwenden, um den Zugriff auf eine bestimmte Ressource zu steuern. Für die Ressource, an welche die Richtlinie angehängt ist, legt die Richtlinie fest, welche Aktionen ein bestimmter Prinzipal unter welchen Bedingungen für diese Ressource ausführen kann. Sie müssen in einer ressourcenbasierten Richtlinie einen Prinzipal angeben. Zu den Prinzipalen können Konten, Benutzer, Rollen, Verbundbenutzer oder gehören. AWS-Services

Um kontoübergreifenden Zugriff zu ermöglichen, können Sie ein gesamtes Konto oder IAM-Entitäten in einem anderen Konto als Prinzipal in einer ressourcenbasierten Richtlinie angeben. Weitere Informationen finden Sie unter Kontoübergreifender Ressourcenzugriff in IAM im IAM-Benutzerhandbuch.

Richtlinienaktionen für DynamoDB

Unterstützt Richtlinienaktionen: Ja

Administratoren können mithilfe von AWS JSON-Richtlinien angeben, wer Zugriff auf was hat. Das heißt, welcher Prinzipal Aktionen für welche Ressourcen und unter welchen Bedingungen ausführen kann.

Das Element `Action` einer JSON-Richtlinie beschreibt die Aktionen, mit denen Sie den Zugriff in einer Richtlinie zulassen oder verweigern können. Nehmen Sie Aktionen in eine Richtlinie auf, um Berechtigungen zur Ausführung des zugehörigen Vorgangs zu erteilen.

Eine Liste der DynamoDB-Aktionen finden Sie unter Von Amazon DynamoDB definierte Aktionen in der Service-Autorisierungs-Referenz.

Richtlinienaktionen in DynamoDB verwenden das folgende Präfix vor der Aktion:

```
aws
```

Um mehrere Aktionen in einer einzigen Anweisung anzugeben, trennen Sie sie mit Kommata:

```
"Action": [
      "aws:action1",
      "aws:action2"
         ]
```

Beispiele für identitätsbasierte DynamoDB-Richtlinien finden Sie unter Beispiele für identitätsbasierte Richtlinien für Amazon DynamoDB.

Richtlinienressourcen für DynamoDB

Unterstützt Richtlinienressourcen: Ja

Administratoren können mithilfe von AWS JSON-Richtlinien angeben, wer Zugriff auf was hat. Das heißt, welcher Prinzipal Aktionen für welche Ressourcen und unter welchen Bedingungen ausführen kann.

Das JSON-Richtlinienelement `Resource` gibt die Objekte an, auf welche die Aktion angewendet wird. Als Best Practice geben Sie eine Ressource mit dem zugehörigen Amazon-Ressourcennamen (ARN) an. Verwenden Sie für Aktionen, die keine Berechtigungen auf Ressourcenebene unterstützen, einen Platzhalter (*), um anzugeben, dass die Anweisung für alle Ressourcen gilt.

```
"Resource": "*"
```

Eine Liste der DynamoDB-Ressourcentypen und ihrer ARNs Eigenschaften finden Sie unter Von Amazon DynamoDB definierte Ressourcen in der Service Authorization Reference. Informationen zu den Aktionen, mit denen Sie den ARN einzelner Ressourcen angeben können, finden Sie unter Von Amazon DynamoDB definierte Aktionen.

Beispiele für identitätsbasierte DynamoDB-Richtlinien finden Sie unter Beispiele für identitätsbasierte Richtlinien für Amazon DynamoDB.

Richtlinien-Bedingungsschlüssel für DynamoDB

Unterstützt servicespezifische Richtlinienbedingungsschlüssel: Ja

Administratoren können mithilfe von AWS JSON-Richtlinien angeben, wer Zugriff auf was hat. Das heißt, welcher Prinzipal Aktionen für welche Ressourcen und unter welchen Bedingungen ausführen kann.

Das Element `Condition` gibt an, wann Anweisungen auf der Grundlage definierter Kriterien ausgeführt werden. Sie können bedingte Ausdrücke erstellen, die Bedingungsoperatoren verwenden, z. B. ist gleich oder kleiner als, damit die Bedingung in der Richtlinie mit Werten in der

Anforderung übereinstimmt. Eine Übersicht aller AWS globalen Bedingungsschlüssel finden Sie unter Kontextschlüssel für AWS globale Bedingungen im IAM-Benutzerhandbuch.

Eine Liste von DynamoDB-Bedingungsschlüsseln finden Sie unter Bedingungsschlüssel für Amazon DynamoDB in der Service-Autorisierungs-Referenz. Informationen dazu, mit welchen Aktionen und Ressourcen Sie einen Bedingungsschlüssel verwenden können, finden Sie unter Von Amazon DynamoDB definierte Aktionen.

Beispiele für identitätsbasierte DynamoDB-Richtlinien finden Sie unter Beispiele für identitätsbasierte Richtlinien für Amazon DynamoDB.

Zugriffskontrolllisten (ACLs) in DynamoDB

Unterstützt ACLs: Nein

Zugriffskontrolllisten (ACLs) steuern, welche Principals (Kontomitglieder, Benutzer oder Rollen) über Zugriffsberechtigungen für eine Ressource verfügen. ACLs ähneln ressourcenbasierten Richtlinien, verwenden jedoch nicht das JSON-Richtliniendokumentformat.

Attributbasierte Zugriffskontrolle (Attribute-Based Access Control, ABAC) mit DynamoDB

Unterstützt ABAC (Tags in Richtlinien): Ja

Die attributbasierte Zugriffskontrolle (ABAC) ist eine Autorisierungsstrategie, bei der Berechtigungen basierend auf Attributen, auch als Tags bezeichnet, definiert werden. Sie können Tags an IAM-Entitäten und AWS Ressourcen anhängen und dann ABAC-Richtlinien so entwerfen, dass Operationen möglich sind, wenn das Tag des Prinzipals mit dem Tag auf der Ressource übereinstimmt.

Um den Zugriff auf der Grundlage von Tags zu steuern, geben Sie im Bedingungselement einer Richtlinie Tag-Informationen an, indem Sie die Schlüssel `aws:ResourceTag/`*`key-name`*, `aws:RequestTag/`*`key-name`*, oder Bedingung `aws:TagKeys` verwenden.

Wenn ein Service alle drei Bedingungsschlüssel für jeden Ressourcentyp unterstützt, lautet der Wert für den Service Ja. Wenn ein Service alle drei Bedingungsschlüssel für nur einige Ressourcentypen unterstützt, lautet der Wert Teilweise.

Weitere Informationen zu ABAC finden Sie unter Definieren von Berechtigungen mit ABAC-Autorisierung im IAM-Benutzerhandbuch. Um ein Tutorial mit Schritten zur Einstellung von ABAC anzuzeigen, siehe Attributbasierte Zugriffskontrolle (ABAC) verwenden im IAM-Benutzerhandbuch.

Verwenden temporärer Anmeldeinformationen mit DynamoDB

Unterstützt temporäre Anmeldeinformationen: Ja

Temporäre Anmeldeinformationen ermöglichen den kurzfristigen Zugriff auf AWS Ressourcen und werden automatisch erstellt, wenn Sie einen Verbund verwenden oder die Rollen wechseln. AWS empfiehlt, temporäre Anmeldeinformationen dynamisch zu generieren, anstatt langfristige Zugriffsschlüssel zu verwenden. Weitere Informationen finden Sie unter Temporäre Anmeldeinformationen in IAM und AWS-Services , die mit IAM funktionieren im IAM-Benutzerhandbuch.

Serviceübergreifende Prinzipal-Berechtigungen für DynamoDB

Unterstützt Forward Access Sessions (FAS): Ja

Forward Access Sessions (FAS) verwenden die Berechtigungen des Prinzipals, der einen aufruft AWS-Service, kombiniert mit der Anforderung, Anfragen AWS-Service an nachgeschaltete Dienste zu stellen. Einzelheiten zu den Richtlinien für FAS-Anforderungen finden Sie unter Zugriffssitzungen weiterleiten.

Servicerollen für DynamoDB

Unterstützt Servicerollen: Ja

Eine Servicerolle ist eine IAM-Rolle, die ein Service annimmt, um Aktionen in Ihrem Namen auszuführen. Ein IAM-Administrator kann eine Servicerolle innerhalb von IAM erstellen, ändern und löschen. Weitere Informationen finden Sie unter Erstellen einer Rolle zum Delegieren von Berechtigungen an einen AWS-Service im IAM-Benutzerhandbuch.

> **Warning**
>
> Das Ändern der Berechtigungen für eine Servicerolle könnte die Funktionalität von DynamoDB beeinträchtigen. Bearbeiten Sie Servicerollen nur, wenn DynamoDB dazu Anleitungen gibt.

Serviceverknüpfte Rollen für DynamoDB

Unterstützt serviceverknüpfte Rollen: Ja

Eine serviceverknüpfte Rolle ist eine Art von Servicerolle, die mit einer verknüpft ist. AWS-Service Der Service kann die Rolle übernehmen, um eine Aktion in Ihrem Namen auszuführen.

Dienstbezogene Rollen werden in Ihrem Dienst angezeigt AWS-Konto und gehören dem Dienst. Ein IAM-Administrator kann die Berechtigungen für Service-verknüpfte Rollen anzeigen, aber nicht bearbeiten.

Details zum Erstellen oder Verwalten von serviceverknüpften Rollen finden Sie unter AWS -Services, die mit IAM funktionieren. Suchen Sie in der Tabelle nach einem Service mit einem `Yes` in der Spalte Service-linked role (Serviceverknüpfte Rolle). Wählen Sie den Link Yes (Ja) aus, um die Dokumentation für die serviceverknüpfte Rolle für diesen Service anzuzeigen.

Unterstützte serviceverknüpfte Rollen in DynamoDB

Die folgenden serviceverknüpften Rollen werden in DynamoDB unterstützt.

- DynamoDB verwendet die serviceverknüpfte Rolle AWSServiceRoleForDynamoDBReplicationfür die globale Tabellenreplikation zwischen. AWS-Regionen Weitere Informationen Sicherheit von globalen DynamoDB-Tabellen zur serviceverknüpften Rolle finden Sie unter AWSServiceRoleForDynamoDBReplication.
- DynamoDB Accelerator (DAX) verwendet die dienstverknüpfte Rolle AWSServiceRoleForDAX für die Konfiguration und Wartung eines DAX-Clusters. Weitere Informationen the section called “Verwenden von serviceverknüpften Rollen für DAX” zur dienstverknüpften AWSServiceRoleForDAX-Rolle finden Sie unter.

Zusätzlich zu diesen serviceverknüpften DynamoDB-Rollen verwendet DynamoDB Application Auto Scaling Service für die automatische Verwaltung der Durchsatzeinstellungen für Tabellen mit dem Modus bereitgestellter Kapazität. Der Application Auto Scaling Scaling-Dienst verwendet die dienstverknüpfte Rolle AWSServiceRoleForApplicationAutoScaling_DynamoDBTable, um die Durchsatzeinstellungen in DynamoDB-Tabellen zu verwalten, für die Auto Scaling aktiviert ist. Weitere Informationen finden Sie unter Serviceverknüpfte Rollen für Application Auto Scaling.

Beispiele für identitätsbasierte Richtlinien für Amazon DynamoDB

Benutzer und Rollen haben standardmäßig nicht die Berechtigung, DynamoDB-Ressourcen zu erstellen oder zu ändern. Ein IAM-Administrator muss IAM-Richtlinien erstellen, die Benutzern die Berechtigung erteilen, Aktionen für die Ressourcen auszuführen, die sie benötigen.

Informationen dazu, wie Sie unter Verwendung dieser beispielhaften JSON-Richtliniendokumente eine identitätsbasierte IAM-Richtlinie erstellen, finden Sie unter Erstellen von IAM-Richtlinien (Konsole) im IAM-Benutzerhandbuch.

Einzelheiten zu den von DynamoDB definierten Aktionen und Ressourcentypen, einschließlich des Formats ARNs für die einzelnen Ressourcentypen, finden Sie unter Aktionen, Ressourcen und Bedingungsschlüssel für Amazon DynamoDB in der Service Authorization Reference.

Themen

- Best Practices für Richtlinien
- Verwenden der DynamoDB-Konsole
- Gewähren der Berechtigung zur Anzeige der eigenen Berechtigungen für Benutzer
- Verwenden von identitätsbasierten Richtlinien mit Amazon DynamoDB

Best Practices für Richtlinien

Identitätsbasierte Richtlinien legen fest, ob jemand DynamoDB-Ressourcen in Ihrem Konto erstellen, darauf zugreifen oder sie löschen kann. Dies kann zusätzliche Kosten für Ihr verursachen AWS-Konto. Beachten Sie beim Erstellen oder Bearbeiten identitätsbasierter Richtlinien die folgenden Richtlinien und Empfehlungen:

- Erste Schritte mit AWS verwalteten Richtlinien und Umstellung auf Berechtigungen mit den geringsten Rechten — Verwenden Sie die AWS verwalteten Richtlinien, die Berechtigungen für viele gängige Anwendungsfälle gewähren, um damit zu beginnen, Ihren Benutzern und Workloads Berechtigungen zu gewähren. Sie sind in Ihrem verfügbar. AWS-Konto Wir empfehlen Ihnen, die Berechtigungen weiter zu reduzieren, indem Sie vom AWS Kunden verwaltete Richtlinien definieren, die speziell auf Ihre Anwendungsfälle zugeschnitten sind. Weitere Informationen finden Sie unter Von AWS verwaltete Richtlinien oder Von AWS verwaltete Richtlinien für Auftragsfunktionen im IAM-Benutzerhandbuch.
- Anwendung von Berechtigungen mit den geringsten Rechten – Wenn Sie mit IAM-Richtlinien Berechtigungen festlegen, gewähren Sie nur die Berechtigungen, die für die Durchführung einer Aufgabe erforderlich sind. Sie tun dies, indem Sie die Aktionen definieren, die für bestimmte Ressourcen unter bestimmten Bedingungen durchgeführt werden können, auch bekannt als die geringsten Berechtigungen. Weitere Informationen zur Verwendung von IAM zum Anwenden von Berechtigungen finden Sie unter Richtlinien und Berechtigungen in IAM im IAM-Benutzerhandbuch.
- Verwenden von Bedingungen in IAM-Richtlinien zur weiteren Einschränkung des Zugriffs – Sie können Ihren Richtlinien eine Bedingung hinzufügen, um den Zugriff auf Aktionen und Ressourcen zu beschränken. Sie können beispielsweise eine Richtlinienbedingung schreiben, um festzulegen, dass alle Anforderungen mithilfe von SSL gesendet werden müssen. Sie können

auch Bedingungen verwenden, um Zugriff auf Serviceaktionen zu gewähren, wenn diese für einen bestimmten Zweck verwendet werden AWS-Service, z. CloudFormation B. Weitere Informationen finden Sie unter IAM-JSON-Richtlinienelemente: Bedingung im IAM-Benutzerhandbuch.

- Verwenden von IAM Access Analyzer zur Validierung Ihrer IAM-Richtlinien, um sichere und funktionale Berechtigungen zu gewährleisten – IAM Access Analyzer validiert neue und vorhandene Richtlinien, damit die Richtlinien der IAM-Richtliniensprache (JSON) und den bewährten IAM-Methoden entsprechen. IAM Access Analyzer stellt mehr als 100 Richtlinienprüfungen und umsetzbare Empfehlungen zur Verfügung, damit Sie sichere und funktionale Richtlinien erstellen können. Weitere Informationen finden Sie unter Richtlinienvalidierung mit IAM Access Analyzer im IAM-Benutzerhandbuch.
- Multi-Faktor-Authentifizierung (MFA) erforderlich — Wenn Sie ein Szenario haben, das IAM-Benutzer oder einen Root-Benutzer in Ihrem System erfordert AWS-Konto, aktivieren Sie MFA für zusätzliche Sicherheit. Um MFA beim Aufrufen von API-Vorgängen anzufordern, fügen Sie Ihren Richtlinien MFA-Bedingungen hinzu. Weitere Informationen finden Sie unter Sicherer API-Zugriff mit MFA im IAM-Benutzerhandbuch.

Weitere Informationen zu bewährten Methoden in IAM finden Sie unter Best Practices für die Sicherheit in IAM im IAM-Benutzerhandbuch.

Verwenden der DynamoDB-Konsole

Für den Zugriff auf die Amazon-DynamoDB-Konsole müssen Sie über einen Mindestsatz von Berechtigungen verfügen. Diese Berechtigungen müssen es Ihnen ermöglichen, Details zu den DynamoDB-Ressourcen in Ihrem aufzulisten und anzuzeigen. AWS-Konto Wenn Sie eine identitätsbasierte Richtlinie erstellen, die strenger ist als die mindestens erforderlichen Berechtigungen, funktioniert die Konsole nicht wie vorgesehen für Entitäten (Benutzer oder Rollen) mit dieser Richtlinie.

Sie müssen Benutzern, die nur die API AWS CLI oder die API aufrufen, keine Mindestberechtigungen für die Konsole gewähren. AWS Stattdessen sollten Sie nur Zugriff auf die Aktionen zulassen, die der API-Operation entsprechen, die die Benutzer ausführen möchten.

Um sicherzustellen, dass Benutzer und Rollen die DynamoDB-Konsole weiterhin verwenden können, fügen Sie den Entitäten auch die DynamoDB `ConsoleAccess` - oder `ReadOnly` AWS verwaltete Richtlinie hinzu. Weitere Informationen finden Sie unter Hinzufügen von Berechtigungen zu einem Benutzer im IAM-Benutzerhandbuch.

Gewähren der Berechtigung zur Anzeige der eigenen Berechtigungen für Benutzer

In diesem Beispiel wird gezeigt, wie Sie eine Richtlinie erstellen, die IAM-Benutzern die Berechtigung zum Anzeigen der eingebundenen Richtlinien und verwalteten Richtlinien gewährt, die ihrer Benutzeridentität angefügt sind. Diese Richtlinie umfasst Berechtigungen zum Ausführen dieser Aktion auf der Konsole oder programmgesteuert mithilfe der OR-API. AWS CLI AWS

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "ViewOwnUserInfo",
            "Effect": "Allow",
            "Action": [
                "iam:GetUserPolicy",
                "iam:ListGroupsForUser",
                "iam:ListAttachedUserPolicies",
                "iam:ListUserPolicies",
                "iam:GetUser"
            ],
            "Resource": ["arn:aws:iam::*:user/${aws:username}"]
        },
        {
            "Sid": "NavigateInConsole",
            "Effect": "Allow",
            "Action": [
                "iam:GetGroupPolicy",
                "iam:GetPolicyVersion",
                "iam:GetPolicy",
                "iam:ListAttachedGroupPolicies",
                "iam:ListGroupPolicies",
                "iam:ListPolicyVersions",
                "iam:ListPolicies",
                "iam:ListUsers"
            ],
            "Resource": "*"
        }
    ]
}
```

Verwenden von identitätsbasierten Richtlinien mit Amazon DynamoDB

Dieses Thema behandelt die Verwendung identitätsbasierter AWS Identity and Access Management (IAM) -Richtlinien mit Amazon DynamoDB und bietet Beispiele. Die folgenden Beispiele zu identitätsbasierten Richtlinien verdeutlichen, wie ein Kontoadministrator IAM-Identitäten (d.h. Benutzern, Gruppen und Rollen) Berechtigungsrichtlinien zuweisen und somit Berechtigungen zum Durchführen von Operationen an Amazon-DynamoDB-Ressourcen erteilen kann.

Dieses Thema besteht aus folgenden Abschnitten:

- Erforderliche IAM-Berechtigungen für die Verwendung der Amazon-DynamoDB-Konsole
- AWS verwaltete (vordefinierte) IAM-Richtlinien für Amazon DynamoDB
- Beispiele für vom Kunden verwaltete Richtlinien

Nachstehend finden Sie ein Beispiel für eine Berechtigungsrichtlinie.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "DescribeQueryScanBooksTable",
            "Effect": "Allow",
            "Action": [
                "dynamodb:DescribeTable",
                "dynamodb:Query",
                "dynamodb:Scan"
            ],
            "Resource": "arn:aws:dynamodb:us-west-2:111122223333:table/Books"
        }
    ]
}
```

Die obige Richtlinie enthält eine Anweisung, die Berechtigungen für drei DynamoDB-Aktionen (`dynamodb:DescribeTable`dynamodb:Query, unddynamodb:Scan) für eine Tabelle in der `us-west-2` AWS Region gewährt, die dem von angegebenen AWS Konto gehört. *account-*

id Der Amazon-Ressourcenname (ARN) in dem `Resource`-Wert gibt die Tabelle an, für die die Berechtigungen zutreffen.

Erforderliche IAM-Berechtigungen für die Verwendung der Amazon-DynamoDB-Konsole

Um mit der DynamoDB-Konsole arbeiten zu können, muss ein Benutzer über Mindestberechtigungen verfügen, die es ihm ermöglichen, mit den DynamoDB-Ressourcen seines AWS Kontos zu arbeiten. Zusätzlich zu diesen DynamoDB-Berechtigungen erfordert die Konsole Berechtigungen von den folgenden Services:

- CloudWatch Amazon-Berechtigungen zur Anzeige von Metriken und Grafiken.
- AWS Data Pipeline Berechtigungen zum Exportieren und Importieren von DynamoDB-Daten.
- AWS Identity and Access Management Berechtigungen für den Zugriff auf Rollen, die für Exporte und Importe erforderlich sind.
- Amazon Simple Notification Service berechtigt, Sie zu benachrichtigen, wenn ein CloudWatch Alarm ausgelöst wird.
- AWS Lambda Berechtigungen zur Verarbeitung von DynamoDB Streams Streams-Datensätzen.

Wenn Sie eine IAM-Richtlinie erstellen, die strenger ist als die mindestens erforderlichen Berechtigungen, funktioniert die Konsole nicht wie vorgesehen für Benutzer mit dieser IAM-Richtlinie. Um sicherzustellen, dass diese Benutzer die DynamoDB-Konsole weiterhin verwenden können, fügen Sie dem Benutzer auch die `AmazonDynamoDBReadOnlyAccess` AWS verwaltete Richtlinie hinzu, wie unter beschrieben. AWS verwaltete (vordefinierte) IAM-Richtlinien für Amazon DynamoDB

Sie müssen Benutzern, die nur die Amazon DynamoDB-API AWS CLI oder die Amazon DynamoDB DynamoDB-API aufrufen, keine Mindestberechtigungen für die Konsole gewähren.

> **Note**
>
> Wenn Sie auf einen VPC-Endpunkt verweisen, müssen Sie auch den DescribeEndpoints API-Aufruf für die anfordernden IAM-Prinzipale mit der IAM-Aktion (dynamodb:) autorisieren. DescribeEndpoints Weitere Informationen finden Sie unter Erforderliche Richtlinie für Endpunkte.

AWS verwaltete (vordefinierte) IAM-Richtlinien für Amazon DynamoDB

AWS adressiert einige gängige Anwendungsfälle durch die Bereitstellung eigenständiger IAM-Richtlinien, die von erstellt und verwaltet werden. AWS Diese AWS verwalteten Richtlinien gewähren die erforderlichen Berechtigungen für allgemeine Anwendungsfälle, sodass Sie nicht erst untersuchen müssen, welche Berechtigungen benötigt werden. Weitere Informationen finden Sie unter AWS - verwaltete Richtlinien im IAM-Benutzerhandbuch.

Die folgenden AWS verwalteten Richtlinien, die Sie Benutzern in Ihrem Konto zuordnen können, sind spezifisch für DynamoDB und nach Anwendungsszenarien gruppiert:

- AmazonDynamoDBReadOnlyAccess— Gewährt schreibgeschützten Zugriff auf DynamoDB-Ressourcen über die. AWS-Managementkonsole
- AmazonDynamoDBFullZugriff — Gewährt vollen Zugriff auf DynamoDB-Ressourcen über die. AWS-Managementkonsole

Sie können diese Richtlinien für AWS verwaltete Berechtigungen überprüfen, indem Sie sich bei der IAM-Konsole anmelden und dort nach bestimmten Richtlinien suchen.

> **Important**
>
> Die bewährte Methode besteht darin, benutzerdefinierte IAM-Richtlinien zu erstellen, die den Benutzern, Rollen oder Gruppen, die diese Berechtigungen benötigen, die geringste Berechtigung gewähren.

Beispiele für vom Kunden verwaltete Richtlinien

In diesem Abschnitt finden Sie Beispiele für Benutzerrichtlinien, die Berechtigungen für verschiedene DynamoDB-Aktionen gewähren. Diese Richtlinien funktionieren, wenn Sie AWS SDKs oder die AWS CLI verwenden. Wenn Sie die Konsole verwenden, müssen Sie zusätzliche konsolenspezifische Berechtigungen erteilen. Weitere Informationen finden Sie unter Erforderliche IAM-Berechtigungen für die Verwendung der Amazon-DynamoDB-Konsole.

> **Note**
>
> Alle folgenden Richtlinienbeispiele verwenden eine der AWS Regionen und enthalten fiktive Konto- IDs und Tabellennamen.

Beispiele:

- IAM-Richtlinie zum Erteilen von Berechtigungen für alle DynamoDB-Aktionen in einer Tabelle
- IAM-Richtlinie zum Gewähren von Schreibgeschützten Berechtigungen für Elemente in einer DynamoDB-Tabelle
- IAM-Richtlinie zum Erteilen des Zugriffs auf eine bestimmte DynamoDB-Tabelle und ihre Indizes
- IAM-Richtlinie zum Lesen, Schreiben, Aktualisieren und Löschen des Zugriffs auf eine DynamoDB-Tabelle
- IAM-Richtlinie zur Trennung von DynamoDB-Umgebungen im selben Konto AWS
- IAM-Richtlinie zum Verhindern des Erwerbs von reservierter DynamoDB-Kapazität
- IAM-Richtlinie zum Gewähren von Lesezugriff für einen DynamoDB Stream (nicht für die Tabelle)
- IAM-Richtlinie, um einer AWS Lambda Funktion den Zugriff auf DynamoDB-Stream-Datensätze zu ermöglichen
- IAM-Richtlinie für Lese- und Schreibzugriff auf einen DynamoDB-Accelerator-(DAX)-Cluster

Das IAM-Benutzerhandbuch umfasst drei zusätzliche DynamoDB-Beispiele:

- Amazon DynamoDB: Gewährt Zugriff auf eine bestimmte Tabelle
- Amazon DynamoDB: Gewährt Zugriff auf bestimmte Spalten
- Amazon DynamoDB: Ermöglicht basierend auf einer Amazon-Cognito-ID den zeilenweisen Zugriff auf DynamoDB

IAM-Richtlinie zum Erteilen von Berechtigungen für alle DynamoDB-Aktionen in einer Tabelle

Die folgende Berechtigungsrichtlinie erteilt Berechtigungen für alle DynamoDB-Aktionen in einer Tabelle namens `Books`. Der im angegebene Ressourcen-ARN `Resource` identifiziert eine Tabelle in einer bestimmten AWS Region. Wenn Sie den Tabellennamen `Books` im`Resource`-ARN durch ein Platzhalterzeichen (*) ersetzen, lassen Sie alle DynamoDB-Aktionen für alle Tabellen im Konto zu. Berücksichtigen Sie sorgfältig die möglichen Auswirkungen auf die Sicherheit, bevor Sie ein Platzhalterzeichen für diese oder eine IAM-Richtlinie verwenden.

JSON

```
{
```

```
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "AllAPIActionsOnBooks",
            "Effect": "Allow",
            "Action": "dynamodb:*",
            "Resource": "arn:aws:dynamodb:us-west-2:123456789012:table/Books"
        }
    ]
}
```

> **Note**
>
> Dies ist ein Beispiel für die Verwendung eines Platzhalterzeichens (*), das Alle Aktionen, einschließlich Administration, Datenvorgänge, Überwachung und Kauf von reservierten DynamoDB-Kapazitäten. Stattdessen empfiehlt es sich, jede Aktion explizit anzugeben, die gewährt werden soll, und nur was dieser Benutzer, die Rolle oder die Gruppe benötigt.

IAM-Richtlinie zum Gewähren von Schreibgeschützten Berechtigungen für Elemente in einer DynamoDB-Tabelle

Die folgende Berechtigungsrichtlinie erteilt ausschließlich Berechtigungen für die `GetItem`, `BatchGetItem`, `Scan`, `Query` und `ConditionCheckItem`DynamoDB-Aktionen und legt somit einen schreibgeschützten Zugriff auf eine Tabelle `Books` fest.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "ReadOnlyAPIActionsOnBooks",
            "Effect": "Allow",
            "Action": [
                "dynamodb:GetItem",
                "dynamodb:BatchGetItem",
                "dynamodb:Scan",
                "dynamodb:Query",
                "dynamodb:ConditionCheckItem"
```

```
            ],
            "Resource": "arn:aws:dynamodb:us-west-2:123456789012:table/Books"
        }
    ]
}
```

IAM-Richtlinie zum Erteilen des Zugriffs auf eine bestimmte DynamoDB-Tabelle und ihre Indizes

Mit der folgenden Richtlinie werden die Berechtigungen für Datenänderungsaktionen für eine DynamoDB-Tabelle mit dem Namen `Books` und alle Indizes dieser Tabelle. Weitere Informationen zu der Funktionsweise von Indizes finden Sie unter Verbessern des Datenzugriffs mit sekundären Indizes in DynamoDB.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "AccessTableAllIndexesOnBooks",
            "Effect": "Allow",
            "Action": [
              "dynamodb:PutItem",
              "dynamodb:UpdateItem",
              "dynamodb:DeleteItem",
              "dynamodb:BatchWriteItem",
              "dynamodb:GetItem",
              "dynamodb:BatchGetItem",
              "dynamodb:Scan",
              "dynamodb:Query",
              "dynamodb:ConditionCheckItem"
            ],
            "Resource": [
                "arn:aws:dynamodb:us-west-2:123456789012:table/Books",
                "arn:aws:dynamodb:us-west-2:123456789012:table/Books/index/*"
            ]
        }
    ]
}
```

IAM-Richtlinie zum Lesen, Schreiben, Aktualisieren und Löschen des Zugriffs auf eine DynamoDB-Tabelle

Verwenden Sie diese Richtlinie, wenn Sie Ihrer Anwendung erlauben müssen, Daten in Amazon-DynamoDB-Tabellen, -Indizes und -Streams zu erstellen, zu lesen, zu aktualisieren und zu löschen. Ersetzen Sie gegebenenfalls den Namen der AWS Region, Ihre Konto-ID und den Tabellennamen oder das Platzhalterzeichen (*).

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "DynamoDBIndexAndStreamAccess",
            "Effect": "Allow",
            "Action": [
                "dynamodb:GetShardIterator",
                "dynamodb:Scan",
                "dynamodb:Query",
                "dynamodb:DescribeStream",
                "dynamodb:GetRecords",
                "dynamodb:ListStreams"
            ],
            "Resource": [
                "arn:aws:dynamodb:us-west-2:123456789012:table/Books/index/*",
                "arn:aws:dynamodb:us-west-2:123456789012:table/Books/stream/*"
            ]
        },
        {
            "Sid": "DynamoDBTableAccess",
            "Effect": "Allow",
            "Action": [
                "dynamodb:BatchGetItem",
                "dynamodb:BatchWriteItem",
                "dynamodb:ConditionCheckItem",
                "dynamodb:PutItem",
                "dynamodb:DescribeTable",
                "dynamodb:DeleteItem",
                "dynamodb:GetItem",
                "dynamodb:Scan",
                "dynamodb:Query",
```

```
                "dynamodb:UpdateItem"
            ],
            "Resource": "arn:aws:dynamodb:us-west-2:123456789012:table/Books"
        },
        {
            "Sid": "DynamoDBDescribeLimitsAccess",
            "Effect": "Allow",
            "Action": "dynamodb:DescribeLimits",
            "Resource": [
                "arn:aws:dynamodb:us-west-2:123456789012:table/Books",
                "arn:aws:dynamodb:us-west-2:123456789012:table/Books/index/*"
            ]
        }
    ]
}
```

Um diese Richtlinie auf alle DynamoDB-Tabellen in allen AWS Regionen für dieses Konto auszudehnen, verwenden Sie einen Platzhalter (*) für die Region und den Tabellennamen. Beispiel:

```
"Resource":[
               "arn:aws:dynamodb:*:123456789012:table/*",
               "arn:aws:dynamodb:*:123456789012:table/*/index/*"
               ]
```

IAM-Richtlinie zur Trennung von DynamoDB-Umgebungen im selben Konto AWS

Angenommen, Sie verfügen über separate Umgebungen, wobei jede Umgebung ihre eigene Version einer Tabelle namens `ProductCatalog` unterhält. Wenn Sie zwei `ProductCatalog` Tabellen in demselben AWS Konto erstellen, kann sich die Arbeit in einer Umgebung aufgrund der Art und Weise, wie Berechtigungen eingerichtet sind, auf die andere Umgebung auswirken. Beispielsweise werden Kontingente für die Anzahl gleichzeitiger Vorgänge auf der Kontrollebene (z. B.`CreateTable`) auf AWS Kontoebene festgelegt.

Daher reduziert jede Aktion in einer Umgebung die Anzahl der Operationen, die in der anderen Umgebung verfügbar sind. Es besteht auch das Risiko, dass der Code in der Umgebung möglicherweise versehentlich auf Tabellen in der anderen Umgebung zugreift.

> **Note**
>
> Wenn Sie Produktions- und Test-Workloads trennen möchten, um den potenziellen „Explosionsradius“ eines Ereignisses zu steuern, empfiehlt es sich, separate AWS -Konten für Test- und Produktions-Workloads zu erstellen. Weitere Informationen finden Sie unter AWS Kontenverwaltung und Trennung.

Weiter gehen wir davon aus, dass Sie über zwei Entwickler verfügen, Amit und Alice, die die `ProductCatalog`-Tabelle testen. Anstatt dass jeder Entwickler ein separates AWS Konto benötigt, können sich Ihre Entwickler dasselbe AWS Testkonto teilen. In diesem Testkonto können Sie eine Kopie derselben Tabelle für jeden Entwickler zur Bearbeitung erstellen, beispielsweise `Alice_ProductCatalog` und `Amit_ProductCatalog`. In diesem Fall können Sie die Benutzer Alice und Amit in dem AWS Konto erstellen, das Sie für die Testumgebung erstellt haben. Sie können dann diesen Benutzern Berechtigungen erteilen, damit sie in den Tabellen, die sie besitzen, DynamoDB-Aktionen durchführen können.

Um diesen Benutzern IAM-Berechtigungen zu erteilen, können Sie Folgendes tun:

- Erstellen Sie eine separate Richtlinie für jeden Benutzer und weisen jede Richtlinie getrennt ihrem Benutzer zu. Sie können beispielsweise die folgende Richtlinie der Benutzerin Alice zuweisen, um ihr Zugriff auf alle DynamoDB-Aktionen in der Tabelle `Alice_ProductCatalog` zu gewähren:

  JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "AllAPIActionsOnAliceTable",
            "Effect": "Allow",
            "Action": [
              "dynamodb:DeleteItem",
              "dynamodb:DescribeContributorInsights",
              "dynamodb:RestoreTableToPointInTime",
              "dynamodb:ListTagsOfResource",
              "dynamodb:CreateTableReplica",
              "dynamodb:UpdateContributorInsights",
              "dynamodb:CreateBackup",
              "dynamodb:DeleteTable",
              "dynamodb:UpdateTableReplicaAutoScaling",
```

```
                "dynamodb:UpdateContinuousBackups",
                "dynamodb:TagResource",
                "dynamodb:DescribeTable",
                "dynamodb:GetItem",
                "dynamodb:DescribeContinuousBackups",
                "dynamodb:BatchGetItem",
                "dynamodb:UpdateTimeToLive",
                "dynamodb:BatchWriteItem",
                "dynamodb:ConditionCheckItem",
                "dynamodb:UntagResource",
                "dynamodb:PutItem",
                "dynamodb:Scan",
                "dynamodb:Query",
                "dynamodb:UpdateItem",
                "dynamodb:DeleteTableReplica",
                "dynamodb:DescribeTimeToLive",
                "dynamodb:RestoreTableFromBackup",
                "dynamodb:UpdateTable",
                "dynamodb:DescribeTableReplicaAutoScaling",
                "dynamodb:GetShardIterator",
                "dynamodb:DescribeStream",
                "dynamodb:GetRecords",
                "dynamodb:DescribeLimits",
                "dynamodb:ListStreams"
            ],
            "Resource": "arn:aws:dynamodb:us-west-2:123456789012:table/
Alice_ProductCatalog/*"
        }
    ]
}
```

Anschließend können Sie eine ähnliche Richtlinie mit einer anderen Ressource (`Amit_ProductCatalog`-Tabelle) für den Benutzer Amit erstellen.

- Anstatt die Richtlinien einzelnen Benutzern zuzuweisen, können Sie IAM-Richtlinienvariablen zum Schreiben einer einzelnen Richtlinie verwenden und sie einer Gruppe anfügen. Sie müssen eine Gruppe erstellen und für dieses Beispiel die Benutzer Alice und Amit der Gruppe hinzufügen. Das folgende Beispiel gewährt Berechtigungen, um alle DynamoDB-Aktionen in der Tabelle `${aws:username}_ProductCatalog` durchzuführen. Die Richtlinienvariable `${aws:username}` wird durch den Benutzernamen des Anforderers bei der Evaluierung der Richtlinie ersetzt. Wenn Alice beispielsweise eine Anforderung für das Hinzufügen eines Elements

sendet, wird die Aktion nur zugelassen, wenn Alice der `Alice_ProductCatalog`-Tabelle Elemente hinzufügt.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "ActionsOnUserSpecificTable",
            "Effect": "Allow",
            "Action": [
              "dynamodb:PutItem",
              "dynamodb:UpdateItem",
              "dynamodb:DeleteItem",
              "dynamodb:BatchWriteItem",
              "dynamodb:GetItem",
              "dynamodb:BatchGetItem",
              "dynamodb:Scan",
              "dynamodb:Query",
              "dynamodb:ConditionCheckItem"
            ],
            "Resource": "arn:aws:dynamodb:us-west-2:123456789012:table/
${aws:username}_ProductCatalog"
        },
        {
            "Sid": "AdditionalPrivileges",
            "Effect": "Allow",
            "Action": [
                "dynamodb:ListTables",
                "dynamodb:DescribeTable",
                "dynamodb:DescribeContributorInsights"
            ],
            "Resource": "arn:aws:dynamodb:us-west-2:123456789012:table/*"
        }
    ]
}
```

> **Note**
>
> Bei der Verwendung von IAM-Richtlinienvariablen müssen Sie explizit die `2012-10-17`-Version der IAM-Zugriffsrichtliniensprache in der Richtlinie angeben. Die Standardversion der IAM-Zugriffsrichtliniensprache (`2008-10-17`) unterstützt keine Richtlinienvariablen.

Anstelle der Identifizierung einer bestimmten Tabelle als Ressource können Sie ein Platzhalterzeichen (*) verwenden, um Berechtigungen für alle Tabellen zu gewähren, in denen der Name dem Namen des Benutzers vorangesetzt wird, der die Anforderung stellt. Siehe folgendes Beispiel.

```
"Resource":"arn:aws:dynamodb:us-west-2:123456789012:table/${aws:username}_*"
```

IAM-Richtlinie zum Verhindern des Erwerbs von reservierter DynamoDB-Kapazität

Mit Amazon DynamoDBs reservierter Kapazität bezahlen Sie im Vorfeld eine einmalige Gebühr und verpflichten sich zur Zahlung für eine Mindestnutzung während eines bestimmten Zeitraums mit erheblichen Einsparungen. Sie können den verwenden AWS-Managementkonsole , um reservierte Kapazität einzusehen und zu erwerben. Möglicherweise möchten Sie jedoch nicht, dass alle Benutzer in Ihrer Organisation reservierte Kapazität erwerben können. Weitere Informationen über reservierte Kapazität finden Sie unter Amazon DynamoDB.

DynamoDB bietet die folgenden API-Operationen für die Steuerung des Zugriffs auf das reservierte Kapazitätsmanagement:

- `dynamodb:DescribeReservedCapacity` – gibt die Käufe der reservierten Kapazität zurück, die momentan bestehen
- `dynamodb:DescribeReservedCapacityOfferings` – gibt Details über die Pläne der reservierten Kapazität zurück, die momentan von AWS angeboten werden.
- `dynamodb:PurchaseReservedCapacityOfferings` – führt einen tatsächlichen Kauf von reservierter Kapazität durch

Der AWS-Managementkonsole verwendet diese API-Aktionen, um Informationen zur reservierten Kapazität anzuzeigen und Käufe zu tätigen. Sie können diese Operationen nicht von einem Anwendungsprogramm abrufen, da sie nur über die Konsole aufgerufen werden können. Sie

können jedoch den Zugriff auf diese Operationen in einer IAM-Berechtigungsrichtlinie zulassen oder verweigern.

Die folgende Richtlinie ermöglicht es Benutzern, Käufe und Angebote für reservierte Kapazität mithilfe von AWS-Managementkonsole einzusehen. Neue Käufe werden jedoch verweigert.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "AllowReservedCapacityDescriptions",
            "Effect": "Allow",
            "Action": [
                "dynamodb:DescribeReservedCapacity",
                "dynamodb:DescribeReservedCapacityOfferings"
            ],
            "Resource": "arn:aws:dynamodb:us-west-2:123456789012:*"
        },
        {
            "Sid": "DenyReservedCapacityPurchases",
            "Effect": "Deny",
            "Action": "dynamodb:PurchaseReservedCapacityOfferings",
            "Resource": "arn:aws:dynamodb:us-west-2:123456789012:*"
        }
    ]
}
```

Beachten Sie, dass diese Richtlinie das Platzhalterzeichen (*) verwendet, um Beschreibungsberechtigungen für alle, zu ermöglichen und den Kauf von reservierter DynamoDB-Kapazität für alle zu verweigern.

IAM-Richtlinie zum Gewähren von Lesezugriff für einen DynamoDB Stream (nicht für die Tabelle)

Wenn Sie DynamoDB Streams für eine Tabelle aktivieren, werden Informationen über jede Änderung an den Elementen in der Tabelle erfasst. Weitere Informationen finden Sie unter Ändern Sie die Datenerfassung für DynamoDB Streams.

In einigen Fällen möchten Sie möglicherweise verhindern, dass eine Anwendung Daten aus einer DynamoDB-Tabelle liest, während weiterhin der Zugriff auf den Stream dieser Tabelle gewährt

wird. Sie können beispielsweise so konfigurieren AWS Lambda , dass ein Stream abgefragt und eine Lambda-Funktion aufgerufen wird, wenn Elementaktualisierungen erkannt werden, und dann zusätzliche Verarbeitungen durchführen.

Die folgenden Aktionen sind für die Steuerung des Zugriffs auf DynamoDB Streams verfügbar:

- `dynamodb:DescribeStream`
- `dynamodb:GetRecords`
- `dynamodb:GetShardIterator`
- `dynamodb:ListStreams`

Das folgende Beispielrichtlinie erteilt Benutzerberechtigungen, um auf die Streams in einer Tabelle mit dem Namen `GameScores` zuzugreifen. Das letzte Platzhalterzeichen (*) in dem ARN entspricht jedem Stream, der mit dieser Tabelle verknüpft ist.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "AccessGameScoresStreamOnly",
            "Effect": "Allow",
            "Action": [
                "dynamodb:DescribeStream",
                "dynamodb:GetRecords",
                "dynamodb:GetShardIterator",
                "dynamodb:ListStreams"
            ],
            "Resource": "arn:aws:dynamodb:us-west-2:123456789012:table/
GameScores/stream/*"
        }
    ]
}
```

Beachten Sie, dass diese Richtlinie Zugriff auf die Streams in der `GameScores`-Tabelle gewährt, aber nicht auf die Tabelle selbst.

IAM-Richtlinie, um einer AWS Lambda Funktion den Zugriff auf DynamoDB-Stream-Datensätze zu ermöglichen

Wenn Sie möchten, dass bestimmte Aktionen auf der Grundlage von Ereignissen in einem DynamoDB-Stream ausgeführt werden, können Sie eine AWS Lambda Funktion schreiben, die durch diese Ereignisse ausgelöst wird. Eine solche Lambda-Funktion benötigt Berechtigungen zum Lesen von Daten aus dem DynamoDB-Stream. Weitere Informationen zur Verwendung der Lambda mit DynamoDB Streams finden Sie unter DynamoDB Streams und -Trigger AWS Lambda.

Um Lambda Berechtigungen zu erteilen, verwenden Sie die Berechtigungsrichtlinie, die der IAM-Rolle der Lambda-Funktion zugeordnet ist (-Ausführungsrolle). Geben Sie diese Richtlinie an, wenn Sie die Lambda-Funktion erstellen.

Sie können beispielsweise die folgenden Berechtigungsrichtlinien mit den Ausführungsrollen verknüpfen, um Lambda-Berechtigungen zu erteilen, die aufgeführten DynamoDB-Streams-Aktionen durchzuführen.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "APIAccessForDynamoDBStreams",
            "Effect": "Allow",
            "Action": [
                "dynamodb:GetRecords",
                "dynamodb:GetShardIterator",
                "dynamodb:DescribeStream",
                "dynamodb:ListStreams"
            ],
            "Resource": "arn:aws:dynamodb:us-west-2:123456789012:table/
GameScores/stream/*"
        }
    ]
}
```

Weitere Informationen finden Sie unter AWS Lambda -Berechtigungen im AWS Lambda - Entwicklerhandbuch.

IAM-Richtlinie für Lese- und Schreibzugriff auf einen DynamoDB-Accelerator-(DAX)-Cluster

Die folgende Richtlinie ermöglicht einen Lese-, Schreib-, Aktualisierungs- und Löschzugriff auf einen DynamoDB-Accelerator-(DAX)-Cluster, der aber nicht der DynamoDB-Tabelle zugeordnet ist. Um diese Richtlinie zu verwenden, ersetzen Sie den Namen der AWS Region, Ihre Konto-ID und den Namen Ihres DAX-Clusters.

> **Note**
>
> Diese Richtlinie ermöglicht den Zugriff auf DAX-Cluster, aber nicht auf die zugehörige DynamoDB-Tabelle. Stellen Sie sicher, dass Ihr DAX-Cluster über die richtige Richtlinie verfügt, um dieselben Vorgänge in der DynamoDB-Tabelle in Ihrem Namen auszuführen.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "AmazonDynamoDBDAXDataOperations",
            "Effect": "Allow",
            "Action": [
                "dax:GetItem",
                "dax:PutItem",
                "dax:ConditionCheckItem",
                "dax:BatchGetItem",
                "dax:BatchWriteItem",
                "dax:DeleteItem",
                "dax:Query",
                "dax:UpdateItem",
                "dax:Scan"
            ],
            "Resource": "arn:aws:dax:eu-west-1:123456789012:cache/MyDAXCluster"
        }
    ]
}
```

Um diese Richtlinie so zu erweitern, dass sie den DAX-Zugriff für alle AWS Regionen für ein Konto abdeckt, verwenden Sie ein Platzhalterzeichen (*) für den Namen der Region.

```
"Resource": "arn:aws:dax:*:123456789012:cache/MyDAXCluster"
```

Fehlerbehebung für Amazon-DynamoDB-Identität und -Zugriff

Verwenden Sie die folgenden Informationen, um häufige Probleme zu diagnostizieren und zu beheben, die beim Arbeiten mit DynamoDB und IAM auftreten könnten.

Themen

- Ich bin nicht autorisiert, eine Aktion in DynamoDB auszuführen.
- Ich bin nicht berechtigt, iam durchzuführen: PassRole
- Ich möchte Personen außerhalb von mir den Zugriff AWS-Konto auf meine DynamoDB-Ressourcen ermöglichen

Ich bin nicht autorisiert, eine Aktion in DynamoDB auszuführen.

Wenn Ihnen AWS-Managementkonsole mitgeteilt wird, dass Sie nicht berechtigt sind, eine Aktion auszuführen, müssen Sie sich an Ihren Administrator wenden, um Unterstützung zu erhalten. Ihr Administrator ist die Person, die Ihnen Ihren Benutzernamen und Ihr Passwort zur Verfügung gestellt hat.

Der folgende Beispielfehler tritt auf, wenn der `mateojackson`-Benutzer versucht, die Konsole zum Anzeigen von Details zu einer fiktiven *`my-example-widget`*-Ressource zu verwenden, jedoch nicht über `aws:`*`GetWidget`*-Berechtigungen verfügt.

```
User: arn:aws:iam::123456789012:user/mateojackson is not authorized to perform:
 aws:GetWidget on resource: my-example-widget
```

In diesem Fall bittet Mateo seinen Administrator um die Aktualisierung seiner Richtlinien, um unter Verwendung der Aktion *`my-example-widget`* auf die Ressource `aws:`*`GetWidget`* zugreifen zu können.

Ich bin nicht berechtigt, iam durchzuführen: PassRole

Wenn Sie die Fehlermeldung erhalten, dass Sie nicht zum Durchführen der `iam:PassRole`-Aktion autorisiert sind, müssen Ihre Richtlinien aktualisiert werden, um eine Rolle an DynamoDB übergeben zu können.

Einige AWS-Services ermöglichen es Ihnen, eine bestehende Rolle an diesen Dienst zu übergeben, anstatt eine neue Servicerolle oder eine dienstverknüpfte Rolle zu erstellen. Hierzu benötigen Sie Berechtigungen für die Übergabe der Rolle an den Dienst.

Der folgende Fehler tritt beispielsweise auf, wenn ein IAM-Benutzer mit dem Namen `marymajor` versucht, die Konsole zu verwenden, um eine Aktion in DynamoDB auszuführen. Die Aktion erfordert jedoch, dass der Service über Berechtigungen verfügt, die durch eine Servicerolle gewährt werden. Mary besitzt keine Berechtigungen für die Übergabe der Rolle an den Dienst.

```
User: arn:aws:iam::123456789012:user/marymajor is not authorized to perform:
 iam:PassRole
```

In diesem Fall müssen die Richtlinien von Mary aktualisiert werden, um die Aktion `iam:PassRole` ausführen zu können.

Wenn Sie Hilfe benötigen, wenden Sie sich an Ihren AWS Administrator. Ihr Administrator hat Ihnen Ihre Anmeldeinformationen zur Verfügung gestellt.

Ich möchte Personen außerhalb von mir den Zugriff AWS-Konto auf meine DynamoDB-Ressourcen ermöglichen

Sie können eine Rolle erstellen, mit der Benutzer in anderen Konten oder Personen außerhalb Ihrer Organisation auf Ihre Ressourcen zugreifen können. Sie können festlegen, wem die Übernahme der Rolle anvertraut wird. Für Dienste, die ressourcenbasierte Richtlinien oder Zugriffskontrolllisten (ACLs) unterstützen, können Sie diese Richtlinien verwenden, um Personen Zugriff auf Ihre Ressourcen zu gewähren.

Weitere Informationen dazu finden Sie hier:

- Informationen dazu, ob DynamoDB diese Funktionen unterstützt, finden Sie unter Funktionsweise von Amazon DynamoDB mit IAM.
- Informationen dazu, wie Sie Zugriff auf Ihre Ressourcen in AWS-Konten Ihrem Besitz gewähren können, finden Sie im IAM-Benutzerhandbuch unter Gewähren des Zugriffs für einen IAM-Benutzer in einem anderen AWS-Konto , dem Sie gehören.
- Informationen dazu, wie Sie Dritten Zugriff auf Ihre Ressourcen gewähren können AWS-Konten, finden Sie AWS-Konten im IAM-Benutzerhandbuch unter Gewähren des Zugriffs für Dritte.
- Informationen dazu, wie Sie über einen Identitätsverbund Zugriff gewähren, finden Sie unter Gewähren von Zugriff für extern authentifizierte Benutzer (Identitätsverbund) im IAM-Benutzerhandbuch.

- Informationen zum Unterschied zwischen der Verwendung von Rollen und ressourcenbasierten Richtlinien für den kontoübergreifenden Zugriff finden Sie unter Kontoübergreifender Ressourcenzugriff in IAM im IAM-Benutzerhandbuch.

IAM-Richtlinie zum Verhindern des Erwerbs von reservierter DynamoDB-Kapazität

Mit Amazon DynamoDBs reservierter Kapazität bezahlen Sie im Vorfeld eine einmalige Gebühr und verpflichten sich zur Zahlung für eine Mindestnutzung während eines bestimmten Zeitraums mit erheblichen Einsparungen. Sie können die verwenden, um reservierte AWS-Managementkonsole Kapazität einzusehen und zu erwerben. Möglicherweise möchten Sie jedoch nicht, dass alle Benutzer in Ihrer Organisation reservierte Kapazität erwerben können. Weitere Informationen über reservierte Kapazität finden Sie unter Amazon DynamoDB.

DynamoDB bietet die folgenden API-Operationen für die Steuerung des Zugriffs auf das reservierte Kapazitätsmanagement:

- `dynamodb:DescribeReservedCapacity` – gibt die Käufe der reservierten Kapazität zurück, die momentan bestehen
- `dynamodb:DescribeReservedCapacityOfferings` – gibt Details über die Pläne der reservierten Kapazität zurück, die momentan von AWS angeboten werden.
- `dynamodb:PurchaseReservedCapacityOfferings` – führt einen tatsächlichen Kauf von reservierter Kapazität durch

Der AWS-Managementkonsole verwendet diese API-Aktionen, um Informationen zur reservierten Kapazität anzuzeigen und Käufe zu tätigen. Sie können diese Operationen nicht von einem Anwendungsprogramm abrufen, da sie nur über die Konsole aufgerufen werden können. Sie können jedoch den Zugriff auf diese Operationen in einer IAM-Berechtigungsrichtlinie zulassen oder verweigern.

Die folgende Richtlinie ermöglicht es Benutzern, Käufe und Angebote für reservierte Kapazität mithilfe von AWS-Managementkonsole einzusehen. Neue Käufe werden jedoch verweigert.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
```

```
        {
            "Sid": "AllowReservedCapacityDescriptions",
            "Effect": "Allow",
            "Action": [
                "dynamodb:DescribeReservedCapacity",
                "dynamodb:DescribeReservedCapacityOfferings"
            ],
            "Resource": "arn:aws:dynamodb:us-west-2:123456789012:*"
        },
        {
            "Sid": "DenyReservedCapacityPurchases",
            "Effect": "Deny",
            "Action": "dynamodb:PurchaseReservedCapacityOfferings",
            "Resource": "arn:aws:dynamodb:us-west-2:123456789012:*"
        }
    ]
}
```

Beachten Sie, dass diese Richtlinie das Platzhalterzeichen (*) verwendet, um Beschreibungsberechtigungen für alle, zu ermöglichen und den Kauf von reservierter DynamoDB-Kapazität für alle zu verweigern.

Verwenden von IAM-Richtlinienbedingungen für die differenzierte Zugriffskontrolle

Wenn Sie in DynamoDB Berechtigungen gewähren, können Sie Bedingungen angeben, die bestimmen, wie eine Berechtigungsrichtlinie wirksam wird.

-Übersicht

In DynamoDB haben Sie die Möglichkeit, beim Erteilen von Berechtigungen mithilfe einer IAM-Richtlinie Bedingungen anzugeben (siehe Identity and Access Management für Amazon DynamoDB). Beispielsweise ist Folgendes möglich:

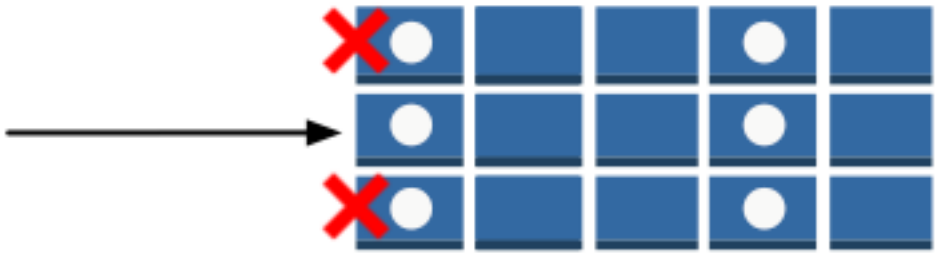
- Erteilen von Berechtigungen, damit den Benutzern schreibgeschützter Zugriff auf bestimmte Elemente und Attribute in einer Tabelle oder einem sekundären Index gewährt wird.
- Erteilen von Berechtigungen, damit den Benutzern lesegeschützter Zugriff auf bestimmte Attribute in einer Tabelle, basierend auf der Identität dieses Benutzers, gewährt wird.

In DynamoDB können Sie Bedingungen in einer IAM-Richtlinie mit Bedingungsschlüsseln angeben, wie im Anwendungsfall in folgendem Abschnitt dargestellt.

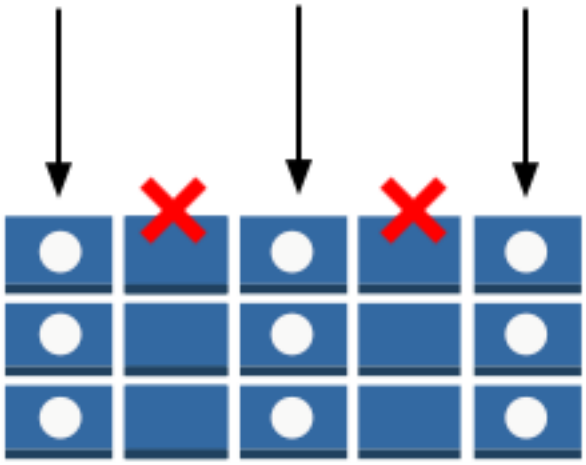
Anwendungsfall für Berechtigungen

Zusätzlich zur Kontrolle des Zugriffs auf DynamoDB-API-Aktionen können Sie auch den Zugriff auf einzelne Datenelemente und Attribute kontrollieren. Sie können z. B. Folgendes tun:

- Erteilen von Berechtigungen für eine Tabelle, aber Einschränken des Zugriffs auf bestimmte Elemente in dieser Tabelle basierend auf bestimmten Primärschlüsselwerten. Ein Beispiel wäre eine Social-Network-Anwendung für Spiele, in der alle gespeicherten Spieldaten eines Benutzers in einer einzelnen Tabelle gespeichert werden, der Benutzer aber keinen Zugriff auf die Datenelemente hat, die er nicht besitzt. Dies wird in der folgenden Darstellung gezeigt:



- Ausblenden von Informationen, so dass nur eine Teilmenge der Attribute für den Benutzer sichtbar ist. Ein Beispiel dafür ist eine Anwendung, die Flugdaten für nahegelegene Flughäfen, basierend auf dem Standort des Benutzers, anzeigt. Namen von Airlines, Ankunfts- und Abflugzeiten und Flugnummer werden alle angezeigt. Jedoch werden Attribute, wie z. B. Pilotennamen oder die Anzahl der Passagiere ausgeblendet, wie in der folgenden Abbildung dargestellt:



Zur Implementierung dieser Art von differenzierter Zugriffskontrolle, schreiben Sie eine IAM-Berechtigungsrichtlinie, die Bedingungen für den Zugriff auf Sicherheitsanmeldeinformationen und den dazugehörigen Berechtigungen angibt. Anschließend wenden Sie die Richtlinie für die -Benutzer, -Gruppen oder -Rollen an, die Sie mithilfe der IAM-Konsole erstellen. Ihre IAM-Richtlinie kann den Zugriff auf einzelne Elemente in einer Tabelle oder auf die Attribute dieser Elemente oder auf beides gleichzeitig beschränken.

Sie können wahlweise den Web-Identitätsverbund verwenden, um den Zugriff von Benutzern zu kontrollieren, die durch Login with Amazon, Facebook oder Google authentifiziert werden. Weitere Informationen finden Sie unter Verwenden des Web-Identitätsverbunds.

Sie verwenden das IAM-`Condition`-Element, um eine differenzierte Zugriffskontrollrichtlinie zu implementieren. Indem ein `Condition`-Element einer Berechtigungsrichtlinie hinzugefügt wird, können Sie den Zugriff auf Elemente und Attribute in DynamoDB-Tabellen und -Indizes, basierend auf Ihren besonderen Geschäftsanforderungen, gewähren oder verweigern.

Im folgenden Video wird die differenzierte Zugriffskontrolle in DynamoDB unter Verwendung von IAM-Richtlinienbedingungen erläutert.

Grundlegendes zur detaillierten Zugriffskontrolle in DynamoDB

Die detaillierte Zugriffskontrolle in DynamoDB ermöglicht es Ihnen, präzise Berechtigungsgrenzen auf mehreren Ebenen festzulegen:

1. Zugriffskontrolle auf Elementebene: Beschränken Sie Benutzer darauf, nur auf Elemente zuzugreifen, die bestimmte Schlüsselwerte enthalten, die in der Regel ihrer Identität oder ihrem Berechtigungsbereich entsprechen.
2. Zugriffskontrolle auf Attributebene: Beschränken Sie, welche Attribute (Spalten) Benutzer anzeigen oder ändern können, sodass Sie vertrauliche Informationen schützen und gleichzeitig Zugriff auf nicht vertrauliche Daten innerhalb derselben Elemente gewähren können.
3. Vorgangsspezifische Kontrollen: Wenden Sie je nach Art des ausgeführten Vorgangs unterschiedliche Berechtigungsregeln an.

Diese Kontrollen werden durch IAM-Richtlinien unter Verwendung von DynamoDB-spezifischen Bedingungsschlüsseln implementiert.

Festlegung von Bedingungen: Verwenden von Bedingungsschlüsseln

AWS stellt eine Reihe vordefinierter Bedingungsschlüssel (AWS allgemeine Bedingungsschlüssel) für alle AWS Dienste bereit, die IAM für die Zugriffskontrolle unterstützen. Sie können beispielsweise den `aws:SourceIp`-Bedingungsschlüssel verwenden, um die IP-Adresse des Anforderers zu prüfen, bevor eine Aktion durchgeführt werden darf. Weitere Informationen und eine Liste der AWS-weiten Schlüssel finden Sie unter Verfügbare Schlüssel für Bedingungen im IAM-Benutzerhandbuch.

Im Folgenden sind die dienstspezifischen DynamoDB-Bedingungsschlüssel aufgeführt, die für DynamoDB gelten.

`dynamodb:LeadingKeys`

Repräsentiert das erste Schlüsselattribut der Tabelle – mit anderen Worten, der Partitionsschlüssel. Der Schlüsselname `LeadingKeys` ist im Plural, sogar wenn der Schlüssel mit Einzelelementaktionen verwendet wird. Zudem müssen Sie den Modifikator `ForAllValues` verwenden, wenn `LeadingKeys` in einer Bedingung genutzt wird.

`dynamodb:Select`

Repräsentiert den Parameter `Select` einer Anforderung. `Select` kann jeden der folgenden Werte annehmen:

- `ALL_ATTRIBUTES`
- `ALL_PROJECTED_ATTRIBUTES`
- `SPECIFIC_ATTRIBUTES`

- `COUNT`

> **Note**
>
> Dieser Bedingungsschlüssel wird zwar häufig mit Abfrage- und Scanvorgängen in Verbindung gebracht, gilt aber für alle DynamoDB-Operationen, die Artikelattribute zurückgeben, und ist für die Steuerung des Attributzugriffs über alle API-Aktionen hinweg unerlässlich. Durch die Verwendung StringEqualsIfExists oder ähnlicher Beschränkungen für diesen Bedingungsschlüssel werden Einschränkungen auf Operationen angewendet, für die dieser Bedingungsschlüssel gilt, während er bei Vorgängen, bei denen er nicht zutrifft, ignoriert wird.

`dynamodb:Attributes`

Stellt eine Liste der Attribute der obersten Ebene dar, auf die durch eine Anfrage zugegriffen wird. Auf ein Attribut der obersten Ebene wird durch eine Anfrage zugegriffen, wenn es oder ein darin enthaltenes verschachteltes Attribut in den Anforderungsparametern angegeben ist. Bei einer `GetItem` Anfrage, die einen Wert `ProjectionExpression` von angibt`"Name, Address.City"`, würde die `dynamodb:Attributes` Liste beispielsweise „Name“ und „Adresse“ enthalten. Wenn der `Attributes` Parameter in einer detaillierten Zugriffskontrollrichtlinie aufgeführt ist, sollten Sie auch Einschränkungen `ReturnValues` und `Select` Parameter in Betracht ziehen, um den eingeschränkten Zugriff auf bestimmte Attribute für mehrere API-Aktionen wie`GetItem`, und sicherzustellen. `Query` `Scan`

> **Note**
>
> Diese Bedingung wird nur für die in der Anfrage angegebenen Attribute (z. B. in a ProjectionExpression) ausgewertet, nicht für Attribute in der Antwort. Wenn in der Anfrage kein Wert angegeben ProjectionExpression wird, werden alle Attribute unabhängig von etwaigen Attributbeschränkungen in der Richtlinie zurückgegeben. Einzelheiten zur ordnungsgemäßen Sicherung des Attributzugriffs finden Sie weiter unten im Abschnitt „Sicherstellen, dass attributbasierte Einschränkungen durchgesetzt werden“.

`dynamodb:ReturnValues`

Stellt den `ReturnValues` Parameter einer Anfrage dar. Abhängig von der API-Aktion `ReturnValues` kann dies einer der folgenden Werte sein:

- ALL_OLD
- UPDATED_OLD
- ALL_NEW
- UPDATED_NEW
- NONE

dynamodb:ReturnConsumedCapacity

Repräsentiert den Parameter ReturnConsumedCapacity einer Anforderung. ReturnConsumedCapacity kann einen der folgenden Werte annehmen:

- TOTAL
- NONE

dynamodb:FirstPartitionKeyValues

Stellt das erste Schlüsselattribut einer Tabelle dar, mit anderen Worten, den ersten Partitionsschlüssel. Der Schlüsselname FirstPartitionKeyValues ist im Plural, sogar wenn der Schlüssel mit Einzelelementaktionen verwendet wird. Darüber hinaus müssen Sie den ForAllValues Modifikator verwenden, wenn Sie ihn FirstPartitionKeyValues in einer Bedingung verwenden. FirstPartitionKeyValuesund LeadingKeys kann austauschbar verwendet werden.

dynamodb:SecondPartitionKeyValues

Ähnlich dynamodb:FirstPartitionKeyValues. Stellt den zweiten Partitionsschlüssel der Ressourcen dar. Der Schlüsselname SecondPartitionKeyValues ist im Plural, sogar wenn der Schlüssel mit Einzelelementaktionen verwendet wird.

dynamodb:ThirdPartitionKeyValues

Ähnlich dynamodb:FirstPartitionKeyValues. Stellt den dritten Partitionsschlüssel der Ressourcen dar. Der Schlüsselname ThirdPartitionKeyValues ist im Plural, sogar wenn der Schlüssel mit Einzelelementaktionen verwendet wird.

dynamodb:FourthPartitionKeyValues

Ähnlich dynamodb:FirstPartitionKeyValues. Stellt den vierten Partitionsschlüssel der Ressourcen dar. Der Schlüsselname FourthPartitionKeyValues ist im Plural, sogar wenn der Schlüssel mit Einzelelementaktionen verwendet wird.

Stellt sicher, dass attributbasierte Einschränkungen durchgesetzt werden

Bei der Verwendung von attributbasierten Bedingungen zur Beschränkung des Zugriffs auf bestimmte Attribute ist es wichtig zu verstehen, wie diese Bedingungen bewertet werden:

- Attributbedingungen werden nur für Attribute ausgewertet, die in der Anfrage angegeben sind, nicht für Attribute in der Antwort.
- Bei Lesevorgängen ohne ProjectionExpression (GetItem, Query, Scan usw.) werden alle Attribute zurückgegeben, unabhängig von den Attributbeschränkungen in Ihrer Richtlinie. Um diese potenzielle Offenlegung vertraulicher Daten zu verhindern, implementieren Sie sowohl Attributbedingungen (`dynamodb:Attributes`) als auch eine Bedingung, die vorschreibt, dass bestimmte Attribute angefordert werden müssen (`dynamodb:Select`).
- Bei Schreibvorgängen (PutItem, UpdateItem, DeleteItem) kann der ReturnValues Parameter vollständige Elemente zurückgeben, wodurch möglicherweise eingeschränkte Attribute verfügbar gemacht werden, selbst wenn der Schreibvorgang selbst Ihrer Richtlinie entspricht. Um diese Offenlegung zu verhindern, implementieren Sie in Ihrer Richtlinie sowohl Attributbedingungen (`dynamodb:Attributes`) als auch Einschränkungen für ReturnValues (`dynamodb:ReturnValues`).

Begrenzen des Benutzerzugriffs

Viele IAM-Berechtigungsrichtlinien erlauben Benutzern, nur auf die Elemente in einer Tabelle zuzugreifen, in denen der Partitions-Schlüsselwert der Benutzer-ID entspricht. Beispielsweise begrenzen die zuvor genannten Grenzwerte der Spielanwendung den Zugriff so, dass Benutzer nur auf Spieldaten Zugriff haben, die mit ihrer Benutzer-ID verknüpft sind. Die IAM-Ersetzungsvariablen `${www.amazon.com:user_id}`, `${graph.facebook.com:id}` und `${accounts.google.com:sub}` enthalten Benutzer-IDs für Login with Amazon, Facebook und Google. Um zu erfahren, wie eine Anwendung sich bei einem dieser Identitätsanbieter anmeldet und diese IDs erhält, klicken Sie auf Verwenden des Web-Identitätsverbunds.

> **Important**
>
> Eine differenzierte Zugriffskontrolle wird nicht unterstützt, um die Replikation globaler Tabellen einzuschränken. Die Anwendung von Richtlinienbedingungen für eine differenzierte Zugriffskontrolle auf Service-Prinzipale oder serviceverknüpfte Rollen von DynamoDB, die für die Replikation globaler Tabellen verwendet werden, kann die Replikation innerhalb einer globalen Tabelle unterbrechen.

> **Note**
>
> Jedes der Beispiele im folgenden Abschnitt legt die `Effect`-Klausel auf `Allow` fest und gibt nur die Aktionen, Ressourcen und Parameter an, die erlaubt sind. Zugriff hat lediglich das, was in der IAM-Richtlinie aufgeführt ist.
> In einigen Fällen ist es möglich, diese Richtlinien umzuschreiben, damit sie auf Verweigerung basieren (dies bedeutet, die `Effect`-Klausel auf `Deny` festzulegen und die gesamte Logik in der Richtlinie umzukehren). Allerdings empfehlen wir, die Nutzung von Richtlinien, die auf einer Verweigerung basieren, mit DynamoDB zu vermeiden, weil es verglichen mit Richtlinien, die auf einer Berechtigung basieren, schwierig ist, sie korrekt zu schreiben. Darüber hinaus können zukünftige Änderungen an der DynamoDB-API (oder Änderungen an den vorhandenen API-Eingaben) eine Richtlinie, die auf Verweigerung basiert, wirkungslos machen.

Beispielrichtlinien: Verwenden von Bedingungen für die differenzierte Zugriffskontrolle

In diesem Abschnitt werden einige Richtlinien für die Implementierung einer differenzierten Zugriffskontrolle auf DynamoDB-Tabellen und Indizes dargestellt.

> **Note**
>
> Alle Beispiele verwenden die Region us-west-2 und enthalten ein fiktives Konto. IDs

Beispiel 1. Grundlegende Zugriffskontrolle auf Basis von Partitionsschlüsseln mit Attributeinschränkungen

Nehmen wir beispielsweise eine mobile Spielanwendung, mit der Spieler aus einer Auswahl verschiedener Spiele wählen und diese auch spielen können. Die App verwendet eine DynamoDB-Tabelle mit dem Namen`GameScores`, um Highscores und andere Benutzerdaten zu verfolgen. Jedes Element in der Tabelle wird durch eine Benutzer-ID und den Namen des Spiels, das der Benutzer gespielt hat, eindeutig identifiziert. Die `GameScores`-Tabelle verfügt über einen Primärschlüssel, der aus einem Partitionsschlüssel (`UserId`) und einem Sortierschlüssel (`GameTitle`) besteht. Benutzer haben nur Zugriff auf Spieldaten, die mit ihrer Benutzer-ID verknüpft sind. Ein Benutzer, der ein Spiel spielen möchte, muss zu einer IAM-Rolle mit dem Namen `GameRole` gehören, der eine Sicherheitsrichtlinie zugewiesen ist.

Um Benutzerberechtigungen in dieser Anwendung zu verwalten, können Sie eine Berechtigungsrichtlinie wie die folgende schreiben:

JSON

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Sid":"AllowAccessToOnlyItemsMatchingUserID",
         "Effect":"Allow",
         "Action":[
            "dynamodb:GetItem",
            "dynamodb:BatchGetItem",
            "dynamodb:Query",
            "dynamodb:PutItem",
            "dynamodb:UpdateItem",
            "dynamodb:DeleteItem",
            "dynamodb:BatchWriteItem"
         ],
         "Resource":[
            "arn:aws:dynamodb:us-west-2:123456789012:table/GameScores"
         ],
         "Condition":{
            "ForAllValues:StringEquals":{
               "dynamodb:LeadingKeys":[
                  "${www.amazon.com:user_id}"
               ],
               "dynamodb:Attributes":[
                  "UserId",
                  "GameTitle",
                  "Wins",
                  "Losses",
                  "TopScore",
                  "TopScoreDateTime"
               ]
            },
            "StringEqualsIfExists":{
               "dynamodb:Select":"SPECIFIC_ATTRIBUTES"
            }
         }
      }
   ]
```

```
}
```

Zusätzlich zum Gewähren von Berechtigungen für bestimmte `Action`-Aktionen (Element `GameScores`) in der Tabelle `Resource` (Element) verwendet das Element `Condition` folgende DynamoDB-spezifische Bedingungsschlüssel, die die Berechtigungen wie folgt begrenzen:

- `dynamodb:LeadingKeys` – Dieser Bedingungsschlüssel gewährt Benutzern ausschließlich den Zugriff auf die Elemente, in denen der Partitions-Schlüsselwert ihrer Benutzer-ID entspricht. Diese `${www.amazon.com:user_id}`-ID ist eine Ersetzungsvariable. Weitere Informationen zu Ersetzungsvariablen finden Sie unter Verwenden des Web-Identitätsverbunds.
- `dynamodb:Attributes` – Dieser Bedingungsschlüssel begrenzt den Zugriff auf angegebene Attribute, sodass nur die Aktionen, die in der Berechtigungsrichtlinie aufgeführt sind, die Werte für diese Attribute zurückgeben können. Zusätzlich stellt die `StringEqualsIfExists`-Klausel sicher, dass die Anwendung immer eine Liste der spezifischen Attribute bereitstellen muss, nach denen gehandelt wird, und dass die Anwendung nicht alle Attribute anfordern kann.

Wenn eine IAM-Richtlinie evaluiert wird, wird das Ergebnis immer entweder True (Zugriff erlaubt) oder False (Zugriff verweigert) sein. Wenn irgendein Teil des `Condition`-Elements False ist, dann wird die gesamte Richtlinie als False gewertet und der Zugriff wird verweigert.

> **Important**
>
> Wenn Sie `dynamodb:Attributes` verwenden, müssen Sie die Namen aller Primärschlüssel und Indexschlüsselattribute für die Tabelle und alle sekundären Indizes, die in der Richtlinie aufgeführt sind, angeben. Andernfalls kann DynamoDB diese Schlüsselattribute nicht verwenden, um die angeforderte Aktion durchzuführen.

IAM-Richtliniendokumente können nur die folgenden Unicode-Zeichen enthalten: horizontale Tabulatorschritt (U+0009), Zeilenvorschub (U+000A), Wagenrücklauf (U+000D) und Zeichen im Bereich von U+0020 bis U+00FF.

Beispiel 2: Erteilen Sie Berechtigungen, die den Zugriff auf Elemente mit einem bestimmten Partitionsschlüsselwert einschränken

Die folgende Berechtigungsrichtlinie erteilt Berechtigungen, die eine Reihe von DynamoDB-Aktionen in der `GamesScore` Tabelle erlauben. Sie verwendet den `dynamodb:LeadingKeys`-

Bedingungsschlüssel, um Benutzeraktionen nur für diejenigen Elemente einzuschränken, deren `UserID`-Partitions-Schlüsselwert der eindeutigen Benutzer-ID von „Login with Amazon“ für diese Anwendung entspricht.

> **Important**
>
> Die Liste von Aktionen schließt Berechtigungen für `Scan` nicht mit ein, weil `Scan` alle Elemente zurückgibt, unabhängig von den Hauptschlüsseln.

JSON

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Sid":"FullAccessToUserItems",
         "Effect":"Allow",
         "Action":[
            "dynamodb:GetItem",
            "dynamodb:BatchGetItem",
            "dynamodb:Query",
            "dynamodb:PutItem",
            "dynamodb:UpdateItem",
            "dynamodb:DeleteItem",
            "dynamodb:BatchWriteItem"
         ],
         "Resource":[
            "arn:aws:dynamodb:us-west-2:123456789012:table/GameScores"
         ],
         "Condition":{
            "ForAllValues:StringEquals":{
               "dynamodb:LeadingKeys":[
                  "${www.amazon.com:user_id}"
               ]
            }
         }
      }
   ]
}
```

> **Note**
>
> Wenn Sie Richtlinienvariablen verwenden, müssen Sie Version 2012-10-17 in der Richtlinie explizit angeben. Die Standardversion der Zugriffsrichtliniensprache, 2008-10-17, unterstützt keine Richtlinienvariablen.

Um schreibgeschützten Zugriff zu implementieren, können Sie alle Aktionen entfernen, die die Daten ändern können. In der folgenden Richtlinie werden nur die Aktionen in die Bedingung integriert, die schreibgeschützten Zugriff erteilen.

JSON

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Sid":"ReadOnlyAccessToUserItems",
         "Effect":"Allow",
         "Action":[
            "dynamodb:GetItem",
            "dynamodb:BatchGetItem",
            "dynamodb:Query"
         ],
         "Resource":[
            "arn:aws:dynamodb:us-west-2:123456789012:table/GameScores"
         ],
         "Condition":{
            "ForAllValues:StringEquals":{
               "dynamodb:LeadingKeys":[
                  "${www.amazon.com:user_id}"
               ]
            }
         }
      }
   ]
}
```

> **Important**
>
> Wenn Sie `dynamodb:Attributes` verwenden, müssen Sie die Namen aller Primärschlüssel und Indexschlüsselattribute für die Tabelle und alle sekundären Indizes, die in der Richtlinie aufgeführt sind, angeben. Andernfalls kann DynamoDB diese Schlüsselattribute nicht verwenden, um die angeforderte Aktion durchzuführen.

Beispiel 3: Erteilen Sie Berechtigungen, die den Zugriff auf bestimmte Attribute in einer Tabelle einschränken

Die folgende Berechtigungsrichtlinie erlaubt den Zugriff auf nur zwei bestimmte Attribute in einer Tabelle, indem sie den `dynamodb:Attributes`-Bedingungsschlüssel hinzufügt. Diese Attribute können in einem bedingten Schreib- oder Scan-Filter gelesen, geschrieben oder evaluiert werden.

JSON

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Sid":"LimitAccessToSpecificAttributes",
         "Effect":"Allow",
         "Action":[
            "dynamodb:UpdateItem",
            "dynamodb:GetItem",
            "dynamodb:Query",
            "dynamodb:BatchGetItem",
            "dynamodb:Scan"
         ],
         "Resource":[
            "arn:aws:dynamodb:us-west-2:123456789012:table/GameScores"
         ],
         "Condition":{
            "ForAllValues:StringEquals":{
               "dynamodb:Attributes":[
                  "UserId",
                  "TopScore"
               ]
            },
            "StringEqualsIfExists":{
```

```
                "dynamodb:Select":"SPECIFIC_ATTRIBUTES",
                "dynamodb:ReturnValues":[
                   "NONE",
                   "UPDATED_OLD",
                   "UPDATED_NEW"
                ]
             }
          }
       }
    ]
}
```

> **Note**
>
> Die Richtlinie verfolgt einen Listen-Ansatz, der den Zugriff auf eine benannte Gruppe von Attributen erteilt. Sie können eine äquivalente Richtlinie schreiben, die stattdessen den Zugriff auf andere Attribute verweigert. Wir empfehlen diesen Ansatz einer Ablehnungsliste nicht. Benutzer können die Namen dieser verweigerten Attribute bestimmen, indem sie dem Prinzip der geringsten Rechte folgen, wie es in Wikipedia unter http://en.wikipedia erklärt wird. org/wiki/Principle_of_least_privilege und mithilfe einer Zulassungsliste alle erlaubten Werte aufzuzählen, anstatt die verweigerten Attribute anzugeben.

Diese Richtlinie erlaubt `PutItem`, `DeleteItem` und `BatchWriteItem` nicht. Diese Aktionen ersetzen immer das gesamte vorherige Element. Dies würde Benutzern ermöglichen, die vorherigen Werte für Attribute, auf die sie nicht zugriffsberechtigt sind, zu löschen.

Die `StringEqualsIfExists`-Klausel in der Berechtigungsrichtlinie gewährleistet Folgendes:

- Wenn der Benutzer den Parameter `Select` angibt, dann muss sein Wert `SPECIFIC_ATTRIBUTES` sein. Diese Anforderung verhindert, dass die API-Aktion jegliche Attribute, die nicht zulässig sind, zurückgibt, z. B. von einer Indexprojektion.
- Wenn der Benutzer den Parameter `ReturnValues` angibt, dann muss sein Wert `NONE`, `UPDATED_OLD` oder `UPDATED_NEW` sein. Dies ist erforderlich, weil die `UpdateItem`-Aktion ebenfalls implizite Lesevorgänge durchführt, um zu überprüfen, ob ein zu ersetzendes Element existiert, und damit vorherige Attributwerte, wenn angefordert, zurückgegeben werden können. Eine solche Einschränkung von `ReturnValues` stellt sicher, dass Benutzer nur die zulässigen Attribute lesen oder schreiben können.

- Die `StringEqualsIfExists`-Klausel gewährleistet, dass nur einer dieser Parameter – `Select` oder `ReturnValues` – pro Anforderung im Kontext der zulässigen Aktionen genutzt werden kann.

Es folgen einige Variationen dieser Richtlinie:

- Um nur Leseaktionen zu erlauben, können Sie `UpdateItem` aus der Liste der zulässigen Aktionen entfernen. Da keine der verbleibenden Aktionen `ReturnValues` akzeptiert, können Sie `ReturnValues` aus der Bedingung entfernen. Sie können auch `StringEqualsIfExists` zu `StringEquals` ändern, weil der Parameter `Select` immer einen Wert hat (`ALL_ATTRIBUTES`, sofern nicht anders angegeben).
- Um nur Leseaktionen zu erlauben, können Sie alles außer `UpdateItem` aus der Liste der zulässigen Aktionen entfernen. Da `UpdateItem` den Parameter `Select` nicht verwendet, können Sie `Select` aus der Bedingung entfernen. Sie müssen auch `StringEqualsIfExists` zu `StringEquals` ändern, weil der Parameter `ReturnValues` immer einen Wert hat (`NONE` sofern nicht anders angegeben).
- Um alle Attribute zu erlauben, deren Name einem Muster entspricht, verwenden Sie `StringLike` anstatt `StringEquals` und verwenden Sie Platzhalter (*) für mehrere Zeichen.

Beispiel 4: Erteilen Sie Berechtigungen, um Aktualisierungen bestimmter Attribute zu verhindern

Die folgende Berechtigungsrichtlinie beschränkt den Benutzerzugriff, so dass nur die Aktualisierung bestimmter Attribute zulässig ist, die von dem `dynamodb:Attributes`-Bedingungsschlüssel identifiziert wurden. Die `StringNotLike`-Bedingung hindert eine Anwendung daran, die Attribute zu aktualisieren, die mithilfe des `dynamodb:Attributes`-Bedingungsschlüssels angegeben wurden.

JSON

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Sid":"PreventUpdatesOnCertainAttributes",
         "Effect":"Allow",
         "Action":[
            "dynamodb:UpdateItem"
         ],
         "Resource":"arn:aws:dynamodb:us-west-2:123456789012:table/GameScores",
         "Condition":{
```

```
            "ForAllValues:StringNotLike":{
               "dynamodb:Attributes":[
                  "FreeGamesAvailable",
                  "BossLevelUnlocked"
               ]
            },
            "StringEqualsIfExists":{
               "dynamodb:Select":"SPECIFIC_ATTRIBUTES",
               "dynamodb:ReturnValues":[
                  "NONE",
                  "UPDATED_OLD",
                  "UPDATED_NEW"
               ]
            }
         }
      }
   ]
}
```

Beachten Sie Folgendes:

- Die `UpdateItem`-Aktion benötigt, wie andere Schreibaktionen, Lesezugriff auf die Elemente, damit sie Werte vor und nach dem Aktualisieren zurückgeben kann. In der Richtlinie beschränken Sie die Aktionen, so dass nur auf Attribute zugegriffen werden kann, die für eine Aktualisierung zulässig sind, indem der `dynamodb:ReturnValues`-Bedingungsschlüssel angegeben wird. Der Bedingungsschlüssel beschränkt `ReturnValues` in der Anforderung auf das Angeben von `NONE`, `UPDATED_OLD` oder `UPDATED_NEW` und schließt `ALL_OLD` oder `ALL_NEW` nicht mit ein.
- Der `StringEqualsIfExists` Operator stellt sicher, dass, falls `dynamodb:Select` oder in der Anfrage vorhanden `dynamodb:ReturnValues` ist, diese mit den angegebenen Werten übereinstimmen muss. Dadurch wird verhindert, dass Operationen vollständige Elemente zurückgeben.
- Wenn Sie die Aktualisierung von Attributen einschränken, sollten Sie auch kontrollieren, welche Daten zurückgegeben werden können, um zu verhindern, dass Informationen über geschützte Attribute offengelegt werden.
- Die `PutItem`- und `DeleteItem`-Aktionen ersetzen ein gesamtes Element und erlauben daher Anwendungen, beliebige Attribute zu ändern. Wenn Sie also eine Anwendung darauf beschränken, nur bestimmte Attribute zu aktualisieren, sollten Sie für diese APIs keine Genehmigung erteilen.

Beispiel 5: Erteilen Sie Berechtigungen, um nur projizierte Attribute in einem Index abzufragen

Die folgende Berechtigungsrichtlinie erlaubt Abfragen über einen sekundären Index (`TopScoreDateTimeIndex`), indem der `dynamodb:Attributes`-Bedingungsschlüssel verwendet wird. Die Richtlinie beschränkt auch Abfragen auf das Anfordern nur bestimmter Attribute, die in den Index projiziert wurden.

Um eine Anwendung aufzufordern, eine Liste von Attributen in der Abfrage anzugeben, gibt die Richtlinie auch den Bedingungsschlüssel `dynamodb:Select` an, um sicherzustellen, dass der Parameter `Select` der `Query`-DynamoDB-Aktion `SPECIFIC_ATTRIBUTES` lautet. Die Liste von Attributen ist auf eine spezifische Liste beschränkt, die mithilfe des `dynamodb:Attributes`-Bedingungsschlüssels bereitgestellt wird.

JSON

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Sid":"QueryOnlyProjectedIndexAttributes",
         "Effect":"Allow",
         "Action":[
            "dynamodb:Query"
         ],
         "Resource":[
            "arn:aws:dynamodb:us-west-2:123456789012:table/GameScores/index/
TopScoreDateTimeIndex"
         ],
         "Condition":{
            "ForAllValues:StringEquals":{
               "dynamodb:Attributes":[
                  "TopScoreDateTime",
                  "GameTitle",
                  "Wins",
                  "Losses",
                  "Attempts"
               ]
            },
            "StringEquals":{
               "dynamodb:Select":"SPECIFIC_ATTRIBUTES"
            }
         }
```

```
      }
   ]
}
```

Die folgende Berechtigungsrichtlinie ist ähnlich, aber die Abfrage muss alle Attribute anfordern, die in den Index projiziert wurden.

JSON

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Sid":"QueryAllIndexAttributes",
         "Effect":"Allow",
         "Action":[
            "dynamodb:Query"
         ],
         "Resource":[
            "arn:aws:dynamodb:us-west-2:123456789012:table/GameScores/index/
TopScoreDateTimeIndex"
         ],
         "Condition":{
            "StringEquals":{
               "dynamodb:Select":"ALL_PROJECTED_ATTRIBUTES"
            }
         }
      }
   ]
}
```

Beispiel 6: Erteilen Sie Berechtigungen, um den Zugriff auf bestimmte Attribute und Partitionsschlüsselwerte einzuschränken

Die folgende Berechtigungsrichtlinie erlaubt bestimmte DynamoDB-Aktionen in einer Tabelle (angegeben im Element `Action`) und einem Tabellenindex (angegeben im Element `Resource`). Die Richtlinie verwendet den `dynamodb:LeadingKeys` Bedingungsschlüssel, um Berechtigungen nur auf die Elemente zu beschränken, deren Partitionsschlüsselwert mit der Facebook-ID des Benutzers übereinstimmt.

JSON

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Sid":"LimitAccessToCertainAttributesAndKeyValues",
         "Effect":"Allow",
         "Action":[
            "dynamodb:UpdateItem",
            "dynamodb:GetItem",
            "dynamodb:Query",
            "dynamodb:BatchGetItem"
         ],
         "Resource":[
            "arn:aws:dynamodb:us-west-2:123456789012:table/GameScores",
            "arn:aws:dynamodb:us-west-2:123456789012:table/GameScores/index/
TopScoreDateTimeIndex"
         ],
         "Condition":{
            "ForAllValues:StringEquals":{
               "dynamodb:LeadingKeys":[
                  "${graph.facebook.com:id}"
               ],
               "dynamodb:Attributes":[
                  "attribute-A",
                  "attribute-B"
               ]
            },
            "StringEqualsIfExists":{
               "dynamodb:Select":"SPECIFIC_ATTRIBUTES",
               "dynamodb:ReturnValues":[
                  "NONE",
                  "UPDATED_OLD",
                  "UPDATED_NEW"
               ]
            }
         }
      }
   ]
}
```

Beachten Sie Folgendes:

- Durch die Richtlinie (`UpdateItem`) zulässige Schreibaktionen können nur Attribut-A oder Attribut-B geändert werden.
- Da die Richtlinie `UpdateItem` zulässt, kann eine Anwendung neue Elemente einfügen und die ausgeblendeten Attribute sind in den neuen Elementen dann Null. Wenn diese Attribute in `TopScoreDateTimeIndex` projiziert werden, bietet die Richtlinie den zusätzlichen Vorteil, dass Abfragen verhindert werden, die Abrufe aus der Tabelle verursachen.
- Anwendungen können nur die in `dynamodb:Attributes` aufgeführten Attribute lesen. Mit dieser vorhandenen Richtlinie muss eine Anwendung den Parameter `Select` auf `SPECIFIC_ATTRIBUTES` in Leseanforderungen festlegen und es können nur Listenattribute angefordert werden. Die Anwendung kann bei Schreibanforderungen `ReturnValues` nicht auf `ALL_OLD` oder `ALL_NEW` festlegen und sie kann keine bedingten Schreibvorgänge basierend auf beliebigen anderen Attributen durchführen.

Beispiel 7: Verweigern Sie Berechtigungen, um den Zugriff auf bestimmte Attribute in einer Tabelle zu beschränken

Die folgende Richtlinie verweigert den Zugriff auf vertrauliche Attribute und stellt sicher, dass diese Einschränkung nicht durch das Auslassen eines Projektionsausdrucks umgangen werden kann. Sie ermöglicht den allgemeinen Zugriff auf die `CustomerData` Tabelle und verweigert gleichzeitig ausdrücklich den Zugriff auf Attribute. `SSN` `CreditCardNumber`

JSON

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Effect":"Allow",
         "Action":[
            "dynamodb:GetItem",
            "dynamodb:Query",
            "dynamodb:Scan"
         ],
         "Resource":"arn:aws:dynamodb:us-west-2:123456789012:table/CustomerData"
      },
      {
         "Effect":"Deny",
```

```
         "Action":[
            "dynamodb:GetItem",
            "dynamodb:Query",
            "dynamodb:Scan"
         ],
         "Resource":"arn:aws:dynamodb:us-west-2:123456789012:table/CustomerData",
         "Condition":{
            "ForAnyValue:StringEquals":{
               "dynamodb:Attributes":[
                  "SSN",
                  "CreditCardNumber"
               ]
            }
         }
      },
      {
         "Effect":"Deny",
         "Action":[
            "dynamodb:GetItem",
            "dynamodb:Query",
            "dynamodb:Scan"
         ],
         "Resource":"arn:aws:dynamodb:us-west-2:123456789012:table/CustomerData",
         "Condition":{
            "StringNotEqualsIfExists":{
               "dynamodb:Select":"SPECIFIC_ATTRIBUTES"
            }
         }
      }
   ]
}
```

Verwandte Themen

- Identity and Access Management für Amazon DynamoDB
- DynamoDB-API-Berechtigungen: Referenzliste für Aktionen, Ressourcen und Bedingungen

Verwenden des Web-Identitätsverbunds

Wenn Sie eine Anwendung schreiben, die für eine große Anzahl von Benutzern bestimmt ist, können Sie optional den Web-Identitätsverbund für die Authentifizierung und Autorisierung verwenden.

Mit dem Web-Identitätsverbund ist es nicht mehr erforderlich, einzelne -Benutzer zu erstellen. Stattdessen können sich Benutzer bei einem Identitätsanbieter anmelden und dann temporäre Sicherheitsanmeldedaten von AWS -Security-Token-Service (AWS STS) abrufen. Die App kann diese Anmeldeinformationen dann verwenden, um auf AWS Dienste zuzugreifen.

Ein Web-Identitätsverbund unterstützt die folgenden Identitätsanbieter:

- Login with Amazon
- Facebook
- Google

Weitere Ressourcen für den Web-Identitätsverbund

Die folgenden Ressourcen können Ihnen dabei helfen, mehr über den Web-Identitätsverbund zu erfahren:

- Der Post Web Identity Federation verwendet AWS SDK für .NET im AWS Entwicklerblog um zu erläutern, wie der Web-Identitätsverbund mit Facebook verwendet wird. Sie enthält Codefragmente in C#, die zeigen, wie eine IAM-Rolle mit Webidentität übernommen wird und wie temporäre Sicherheitsanmeldedaten für den Zugriff auf eine Ressource verwendet werden. AWS
- Die AWS Mobile SDK for iOS und die AWS Mobile SDK für Android enthalten Beispiel-Apps. Sie enthalten Code, der zeigt, wie die Identitätsanbieter aufgerufen werden und wie dann die Informationen von diesen Anbietern verwendet werden, um temporäre Sicherheitsanmeldeinformationen zu beziehen und zu nutzen.
- Der Artikel Web Identity Federation with Mobile Applications behandelt den Web-Identitätsverbund und zeigt ein Beispiel dafür, wie der Web-Identitätsverbund für den Zugriff auf eine Ressource verwendet werden kann. AWS

Beispielrichtlinie für einen Web-Identitätsverbund

Um zu zeigen, wie Sie den Web-Identitätsverbund mit DynamoDB verwenden können, schauen Sie sich noch einmal die GameScoresTabelle an, die in eingeführt wurde. Verwenden von IAM-Richtlinienbedingungen für die differenzierte Zugriffskontrolle Hier ist der Primärschlüssel für. GameScores

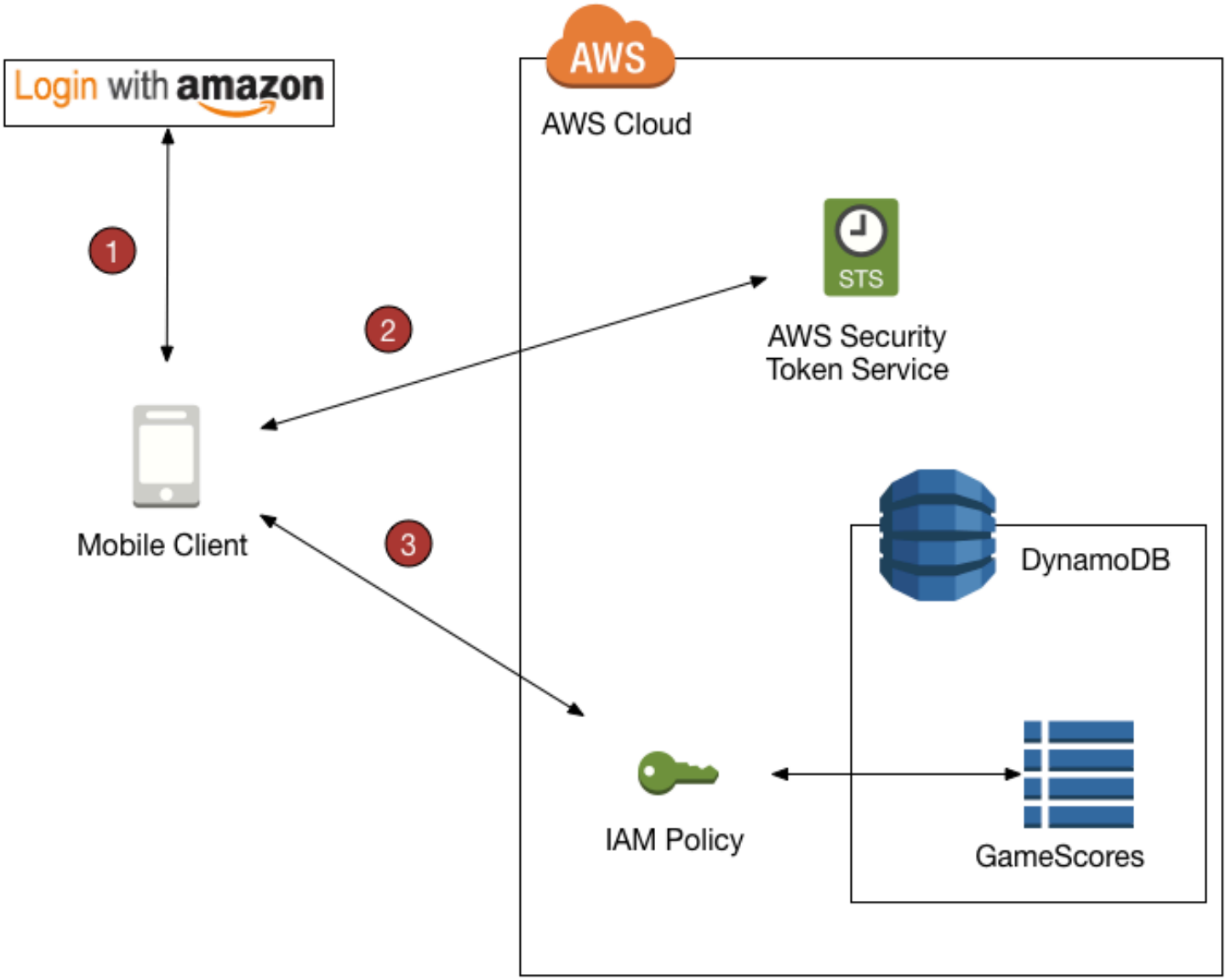
Tabellenname	Primärschlüsseltyp	Partitions-Schlüsselname und -Typ	Sortierschlüsselname und -Typ
GameScores (UserId, GameTitle, ...)	Zusammengesetzt	Name des Attributs: UserId Typ: Zeichenfolge	Name des Attributs: GameTitle Typ: Zeichenfolge

Angenommen, dass eine mobile Spielanwendung diese Tabelle verwendet und diese Anwendung Tausenden oder sogar Millionen von Benutzern unterstützen muss. Bei dieser Größenordnung wird es sehr schwierig, einzelne App-Benutzer zu verwalten und sicherzustellen, dass jeder Benutzer nur auf seine eigenen Daten in der GameScoresTabelle zugreifen kann. Glücklicherweise verfügen viele Benutzer bereits über Konten bei einem externen Identitätsanbieter wie Facebook, Google oder Login with Amazon. So ist es durchaus sinnvoll, einen dieser Anbieter für die Authentifizierungsaufgaben zu nutzen.

Um dies mit dem Web-Identitätsverbund durchzuführen, muss der Anwendungsentwickler die Anwendung bei einem Identitätsanbieter (wie Login with Amazon) registrieren und eine eindeutige Anwendungs-ID erhalten. Als nächstes muss der Entwickler eine IAM-Rolle erstellen. (In diesem Beispiel ist diese Rolle benannt GameRole.) Der Rolle muss ein IAM-Richtliniendokument beigefügt sein, in dem die Bedingungen angegeben sind, unter denen die App auf die GameScoresTabelle zugreifen kann.

Wenn ein Benutzer ein Spiel spielen möchte, meldet er sich mit seinem „Login with Amazon“-Konto bei der Spielanwendung an. Die App ruft dann AWS -Security-Token-Service (AWS STS) auf, gibt das Login mit der Amazon-App-ID an und fordert die Mitgliedschaft bei an GameRole. AWS STS gibt temporäre AWS Anmeldeinformationen an die App zurück und ermöglicht ihr, vorbehaltlich des GameRoleRichtliniendokuments, den Zugriff auf die GameScoresTabelle.

Das folgende Diagramm zeigt, wie diese einzelnen Komponenten zusammenpassen.



Übersicht eines Web-Identitätsverbunds

1. Die Anwendung ruft einen externen Identitätsanbieter auf, um den Benutzer und die Anwendung zu authentifizieren. Der Identitätsanbieter gibt ein Web-Identitäts-Token an die Anwendung zurück.
2. Die App ruft das Web-Identity-Token auf AWS STS und übergibt es als Eingabe. AWS STS autorisiert die App und gibt ihr temporäre AWS Zugangsdaten. Die App darf gemäß der Sicherheitsrichtlinie der Rolle eine IAM-Rolle (GameRole) annehmen und auf AWS Ressourcen zugreifen.
3. Die App ruft DynamoDB auf, um auf die GameScoresTabelle zuzugreifen. Da sie das übernommen hat GameRole, unterliegt die App den Sicherheitsrichtlinien, die mit dieser Rolle

verknüpft sind. Das Richtliniendokument verhindert, dass die Anwendung auf Daten zugreift, die dem Benutzer nicht gehören.

Hier ist noch einmal die Sicherheitsrichtlinie dafür GameRole, die gezeigt wurde in[Verwenden von IAM-Richtlinienbedingungen für die differenzierte Zugriffskontrolle](#):

JSON

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Sid":"AllowAccessToOnlyItemsMatchingUserID",
         "Effect":"Allow",
         "Action":[
            "dynamodb:GetItem",
            "dynamodb:BatchGetItem",
            "dynamodb:Query",
            "dynamodb:PutItem",
            "dynamodb:UpdateItem",
            "dynamodb:DeleteItem",
            "dynamodb:BatchWriteItem"
         ],
         "Resource":[
            "arn:aws:dynamodb:us-west-2:123456789012:table/GameScores"
         ],
         "Condition":{
            "ForAllValues:StringEquals":{
               "dynamodb:LeadingKeys":[
                  "${www.amazon.com:user_id}"
               ],
               "dynamodb:Attributes":[
                  "UserId",
                  "GameTitle",
                  "Wins",
                  "Losses",
                  "TopScore",
                  "TopScoreDateTime"
               ]
            },
            "StringEqualsIfExists":{
               "dynamodb:Select":"SPECIFIC_ATTRIBUTES"
```

```
            }
         }
      }
   ]
}
```

Die `Condition` Klausel bestimmt, welche Elemente in der App sichtbar GameScoressind. Dies geschieht durch das Vergleichen von Login with Amazon mit den `UserId`-Partitions-Schlüsselwerten in `GameScores`. Nur die Elemente des aktuellen Benutzers können verarbeitet werden, indem Sie eine der DynamoDB-Aktionen verwenden, die in dieser Richtlinie aufgeführt sind. Auf andere Elemente in der Tabelle kann nicht zugegriffen werden. Außerdem kann nur auf die spezifischen Attribute, die in der Richtlinie aufgeführt sind, zugegriffen werden.

Vorbereiten der Nutzung des Web-Identitätsverbunds

Wenn Sie ein Anwendungsentwickler sind und den Web-Identitätsverbund für Ihre Anwendung verwenden möchten, führen Sie die folgenden Schritte aus:

1. Anmelden als Entwickler bei einem externen Identitätsanbieter. Die folgenden externen Links stellen Informationen zur Anmeldung mit unterstützten Identitätsanbietern bereit:
   - Login with Amazon-Entwicklerzentrum
   - Registrierung auf der Facebook-Seite
   - Verwenden von OAuth 2.0 für den Zugriff APIs auf Google auf der Google-Website
2. Registrieren der Anwendung bei dem Identitätsanbieter. Wenn Sie dies tun, stellt der Anbieter Ihnen eine ID zur Verfügung, die in Ihrer Anwendung eindeutig ist. Wenn Sie möchten, dass Ihre Anwendung mit mehreren Identitätsanbietern arbeitet, müssen Sie eine Anwendungs-ID von jedem einzelnen Anbieter abrufen.
3. Erstellen Sie eine oder mehrere IAM-Rollen. Sie benötigen eine Rolle für jeden Identitätsanbieter für jede Anwendung. Sie können beispielsweise eine Rolle erstellen, die von einer Anwendung übernommen werden kann, in der der Benutzer sich mit Login with Amazon angemeldet hat, eine zweite Rolle für die gleiche Anwendung, in der der Benutzer sich mit Facebook angemeldet hat und eine dritte Rolle für die Anwendung, in der sich der Benutzer mit Google angemeldet hat.

   Im Rahmen des Rollenerstellungsprozesses, müssen Sie eine IAM-Richtlinie der Rolle zuordnen. Ihr Richtliniendokument sollte die DynamoDB-Ressourcen, die für Ihre Anwendung erforderlich sind, und die Berechtigungen für den Zugriff auf diese Ressourcen definieren.

Weitere Informationen finden Sie unter Informationen zum Web-Identitätsverbund im IAM-Benutzerhandbuch.

> **Note**
>
> Als Alternative können Sie Amazon Cognito verwenden. AWS -Security-Token-Service Amazon Cognito ist der bevorzugte Service für die Verwaltung von temporären Anmeldeinformationen für mobile Anwendungen. Weitere Informationen finden Sie unter Abrufen von Anmeldeinformationen im Entwicklerhandbuch zu Amazon Cognito.

Generieren einer IAM-Richtlinie mit der DynamoDB-Konsole

Die DynamoDB-Konsole kann Ihnen helfen, eine IAM-Richtlinie für die Verwendung mit dem Web-Identitätsverbund zu erstellen. Dazu wählen Sie eine DynamoDB-Tabelle aus und geben den Identitätsanbieter, Aktionen und Attribute an, die in der Richtlinie enthalten sein sollen. Die DynamoDB-Konsole erstellt dann eine Richtlinie, die Sie einer IAM-Rolle zuordnen können.

1. Melden Sie sich bei der an AWS-Managementkonsole und öffnen Sie die DynamoDB-Konsole unter. https://console.aws.amazon.com/dynamodb/
2. Wählen Sie im Navigationsbereich Tables (Tabellen) aus.
3. Wählen Sie in der Liste der Tabellen die Tabelle aus, für die Sie die IAM-Richtlinie erstellen möchten.
4. Wählen Sie die Schaltfläche Aktionen und Zugriffskontrollrichtlinie erstellen aus.
5. Wählen Sie den Identitätsanbieter, Aktionen und Attribute für die Richtlinie aus.

   Wenn Sie die gewünschten Einstellungen vorgenommen haben, wählen Sie Richtlinie generieren aus. Die erstellte Richtlinie wird angezeigt.
6. Wählen Sie auf Dokumentation anzeigen aus und befolgen Sie die erforderlichen Schritte, um die generierte Richtlinie einer IAM-Rolle zuzuordnen.

Schreiben Ihrer Anwendung, um den Web-Identitätsverbund zu nutzen

Um den Web-Identitätsverbund zu verwenden, muss die Anwendung die IAM-Rolle übernehmen, die Sie erstellt haben. Von diesem Punkt an berücksichtigt die Anwendung die Zugriffsrichtlinie, die mit der Rolle verknüpft ist.

Wenn Ihre Anwendung den Web-Identitätsverbund zur Laufzeit nutzt, muss sie die folgenden Schritte befolgen:

1. Authentifizieren mit einem externen Identitätsanbieter. Ihre Anwendung muss den Identitätsanbieter mithilfe der Schnittstelle, die er bereitstellt, aufrufen. Die genaue Art und Weise, wie Sie den Benutzer authentifizieren, hängt von dem Anbieter und der Plattform ab, auf der Sie die Anwendung ausführen. Wenn der Benutzer nicht bereits angemeldet ist, kümmert sich der Identitätsanbieter in der Regel um die Anzeige einer Anmeldeseite dieses Anbieters.

   Nachdem der Identitätsanbieter den Benutzer authentifiziert hat, gibt der Anbieter ein Web-Identitäts-Token an Ihre Anwendung zurück. Das Format dieses Tokens hängt von dem Anbieter ab, ist in der Regel aber eine sehr lange Zeichenfolge.
2. Besorgen Sie sich temporäre AWS Sicherheitsanmeldeinformationen. Zu diesem Zweck sendet die App eine `AssumeRoleWithWebIdentity`-Anforderung an AWS -Security-Token-Service (AWS STS). Diese Anforderung enthält Folgendes:
   - Das Web-Identitäts-Token aus dem vorherigen Schritt
   - Die Anwendungs-ID des Identitätsanbieters
   - Der Amazon-Ressourcenname (ARN) der IAM;-Rolle, die Sie für diesen Identitätsanbieter für diese Anwendung erstellt haben

   AWS STS gibt eine Reihe von AWS Sicherheitsanmeldeinformationen zurück, die nach einer bestimmten Zeit ablaufen (standardmäßig 3.600 Sekunden).

   Das Folgende ist eine Beispielanforderung und -antwort von einer `AssumeRoleWithWebIdentity`-Aktion in AWS STS. Das Web-Identitäts-Token wurde von dem „Login with Amazon“-Identitätsanbieter erhalten.

```
GET / HTTP/1.1
Host: sts.amazonaws.com
Content-Type: application/json; charset=utf-8
URL: https://sts.amazonaws.com/?ProviderId=www.amazon.com
&DurationSeconds=900&Action=AssumeRoleWithWebIdentity
&Version=2011-06-15&RoleSessionName=web-identity-federation
&RoleArn=arn:aws:iam::123456789012:role/GameRole
&WebIdentityToken=Atza|IQEBLjAsAhQluyKqyBiYZ8-kclvGTYM81e...(remaining characters
 omitted)
```

```
<AssumeRoleWithWebIdentityResponse
  xmlns="https://sts.amazonaws.com/doc/2011-06-15/">
  <AssumeRoleWithWebIdentityResult>
    <SubjectFromWebIdentityToken>amzn1.account.AGJZDKHJKAUUSW6C44CHPEXAMPLE</
SubjectFromWebIdentityToken>
    <Credentials>
      <SessionToken>AQoDYXdzEMf//////////wEa8AP6nNDwcSLnf+cHupC...(remaining
 characters omitted)</SessionToken>
      <SecretAccessKey>8Jhi60+EWUUbbUShTEsjTxqQtM8UKvsM6XAjdA==</SecretAccessKey>
      <Expiration>2013-10-01T22:14:35Z</Expiration>
      <AccessKeyId>06198791C436IEXAMPLE</AccessKeyId>
    </Credentials>
    <AssumedRoleUser>
      <Arn>arn:aws:sts::123456789012:assumed-role/GameRole/web-identity-federation</
Arn>
      <AssumedRoleId>AROAJU4SA2VW5SZRF2YMG:web-identity-federation</AssumedRoleId>
    </AssumedRoleUser>
  </AssumeRoleWithWebIdentityResult>
  <ResponseMetadata>
    <RequestId>c265ac8e-2ae4-11e3-8775-6969323a932d</RequestId>
  </ResponseMetadata>
</AssumeRoleWithWebIdentityResponse>
```

3. Greifen Sie auf AWS Ressourcen zu. Die Antwort von AWS STS enthält Informationen, die die Anwendung benötigt, um auf DynamoDB-Ressourcen zuzugreifen:
   - Die Felder `AccessKeyID`-, `SecretAccessKey`- und `SessionToken` beinhalten Sicherheitsanmeldeinformationen, die nur für diesen Benutzer und diese Anwendung gültig sind.
   - Das `Expiration`-Feld gibt die Frist für diese Anmeldeinformationen an, nach der sie nicht mehr gültig sind.
   - Das `AssumedRoleId`-Feld enthält den Namen einer sitzungsspezifischen IAM-Rolle, der von der Anwendung übernommen wurde. Die Anwendung berücksichtigt die Zugriffskontrollen im IAM–Richtliniendokument für die Dauer dieser Sitzung.
   - Das `SubjectFromWebIdentityToken`-Feld enthält die eindeutige ID, die in einer IAM-Richtlinienvariablen für diesen bestimmten Identitätsanbieter erscheint. Die Folgenden sind die IAM-Richtlinienvariablen für unterstützte Anbieter und einige Beispielwerte für diese:

Richtlinienvariable	Beispielwert
`${www.amazon.com:user_id}`	`amzn1.account.AGJZDKHJKAUUSW6C44CHPEXAMPLE`
`${graph.facebook.com:id}`	`123456789`
`${accounts.google.com:sub}`	`12345678901234567890`

Beispiele für IAM-Richtlinien, in denen diese Richtlinienvariablen verwendet werden, finden Sie unter Beispielrichtlinien: Verwenden von Bedingungen für die differenzierte Zugriffskontrolle.

Weitere Informationen zur AWS STS Generierung temporärer Zugangsdaten finden Sie unter Temporäre Sicherheitsanmeldedaten anfordern im IAM-Benutzerhandbuch.

DynamoDB-API-Berechtigungen: Referenzliste für Aktionen, Ressourcen und Bedingungen

Identity and Access Management für Amazon DynamoDBWenn Sie eine Berechtigungsrichtlinie für eine IAM-Identität (identitätsbasierte Richtlinie) verfassen, können Sie die Liste von Aktionen, Ressourcen und Bedingungsschlüssel für Amazon DynamoDB im IAM-Benutzerhandbuch-Referenz verwenden. Auf der Seite werden alle DynamoDB-API-Operationen, die entsprechenden Aktionen, für die Sie Berechtigungen zur Ausführung der Aktion erteilen können, und die AWS Ressource, für die Sie die Berechtigungen erteilen können, aufgeführt. Die Aktionen geben Sie im Feld `Action` und den Wert für die Ressource im Feld `Resource` der Richtlinie an.

Sie können in Ihren DynamoDB-Richtlinien AWS-weite Bedingungsschlüssel verwenden, um Bedingungen auszudrücken. Eine vollständige Liste der AWS-weiten Schlüssel finden Sie in der Referenz zu den IAM-JSON-Richtlinienelementen im IAM-Benutzerhandbuch.

Zusätzlich zu den AWS-weiten Bedingungsschlüsseln verfügt DynamoDB über eigene spezifische Schlüssel, die Sie in Bedingungen verwenden können. Weitere Informationen finden Sie unter Verwenden von IAM-Richtlinienbedingungen für die differenzierte Zugriffskontrolle.

Verwandte Themen

- Identity and Access Management für Amazon DynamoDB

- Verwenden von IAM-Richtlinienbedingungen für die differenzierte Zugriffskontrolle

Compliance-Validierung nach Branche für DynamoDB

Informationen darüber, ob AWS-Service ein AWS-Services in den Geltungsbereich bestimmter Compliance-Programme fällt, finden Sie unter Umfang nach Compliance-Programm AWS-Services unter . Wählen Sie dort das Compliance-Programm aus, an dem Sie interessiert sind. Allgemeine Informationen finden Sie unter AWS Compliance-Programme AWS .

Sie können Prüfberichte von Drittanbietern unter herunterladen AWS Artifact. Weitere Informationen finden Sie unter Berichte herunterladen unter .

Ihre Verantwortung für die Einhaltung der Vorschriften bei der Nutzung AWS-Services hängt von der Vertraulichkeit Ihrer Daten, den Compliance-Zielen Ihres Unternehmens und den geltenden Gesetzen und Vorschriften ab. Weitere Informationen zu Ihrer Verantwortung für die Einhaltung der Vorschriften bei der Nutzung AWS-Services finden Sie in der AWS Sicherheitsdokumentation.

Ausfallsicherheit und Notfallwiederherstellung in Amazon DynamoDB

Die AWS globale Infrastruktur basiert auf AWS Regionen und Availability Zones. AWS Regionen bieten mehrere physisch getrennte und isolierte Availability Zones, die über Netzwerke mit niedriger Latenz, hohem Durchsatz und hoher Redundanz miteinander verbunden sind. Mithilfe von Availability Zones können Sie Anwendungen und Datenbanken erstellen und ausführen, die automatisch Failover zwischen Availability Zones ausführen, ohne dass es zu Unterbrechungen kommt. Availability Zones sind besser hoch verfügbar, fehlertoleranter und skalierbarer als herkömmliche Infrastrukturen mit einem oder mehreren Rechenzentren.

Falls Sie Ihre Daten oder Anwendungen über größere geografische Distanzen hinweg replizieren müssen, verwenden Sie lokale AWS -Regionen. Eine AWS lokale Region ist ein einzelnes Rechenzentrum, das eine bestehende AWS Region ergänzen soll. Wie alle AWS Regionen sind AWS lokale Regionen vollständig von anderen AWS Regionen isoliert.

Weitere Informationen zu AWS Regionen und Availability Zones finden Sie unter AWS Globale Infrastruktur.

Amazon DynamoDB repliziert Ihre Daten automatisch in drei Availability Zones in einer Region und bietet so eine integrierte hohe Beständigkeit und ein SLA mit einer Verfügbarkeit von 99,99%.

Darüber hinaus bietet DynamoDB mehrere Funktionen, mit denen Sie Ihre Anforderungen an Datenstabilität und Backup erfüllen können.

On-Demand-Backup und Wiederherstellung

DynamoDB bietet On-Demand-Sicherungsfunktionen. Es ermöglicht Ihnen, vollständige Sicherungen Ihrer Tabellen für die langfristige Aufbewahrung und Archivierung zu erstellen. Weitere Informationen finden Sie unter On-Demand-Backup und Wiederherstellung für DynamoDB.

Point-in-time Wiederherstellung

Point-in-time Recovery schützt Ihre DynamoDB-Tabellen vor versehentlichen Schreib- oder Löschvorgängen. Mit der zeitpunktbezogenen Wiederherstellung müssen Sie sich keine Gedanken über das Erstellen, Warten oder Planen von On-Demand-Backups machen. Weitere Informationen finden Sie unter Point-in-time Recovery for DynamoDB.

Globale Tabellen, die über AWS -Regionen hinweg synchronisiert werden

DynamoDB verteilt die Daten und den Datenverkehr für Ihre Tabellen automatisch auf eine ausreichende Anzahl von Servern entsprechend Ihren Anforderungen an Durchsatz und Speicherung. Außerdem ist für eine konsistente und schnelle Leistung gesorgt. Alle Ihre Daten werden auf Solid-State-Festplatten (SSDs) gespeichert und automatisch über mehrere Availability Zones in einer AWS Region repliziert, wodurch eine integrierte Hochverfügbarkeit und Datenbeständigkeit gewährleistet wird. Sie können globale Tabellen verwenden, um DynamoDB-Tabellen regionsübergreifend AWS synchron zu halten.

Infrastruktursicherheit in Amazon DynamoDB

Als verwalteter Service ist Amazon DynamoDB durch die AWS globalen Netzwerksicherheitsverfahren geschützt, die unter Infrastrukturschutz im AWS Well-Architected Framework beschrieben sind.

Sie verwenden AWS veröffentlichte API-Aufrufe, um über das Netzwerk auf DynamoDB zuzugreifen. Clients können TLS (Transport Layer Security) Version 1.2 oder 1.3 verwenden. Clients müssen außerdem Cipher Suites mit PFS (Perfect Forward Secrecy) wie DHE (Ephemeral Diffie-Hellman) oder ECDHE (Elliptic Curve Ephemeral Diffie-Hellman) unterstützen. Die meisten modernen Systeme wie Java 7 und höher unterstützen diese Modi. Außerdem müssen Anforderungen mit einer Zugriffsschlüssel-ID und einem geheimen Zugriffsschlüssel signiert sein, der einem IAM-Prinzipal

zugeordnet ist. Alternativ können Sie mit AWS -Security-Token-Service (AWS STS) temporäre Sicherheitsanmeldeinformationen erstellen, um die Anforderungen zu signieren.

Sie können auch einen VPC-Endpunkt (Virtual Private Cloud) für DynamoDB verwenden, um Amazon-EC2-Instances in Ihrer VPC die Verwendung privater IP-Adressen für den Zugriff auf DynamoDB zu ermöglichen, sodass sie nicht im öffentlichen Internet verfügbar sind. Weitere Informationen finden Sie unter Verwenden von Amazon-VPC-Endpunkten für den Zugriff auf DynamoDB.

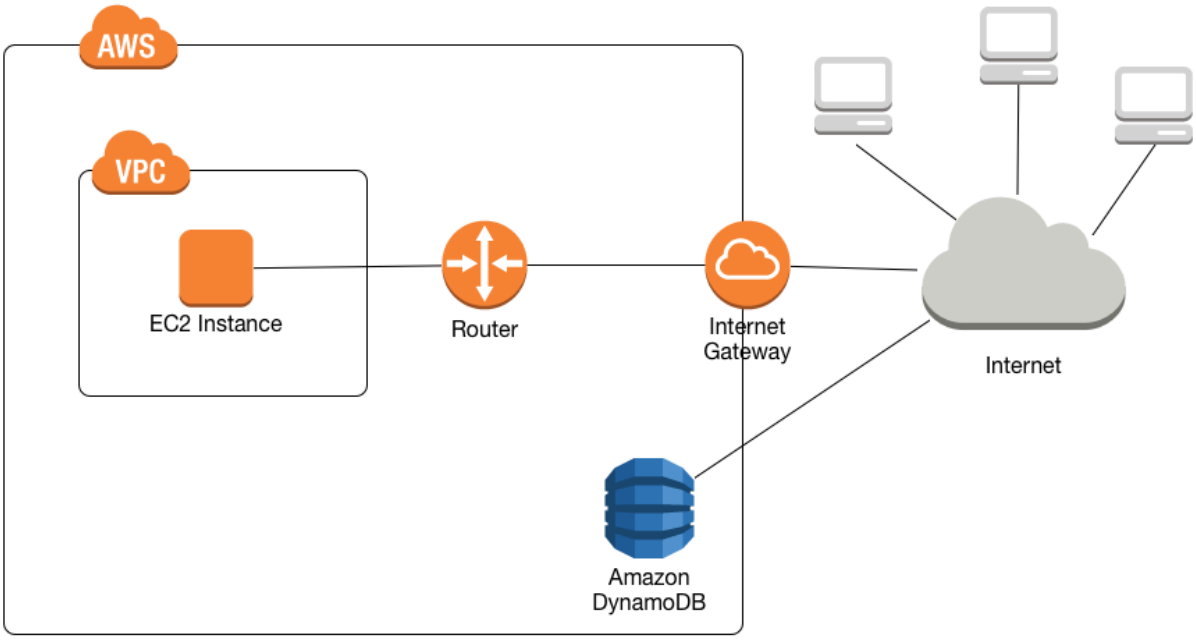
Verwenden von Amazon-VPC-Endpunkten für den Zugriff auf DynamoDB

Aus Sicherheitsgründen führen viele AWS Kunden ihre Anwendungen in einer Amazon Virtual Private Cloud Cloud-Umgebung (Amazon VPC) aus. Mit Amazon VPC können Sie Amazon-EC2-Instances in einer Virtual Private Cloud starten, die von anderen Netzwerken logisch isoliert ist – einschließlich des öffentlichen Internets. Mit einer Amazon VPC können Sie den zugehörigen IP-Adressbereich, die Subnetze, Routing-Tabellen, Netzwerk-Gateways und Sicherheitseinstellungen steuern.

> **Note**
>
> Wenn Sie Ihre AWS-Konto nach dem 4. Dezember 2013 erstellt haben, haben Sie in jeder AWS-Region bereits eine Standard-VPC. Eine Standard-VPC ist sofort einsatzbereit - Sie können Ihre Standard-VPC umgehend einsetzen, ohne weitere Konfigurationsschritte ausführen zu müssen.
> Weitere Informationen finden Sie unter Standard-VPC und Standard-Subnetze im Amazon-VPC-Benutzerhandbuch.

Um auf das öffentliche Internet zuzugreifen, muss Ihre VPC über ein Internet-Gateway verfügen, einen virtuellen Router, der Ihr VPC mit dem Internet verbindet. Auf diese Weise können Anwendungen, die in Ihrem VPC in Amazon EC2 ausgeführt werden, auf Internetressourcen wie Amazon DynamoDB zugreifen.

Standardmäßig wird für die Kommunikation zu und von DynamoDB das HTTPS-Protokoll verwendet, das den Netzwerkverkehr durch Verschlüsselung SSL/TLS schützt. Das folgende Diagramm zeigt eine Amazon-EC2-Instance in einer VPC, die auf DynamoDB zugreift, indem DynamoDB ein Internet-Gateway anstelle von VPC-Endpunkten verwendet.



Viele Kunden haben legitime Bedenken hinsichtlich Datenschutz und Sicherheit, was das Senden und Empfangen von Daten über das öffentliche Internet angeht. Kunden können dieses Problem durch die Verwendung eines Virtual Private Network (VPN) lösen, um sämtlichen DynamoDB-Netzwerkdatenverkehr über die eigene Unternehmensnetzwerkinfrastruktur zu leiten. Dieser Ansatz kann jedoch zu Herausforderungen hinsichtlich Bandbreite und Verfügbarkeit führen.

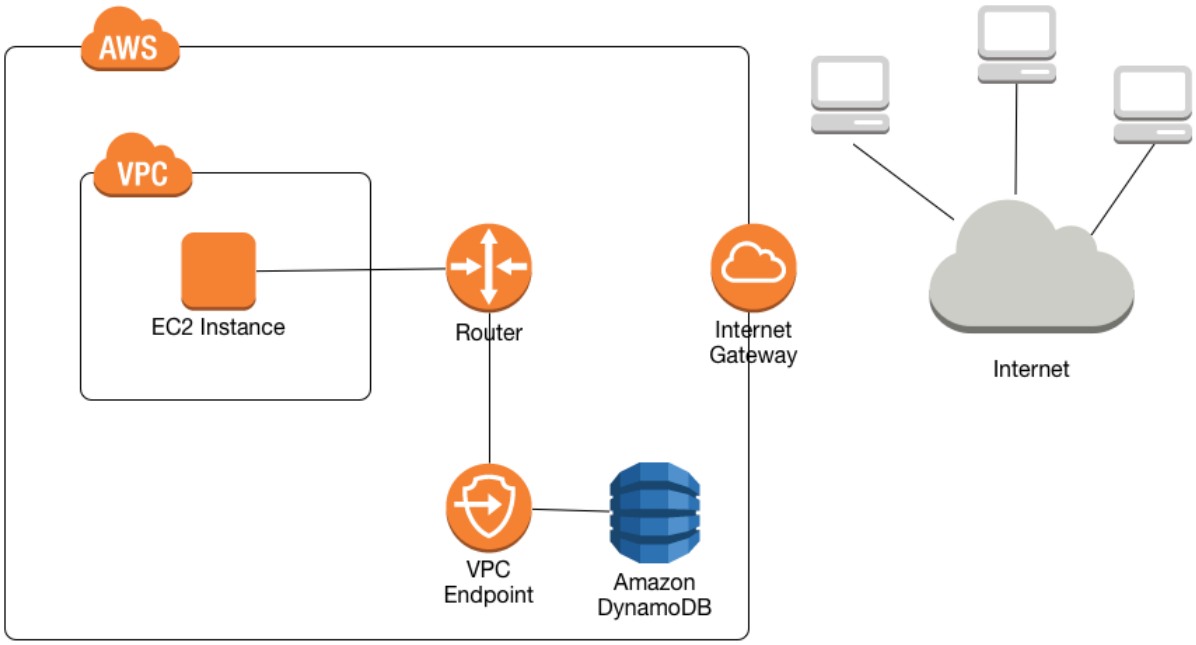
VPC-Endpunkte für DynamoDB können diese Herausforderungen bewältigen helfen. Ein VPC-Endpunkt für DynamoDB ermöglicht den Amazon-EC2-Instances in Ihrer VPC die Verwendung privater IP-Adressen für den Zugriff auf DynamoDB, sodass sie nicht im öffentlichen Internet sichtbar sind. Ihre EC2-Instances benötigen keine öffentlichen IP-Adressen und Sie benötigen keinen Internet-Gateway, kein NAT-Gerät und kein virtuelles privates Gateway in Ihrer VPC. Sie steuern den Zugriff auf DynamoDB mittels Endpunktrichtlinien. Der Verkehr zwischen Ihrer VPC und dem AWS Service verlässt das Amazon-Netzwerk nicht.

> **Note**
>
> Selbst wenn Sie öffentliche IP-Adressen verwenden, wird die gesamte VPC-Kommunikation zwischen Instances und Diensten, in denen gehostet AWS wird, innerhalb des AWS Netzwerks privat gehalten. Pakete, die aus dem AWS Netzwerk mit einem Ziel im

AWS Netzwerk stammen, verbleiben im AWS globalen Netzwerk, mit Ausnahme des Datenverkehrs in oder aus AWS China Regionen.

Wenn Sie einen VPC-Endpunkt für DynamoDB erstellen, werden alle Anfragen für einen DynamoDB-Endpunkt in der Region (z. B. dynamodb.us-west-2.amazonaws.com) an einen privaten DynamoDB-Endpunkt innerhalb des Amazon-Netzwerks geleitet. Sie müssen Ihre Anwendungen, die auf EC2-Instances in Ihrer VPC ausgeführt werden, nicht ändern. Der Endpunktname bleibt gleich, aber die Route zu DynamoDB bleibt vollständig im Amazon-Netzwerk und greift nicht auf das öffentliche Internet zu.

Das folgende Diagramm zeigt, wie eine EC2-Instance in einer VPC einen VPC-Endpunkt verwenden kann, um auf DynamoDB zuzugreifen.



Weitere Informationen finden Sie unter the section called “Tutorial: Verwendung eines VPC-Endpunkts für DynamoDB”.

Freigabe von Amazon-VPC-Endpunkten und DynamoDB

Um Zugriff auf DynamoDB über den Gateway-Endpunkt eines VPC-Subnetzes ermöglichen zu können, müssen Sie über Eigentümerkontoberechtigungen für dieses VPC-Subnetz verfügen.

Sobald dem Gateway-Endpunkt des VPC-Subnetzes Zugriff auf DynamoDB gewährt wurde, kann jedes AWS -Konto, das über Zugriff auf dieses Subnetz verfügt, DynamoDB verwenden. Dies bedeutet, dass alle Kontonutzer innerhalb des VPC-Subnetzes alle DynamoDB-Tabellen verwenden können, auf die sie Zugriff haben. Dies umfasst DynamoDB-Tabellen, die einem anderen Konto als dem VPC-Subnetz zugeordnet sind. Der Eigentümer des VPC-Subnetzes kann bestimmte Benutzer innerhalb des Subnetzes weiterhin nach eigenem Ermessen daran hindern, DynamoDB über den Gateway-Endpunkt zu nutzen.

Tutorial: Verwendung eines VPC-Endpunkts für DynamoDB

In diesem Abschnitt werden Sie schrittweise durch die Einrichtung und Verwendung eines VPC-Endpunkts für DynamoDB geführt.

Themen

- Schritt 1: Starten einer Amazon-EC2-Instance
- Schritt 2: Konfigurieren Ihrer Amazon-EC2-Instance
- Schritt 3: Erstellen eines VPC-Endpunkts für DynamoDB
- Schritt 4: (Optional): Bereinigen

Schritt 1: Starten einer Amazon-EC2-Instance

In diesem Schritt starten Sie eine Amazon-EC2-Instance in Ihrer Standard-Amazon-VPC. Anschließend können Sie einen VPC Endpunkt für DynamoDB erstellen und verwenden.

1. Öffnen Sie die Amazon-EC2-Konsole unter https://console.aws.amazon.com/ec2/.
2. Wählen Sie Launch Instance aus und gehen Sie folgendermaßen vor:

   Schritt 1: Auswählen eines Amazon Machine Images (AMI)

   - Gehen Sie oben in der Liste von AMIs zu Amazon Linux AMI und wählen Sie Select aus.

   Schritt 2: Auswählen eines Instance-Typs

   - Wählen Sie oben in der Liste der Instance-Typen t2.micro aus.
   - Wählen Sie Next: Configure Instance Details aus.

   Schritt 3: Konfigurieren der Instance-Details

- Navigieren Sie zu Network und wählen Sie Ihre Standard-VPC aus.

  Wählen Sie Next: Add Storage aus.

  Schritt 4: Hinzufügen von Speicher

- Überspringen Sie diesen Schritt, indem Sie Next: Tag Instance auswählen.

  Schritt 5: Kennzeichnen der Instance

- Überspringen Sie diesen Schritt, indem Sie Next: Configure Security Group auswählen.

  Schritt 6: Konfigurieren einer Sicherheitsgruppe

- Wählen Sie Bestehende Sicherheitsgruppe auswählen aus.
- Wählen Sie in der Liste der Sicherheitsgruppen default aus. Dies ist die Standard-Sicherheitsgruppe für Ihre VPC.
- Wählen Sie Next: Review and Launch aus.

  Schritt 7: Prüfen eines Starts einer Instance

- Wählen Sie Launch (Starten) aus.

3. Führen Sie im Fenster Select an existing key pair or create a new key pair einen der folgenden Schritte aus:

   - Wenn Sie nicht über ein Amazon-EC2-Schlüsselpaar verfügen, wählen Sie Einen neuen Schlüsselpaar erstellen aus und befolgen Sie die Anweisungen. Sie werden dazu aufgefordert eine private Schlüsseldatei herunterzuladen (.pem-Datei). Sie benötigen diese Datei später, wenn Sie sich bei Ihrer Amazon-EC2-Instance anmelden.
   - Wenn Sie bereits über ein bestehendes Amazon-EC2-Schlüsselpaar verfügen, navigieren Sie zu Schlüsselpaar wählen und wählen Sie Ihr Schlüsselpaar aus der Liste aus. Beachten Sie, dass Sie bereits über die private Schlüsseldatei (.pem-Datei) verfügen müssen, um sich bei Ihrer Amazon-EC2-Instance anmelden zu können.

4. Wenn Sie Ihr Schlüsselpaar konfiguriert haben, wählen Sie Launch Instances (Instances starten) aus.

5. Kehren Sie zur Startseite der Amazon-EC2-Konsole zurück und wählen Sie die Instance, die Sie gestartet haben. Suchen Sie im unteren Bereich der Registerkarte Beschreiben die Public DNS für Ihre Instance. Beispiel: `ec2-00-00-00-00.us-east-1.compute.amazonaws.com`.

   Notieren Sie sich diesen öffentlichen DNS-Namen, da Sie ihn für den nächsten Schritt in diesem Tutorial benötigen (Schritt 2: Konfigurieren Ihrer Amazon-EC2-Instance).

> **Note**
>
> Es dauert einige Minuten, bis Ihre Amazon-EC2-Instance verfügbar wird. Überprüfen Sie, ob Instance-Zustand `running` wird und alle Status-Kontrollen bestanden hat, bevor Sie mit dem nächsten Schritt fortfahren.

Schritt 2: Konfigurieren Ihrer Amazon-EC2-Instance

Wenn Ihre Amazon-EC2-Instance verfügbar ist, können Sie sich bei ihr anmelden und auf die erstmalige Verwendung vorbereiten.

> **Note**
>
> In den folgenden Schritten wird davon ausgegangen, dass Sie eine Verbindung mit Ihrer Amazon-EC2-Instance von einem Linux-Computer aus herstellen. Informationen über andere Möglichkeiten zum Herstellen einer Verbindung finden Sie unter Herstellen einer Verbindung mit Ihrer Linux-Instance im Amazon-EC2-Benutzerhandbuch.

1. Sie müssen eingehenden SSH-Datenverkehr für Ihre Amazon-EC2-Instance autorisieren. Dazu erstellen Sie eine neue EC2-Sicherheitsgruppe und weisen anschließend die Sicherheitsgruppe Ihrer EC2-Instance zu.

   a. Wählen Sie im Navigationsbereich Sicherheitsgruppen aus.

   b. Wählen Sie Sicherheitsgruppen erstellen aus. Führen Sie im Fenster Sicherheitsgruppen erstellen Folgendes aus:

      - Sicherheitsgruppenname – Geben Sie einen Namen für die Sicherheitsgruppe ein. Zum Beispiel: `my-ssh-access`
      - Description – Geben Sie eine kurze Beschreibung für die Sicherheitsgruppe ein.

- VPC – Wählen Sie Ihre Standard-VPC aus.
- Wählen Sie im Abschnitt Sicherheitsgruppen-Regeln die Option Regel hinzufügen aus und führen Sie Folgendes aus:
  - Type – Wählen Sie „SSH“ aus.
  - Source – Wählen Sie „My IP“ aus.

Wenn Sie die gewünschten Einstellungen vorgenommen haben, wählen Sie Erstellen.

c. Wählen Sie im Navigationsbereich Instances aus.

d. Wählen Sie die Amazon-EC2-Instance aus, die Sie in Schritt 1: Starten einer Amazon-EC2-Instance gestartet haben.

e. Wählen Sie Aktionen --> Netzwerk --> Sicherheitsgruppen ändern aus.

f. Wählen Sie in Sicherheitsgruppen ändern die Sicherheitsgruppe aus, die Sie früher in diesem Verfahren erstellt haben (z. B. `my-ssh-access`). Die vorhandene `default`-Sicherheitsgruppe sollte ebenfalls ausgewählt werden. Wenn Sie die gewünschten Einstellungen vorgenommen haben, wählen Sie Zuweisen von Sicherheitsgruppen aus.

2. Verwenden Sie den Befehl `ssh`, wie im folgenden Beispiel gezeigt, um sich bei der Amazon-EC2-Instance anzumelden.

```
ssh -i my-keypair.pem ec2-user@public-dns-name
```

Sie werden Ihre private Schlüsseldatei (.pem-Datei) und den öffentlichen DNS-Namen Ihrer Instance angeben müssen. (Siehe Schritt 1: Starten einer Amazon-EC2-Instance).

Die Anmelde-ID lautet `ec2-user`. Es ist kein Passwort erforderlich.

3. Konfigurieren Sie Ihre AWS Anmeldeinformationen wie im folgenden Beispiel gezeigt. Geben Sie bei Aufforderung Ihre AWS -Zugriffsschlüssel-ID, den geheimen Schlüssel und den Namen der Standardregion ein.

```
aws configure
```

```
AWS Access Key ID [None]: AKIAIOSFODNN7EXAMPLE
AWS Secret Access Key [None]: wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY
Default region name [None]: us-east-1
Default output format [None]:
```

Sie können nun einen VPC-Endpunkt für DynamoDB erstellen.

Schritt 3: Erstellen eines VPC-Endpunkts für DynamoDB

In diesem Schritt erstellen Sie einen VPC-Endpunkt für DynamoDB und testen diesen, um sicherzustellen, dass er funktioniert.

1. Überprüfen Sie, ob Sie unter Verwendung des folgenden öffentlichen Endpunkts mit DynamoDB kommunizieren können, bevor Sie beginnen.

   ```
   aws dynamodb list-tables
   ```

   Die Ausgabe zeigt eine Liste der DynamoDB-Tabellen, die Sie zurzeit besitzen. (Wenn Sie keine Tabellen besitzen, ist die Liste leer.)
2. Stellen Sie sicher, dass DynamoDB ein verfügbarer Dienst für die Erstellung von VPC-Endpoints in der aktuellen Region ist. AWS (Der Befehl wird in Fettschrift angezeigt, gefolgt von einer Beispielausgabe.)

   ```
   aws ec2 describe-vpc-endpoint-services
   ```

   ```
   {
       "ServiceNames": [
           "com.amazonaws.us-east-1.s3",
           "com.amazonaws.us-east-1.dynamodb"
       ]
   }
   ```

   In der Beispielausgabe ist DynamoDB als Service verfügbar. Sie können daher einen VPC-Endpunkt für diesen Service erstellen.
3. Ermitteln Sie den VPC-Bezeichner.

   ```
   aws ec2 describe-vpcs
   ```

   ```
   {
       "Vpcs": [
           {
               "VpcId": "vpc-0bbc736e",
               "InstanceTenancy": "default",
               "State": "available",
   ```

```
            "DhcpOptionsId": "dopt-8454b7e1",
            "CidrBlock": "172.31.0.0/16",
            "IsDefault": true
        }
    ]
}
```

In der Beispielausgabe ist die VPC-ID `vpc-0bbc736e`.

4. Erstellen des VPC-Endpunkts Geben Sie für den Parameter `--vpc-id` die VPC-ID aus dem vorherigen Schritt an. Verwenden Sie den `--route-table-ids`-Parameter, um den Endpunkt mit den Routentabellen zu verknüpfen.

```
aws ec2 create-vpc-endpoint --vpc-id vpc-0bbc736e --service-name com.amazonaws.us-
east-1.dynamodb --route-table-ids rtb-11aa22bb
```

```
{
    "VpcEndpoint": {
        "PolicyDocument": "{\"Version\":\"2008-10-17\",\"Statement\":[{\"Effect\":
\"Allow\",\"Principal\":\"*\",\"Action\":\"*\",\"Resource\":\"*\"}]}",
        "VpcId": "vpc-0bbc736e",
        "State": "available",
        "ServiceName": "com.amazonaws.us-east-1.dynamodb",
        "RouteTableIds": [
            "rtb-11aa22bb"
        ],
        "VpcEndpointId": "vpce-9b15e2f2",
        "CreationTimestamp": "2017-07-26T22:00:14Z"
    }
}
```

5. Überprüfen Sie, ob Sie über den VPC-Endpunkt auf DynamoDB zugreifen können:

```
aws dynamodb list-tables
```

Wenn Sie möchten, können Sie einige andere AWS CLI Befehle für DynamoDB ausprobieren. Weitere Informationen finden Sie in der AWS CLI -Befehlsreferenz.

Schritt 4: (Optional): Bereinigen

Wenn Sie die Ressourcen löschen möchten, die Sie in diesem Tutorial erstellt haben, führen Sie die folgenden Schritte aus:

So entfernen Sie Ihren VPC-Endpunkt für DynamoDB

1. Melden Sie sich bei Ihrer Amazon-EC2-Instance an.
2. Ermitteln Sie die VPC-Endpunkt-ID.

```
aws ec2 describe-vpc-endpoints
```

```
{
    "VpcEndpoint": {
        "PolicyDocument": "{\"Version\":\"2008-10-17\",\"Statement\":[{\"Effect\":
\"Allow\",\"Principal\":\"*\",\"Action\":\"*\",\"Resource\":\"*\"}]}",
        "VpcId": "vpc-0bbc736e",
        "State": "available",
        "ServiceName": "com.amazonaws.us-east-1.dynamodb",
        "RouteTableIds": [],
        "VpcEndpointId": "vpce-9b15e2f2",
        "CreationTimestamp": "2017-07-26T22:00:14Z"
    }
}
```

   In der Beispielausgabe ist die ID des VPC-Endpunkts `vpce-9b15e2f2`.
3. Löschen Sie den VPC-Endpunkt.

```
aws ec2 delete-vpc-endpoints --vpc-endpoint-ids vpce-9b15e2f2
```

```
{
    "Unsuccessful": []
}
```

   Das leere Array `[]` zeigt den Erfolg an (keine fehlgeschlagenen Anforderungen).

So beenden Sie Ihre Amazon-EC2-Instance

1. Öffnen Sie die Amazon-EC2-Konsole unter https://console.aws.amazon.com/ec2/.

2. Wählen Sie im Navigationsbereich Instances aus.
3. Wählen Sie Ihre Amazon EC2-Instance aus.
4. Wählen Sie Actions, Instance State und Terminate aus.
5. Wählen Sie im Bestätigungsfenster Yes, Terminate aus.

AWS PrivateLink für DynamoDB

Mit AWS PrivateLink for DynamoDB können Sie Amazon VPC-Schnittstellenendpunkte (Schnittstellenendpunkte) in Ihrer Virtual Private Cloud (Amazon VPC) bereitstellen. Auf diese Endpunkte kann direkt von Anwendungen aus zugegriffen werden, die sich vor Ort befinden Direct Connect, über VPN und/oder auf andere Weise AWS-Region über Amazon VPC-Peering. Mit AWS PrivateLink und Schnittstellenendpunkten können Sie die private Netzwerkkonnektivität zwischen Ihren Anwendungen und DynamoDB vereinfachen.

Anwendungen in Ihrer VPC benötigen keine öffentlichen IP-Adressen, um mit DynamoDB über VPC-Schnittstellenendpunkte für DynamoDB-Operationen zu kommunizieren. Schnittstellenendpunkte werden durch eine oder mehrere elastische Netzwerkschnittstellen (ENIs) repräsentiert, denen private IP-Adressen aus Subnetzen in Ihrer Amazon VPC zugewiesen wurden. Anforderungen an DynamoDB über Schnittstellenendpunkte bleiben im Amazon-Netzwerk. Sie können auch von lokalen Anwendungen aus über AWS Direct Connect oder AWS Virtual Private Network () auf Schnittstellenendpunkte in Ihrer Amazon VPC zugreifen.Site-to-Site VPN Weitere Informationen darüber, wie Sie Ihre Amazon VPC mit Ihrem On-Premises-Netzwerk verbinden, finden Sie im Direct Connect -Benutzerhandbuch und im AWS Site-to-Site VPN -Benutzerhandbuch.

Allgemeine Informationen zu Schnittstellenendpunkten finden Sie unter Interface Amazon VPC endpoints (AWS PrivateLink) im Handbuch.AWS PrivateLink AWS PrivateLink wird auch für Amazon DynamoDB Streams-Endpunkte unterstützt. Weitere Informationen finden Sie unter the section called “AWS PrivateLink für DynamoDB Streams”.

Themen

- Arten von Amazon-VPC-Endpunkten für Amazon DynamoDB
- Überlegungen bei der Verwendung AWS PrivateLink für Amazon DynamoDB
- Erstellen eines Amazon VPC-Endpunkts
- Zugreifen auf Schnittstellenendpunkte in Amazon DynamoDB
- Zugreifen auf DynamoDB-Tabellen und -Control-API-Operationen über DynamoDB-Schnittstellenendpunkte

- Aktualisieren einer lokalen DNS-Konfiguration
- Erstellen einer Amazon-VPC-Endpunktrichtlinie für DynamoDB
- Verwenden von DynamoDB-Endpunkten mit privatem Zugriff AWS-Managementkonsole
- AWS PrivateLink für DynamoDB Streams
- AWS PrivateLink Für DynamoDB Accelerator (DAX) verwenden

Arten von Amazon-VPC-Endpunkten für Amazon DynamoDB

Sie können zwei Arten von Amazon VPC-Endpunkten für den Zugriff auf Amazon DynamoDB verwenden: Gateway-Endpunkte und Schnittstellenendpunkte (durch Verwenden). AWS PrivateLink Ein Gateway-Endpunkt ist ein Gateway, das Sie in Ihrer Routing-Tabelle angeben, um von Ihrer Amazon VPC aus über das Netzwerk auf DynamoDB zuzugreifen. AWS Schnittstellenendpunkte erweitern die Funktionalität von Gateway-Endpunkten, indem sie private IP-Adressen verwenden, um Anfragen von Ihrer Amazon VPC, vor Ort oder von einer Amazon VPC in einer anderen AWS-Region mithilfe von Amazon VPC Peering an DynamoDB weiterzuleiten oder. AWS Transit Gateway Weitere Informationen finden Sie unter Was ist Amazon-VPC Peering? und Vergleich zwischen Transit-Gateway und Amazon-VPC-Peering.

Schnittstellenendpunkte sind mit Gateway-Endpunkten kompatibel. Wenn Sie einen vorhandenen Gateway-Endpunkt in der Amazon VPC haben, können Sie beide Arten von Endpunkten in derselben Amazon VPC verwenden.

Gateway-Endpunkte für DynamoDB	Schnittstellenendpunkte für DynamoDB
In beiden Fällen verbleibt Ihr Netzwerkverkehr im Netzwerk. AWS	
Verwenden von öffentlichen IP-Adressen in Amazon DynamoDB	Verwenden von privaten IP-Adressen aus Ihrer Amazon VPC für den Zugriff auf Amazon DynamoDB
Erlauben keinen On-Premises-Zugriff	Erlaubt On-Premises-Zugriff
Erlauben Sie keinen Zugriff von einem anderen AWS-Region	Erlauben Sie den Zugriff von einem Amazon VPC-Endpunkt auf einem anderen mithilfe AWS-Region von Amazon VPC-Peering oder AWS Transit Gateway

Gateway-Endpunkte für DynamoDB	Schnittstellenendpunkte für DynamoDB
Nicht berechnet	Berechnet

Weitere Informationen zu Gateway-Endpunkten finden Sie unter Gateway-Endpunkte in der Amazon VPC im AWS PrivateLink -Handbuch.

Überlegungen bei der Verwendung AWS PrivateLink für Amazon DynamoDB

Überlegungen zu Amazon VPC gelten AWS PrivateLink für Amazon DynamoDB. Weitere Informationen finden Sie unter Überlegungen zu Schnittstellenendpunkten und AWS PrivateLink -Kontingente im Handbuch zu AWS PrivateLink . Darüber hinaus gelten die folgenden Einschränkungen.

AWS PrivateLink für Amazon DynamoDB unterstützt Folgendes nicht:

- Transport Layer Security (TLS) 1.1
- Private und Hybrid Domain Name System (DNS)-Services

Sie können bis zu 50.000 Anfragen pro Sekunde für jeden von Ihnen aktivierten AWS PrivateLink Endpunkt einreichen.

> **Note**
>
> Timeouts der Netzwerkkonnektivität zu AWS PrivateLink Endpunkten fallen nicht in den Geltungsbereich der DynamoDB-Fehlerantworten und müssen von Ihren Anwendungen, die eine Verbindung zu den Endpunkten herstellen, angemessen behandelt werden. PrivateLink

Erstellen eines Amazon VPC-Endpunkts

Informationen zum Erstellen eines Amazon-VPC-Schnittstellenendpunkts finden Sie unter Erstellen eines Amazon-VPC-Endpunkts im AWS PrivateLink -Handbuch.

Zugreifen auf Schnittstellenendpunkte in Amazon DynamoDB

Wenn Sie einen Schnittstellenendpunkt erstellen, generiert Amazon DynamoDB zwei Arten von endpunktspezifischen DynamoDB-DNS-Namen: regional und zonengebunden.

- Ein regionaler DNS-Name enthält eine eindeutige Amazon VPC-Endpunkt-ID, eine Service-ID AWS-Region, das und `vpce.amazonaws.com` in seinem Namen. Beispielsweise könnte der generierte DNS-Name für die Amazon-VPC-Endpunkt-ID *`vpce-1a2b3c4d`* mit *`vpce-1a2b3c4d-5e6f`*`.dynamodb.us-east-1.vpce.amazonaws.com` vergleichbar sein.
- Ein zonengebundener DNS-Name enthält die Availability Zone, z. B. *`vpce-1a2b3c4d-5e6f`***`-us-east-1a.`**`dynamodb.us-east-1.vpce.amazonaws.com`. Sie können diese Option verwenden, wenn Ihre Architektur Availability Zones isoliert. Sie könnten sie beispielsweise zur Fehlereingrenzung oder zur Senkung der regionalen Datenübertragungskosten verwenden.

> **Note**
>
> Um eine optimale Zuverlässigkeit zu erreichen, empfehlen wir, Ihren Service in mindestens drei Availability Zones bereitzustellen.

Zugreifen auf DynamoDB-Tabellen und -Control-API-Operationen über DynamoDB-Schnittstellenendpunkte

Sie können das AWS CLI oder verwenden AWS SDKs , um auf DynamoDB-Tabellen zuzugreifen und API-Operationen über DynamoDB-Schnittstellenendpunkte zu steuern.

AWS CLI Beispiele

Verwenden Sie die Parameter und, um über DynamoDB-Schnittstellenendpunkte in Befehlen auf DynamoDB-Tabellen oder DynamoDB-Steuerungs-API-Operationen zuzugreifen. AWS CLI `--region` `--endpoint-url`

Beispiel: Erstellen eines VPC-Endpunkts

```
aws ec2 create-vpc-endpoint \
--region us-east-1 \
--service-name com.amazonaws.us-east-1.dynamodb \
--vpc-id client-vpc-id \
```

```
--subnet-ids client-subnet-id \
--vpc-endpoint-type Interface \
--security-group-ids client-sg-id
```

Beispiel: Ändern eines VPC-Endpunkts

```
aws ec2 modify-vpc-endpoint \
--region us-east-1 \
--vpc-endpoint-id client-vpc-endpoint-id \
--policy-document policy-document \ #example optional parameter
--add-security-group-ids security-group-ids \ #example optional parameter
# any additional parameters needed, see Privatelink documentation for more details
```

Beispiel: Auflisten von Tabellen mit einer Endpunkt-URL

Ersetzen Sie im folgenden Beispiel die Region `us-east-1` und den DNS-Namen der VPC-Endpunkt-ID `vpce-1a2b3c4d-5e6f.dynamodb.us-east-1.vpce.amazonaws.com` durch Ihre eigenen Angaben.

```
aws dynamodb --region us-east-1 --endpoint https://vpce-1a2b3c4d-5e6f.dynamodb.us-
east-1.vpce.amazonaws.com list-tables
```

AWS SDK-Beispiele

Um auf DynamoDB-Tabellen oder DynamoDB-Steuerungs-API-Operationen über DynamoDB-Schnittstellenendpunkte zuzugreifen, wenn Sie die verwenden AWS SDKs, aktualisieren Sie Ihre Version auf die neueste Version. SDKs Konfigurieren Sie Ihre Clients anschließend so, dass sie eine Endpunkt-URL verwenden, um über DynamoDB-Schnittstellenendpunkte auf eine Tabelle oder Control-API-Operation in DynamoDB zuzugreifen.

SDK for Python (Boto3)

Beispiel: Verwenden einer Endpunkt-URL für den Zugriff auf eine DynamoDB-Tabelle

Ersetzen Sie im folgenden Beispiel die Region `us-east-1` und die VPC-Endpunkt-ID `https://vpce-1a2b3c4d-5e6f.dynamodb.us-east-1.vpce.amazonaws.com` durch Ihre eigenen Informationen.

```
ddb_client = session.client(
service_name='dynamodb',
region_name='us-east-1',
```

```
endpoint_url='https://vpce-1a2b3c4d-5e6f.dynamodb.us-east-1.vpce.amazonaws.com'
)
```

SDK for Java 1.x

Beispiel: Verwenden einer Endpunkt-URL für den Zugriff auf eine DynamoDB-Tabelle

Ersetzen Sie im folgenden Beispiel die Region `us-east-1` und die VPC-Endpunkt-ID `https://vpce-1a2b3c4d-5e6f.dynamodb.us-east-1.vpce.amazonaws.com` durch Ihre eigenen Informationen.

```
//client build with endpoint config
final AmazonDynamoDB dynamodb =
 AmazonDynamoDBClientBuilder.standard().withEndpointConfiguration(
        new AwsClientBuilder.EndpointConfiguration(
                "https://vpce-1a2b3c4d-5e6f.dynamodb.us-east-1.vpce.amazonaws.com",
                Regions.DEFAULT_REGION.getName()
        )
).build();
```

SDK for Java 2.x

Beispiel: Verwenden einer Endpunkt-URL für den Zugriff auf eine DynamoDB-Tabelle

Ersetzen Sie im folgenden Beispiel die Region us-east-1 und die VPC-Endpunkt-ID https://vpce-1a2b3c4d-5e6f.dynamodb.us-east-1.vpce.amazonaws.com durch Ihre eigenen Informationen.

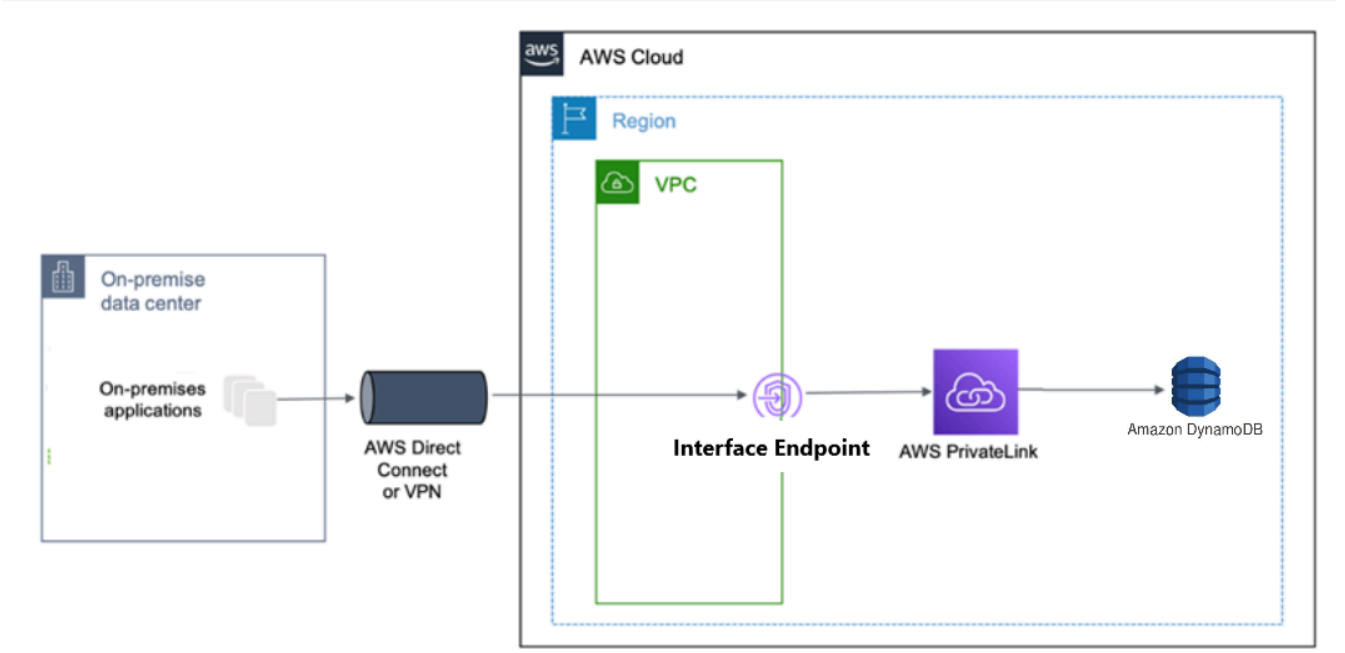
```
Region region = Region.US_EAST_1;
dynamoDbClient = DynamoDbClient.builder().region(region)
.endpointOverride(URI.create("https://vpce-1a2b3c4d-5e6f.dynamodb.us-
east-1.vpce.amazonaws.com"))
.build()
```

Aktualisieren einer lokalen DNS-Konfiguration

Wenn Sie endpunktspezifische DNS-Namen für den Zugriff auf die Schnittstellenendpunkte für DynamoDB verwenden, müssen Sie Ihren lokalen DNS-Resolver nicht aktualisieren. Sie können den endpunktspezifischen DNS-Namen mit der privaten IP-Adresse des Schnittstellenendpunkts aus der öffentlichen DynamoDB-DNS-Domain auflösen.

Verwenden von Schnittstellenendpunkten für den Zugriff auf DynamoDB ohne Gateway-Endpunkt oder Internet-Gateway in der Amazon VPC

Schnittstellenendpunkte in Ihrer Amazon VPC können sowohl Anwendungen in der Amazon VPC als auch On-Premises-Anwendungen über das Amazon-Netzwerk an DynamoDB weiterleiten, wie im folgenden Diagramm dargestellt.

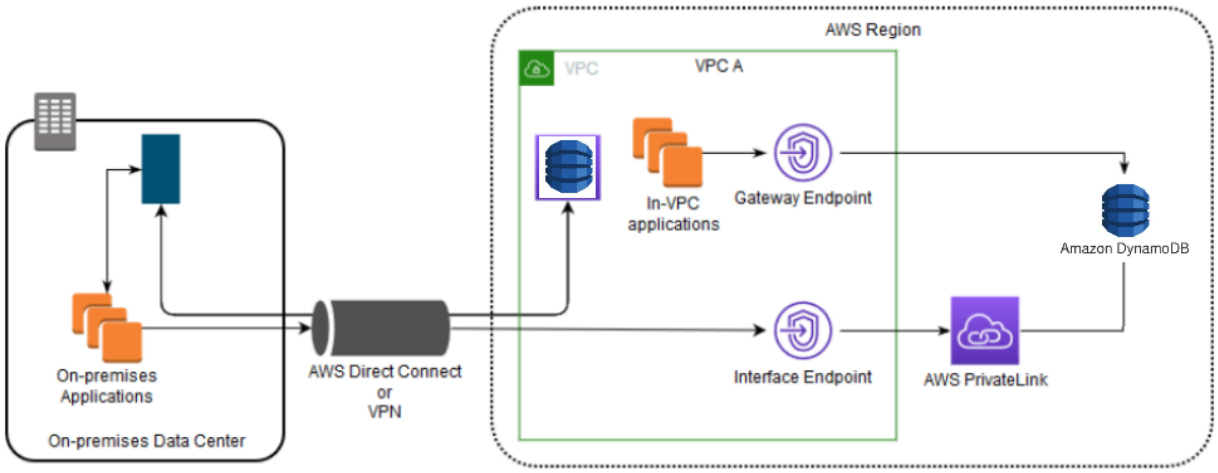


Das Diagramm veranschaulicht folgende Vorgänge:

- Ihr lokales Netzwerk verwendet Direct Connect oder, um eine Verbindung Site-to-Site VPN zu Amazon VPC A herzustellen.
- Ihre On-Premises-Anwendungen und die Anwendungen in Amazon VPC A verwenden endpunktspezifische DNS-Namen, um über den DynamoDB-Schnittstellenendpunkt auf DynamoDB zuzugreifen.
- Lokale Anwendungen senden Daten über Direct Connect (oder Site-to-Site VPN) an den Schnittstellenendpunkt in der Amazon VPC. AWS PrivateLink verschiebt die Daten vom Schnittstellenendpunkt über das Netzwerk zu DynamoDB. AWS
- VPC-Anwendungen innerhalb von Amazon senden auch Traffic an den Schnittstellenendpunkt. AWS PrivateLink verschiebt die Daten vom Schnittstellenendpunkt über das Netzwerk zu DynamoDB. AWS

Gemeinsames Verwenden von Gateway-Endpunkten und Schnittstellenendpunkten in derselben Amazon VPC für den Zugriff auf DynamoDB

Sie können Schnittstellenendpunkte erstellen und den vorhandenen Gateway-Endpunkt in derselben Amazon VPC beibehalten, wie das folgende Diagramm zeigt. Mit diesem Ansatz erlauben Sie Anwendungen in der Amazon VPC, weiterhin über den Gateway-Endpunkt auf DynamoDB zuzugreifen, was nicht in Rechnung gestellt wird. Dann würden nur Ihre On-Premises-Anwendungen Schnittstellenendpunkte für den Zugriff auf DynamoDB verwenden. Um auf diese Weise auf DynamoDB zuzugreifen, müssen Sie Ihre On-Premises-Anwendungen aktualisieren, damit endpunktspezifische DNS-Namen für DynamoDB verwendet werden.



Das Diagramm veranschaulicht folgende Vorgänge:

- Lokale Anwendungen verwenden endpunktspezifische DNS-Namen, um Daten über Direct Connect (oder) an den Schnittstellenendpunkt innerhalb der Amazon VPC zu senden. Site-to-Site VPN AWS PrivateLink verschiebt die Daten vom Schnittstellenendpunkt über das Netzwerk zu DynamoDB. AWS
- Mithilfe von standardmäßigen regionalen DynamoDB-Namen senden VPC-Anwendungen in Amazon Daten an den Gateway-Endpunkt, der über das Netzwerk eine Verbindung zu DynamoDB herstellt. AWS

Weitere Informationen zu Gateway-Endpunkten finden Sie unter Gateway-Endpunkte in der Amazon VPC im Amazon-VPC-Benutzerhandbuch.

Erstellen einer Amazon-VPC-Endpunktrichtlinie für DynamoDB

Sie können eine Endpunktrichtlinie an Ihren Amazon-VPC-Endpunkt anhängen, der den Zugriff auf DynamoDB kontrolliert. Die Richtlinie gibt die folgenden Informationen an:

- Der AWS Identity and Access Management (IAM-) Principal, der Aktionen ausführen kann
- Aktionen, die ausgeführt werden können
- Ressourcen, für die Aktionen ausgeführt werden können

Themen

- Beispiel: Beschränken des Zugriffs auf eine bestimmte Tabelle von einem Amazon-VPC-Endpunkt aus

Beispiel: Beschränken des Zugriffs auf eine bestimmte Tabelle von einem Amazon-VPC-Endpunkt aus

Sie können eine Endpunktrichtlinie erstellen, die den Zugriff auf spezifische DynamoDB-Tabellen einschränkt. Diese Art von Richtlinie ist nützlich, wenn Sie andere AWS-Services in Ihrer Amazon VPC haben, die Tabellen verwenden. Die folgende Tabellenrichtlinie schränkt den Zugriff ausschließlich auf *DOC-EXAMPLE-TABLE* ein. Um diese Endpunktrichtlinie zu verwenden, ersetzen Sie *DOC-EXAMPLE-TABLE* durch den Namen Ihrer Tabelle.

JSON

```
{
"Version":"2012-10-17",
  "Id": "Policy1216114807515",
  "Statement": [
    { "Sid": "Access-to-specific-table-only",
      "Principal": "*",
      "Action": [
        "dynamodb:GetItem",
        "dynamodb:PutItem"
      ],
      "Effect": "Allow",
      "Resource": ["arn:aws:dynamodb:us-east-1:111122223333:table/DOC-EXAMPLE-
TABLE",
```

```
                "arn:aws:dynamodb:us-east-1:111122223333:table/DOC-EXAMPLE-
TABLE/*"]
    }
  ]
}
```

Verwenden von DynamoDB-Endpunkten mit privatem Zugriff AWS-Managementkonsole

Sie müssen die DNS-Konfiguration für DynamoDB und DynamoDB Streams einrichten, wenn Sie VPC-Endpunkte mit der DynamoDB-Konsole in AWS-Managementkonsole Private Access verwenden.

Um DynamoDB für den Zugriff in AWS-Managementkonsole Private Access zu konfigurieren, müssen Sie die folgenden zwei VPC-Endpoints erstellen:

- `com.amazonaws.<region>.dynamodb`
- `com.amazonaws.<region>.dynamodb-streams`

Wenn Sie die VPC-Endpunkte erstellen, navigieren Sie zur Route53-Konsole und erstellen Sie mithilfe des regionalen Endpunkts `dynamodb.us-east-1.amazonaws.com` eine privat gehostete Zone für DynamoDB.

Erstellen Sie die folgenden zwei Aliasdatensätze in der privat gehosteten Zone:

- `dynamodb.<region>.amazonaws.com` für die Weiterleitung des Datenverkehrs an den VPC-Endpunkt `com.amazonaws.<region>.dynamodb`.
- `streams.dynamodb.<region>.amazonaws.com` für die Weiterleitung des Datenverkehrs an den VPC-Endpunkt `com.amazonaws.<region>.dynamodb-streams`.

AWS PrivateLink für DynamoDB Streams

Mit AWS PrivateLink for Amazon DynamoDB Streams können Sie Amazon VPC-Schnittstellenendpunkte (Schnittstellenendpunkte) in Ihrer virtuellen privaten Cloud (Amazon VPC) bereitstellen. Auf diese Endpunkte kann direkt von Anwendungen aus zugegriffen werden, die sich vor Ort befinden Direct Connect, über VPN und/oder auf andere Weise AWS-Region über Amazon

VPC-Peering. Mithilfe von Endpunkten AWS PrivateLink und Schnittstellen können Sie die private Netzwerkkonnektivität von Ihren Anwendungen zu DynamoDB Streams vereinfachen.

Anwendungen in Ihrer in einer Amazon VPC benötigen keine öffentlichen IP-Adressen, um mit DynamoDB Streams über Amazon-VPC-Schnittstellenendpunkte für Operationen in DynamoDB Streams zu kommunizieren. Schnittstellenendpunkte werden durch eine oder mehrere elastische Netzwerkschnittstellen (ENIs) repräsentiert, denen private IP-Adressen aus Subnetzen in Ihrer Amazon VPC zugewiesen wurden. Anforderungen an DynamoDB Streams über Schnittstellenendpunkte bleiben im Amazon-Netzwerk. Sie können auch von lokalen Anwendungen aus über Direct Connect oder AWS Virtual Private Network (AWS VPN) auf Schnittstellenendpunkte in Ihrer Amazon VPC zugreifen. Weitere Informationen darüber, wie Sie Ihr Netzwerk AWS Virtual Private Network mit Ihrem lokalen Netzwerk verbinden können, finden Sie im Benutzerhandbuch und im Direct Connect Benutzerhandbuch.AWS Site-to-Site VPN

Allgemeine Informationen zu Schnittstellenendpunkten finden Sie unter Amazon-VPC-Schnittstellenendpunkte (AWS PrivateLink).

> **Note**
>
> Für DynamoDB Streams werden nur Schnittstellenendpunkte unterstützt. Gateway-Endpunkte werden nicht unterstützt.

Themen

- Überlegungen bei der Verwendung AWS PrivateLink für Amazon DynamoDB DynamoDB-Streams
- Erstellen eines Amazon VPC-Endpunkts
- Zugreifen auf Schnittstellenendpunkte in Amazon DynamoDB Streams
- Zugreifen auf API-Operationen in DynamoDB Streams über DynamoDB-Streams-Schnittstellenendpunkte
- AWS SDK-Beispiele
- Erstellen einer Amazon-VPC-Endpunktrichtlinie für DynamoDB Streams
- Verwenden von DynamoDB-Endpunkten mit privatem Zugriff AWS-Managementkonsole

Überlegungen bei der Verwendung AWS PrivateLink für Amazon DynamoDB DynamoDB-Streams

Überlegungen zu Amazon VPC gelten AWS PrivateLink für Amazon DynamoDB DynamoDB-Streams. Weitere Informationen finden Sie unter Überlegungen zu Schnittstellenendpunkten und AWS PrivateLink -Kontingente. Beachten Sie dabei die folgenden Einschränkungen.

AWS PrivateLink für Amazon DynamoDB Streams unterstützt Folgendes nicht:

- Transport Layer Security (TLS) 1.1
- Private und Hybrid Domain Name System (DNS)-Services

> **Note**
>
> Timeouts der Netzwerkkonnektivität zu AWS PrivateLink Endpunkten fallen nicht in den Geltungsbereich der DynamoDB Streams Streams-Fehlerantworten und müssen von Ihren Anwendungen, die eine Verbindung zu den Endpunkten herstellen, angemessen behandelt werden. AWS PrivateLink

Erstellen eines Amazon VPC-Endpunkts

Informationen zum Erstellen eines Amazon-VPC-Schnittstellenendpunkts finden Sie unter Erstellen eines Amazon-VPC-Endpunkts im AWS PrivateLink -Handbuch.

Zugreifen auf Schnittstellenendpunkte in Amazon DynamoDB Streams

Wenn Sie einen Schnittstellenendpunkt erstellen, generiert DynamoDB zwei Arten von endpunktspezifischen DNS-Namen für DynamoDB Streams: regional und zonengebunden.

- Ein regionaler DNS-Name enthält eine eindeutige Amazon VPC-Endpunkt-ID, eine Service-ID AWS-Region, das und `vpce.amazonaws.com` in seinem Namen. Beispielsweise könnte der generierte DNS-Name für die Amazon-VPC-Endpunkt-ID *`vpce-1a2b3c4d`* mit *`vpce-1a2b3c4d-5e6f`*`.streams.dynamodb.us-east-1.vpce.amazonaws.com` vergleichbar sein.
- Ein zonengebundener DNS-Name enthält die Availability Zone, z. B. *`vpce-1a2b3c4d-5e6f`***`-us-east-1a.`**`streams.dynamodb.us-east-1.vpce.amazonaws.com`. Sie können diese Option

verwenden, wenn Ihre Architektur Availability Zones isoliert. Sie könnten sie beispielsweise zur Fehlereingrenzung oder zur Senkung der regionalen Datenübertragungskosten verwenden.

Zugreifen auf API-Operationen in DynamoDB Streams über DynamoDB-Streams-Schnittstellenendpunkte

Sie können das AWS CLI oder verwenden AWS SDKs , um über Endpunkte der DynamoDB Streams Streams-Schnittstelle auf DynamoDB Streams Streams-API-Operationen zuzugreifen.

AWS CLI Beispiele

Verwenden Sie die Parameter und, um über die Endpunkte der DynamoDB-Streams-Schnittstelle in AWS CLI Befehlen auf DynamoDB Streams oder API-Operationen zuzugreifen. `--region` `--endpoint-url`

Beispiel: Erstellen eines VPC-Endpunkts

```
aws ec2 create-vpc-endpoint \
--region us-east-1 \
--service-name com.amazonaws.us-east-1.dynamodb-streams \
--vpc-id client-vpc-id \
--subnet-ids client-subnet-id \
--vpc-endpoint-type Interface \
--security-group-ids client-sg-id
```

Beispiel: Ändern eines VPC-Endpunkts

```
aws ec2 modify-vpc-endpoint \
--region us-east-1 \
--vpc-endpoint-id client-vpc-endpoint-id \
--policy-document policy-document \ #example optional parameter
--add-security-group-ids security-group-ids \ #example optional parameter
# any additional parameters needed, see Privatelink documentation for more details
```

Beispiel: Auflisten von Streams mit einer Endpunkt-URL

Ersetzen Sie im folgenden Beispiel die Region `us-east-1` und den DNS-Namen der VPC-Endpunkt-ID `vpce-1a2b3c4d-5e6f.streams.dynamodb.us-east-1.vpce.amazonaws.com` durch Ihre eigenen Angaben.

```
aws dynamodbstreams --region us-east-1 –endpoint https://
vpce-1a2b3c4d-5e6f.streams.dynamodb.us-east-1.vpce.amazonaws.com list-streams
```

AWS SDK-Beispiele

Um auf Amazon DynamoDB Streams API-Operationen über die DynamoDB Streams Streams-Schnittstellenendpunkte zuzugreifen, wenn Sie die verwenden AWS SDKs, aktualisieren Sie Ihre SDKs Version auf die neueste Version. Konfigurieren Sie Ihre Clients anschließend so, dass sie eine Endpunkt-URL für die API-Operation von DynamoDB Streams über DynamoDB-Streams-Schnittstellenendpunkte verwenden.

SDK for Python (Boto3)

Beispiel: Verwenden einer Endpunkt-URL für den Zugriff auf einen DynamoDB-Stream

Ersetzen Sie im folgenden Beispiel die Region `us-east-1` und die VPC-Endpunkt-ID `https://vpce-1a2b3c4d-5e6f.streams.dynamodb.us-east-1.vpce.amazonaws.com` durch Ihre eigenen Informationen.

```
ddb_streams_client = session.client(
service_name='dynamodbstreams',
region_name='us-east-1',
endpoint_url='https://vpce-1a2b3c4d-5e6f.streams.dynamodb.us-
east-1.vpce.amazonaws.com'
)
```

SDK for Java 1.x

Beispiel: Verwenden einer Endpunkt-URL für den Zugriff auf einen DynamoDB-Stream

Ersetzen Sie im folgenden Beispiel die Region `us-east-1` und die VPC-Endpunkt-ID `https://vpce-1a2b3c4d-5e6f.streams.dynamodb.us-east-1.vpce.amazonaws.com` durch Ihre eigenen Informationen.

```
//client build with endpoint config
final AmazonDynamoDBStreams dynamodbstreams =
 AmazonDynamoDBStreamsClientBuilder.standard().withEndpointConfiguration(
        new AwsClientBuilder.EndpointConfiguration(
                "https://vpce-1a2b3c4d-5e6f.streams.dynamodb.us-
east-1.vpce.amazonaws.com",
```

```
                Regions.DEFAULT_REGION.getName()
        )
).build();
```

SDK for Java 2.x

Beispiel: Verwenden einer Endpunkt-URL für den Zugriff auf einen DynamoDB-Stream

Ersetzen Sie im folgenden Beispiel die Region `us-east-1` und die VPC-Endpunkt-ID `https://vpce-1a2b3c4d-5e6f.streams.dynamodb.us-east-1.vpce.amazonaws.com` durch Ihre eigenen Informationen.

```
Region region = Region.US_EAST_1;
dynamoDbStreamsClient = DynamoDbStreamsClient.builder().region(region)
.endpointOverride(URI.create("https://vpce-1a2b3c4d-5e6f.streams.dynamodb.us-
east-1.vpce.amazonaws.com"))
.build()
```

Erstellen einer Amazon-VPC-Endpunktrichtlinie für DynamoDB Streams

Sie können eine Endpunktrichtlinie an Ihren Amazon-VPC-Endpunkt anhängen, der den Zugriff auf DynamoDB Streams kontrolliert. Die Richtlinie gibt die folgenden Informationen an:

- Der AWS Identity and Access Management (IAM-) Principal, der Aktionen ausführen kann
- Aktionen, die ausgeführt werden können
- Ressourcen, für die Aktionen ausgeführt werden können

Themen

- Beispiel: Beschränken des Zugriffs auf einen bestimmten Streams von einem Amazon-VPC-Endpunkt aus

Beispiel: Beschränken des Zugriffs auf einen bestimmten Streams von einem Amazon-VPC-Endpunkt aus

Sie können eine Endpunktrichtlinie erstellen, die den Zugriff auf spezifische DynamoDB Streams einschränkt. Diese Art von Richtlinie ist nützlich, wenn Sie andere AWS-Services in Ihrer Amazon VPC haben, die DynamoDB Streams verwenden. Die folgende Stream-Richtlinie schränkt den Zugriff nur auf den Stream *2025-02-20T11:22:33.444* ein, der *DOC-EXAMPLE-TABLE* angehängt ist.

Um diese Endpunktrichtlinie zu verwenden, ersetzen Sie *DOC-EXAMPLE-TABLE* durch den Namen Ihrer Tabelle und *2025-02-20T11:22:33.444* durch die Stream-Bezeichnung.

JSON

```
{
"Version":"2012-10-17",
  "Id": "Policy1216114807515",
  "Statement": [
    { "Sid": "Access-to-specific-stream-only",
      "Principal": "*",
      "Action": [
        "dynamodb:DescribeStream",
        "dynamodb:GetRecords"
      ],
      "Effect": "Allow",
      "Resource": ["arn:aws:dynamodb:us-east-1:111122223333:table/table-name/
stream/2025-02-20T11:22:33.444"]
    }
  ]
}
```

> **Note**
>
> Gateway-Endpunkte werden in DynamoDB Streams nicht unterstützt.

Verwenden von DynamoDB-Endpunkten mit privatem Zugriff AWS-Managementkonsole

Sie müssen die DNS-Konfiguration für DynamoDB und DynamoDB Streams einrichten, wenn Sie VPC-Endpunkte mit der DynamoDB-Konsole in AWS-Managementkonsole Private Access verwenden.

Um DynamoDB für den Zugriff in AWS-Managementkonsole Private Access zu konfigurieren, müssen Sie die folgenden zwei VPC-Endpoints erstellen:

- `com.amazonaws.<region>.dynamodb`
- `com.amazonaws.<region>.dynamodb-streams`

Wenn Sie die VPC-Endpunkte erstellen, navigieren Sie zur Route53-Konsole und erstellen Sie mithilfe des regionalen Endpunkts `dynamodb.us-east-1.amazonaws.com` eine privat gehostete Zone für DynamoDB.

Erstellen Sie die folgenden zwei Aliasdatensätze in der privat gehosteten Zone:

- `dynamodb.<region>.amazonaws.com` für die Weiterleitung des Datenverkehrs an den VPC-Endpunkt `com.amazonaws.<region>.dynamodb`.
- `streams.dynamodb.<region>.amazonaws.com` für die Weiterleitung des Datenverkehrs an den VPC-Endpunkt `com.amazonaws.<region>.dynamodb-streams`.

AWS PrivateLink Für DynamoDB Accelerator (DAX) verwenden

AWS PrivateLink für DynamoDB Accelerator (DAX) können Sie sicher auf DAX-Management APIs wie `CreateClusterDescribeClusters`, und `DeleteCluster` über private IP-Adressen innerhalb Ihrer Virtual Private Cloud (VPC) zugreifen. Mit dieser Funktion können Sie privat von Ihren Anwendungen aus auf DAX-Dienste zugreifen, ohne dass der Datenverkehr dem öffentlichen Internet ausgesetzt wird.

DAX PrivateLink unterstützt Dual-Stack-Endpunkte (`dax.{region}.api.aws`) und ermöglicht so sowohl IPv4 Konnektivität als auch Konnektivität. IPv6 Mit AWS PrivateLink for DAX können Kunden über private DNS-Namen auf den Service zugreifen. Die Unterstützung von Dual-Stack-Endpunkten gewährleistet transparente Konnektivität und gewährleistet gleichzeitig den Datenschutz im Netzwerk. Auf diese Weise können Sie sowohl über das öffentliche Internet als auch über VPC-Endpunkte auf DAX zugreifen, ohne Änderungen an Ihrer SDK-Konfiguration vornehmen zu müssen.

Überlegungen bei der Verwendung von AWS PrivateLink DynamoDB Accelerator (DAX)

Bei der Implementierung AWS PrivateLink für DynamoDB Accelerator (DAX) müssen mehrere wichtige Überlegungen berücksichtigt werden.

Bevor Sie einen Schnittstellenendpunkt für DAX einrichten, sollten Sie Folgendes berücksichtigen:

- Endpunkte der DAX-Schnittstelle unterstützen nur den Zugriff auf das DAX-Management APIs innerhalb derselben AWS-Region Schnittstelle. Sie können keinen Schnittstellenendpunkt verwenden, um auf das DAX-Management APIs in anderen Regionen zuzugreifen.

- Um AWS-Managementkonsole privat auf das DAX-Management zuzugreifen, müssen Sie möglicherweise zusätzliche VPC-Endpunkte für Dienste wie `com.amazonaws.region.console` und verwandte Dienste erstellen.
- Die Erstellung und Nutzung eines Schnittstellenendpunkts zu DAX wird Ihnen in Rechnung gestellt. Preisinformationen finden Sie unter AWS PrivateLink Preise.

Wie AWS PrivateLink funktioniert mit DAX

Wenn Sie einen Schnittstellenendpunkt für DAX erstellen:

1. AWS erstellt in jedem Subnetz, das Sie für den Schnittstellenendpunkt aktivieren, eine Endpunkt-Netzwerkschnittstelle.
2. Dabei handelt es sich um vom Anforderer verwaltete Netzwerkschnittstellen, die als Einstiegspunkte für den Datenverkehr dienen, der für DAX bestimmt ist.
3. Sie können dann über private IP-Adressen innerhalb Ihrer VPC auf DAX zugreifen.
4. Diese Architektur ermöglicht es Ihnen, VPC-Sicherheitsgruppen zu verwenden, um den Zugriff auf die Endpoints zu verwalten.
5. Anwendungen können über ihre jeweiligen Schnittstellenendpunkte innerhalb einer VPC sowohl auf DynamoDB als auch auf DAX zugreifen und gleichzeitig lokalen Anwendungen die Verbindung über Direct Connect oder VPN ermöglichen.
6. Dies bietet ein konsistentes Konnektivitätsmodell für beide Dienste, vereinfacht die Architektur und verbessert die Sicherheit, da der Datenverkehr im Netzwerk gehalten wird. AWS

Schnittstellen-Endpunkte für DAX erstellen

Sie können einen Schnittstellenendpunkt erstellen, um mithilfe des AWS SDK oder der AWS-Managementkonsole AWS API eine Verbindung zu DAX herzustellen. CloudFormation

Um mithilfe der Konsole einen Schnittstellenendpunkt für DAX zu erstellen

1. Navigieren Sie zur Amazon VPC-Konsole unter https://console.aws.amazon.com/vpc/.
2. Wählen Sie im Navigationsbereich Endpunkte aus.
3. Klicken Sie auf Endpunkt erstellen.
4. Wählen Sie als Servicekategorie AWS-Servicesund als Servicename suchen Sie nach und wählen Sie aus`com.amazonaws.region.dax`.

5. Wählen Sie für VPC die VPC aus, von der aus Sie auf DAX zugreifen möchten, und wählen Sie für Subnetze die Subnetze aus, in denen die Endpunkt-Netzwerkschnittstellen erstellt AWS werden sollen.
6. Wählen oder erstellen Sie für Sicherheitsgruppen Sicherheitsgruppen, die den Endpunkt-Netzwerkschnittstellen zugeordnet werden sollen.
7. Behalten Sie für Policy die Standardeinstellung „Vollzugriff“ bei oder passen Sie sie nach Bedarf an.
8. Wählen Sie DNS-Name aktivieren aus, um privates DNS für den Endpunkt zu aktivieren. Lassen Sie den privaten DNS-Namen aktiviert, um Änderungen an der SDK-Konfiguration zu verhindern. Wenn diese Option aktiviert ist, können Ihre Anwendungen weiterhin den Standard-DNS-Namen des Dienstes verwenden (Beispiel:`dax.region.amazonaws.com`). AWS erstellt eine private gehostete Zone in Ihrer VPC, die diesen Namen in die private IP-Adresse Ihres Endpunkts auflöst.

   > **Note**
   >
   > Verwenden Sie bei Bedarf regionale DNS-Namen. Die Verwendung zonaler DNS-Namen wird nicht empfohlen. Wählen Sie außerdem Subnetze aus 3 oder mehr aus AZs , um eine maximale Verfügbarkeit zu gewährleisten. PrivateLink

9. Wählen Sie Endpunkt erstellen aus.

Um einen Schnittstellenendpunkt für DAX mit dem zu erstellen AWS CLI

Verwenden Sie den `create-vpc-endpoint` Befehl mit dem `vpc-endpoint-type` Parametersatz auf `Interface` und dem `service-name` Parametersatz auf`com.amazonaws.region.dax`.

```
aws ec2 create-vpc-endpoint \
    --vpc-id vpc-ec43eb89 \
    --vpc-endpoint-type Interface \
    --service-name com.amazonaws.us-east-1.dax \
    --subnet-ids subnet-abcd1234 subnet-1a2b3c4d \
    --security-group-ids sg-1a2b3c4d \
    --private-dns-enabled
```

Weitere Ressourcen

Weitere Informationen zu AWS PrivateLink VPC-Endpunkten finden Sie in den folgenden Ressourcen:

- AWS PrivateLink für DynamoDB
- AWS PrivateLink für DynamoDB Streams
- Connect Sie Ihre VPC mit Diensten über AWS PrivateLink
- Vereinfachen Sie die private Konnektivität zu DynamoDB mit AWS PrivateLink
- AWS PrivateLink Whitepaper

Konfigurations- und Schwachstellenanalyse in Amazon DynamoDB

AWS kümmert sich um grundlegende Sicherheitsaufgaben wie das Patchen von Gastbetriebssystemen (OS) und Datenbanken, die Firewall-Konfiguration und die Notfallwiederherstellung. Diese Verfahren wurden von qualifizierten Dritten überprüft und zertifiziert. Weitere Informationen finden Sie in den folgenden Ressourcen:

- Compliance-Validierung für Amazon DynamoDB
- Modell der geteilten Verantwortung
- Amazon Web Services: Übersicht über Sicherheitsverfahren (Whitepaper)

Die folgenden bewährten Sicherheitsmethoden beinhalten auch die Adresskonfiguration und Schwachstellenanalyse in Amazon DynamoDB:

- Überwachen Sie die DynamoDB-Konformität mit AWS-Config-Regeln
- Überwachen Sie die DynamoDB-Konfiguration mit AWS Config

Bewährte Methoden für die Sicherheit für Amazon DynamoDB

Amazon DynamoDB enthält eine Reihe von Sicherheitsfunktionen, die Sie bei der Entwicklung und Implementierung Ihrer eigenen Sicherheitsrichtlinien berücksichtigen sollten. Die folgenden bewährten Methoden sind allgemeine Richtlinien und keine vollständige Sicherheitslösung. Da diese bewährten Methoden für Ihre Umgebung möglicherweise nicht angemessen oder ausreichend sind, sollten Sie sie als hilfreiche Überlegungen und nicht als bindend ansehen.

Themen

- Bewährte Methoden für vorbeugende DynamoDB-Sicherheitsmaßnahmen
- DynamoDB Bewährte Methoden für Sicherheitsmaßnahmen entdecken

Bewährte Methoden für vorbeugende DynamoDB-Sicherheitsmaßnahmen

Die folgenden bewährten Methoden können Ihnen helfen, Sicherheitsvorfälle in Amazon DynamoDB zu antizipieren und zu verhindern.

Verschlüsselung im Ruhezustand

DynamoDB verschlüsselt im Ruhezustand alle Benutzerdaten, die in Tabellen, Indexen, Streams und Backups gespeichert sind, mithilfe von Verschlüsselungsschlüsseln, die in AWS Key Management Service (AWS KMS) gespeichert sind. Dies bietet eine zusätzliche Datenschutzebene, indem Ihre Daten vor unbefugtem Zugriff auf den zugrunde liegenden Speicher geschützt werden.

Sie können angeben, ob DynamoDB einen AWS-eigener Schlüssel (Standardverschlüsselungstyp), einen oder einen Von AWS verwalteter Schlüssel vom Kunden verwalteten Schlüssel zum Verschlüsseln von Benutzerdaten verwenden soll. Weitere Informationen finden Sie unter Amazon DynamoDB-Verschlüsselung im Ruhezustand.

Verwenden von IAM-Rollen zum Authentifizieren des Zugriffs auf DynamoDB

Damit Benutzer, Anwendungen und andere AWS Dienste auf DynamoDB zugreifen können, müssen sie gültige AWS Anmeldeinformationen in ihren AWS API-Anfragen angeben. Sie sollten AWS Anmeldeinformationen nicht direkt in der Anwendung oder EC2-Instance speichern. Dies sind langfristige Anmeldeinformationen, die nicht automatisch rotiert werden und daher erhebliche geschäftliche Auswirkungen haben können, wenn sie kompromittiert werden. Mit einer IAM-Rolle können Sie temporäre Zugriffsschlüssel abrufen, die für den Zugriff auf AWS Dienste und Ressourcen verwendet werden können.

Weitere Informationen finden Sie unter Identity and Access Management für Amazon DynamoDB.

Verwenden von IAM-Richtlinien für die DynamoDB-Basisautorisierung

Bei der Erteilung von Berechtigungen entscheiden Sie, wer sie erhält, für welche DynamoDB APIs sie Berechtigungen erhalten und welche spezifischen Aktionen Sie für diese Ressourcen zulassen möchten. Die Implementierung der geringsten Rechte ist der Schlüssel zur Verringerung

des Sicherheitsrisikos und der Auswirkungen, die sich aus Fehlern oder böswilligen Absichten ergeben können.

Weisen Sie IAM-Identitäten (d. h. Benutzern, Gruppen und Rollen) Berechtigungsrichtlinien zu und erteilen Sie somit Berechtigungen zum Ausführen von Operationen für DynamoDB-Ressourcen.

Sie können dies wie folgt tun:

- AWS Verwaltete (vordefinierte) Richtlinien
- Kundenverwaltete Richtlinien

Verwenden von IAM-Richtlinienbedingungen für eine differenzierte Zugriffssteuerung

Wenn Sie in DynamoDB Berechtigungen gewähren, können Sie Bedingungen angeben, die bestimmen, wie eine Berechtigungsrichtlinie wirksam wird. Die Implementierung der geringsten Rechte ist der Schlüssel zur Verringerung des Sicherheitsrisikos und der Auswirkungen, die sich aus Fehlern oder böswilligen Absichten ergeben können.

Sie können Bedingungen angeben, wenn Sie Berechtigungen mithilfe einer IAM-Richtlinie erteilen. Sie können z. B. Folgendes tun:

- Erteilen Sie Berechtigungen, um Benutzern schreibgeschützten Zugriff auf bestimmte Elemente und Attribute in einer Tabelle oder einem sekundären Index zu gewähren.
- Erteilen von Berechtigungen, damit den Benutzern lesegeschützter Zugriff auf bestimmte Attribute in einer Tabelle, basierend auf der Identität dieses Benutzers, gewährt wird.

Weitere Informationen finden Sie unter Verwenden von IAM-Richtlinienbedingungen für die differenzierte Zugriffskontrolle.

Verwenden eines VPC-Endpunkts und Richtlinien für den Zugriff auf DynamoDB

Wenn Sie nur innerhalb einer Virtual Private Cloud (VPC) Zugriff auf DynamoDB benötigen, sollten Sie einen VPC-Endpunkt verwenden, um den Zugriff nur von der erforderlichen VPC aus zu beschränken. Dadurch wird verhindert, dass Datenverkehr durch das offene Internet gelangt und dieser Umgebung unterliegt.

Wenn Sie einen VPC Endpunkt für DynamoDB verwenden, können Sie den Zugriff mithilfe der folgenden Optionen steuern und einschränken:

- VPC Endpunktrichtlinien — Diese Richtlinien werden auf den DynamoDB-VPC-Endpunkt angewendet. Mit ihnen können Sie den API-Zugriff auf die DynamoDB-Tabelle steuern und einschränken.

- IAM-Richtlinien – Mit der `aws:sourceVpce`-Bedingung für Richtlinien, die Benutzer, Gruppen oder Rollen zugeordnet sind, können Sie erzwingen, dass der gesamte Zugriff auf die DynamoDB-Tabelle über den angegebenen VPC-Endpunkt erfolgt.

Weitere Informationen finden Sie unter Endpunkte für Amazon DynamoDB.

Clientseitige Verschlüsselung in Betracht ziehen

Wir empfehlen Ihnen, Ihre Verschlüsselungsstrategie zu planen, bevor Sie Ihre Tabelle in DynamoDB implementieren. Wenn Sie sensible oder vertrauliche Daten in DynamoDB speichern, sollten Sie erwägen, eine clientseitige Verschlüsselung in Ihren Plan aufzunehmen. Auf diese Weise können Sie Daten so nah wie möglich an ihrem Ursprung verschlüsseln und den Schutz der Daten während ihres gesamten Lebenszyklus gewährleisten. Die Verschlüsselung Ihrer sensiblen Daten während der Übertragung und im Ruhezustand stellt sicher, dass Ihre Klartextdaten nicht für Dritte verfügbar sind.

Das AWS Database Encryption SDK for DynamoDB ist eine Software-Bibliothek, die Ihnen dabei hilft, Ihre Tabellendaten zu schützen, bevor Sie sie an DynamoDB senden. Es verschlüsselt, signiert, verifiziert und entschlüsselt Ihre DynamoDB-Tabellenelemente. Sie steuern, welche Attribute verschlüsselt und signiert werden.

Überlegungen zum Primärschlüssel

Verwenden Sie in Ihrem Primärschlüssel für Ihre Tabelle und globale sekundäre Indizes keine vertraulichen Namen oder sensiblen Klartextdaten. Schlüsselnamen werden in Ihrer Tabellendefinition angezeigt. Die Namen der Primärschlüssel sind beispielsweise für jeden zugänglich, der über Anrufberechtigungen verfügt DescribeTable. Schlüsselwerte können in Ihren AWS CloudTrailund anderen Protokollen auftauchen. Darüber hinaus verwendet DynamoDB die Schlüsselwerte, um Daten zu verteilen und Anfragen weiterzuleiten, und AWS Administratoren können die Werte beachten, um die Integrität des Dienstes aufrechtzuerhalten.

Wenn Sie vertrauliche Daten in Ihrer Tabelle oder Ihren GSI-Schlüsselwerten verwenden müssen, empfehlen wir die Verwendung von end-to-end Client-Verschlüsselung. Auf diese Weise können Sie Schlüssel-Wert-Verweise für Ihre Daten durchführen und gleichzeitig sicherstellen, dass sie in Ihren DynamoDB-bezogenen Protokollen niemals unverschlüsselt angezeigt werden. Eine Möglichkeit, dies zu erreichen, ist die Verwendung des AWS -Datenverschlüsselungs-SDK für DynamoDB, das aber nicht erforderlich ist. Wenn Sie Ihre eigene Lösung verwenden, sollte immer einen ausreichend sicherer Verschlüsselungsalgorithmus verwendet werden. Sie sollten keine nicht-kryptografische Option wie einen Hash verwenden, da dieser in den meisten Situationen als nicht ausreichend sicher betrachtet wird.

Wenn die Schlüsselnamen Ihrer Primärschlüssel sensibel sind, empfehlen wir, stattdessen `pk` und `sk` zu verwenden. Das ist eine allgemeine bewährte Methode, die für ein flexibles Design Ihres Partitionsschlüssels sorgt.

Wenden Sie sich immer an Ihre Sicherheitsexperten oder Ihr AWS Kundenbetreuungsteam, wenn Sie sich nicht sicher sind, was die richtige Wahl wäre.

DynamoDB Bewährte Methoden für Sicherheitsmaßnahmen entdecken

Die folgenden bewährten Methoden für Amazon DynamoDB können dabei helfen, potenzielle Sicherheitsschwächen und Vorfälle zu erkennen.

Wird verwendet AWS CloudTrail , um die Verwendung von AWS verwalteten KMS-Schlüsseln zu überwachen

Wenn Sie einen Von AWS verwalteter Schlüsselfür die Verschlüsselung im Ruhezustand verwenden, wird die Verwendung dieses Schlüssels angemeldet AWS CloudTrail. CloudTrail bietet Einblick in die Benutzeraktivitäten, indem es die auf Ihrem Konto ausgeführten Aktionen aufzeichnet. CloudTrail zeichnet wichtige Informationen zu jeder Aktion auf, einschließlich der Person, die die Anfrage gestellt hat, die verwendeten Dienste, die durchgeführten Aktionen, die Parameter für die Aktionen und die vom AWS Dienst zurückgegebenen Antwortelemente. Diese Informationen helfen Ihnen dabei, Änderungen an Ihren AWS Ressourcen nachzuverfolgen und betriebliche Probleme zu beheben. CloudTrail macht es einfacher, die Einhaltung interner Richtlinien und regulatorischer Standards sicherzustellen.

Sie können es verwenden CloudTrail , um die Verwendung von Schlüsseln zu überprüfen. CloudTrail erstellt Protokolldateien, die einen Verlauf der AWS API-Aufrufe und verwandter Ereignisse für Ihr Konto enthalten. Diese Protokolldateien enthalten alle AWS KMS API-Anfragen AWS-Managementkonsole, die mit den Befehlszeilentools AWS SDKs, und gestellt wurden, sowie alle API-Anfragen, die über integrierte AWS Dienste gestellt wurden. Sie können diese Protokolldateien verwenden, um Informationen über den Zeitpunkt der Verwendung des KMS-Schlüssels zu erhalten, sowie Informationen über die angeforderte Operation, die Identität des Auftraggebers und die IP-Adresse, von der aus die Anforderung gesendet wurde. Weitere Informationen finden Sie unter Protokollierung von AWS KMS -API-Aufrufen mit AWS CloudTrail im AWS CloudTrail -Benutzerhandbuch.

Überwachen Sie DynamoDB-Operationen mit CloudTrail

CloudTrail kann sowohl Ereignisse auf der Steuerungsebene als auch Ereignisse auf der Datenebene überwachen. Mit Operationen auf Steuerebene können Sie DynamoDB-Tabellen erstellen und verwalten. Außerdem ermöglichen sie die Arbeit mit Indexen, Streams und anderen Objekten, die von Tabellen abhängen. Mit Operationen auf Datenebene können Sie Aktionen zum Erstellen, Lesen, Aktualisieren und Löschen (auch als CRUD bezeichnet) für Daten in einer Tabelle ausführen. Einige Operationen auf Datenebene ermöglichen außerdem das Lesen von Daten aus einem Sekundär-Index. Um die Protokollierung von Ereignissen auf der Datenebene zu aktivieren CloudTrail, müssen Sie die Protokollierung der API-Aktivitäten auf der Datenebene in aktivieren CloudTrail. Weitere Informationen finden Sie unter Protokollieren von Datenereignissen für Trails.

Wenn eine Aktivität in DynamoDB auftritt, wird diese Aktivität zusammen mit anderen AWS Dienstereignissen in der CloudTrail Ereignishistorie in einem Ereignis aufgezeichnet. Weitere Informationen finden Sie unter Protokollieren von DynamoDB-Vorgängen mithilfe von AWS CloudTrail. Sie können aktuelle Ereignisse in Ihrem AWS Konto anzeigen, suchen und herunterladen. Weitere Informationen finden Sie im AWS CloudTrail Benutzerhandbuch unter Ereignisse mit CloudTrail Ereignisverlauf anzeigen.

Für eine fortlaufende Aufzeichnung von Ereignissen in Ihrem AWS Konto, einschließlich Ereignissen für DynamoDB, erstellen Sie einen Trail. Ein Trail ermöglicht CloudTrail die Übermittlung von Protokolldateien an einen Amazon Simple Storage Service (Amazon S3) -Bucket. Wenn Sie einen Trail auf der Konsole erstellen, gilt der Trail standardmäßig für alle AWS Regionen. Der Trail protokolliert Ereignisse aus allen Regionen in der AWS -Partition und stellt die Protokolldateien für den von Ihnen angegebenen S3 Bucket bereit. Darüber hinaus können Sie andere AWS Dienste konfigurieren, um die in den CloudTrail Protokollen gesammelten Ereignisdaten weiter zu analysieren und darauf zu reagieren.

Verwenden von DynamoDB Streams zum Überwachen von Operationen auf Datenebene

DynamoDB ist integriert, AWS Lambda sodass Sie Trigger erstellen können — Codeteile, die automatisch auf Ereignisse in DynamoDB Streams reagieren. Mit Auslösern können Sie Anwendungen erstellen, die auf Datenänderungen in DynamoDB-Tabellen reagieren.

Wenn Sie DynamoDB Streams in einer Tabelle aktivieren, können Sie den Amazon-Ressourcennamen (ARN) des Streams einer Lambda-Funktion zuordnen, die Sie schreiben. Unmittelbar nach der Änderung eines Elements in der Tabelle erscheint ein neuer Datensatz im Stream der Tabelle. AWS Lambda fragt den Stream ab und ruft Ihre Lambda-Funktion

synchron auf, wenn sie neue Stream-Datensätze erkennt. Die Lambda-Funktion kann jede, von Ihnen festgelegte Aktion durchführen, z. B. Senden einer Benachrichtigung oder Initiieren eines Workflows.

Ein Beispiel finden Sie unter Tutorial: Verwenden AWS Lambda mit Amazon DynamoDB Streams. In diesem Beispiel wird eine DynamoDB-Ereigniseingabe empfangen, die darin enthaltenen Nachrichten verarbeitet und einige der eingehenden Ereignisdaten in Amazon CloudWatch Logs geschrieben.

Überwachen Sie die DynamoDB-Konfiguration mit AWS Config

Mit AWS Config können Sie Konfigurationsänderungen Ihrer AWS -Ressourcen kontinuierlich überwachen und aufzeichnen. Sie können es auch verwenden, um Ihre AWS Config AWS Ressourcen zu inventarisieren. Wenn eine Änderung eines früheren Status erkannt wird, kann eine Amazon-Simple-Notification-Service (Amazon SNS)-Benachrichtigung gesendet werden, mit der Sie überprüfen und Maßnahmen ergreifen können. Folgen Sie den Anweisungen unter Einrichtung AWS Config mit der Konsole und stellen Sie sicher, dass DynamoDB-Ressourcentypen enthalten sind.

Sie können konfigurieren AWS Config , dass Konfigurationsänderungen und Benachrichtigungen zu einem Amazon SNS SNS-Thema gestreamt werden. Wenn zum Beispiel eine Ressource aktualisiert wird, können Sie eine Benachrichtigung an Ihre E-Mail-Adresse erhalten, damit Sie die Änderungen sehen können. Sie können auch benachrichtigt werden, wenn Ihre benutzerdefinierten oder verwalteten Regeln anhand Ihrer Ressourcen AWS Config bewertet werden.

Ein Beispiel finden Sie im Thema Benachrichtigungen, die AWS Config an Amazon SNS gesendet werden im AWS Config Entwicklerhandbuch.

Überwachen Sie die Einhaltung der Regeln durch DynamoDB AWS Config

AWS Config verfolgt kontinuierlich die Konfigurationsänderungen, die an Ihren Ressourcen vorgenommen werden. Es prüft, ob diese Änderungen gegen eine Bedingung Ihrer Regeln verstoßen. Wenn eine Ressource gegen eine Regel verstößt, werden die AWS Config Ressource und die Regel als nicht konform gekennzeichnet.

Anhand der AWS Config Auswertung Ihrer Ressourcenkonfigurationen können Sie beurteilen, wie gut Ihre Ressourcenkonfigurationen internen Praktiken, Branchenrichtlinien und Vorschriften entsprechen. AWS Config stellt AWS verwaltete Regeln bereit. Dabei handelt es sich um vordefinierte, anpassbare Regeln, AWS Config anhand derer bewertet wird, ob Ihre AWS Ressourcen den gängigen bewährten Methoden entsprechen.

Markieren Sie Ihre DynamoDB-Ressourcen zur Identifizierung und Automatisierung

Sie können Ihren AWS Ressourcen Metadaten in Form von Tags zuweisen. Jedes Tag ist eine einfache Bezeichnung, die aus einem benutzerdefinierten Schlüssel und einem optionalen Wert besteht, der das Verwalten, Suchen und Filtern von Ressourcen erleichtern kann.

Tagging ermöglicht die Implementierung gruppierter Steuerelemente. Obwohl es keine inhärenten Typen von Tags gibt, können Sie Ressourcen nach Zweck, Besitzer, Umgebung oder anderen Kriterien kategorisieren. Im Folgenden sind einige Beispiele aufgeführt:

- Sicherheit – Wird verwendet, um Anforderungen wie Verschlüsselung zu bestimmen.
- Vertraulichkeit – Eine Kennung für die spezifische Datenvertraulichkeitsebene, die eine Ressource unterstützt
- Umgebung – Wird verwendet, um zwischen Entwicklungs-, Test- und Produktionsinfrastruktur zu unterscheiden.

Weitere Informationen finden Sie unter AWS Markierungsstrategien und Markierung für DynamoDB.

Überwachen Sie Ihre Nutzung von Amazon DynamoDB in Bezug auf bewährte Sicherheitsmethoden mithilfe von. AWS Security Hub CSPM

Security Hub CSPM verwendet Sicherheitskontrollen, um Ressourcenkonfigurationen und Sicherheitsstandards zu bewerten und Sie bei der Einhaltung verschiedener Compliance-Frameworks zu unterstützen.

Weitere Informationen zur Verwendung von Security Hub CSPM zur Evaluierung von DynamoDB-Ressourcen finden Sie unter Amazon DynamoDB-Steuerelemente im Benutzerhandbuch.AWS Security Hub CSPM

Überwachen und Protokollieren in DynamoDB

Die Überwachung ist eine wichtige Komponente für die Wahrung der Zuverlässigkeit, Verfügbarkeit und Leistung von DynamoDB und Ihrer AWS-Lösungen. Sie sollten aus allen Komponenten Ihrer AWS-Lösungen Überwachungsdaten erfassen, damit Sie einen Fehler mit mehreren Ursachen leichter debuggen können.

Themen

- Überwachungsplan
- Leistungsbasislinie
- Integrierte Services
- Automatisierte Überwachungstools
- Überwachen von Metriken in DynamoDB mit Amazon CloudWatch
- Protokollieren von DynamoDB-Vorgängen mithilfe von AWS CloudTrail
- Analysieren des Datenzugriffs mithilfe von CloudWatch Contributor Insights für DynamoDB

Überwachungsplan

Bevor Sie mit der Überwachung von DynamoDB beginnen, sollten Sie einen Überwachungsplan mit Antworten auf die folgenden Fragen erstellen:

- Was sind Ihre Ziele bei der Überwachung?
- Welche Ressourcen werden überwacht?
- Wie oft werden diese Ressourcen überwacht?
- Welche Überwachungstools werden verwendet?
- Wer soll die Überwachungsaufgaben ausführen?
- Wer soll benachrichtigt werden, wenn Fehler auftreten?

Leistungsbasislinie

Im nächsten Schritt legen Sie einen Ausgangswert für normale DynamoDB-Leistung in Ihrer Umgebung fest, indem Sie die Leistung zu verschiedenen Zeiten und unter verschiedenen Lastbedingungen messen. Wenn Sie DynamoDB überwachen, sollten Sie in Betracht ziehen,

historische Überwachungsdaten zu speichern. Diese Daten bieten eine Basis für den Vergleich mit aktuellen Leistungsdaten, zur Erkennung normaler Leistungsmuster und von Leistungsanomalien sowie zur Entwicklung von Verfahren für den Umgang mit Problemen. Zur Festlegung eines Grundwertes sollten Sie mindestens die folgenden Elemente überwachen:

- Anzahl der in einem bestimmten Zeitraum verbrauchten Lese- oder Schreibkapazitätseinheiten, um nachverfolgen zu können, in welchem Maß Ihre bereitgestellte Durchsatzkapazität verwendet wird
- Anforderungen, die die Lese- oder Schreibkapazität einer Tabelle während des angegebenen Zeitraums überschritten haben, so dass Sie ermitteln können, welche Anforderungen die bereitgestellten Durchsatzkontingente für eine Tabelle übersteigen
- Systemfehler, damit Sie feststellen können, ob Anforderungen zu einem Fehler geführt haben

Integrierte Services

DynamoDB überwacht automatisch Tabellen für Sie und meldet die Metriken über Amazon CloudWatch. Darüber hinaus lässt sich DynamoDB in die folgenden AWS-Services integrieren, die Ihnen helfen, Ihre DynamoDB-Ressourcen zu überwachen und Fehler zu beheben.

- AWS CloudTrail erfasst API-Aufrufe und zugehörige Ereignisse, die von oder im Namen Ihres AWS-Konto-Kontos erfolgten, und übermittelt die Protokolldateien an einen von Ihnen angegebenen Amazon-S3-Bucket. Weitere Informationen finden Sie unter Protokollieren von DynamoDB-Vorgängen mithilfe von AWS CloudTrail.
- Contributor Insights ist ein Diagnose-Tool, mit dem die Schlüssel mit den meisten Zugriffen oder Drosselungen in einer Tabelle oder einem Index schnell identifiziert werden können. Weitere Informationen finden Sie unter Analysieren des Datenzugriffs mithilfe von CloudWatch Contributor Insights für DynamoDB.

Automatisierte Überwachungstools

AWS bietet verschiedene Tools für die Überwachung von DynamoDB. Wir empfehlen, dass Sie die Überwachungsaufgaben möglichst automatisieren. Sie können die folgenden automatisierten Tools zur Überwachung von DynamoDB verwenden und möglicherweise auftretende Probleme melden:

- Amazon-CloudWatch-Alarme: Überwachen Sie eine Metrik über einen bestimmten, von Ihnen definierten Zeitraum und führen Sie einzelne oder mehrere Aktionen durch, die vom Wert der Metrik im Vergleich zu einem festgelegten Schwellenwert in einer Reihe von Zeiträumen abhängen.

Bei der Aktion handelt es sich um eine Benachrichtigung, die an ein Amazon-Simple-Notification-Service (Amazon-SNS)-Thema oder eine Amazon-EC2-Auto-Scaling-Richtlinie gesendet wird. CloudWatch-Alarme rufen keine Aktionen auf, nur weil sie einen bestimmten Status aufweisen. Der Status muss geändert und für eine bestimmte Anzahl an Zeiträumen aufrechterhalten worden sein. Weitere Informationen finden Sie unter Überwachen von Metriken in DynamoDB mit Amazon CloudWatch.

- AWS CloudTrail-Protokollüberwachung: Teilen Sie Protokolldateien zwischen Konten, überwachen Sie AWS CloudTrail-Protokolldateien in Echtzeit, indem Sie sie an AWS CloudTrail-Protokolle senden, schreiben Sie Anwendungen zur Protokollverarbeitung in Java und vergewissern Sie sich, dass keine Änderungen an den Protokolldaten vorgenommen wurden, nachdem sie von AWS CloudTrail übermittelt wurden. Weitere Informationen finden Sie unter Was ist Amazon CloudWatch Logs im Amazon AWS CloudTrail-Benutzerhandbuch.

Überwachen von Metriken in DynamoDB mit Amazon CloudWatch

Sie können DynamoDB mithilfe von DynamoDB überwachen CloudWatch, das Rohdaten aus DynamoDB sammelt und in lesbare Metriken nahezu in Echtzeit verarbeitet. Diese Statistiken werden für einen bestimmten Zeitraum aufbewahrt, damit Sie auf historische Informationen zugreifen können, um einen besseren Überblick über die Leistung Ihrer Webanwendung oder Ihres Dienstes zu erhalten. Standardmäßig werden DynamoDB-Metrikdaten automatisch an CloudWatch gesendet. Weitere Informationen finden Sie unter Was ist Amazon CloudWatch? und Aufbewahrung von Kennzahlen im CloudWatch Amazon-Benutzerhandbuch.

Themen

- Wie verwende ich DynamoDB-Metriken?
- Metriken in der Konsole anzeigen CloudWatch
- Metriken anzeigen im AWS CLI
- DynamoDB-Metriken und -Dimensionen
- CloudWatch Alarme in DynamoDB erstellen

Wie verwende ich DynamoDB-Metriken?

Die von DynamoDB gemeldeten Metriken bieten Informationen, die Sie auf unterschiedliche Weise analysieren können. In der folgenden Liste finden Sie einige häufige Verwendungszwecke für die Metriken. Es handelt sich dabei um Vorschläge für den Einstieg und nicht um eine umfassende Liste.

Wie verwende ich DynamoDB-Metriken?

Wie kann ich ...?	Relevante Metriken
Wie kann ich die Rate der TTL-Löschvorgänge in meiner Tabelle überwachen?	Sie können `TimeToLiveDeletedItemCount` über den angegebenen Zeitraum überwachen, um die TTL-Löschrate in Ihrer Tabelle zu verfolgen. Ein Beispiel für eine serverlose Anwendung, die die `TimeToLiveDeletedItemCount` Metrik verwendet, finden Sie unter Automatisches Archivieren von Elementen in S3 mithilfe von DynamoDB Time to Live (TTL) with AWS Lambda und Amazon Data Firehose.
Wie kann ich feststellen, wie viel von meinem bereitgestellten Durchsatz genutzt wird?	Sie können `ConsumedReadCapacityUnits` oder `ConsumedWriteCapacityUnits` über den angegebenen Zeitraum überwachen, um nachzuverfolgen, in welchem Maß Ihre bereitgestellte Durchsatzkapazität verwendet wird.
Wie kann ich feststellen, welche Anforderungen die Kontingente für den bereitgestellten Durchsatz einer Tabelle überschreiten?	`ThrottledRequests` wird um eins erhöht, wenn ein Ereignis innerhalb einer Anforderung die Grenze eines bereitgestellten Durchsatzkontingents überschreitet. Um zu erfahren, durch welches Ereignis eine Anforderung gedrosselt wird, vergleichen Sie `ThrottledRequests` mit den Metriken `ReadThrottleEvents` und `WriteThrottleEvents` für die Tabelle und ihre Indizes.
Wie kann ich feststellen, wenn ein Systemfehler aufgetreten ist?	Sie können `SystemErrors` überwachen, um festzustellen, ob Anforderungen zu einem HTTP 500-Fehler (Serverfehler) geführt haben. In der Regel sollte diese Metrik gleich Null sein. Ist dies nicht der Fall, können Sie dies untersuchen.
Wie kann ich den Latenzwert für meine Tabellenoperationen überwachen?	Sie können den Wert `SuccessfulRequestLatency` überwachen, indem Sie die durchschnittliche Latenz und die mittlere Latenz anhand von Perzentilmetriken verfolgen (p50).

Wie kann ich ...?	Relevante Metriken
	Gelegentliche Latenzspitzen sind kein Grund zur Sorge. Wenn die durchschnittliche Latenz oder p50 (der Median) jedoch hoch ist, liegt möglicherweise ein Problem zugrunde, das Sie lösen müssen. Weitere Informationen finden Sie unter Beheben von Latenzproblemen in Amazon DynamoDB.

Metriken in der Konsole anzeigen CloudWatch

Metriken werden zunächst nach dem Service-Namespace und anschließend nach den verschiedenen Dimensionskombinationen in den einzelnen Namespaces gruppiert.

Um Metriken in der CloudWatch Konsole anzuzeigen

1. Öffnen Sie die CloudWatch Konsole unter https://console.aws.amazon.com/cloudwatch/.
2. Wählen Sie im Navigationsbereich die Option Metriken und anschließend „Alle Metriken“ aus.
3. Wählen Sie den Namespace DynamoDB aus. Sie können auch den Namespace Usage (Verwendung) auswählen, um DynamoDB-Nutzungsmetriken anzuzeigen. Weitere Informationen zu Nutzungsmetriken finden Sie unter AWS -Nutzungsmetriken.
4. Die Registerkarte Durchsuchen zeigt alle Metriken für diese Dimension im Namespace an.
5. (Optional) Um das Metrikdiagramm zu einem CloudWatch Dashboard hinzuzufügen, wählen Sie Aktionen, Zum Dashboard hinzufügen aus.

Metriken anzeigen im AWS CLI

Verwenden Sie den CloudWatch Befehl AWS CLI, um Metrikinformationen mit dem abzurufen`list-metrics`. Im folgenden Beispiel listen Sie alle Metriken im `AWS/DynamoDB`-Namespace auf.

```
aws cloudwatch list-metrics --namespace "AWS/DynamoDB"
```

Um Metrikstatistiken zu erhalten, verwenden Sie den Befehl `get-metric-statistics`. Der folgende Befehl ruft `ConsumedReadCapacityUnits`-Statistiken für die Tabelle `ProductCatalog` über den jeweiligen 24-Stunden-Zeitraum mit einer Granularität von 5 Minuten ab.

```
aws cloudwatch get-metric-statistics –namespace AWS/DynamoDB \
     –metric-name ConsumedReadCapacityUnits \
     –start-time 2023-11-01T00:00:00Z \
     –end-time 2023-11-02T00:00:00Z \
     –period 360 \
     –statistics Average \
     –dimensions Name=TableName,Value=ProductCatalog
```

Beispielausgabe:

```
{
    "Datapoints": [
        {
            "Timestamp": "2023-11-01T 09:18:00+00:00",
            "Average": 20,
            "Unit": "Count"
        },
        {
            "Timestamp": "2023-11-01T 04:36:00+00:00",
            "Average": 22.5,
            "Unit": "Count"
        },
        {
            "Timestamp": "2023-11-01T 15:12:00+00:00",
            "Average": 20,
            "Unit": "Count"
        }, ...
        {
            "Timestamp": "2023-11-01T 17:30:00+00:00",
            "Average": 25,
            "Unit": "Count"
        }
    ],
    "Label": " ConsumedReadCapacityUnits "
}
```

DynamoDB-Metriken und -Dimensionen

Wenn Sie mit DynamoDB interagieren, sendet es Metriken und Dimensionen an. CloudWatch

DynamoDB-Ausgaben verbrauchten den bereitgestellten Durchsatz für Zeiträume von einer Minute. Auto Scaling wird ausgelöst, wenn Ihre verbrauchte Kapazität für zwei aufeinanderfolgende

Minuten die konfigurierte Zielauslastung überschreitet. CloudWatch Alarme können eine kurze Verzögerung von bis zu einigen Minuten haben, bevor sie die auto Skalierung auslösen. Diese Verzögerung gewährleistet eine genaue CloudWatch metrische Auswertung. Wenn die Spitzen des verbrauchten Durchsatzes in Abständen von mehr als einer Minute auftreten, wird Auto Scaling möglicherweise nicht ausgelöst. In ähnlicher Weise kann es zu einer Herunterskalierung kommen, wenn 15 aufeinanderfolgende Datenpunkte unter der Zielauslastung liegen. In beiden Fällen wird die UpdateTableAPI aufgerufen, nachdem Auto Scaling ausgelöst wurde. Die Aktualisierung der bereitgestellten Kapazität für die Tabelle oder den Index kann dann einige Minuten dauern. Während dieses Zeitraums werden alle Anforderungen gedrosselt, die die zuvor bereitgestellte Kapazität der Tabellen überschreiten.

Anzeigen von -Metriken und -Dimensionen

CloudWatch zeigt die folgenden Metriken für DynamoDB an:

DynamoDB-Metriken

> **Note**
>
> Amazon CloudWatch aggregiert diese Metriken in Intervallen von einer Minute:
>
> - `ConditionalCheckFailedRequests`
> - `ConsumedReadCapacityUnits`
> - `ConsumedWriteCapacityUnits`
> - `ReadAccountLimitThrottleEvents`
> - `ReadKeyRangeThroughputThrottleEvents`
> - `ReadMaxOnDemandThroughputThrottleEvents`
> - `ReadProvisionedThroughputThrottleEvents`
> - `ReadThrottleEvents`
> - `ReturnedBytes`
> - `ReturnedItemCount`
> - `ReturnedRecordsCount`
> - `SuccessfulRequestLatency`
> - `SystemErrors`
> - `TimeToLiveDeletedItemCount`
> - `ThrottledRequests`

- `TransactionConflict`
- `UserErrors`
- `WriteAccountLimitThrottleEvents`
- `WriteKeyRangeThroughputThrottleEvents`
- `WriteMaxOnDemandThroughputThrottleEvents`
- `WriteProvisionedThroughputThrottleEvents`
- `WriteThrottleEvents`
- `FaultInjectionServiceInducedErrors`

Für alle anderen DynamoDB-Metriken beträgt die Aggregationsgranularität fünf Minuten.

Nicht alle Statistiken, wie Durchschnitt oder Summe gelten für jede Metrik. Alle diese Werte sind jedoch über die Amazon DynamoDB DynamoDB-Konsole oder über die CloudWatch Konsole oder AWS SDKs für alle AWS CLI Metriken verfügbar.

In der folgenden Liste ist zu jeder Metrik eine Reihe gültiger Statistiken aufgeführt, die auf diese Metrik anwendbar sind.

Liste der verfügbaren Metriken

- AccountMaxReads
- AccountMaxTableLevelReads
- AccountMaxTableLevelWrites
- AccountMaxWrites
- AccountProvisionedReadCapacityUtilization
- AccountProvisionedWriteCapacityUtilization
- AgeOfOldestUnreplicatedRecord
- ConditionalCheckFailedRequests
- ConsumedChangeDataCaptureUnits
- ConsumedReadCapacityUnits
- ConsumedWriteCapacityUnits
- FailedToReplicateRecordCount

- MaxProvisionedTableReadCapacityUtilization
- MaxProvisionedTableWriteCapacityUtilization
- OnDemandMaxReadRequestUnits
- OnDemandMaxWriteRequestUnits
- OnlineIndexConsumedWriteCapacity
- OnlineIndexPercentageProgress
- OnlineIndexThrottleEvents
- PendingReplicationCount
- ProvisionedReadCapacityUnits
- ProvisionedWriteCapacityUnits
- ReadAccountLimitThrottleEvents
- ReadKeyRangeThroughputThrottleEvents
- ReadMaxOnDemandThroughputThrottleEvents
- ReadProvisionedThroughputThrottleEvents
- ReadThrottleEvents
- ReplicationLatency
- ReturnedBytes
- ReturnedItemCount
- ReturnedRecordsCount
- SuccessfulRequestLatency
- SystemErrors
- TimeToLiveDeletedItemCount
- ThrottledPutRecordCount
- ThrottledRequests
- TransactionConflict
- UserErrors
- WriteAccountLimitThrottleEvents
- WriteKeyRangeThroughputThrottleEvents

- WriteMaxOnDemandThroughputThrottleEvents
- WriteProvisionedThroughputThrottleEvents
- WriteThrottleEvents
- Nutzungsmetriken
- FaultInjectionServiceInducedErrors

AccountMaxReads

Die maximale Anzahl von Lesekapazitätseinheiten, die von einem Konto verwendet werden können. Dieses Limit gilt nicht für On-Demand-Tabellen oder globale sekundäre Indizes.

Einheiten: `Count`

Gültige Statistiken:

- `Maximum` – Die maximale Anzahl von Lesekapazitätseinheiten, die von einem Konto verwendet werden können.

AccountMaxTableLevelReads

Die maximale Anzahl von Lesekapazitätseinheiten, die von einer Tabelle oder einem globalen sekundären Index eines Kontos verwendet werden können. Bei On-Demand-Tabellen werden durch diesen Wert die maximalen Leseanforderungseinheiten begrenzt, die eine Tabelle oder ein globaler sekundärer Index verwenden kann.

Einheiten: `Count`

Gültige Statistiken:

- `Maximum` – Die maximale Anzahl von Lesekapazitätseinheiten, die von einer Tabelle oder einem globalen sekundären Index des Kontos verwendet werden können.

AccountMaxTableLevelWrites

Die maximale Anzahl von Schreibkapazitätseinheiten, die von einer Tabelle oder einem globalen sekundären Index eines Kontos verwendet werden können. Bei On-Demand-Tabellen werden durch diesen Wert die maximalen Schreibanforderungseinheiten begrenzt, die eine Tabelle oder ein globaler sekundärer Index verwenden kann.

Einheiten: `Count`

Gültige Statistiken:

- `Maximum` – Die maximale Anzahl von Schreibkapazitätseinheiten, die von einer Tabelle oder einem globalen sekundären Index des Kontos verwendet werden können.

AccountMaxWrites

Die maximale Anzahl von Schreibkapazitätseinheiten, die von einem Konto verwendet werden können. Dieses Limit gilt nicht für On-Demand-Tabellen oder globale sekundäre Indizes.

Einheiten: `Count`

Gültige Statistiken:

- `Maximum` – Die maximale Anzahl von Schreibkapazitätseinheiten, die von einem Konto verwendet werden können.

AccountProvisionedReadCapacityUtilization

Prozentsatz der bereitgestellten Lesekapazitätseinheiten, die vom Konto genutzt werden.

Einheiten: `Percent`

Gültige Statistiken:

- `Maximum` – Der maximale Prozentsatz der bereitgestellten Lesekapazitätseinheiten, die vom Konto genutzt werden.
- `Minimum` – Der Mindestprozentsatz der bereitgestellten Lesekapazitätseinheiten, die vom Konto genutzt werden.
- `Average` – Der durchschnittliche Prozentsatz der bereitgestellten Lesekapazitätseinheiten, die vom Konto genutzt werden. Die Metrik wird für Intervallen von fünf Minuten veröffentlicht. Wenn Sie die bereitgestellten Lesekapazitätseinheiten schnell anpassen, spiegelt diese Statistik möglicherweise nicht den tatsächlichen Durchschnitt wider.

AccountProvisionedWriteCapacityUtilization

Prozentsatz der bereitgestellten Schreibkapazitätseinheiten, die vom Konto genutzt werden.

Einheiten: `Percent`

Gültige Statistiken:

- `Maximum` – Der maximale Prozentsatz der bereitgestellten Schreibkapazitätseinheiten, die vom Konto genutzt werden.
- `Minimum` – Der minimale Prozentsatz der bereitgestellten Schreibkapazitätseinheiten, die vom Konto genutzt werden.
- `Average` – Der durchschnittliche Prozentsatz der bereitgestellten Schreibkapazitätseinheiten, die vom Konto genutzt werden. Die Metrik wird für Intervallen von fünf Minuten veröffentlicht. Wenn Sie die bereitgestellten Schreibkapazitätseinheiten schnell anpassen, spiegelt diese Statistik möglicherweise nicht den tatsächlichen Durchschnitt wider.

AgeOfOldestUnreplicatedRecord

Die verstrichene Zeit, seit ein Datensatz, der noch in den Kinesis-Datenstrom repliziert werden muss, zuerst in der DynamoDB-Tabelle angezeigt wurde.

Einheiten: `Milliseconds`

Maße: `TableName, DelegatedOperation`

Gültige Statistiken:

- `Maximum`.
- `Minimum`.
- `Average`.

ConditionalCheckFailedRequests

Die Anzahl der fehlgeschlagenen Versuche, bedingte Schreibvorgänge durchzuführen. Mit den `PutItem`-, `UpdateItem`- und `DeleteItem`-Operationen können Sie eine logische Bedingung angeben, die als wahr ausgewertet werden muss, bevor die Operation fortgesetzt werden kann. Wenn diese Bedingung als falsch ausgewertet wird, wird `ConditionalCheckFailedRequests` um eins erhöht. `ConditionalCheckFailedRequests` wird auch für PartiQL-Update- und Lösch-Anweisungen um eins erhöht, wenn eine logische Bedingung bereitgestellt wird und diese Bedingung als falsch ausgewertet wird.

> **Note**
>
> Ein fehlgeschlagener bedingter Schreibvorgang führt zu einem HTTP 400-Fehler (Bad Request). Diese Ereignisse spiegeln sich in der `ConditionalCheckFailedRequests`-Metrik, aber nicht in der `UserErrors`-Metrik wieder

Einheiten: `Count`

Maße: `TableName`

Gültige Statistiken:

- `Minimum`
- `Maximum`
- `Average`
- `SampleCount`
- `Sum`

ConsumedChangeDataCaptureUnits

Die Anzahl der verbrauchten Änderungsdatenerfassungseinheiten.

Einheiten: `Count`

Maße: `TableName, DelegatedOperation`

Gültige Statistiken:

- `Minimum`
- `Maximum`
- `Average`

ConsumedReadCapacityUnits

Anzahl der in einem bestimmten Zeitraum verbrauchten Lesekapazitätseinheiten für bereitgestellte Kapazität und On-Demand-Kapazität, um nachverfolgen zu können, in welchem Maß Ihre

Durchsatzkapazität verwendet wird Sie können die gesamte verbrauchte Lesekapazität für eine Tabelle und alle ihre globalen sekundären Indizes oder für einen bestimmten globalen sekundären Index abrufen. Weitere Informationen finden Sie unter Lese-/Schreibkapazitätsmodus.

Die `TableName`-Dimension gibt die `ConsumedReadCapacityUnits` für die Tabelle, aber nicht für globale sekundäre Indizes zurück. Zum Anzeigen von `ConsumedReadCapacityUnits` für einen globalen sekundären Index, müssen Sie sowohl `TableName` als auch `GlobalSecondaryIndexName` angeben.

> **Note**
>
> Das bedeutet, dass kurze, intensive Spitzen des Kapazitätsverbrauchs, die nur eine Sekunde andauern, möglicherweise nicht korrekt in der CloudWatch Grafik wiedergegeben werden, was möglicherweise zu einer niedrigeren scheinbaren Verbrauchsrate für diese Minute führen kann.
> Verwenden der `Sum`, um den verbrauchten Durchsatz zu berechnen. Rufen Sie zum Beispiel den `Sum`-Wert über einen Zeitraum von einer Minute ab und dividieren Sie ihn durch die Anzahl der Sekunden in einer Minute (60), um die durchschnittlichen `ConsumedReadCapacityUnits` pro Sekunde zu berechnen. Sie können den berechneten Wert mit dem bereitgestellten Durchsatzwert vergleichen, den Sie DynamoDB bereitstellen.

Einheiten: `Count`

Maße: `TableName, GlobalSecondaryIndexName`

Gültige Statistiken:

- `Minimum` – Die Mindestanzahl der Lesekapazitätseinheiten, die von einer einzelnen Anforderung an die Tabelle oder den Index verbraucht werden.
- `Maximum` – Die Anzahl der Lesekapazitätseinheiten, die von einer einzelnen Anforderung an die Tabelle oder den Index verbraucht werden.
- `Average` – Die durchschnittliche Lesekapazität pro Anforderung.

> **Note**
>
> Der `Average`-Wert wird von Perioden der Inaktivität beeinflusst, in denen der Stichprobenwert Null ist.

- `Sum` – Die verbrauchten Einheiten der gesamten Lesekapazität. Die nützlichste Statistik für diese Metrik ist `ConsumedReadCapacityUnits`.
- `SampleCount` – Stellt die Häufigkeit dar, mit der die Metrik ausgegeben wird. Selbst bei Tabellen ohne Datenverkehr wird der Wert `SampleCount` regelmäßig ausgegeben, wobei die Stichprobenwerte immer null betragen.

> **Note**
>
> Der `SampleCount`-Wert wird von Perioden der Inaktivität beeinflusst, in denen der Stichprobenwert Null ist.

ConsumedWriteCapacityUnits

Anzahl der in einem bestimmten Zeitraum verbrauchten Schreibkapazitätseinheiten für bereitgestellte Kapazität und On-Demand-Kapazität, um nachverfolgen zu können, in welchem Maß Ihre Durchsatzkapazität verwendet wird Sie können die gesamte verbrauchte Schreibkapazität für eine Tabelle und alle ihre globalen sekundären Indizes oder für einen bestimmten globalen sekundären Index abrufen. Weitere Informationen finden Sie unter Lese-/Schreibkapazitätsmodus.

Die `TableName`-Dimension gibt die `ConsumedWriteCapacityUnits` für die Tabelle, aber nicht für globale sekundäre Indizes zurück. Zum Anzeigen von `ConsumedWriteCapacityUnits` für einen globalen sekundären Index, müssen Sie sowohl `TableName` als auch `GlobalSecondaryIndexName` angeben. Die `Source` Dimension kann einen von zwei Werten zurückgeben: `Customer` und`GlobalTable`. Replizierte Schreibvorgänge haben `ConsumedWriteCapacityUnits` mit der Quelle zu `GlobalTable` tun, regionale Tabellenschreibvorgänge dagegen `ConsumedWriteCapacityUnits` mit der Quelle`Customer`.

> **Note**
>
> Verwenden der `Sum`, um den verbrauchten Durchsatz zu berechnen. Rufen Sie zum Beispiel den `Sum`-Wert über einen Zeitraum von einer Minute ab und dividieren Sie ihn durch die Anzahl der Sekunden in einer Minute (60), um den durchschnittlichen Wert für `ConsumedWriteCapacityUnits` pro Sekunde zu berechnen (wobei zu beachten ist, dass dieser Durchschnitt keine großen, sondern kurzen Spitzen in der Schreibaktivität hervorhebt, die während dieser Minute aufgetreten ist). Sie können den berechneten Wert mit dem bereitgestellten Durchsatzwert vergleichen, den Sie DynamoDB bereitstellen.

Einheiten: `Count`

Maße: `TableName, GlobalSecondaryIndexName, Source`

Gültige Statistiken:

- `Minimum` – Die Mindestanzahl der Schreibkapazitätseinheiten, die von einer einzelnen Anforderung an die Tabelle oder den Index verbraucht werden.
- `Maximum` – Die Anzahl der Schreibkapazitätseinheiten, die von einer einzelnen Anforderung an die Tabelle oder den Index verbraucht werden.
- `Average` – Die durchschnittliche Schreibkapazität pro Anforderung.

  > **Note**
  >
  > Der `Average`-Wert wird von Perioden der Inaktivität beeinflusst, in denen der Stichprobenwert Null ist.

- `Sum` – Die verbrauchten Schreibkapazitätseinheiten. Die nützlichste Statistik für diese Metrik ist `ConsumedWriteCapacityUnits`.
- `SampleCount` – Stellt die Häufigkeit dar, mit der die Metrik ausgegeben wird. Selbst bei Tabellen ohne Datenverkehr wird der Wert `SampleCount` regelmäßig ausgegeben, wobei die Stichprobenwerte immer null betragen.

  > **Note**
  >
  > Der `SampleCount`-Wert wird von Perioden der Inaktivität beeinflusst, in denen der Stichprobenwert Null ist.

FailedToReplicateRecordCount

Die Anzahl der Datensätze, die DynamoDB nicht in Ihren Kinesis-Datenstrom replizieren konnte.

Einheiten: `Count`

Maße: `TableName`, `DelegatedOperation`

Gültige Statistiken:

- `Sum`

MaxProvisionedTableReadCapacityUtilization

Der Prozentsatz der bereitgestellten Lesekapazität, die von der höchsten bereitgestellten Lesetabelle oder dem globalen sekundären Index eines Kontos genutzt wird.

Einheiten: `Percent`

Gültige Statistiken:

- `Maximum` – Der maximale Prozentsatz der bereitgestellten Lesekapazitätseinheiten, die von der höchsten bereitgestellten Lesetabelle oder dem globalen sekundären Index eines Kontos genutzt werden.
- `Minimum` – Der minimale Prozentsatz der bereitgestellten Lesekapazitätseinheiten, die von der höchsten bereitgestellten Lesetabelle oder dem globalen sekundären Index eines Kontos genutzt werden.
- `Average` – Der durchschnittliche Prozentsatz der bereitgestellten Lesekapazitätseinheiten, die von der höchsten bereitgestellten Lesetabelle oder dem globalen sekundären Index des Kontos verwendet werden. Die Metrik wird für Intervallen von fünf Minuten veröffentlicht. Wenn Sie die bereitgestellten Lesekapazitätseinheiten schnell anpassen, spiegelt diese Statistik möglicherweise nicht den tatsächlichen Durchschnitt wider.

MaxProvisionedTableWriteCapacityUtilization

Der Prozentsatz der bereitgestellten Schreibkapazität, die von der höchsten bereitgestellten Schreibtabelle oder dem globalen sekundären Index eines Kontos genutzt wird.

Einheiten: `Percent`

Gültige Statistiken:

- `Maximum` – Der maximale Prozentsatz der bereitgestellten Schreibkapazitätseinheiten, die von der höchsten bereitgestellten Schreibtabelle oder dem globalen sekundären Index eines Kontos verwendet werden.
- `Minimum` – Der Mindestprozentsatz der bereitgestellten Schreibkapazitätseinheiten, die von der höchsten bereitgestellten Schreibtabelle oder dem globalen sekundären Index eines Kontos verwendet werden.
- `Average` – Der durchschnittliche Prozentsatz der bereitgestellten Schreibkapazitätseinheiten, die von der höchsten bereitgestellten Schreibtabelle oder dem globalen sekundären Index des

Kontos verwendet werden. Die Metrik wird für Intervallen von fünf Minuten veröffentlicht. Wenn Sie die bereitgestellten Schreibkapazitätseinheiten schnell anpassen, spiegelt diese Statistik möglicherweise nicht den tatsächlichen Durchschnitt wider.

OnDemandMaxReadRequestUnits

Die Anzahl der angegebenen On-Demand-Leseanforderungseinheiten für eine Tabelle oder einen globalen sekundären Index.

Wenn Sie die `OnDemandMaxReadRequestUnits` einer Tabelle sehen möchten, müssen Sie den `TableName` angeben. Zum Anzeigen von `OnDemandMaxReadRequestUnits` für einen globalen sekundären Index, müssen Sie sowohl `TableName` als auch `GlobalSecondaryIndexName` angeben.

Einheiten: Anzahl

Maße: `TableName`, `GlobalSecondaryIndexName`

Gültige Statistiken:

- `Minimum` – Die niedrigste Einstellung für On-Demand-Leseanforderungseinheiten. Wenn Sie `UpdateTable` verwenden, um die Lesekapazität zu erhöhen, zeigt diese Metrik den niedrigsten Wert der On-Demand-`ReadRequestUnits` während dieses Zeitraums an.
- `Maximum` – Die höchste Einstellung für On-Demand-Leseanforderungseinheiten. Wenn Sie `UpdateTable` verwenden, um die Lesekapazität zu verringern, zeigt diese Metrik den niedrigsten Wert der On-Demand-`ReadRequestUnits` während dieses Zeitraums an.
- `Average` – Die durchschnittlichen Einheiten für On-Demand-Leseanforderungen. Die `OnDemandMaxReadRequestUnits`-Metrik wird in Intervallen von fünf Minuten veröffentlicht. Wenn Sie die On-Demand-Lesekapazitätseinheiten schnell anpassen, spiegelt diese Statistik möglicherweise nicht den tatsächlichen Durchschnitt wider.

OnDemandMaxWriteRequestUnits

Die Anzahl der angegebenen On-Demand-Schreibanforderungseinheiten für eine Tabelle oder einen globalen sekundären Index.

Wenn Sie die `OnDemandMaxWriteRequestUnits` einer Tabelle sehen möchten, müssen Sie den `TableName` angeben. Zum Anzeigen von `OnDemandMaxWriteRequestUnits` für einen globalen

sekundären Index, müssen Sie sowohl `TableName` als auch `GlobalSecondaryIndexName`. angeben.

Einheiten: `Count`

Maße: `TableName`, `GlobalSecondaryIndexName`

Gültige Statistiken:

- `Minimum` – Die niedrigste Einstellung für On-Demand-Schreibanforderungseinheiten. Wenn Sie `UpdateTable` verwenden, um die Schreibkapazität zu erhöhen, zeigt diese Metrik den niedrigsten Wert der On-Demand-`WriteRequestUnits` während dieses Zeitraums an.
- `Maximum` – Die höchste Einstellung für On-Demand-Leseanforderungseinheiten. Wenn Sie `UpdateTable` verwenden, um die Schreibkapazität zu verringern, zeigt diese Metrik den niedrigsten Wert der On-Demand-`WriteRequestUnits` während dieses Zeitraums an.
- `Average` – Die durchschnittlichen Einheiten für On-Demand-Schreibanforderungen. Die `OnDemandMaxWriteRequestUnits`-Metrik wird in Intervallen von fünf Minuten veröffentlicht. Wenn Sie die bereitgestellten Schreibkapazitätseinheiten schnell anpassen, spiegelt diese Statistik möglicherweise nicht den tatsächlichen Durchschnitt wider.

OnlineIndexConsumedWriteCapacity

Es wird erwartet, dass diese Metrik während der Indexerstellung 0 anzeigt. Diese Metrik zeigte zuvor die Anzahl der Schreibkapazitätseinheiten, die beim Hinzufügen eines neuen globalen sekundären Indexes zu einer Tabelle verbraucht wurden.

Einheiten: `Count`

Maße: `TableName, GlobalSecondaryIndexName`

Gültige Statistiken:

- `Minimum`
- `Maximum`
- `Average`
- `SampleCount`
- `Sum`

OnlineIndexPercentageProgress

Der Prozentsatz der Fertigstellung, wenn ein neuer globaler sekundärer Index zu einer Tabelle hinzugefügt wird. DynamoDB muss zunächst Ressourcen für den neuen Index zuweisen und dann Attribute aus der Tabelle in den Index ausfüllen. Dieser Vorgang kann für große Tabellen eine lange Zeit dauern. Sie sollten diese Statistik überwachen, um den relativen Fortschritt anzuzeigen, während DynamoDB den Index erstellt.

Einheiten: `Count`

Maße: `TableName, GlobalSecondaryIndexName`

Gültige Statistiken:

- `Minimum`
- `Maximum`
- `Average`
- `SampleCount`
- `Sum`

OnlineIndexThrottleEvents

Es wird erwartet, dass diese Metrik bei der Indexerstellung 0 anzeigt. Diese Metrik zeigte zuvor die Anzahl von Schreibdrosselungsereignissen, die beim Hinzufügen eines neuen globalen sekundären Indexes zu einer Tabelle auftreten.

Einheiten: `Count`

Maße: `TableName, GlobalSecondaryIndexName`

Gültige Statistiken:

- `Minimum`
- `Maximum`
- `Average`
- `SampleCount`
- `Sum`

PendingReplicationCount

Metrik für Version 2017.11.29 (Legacy) der globalen Tabellen (nur globale Tabellen). Die Anzahl von Elementaktualisierungen, die zwar in eine Replikattabelle, aber noch nicht in ein anderes Replikat der globalen Tabelle geschrieben wurden.

Einheiten: `Count`

Maße: `TableName, ReceivingRegion`

Gültige Statistiken:

- `Average`
- `Sample Count`
- `Sum`

ProvisionedReadCapacityUnits

Die Anzahl der bereitgestellten Lesekapazitätseinheiten für eine Tabelle oder einen globalen sekundären Index. Die `TableName`-Dimension gibt `ProvisionedReadCapacityUnits` für die Tabelle, aber nicht für globale sekundäre Indizes zurück. Zum Anzeigen von `ProvisionedReadCapacityUnits` für einen globalen sekundären Index, müssen Sie sowohl `TableName` als auch `GlobalSecondaryIndexName` angeben.

Einheiten: `Count`

Maße: `TableName`, `GlobalSecondaryIndexName`

Gültige Statistiken:

- `Minimum` – Die niedrigste Einstellung für bereitgestellte Lesekapazität. Wenn Sie `UpdateTable` verwenden, um die Lesekapazität zu erhöhen, zeigt diese Metrik den niedrigsten Wert der bereitgestellten `ReadCapacityUnits` während dieses Zeitraums an.
- `Maximum` – Die höchste Einstellung für bereitgestellte Lesekapazität. Wenn Sie `UpdateTable` verwenden, um die Lesekapazität zu verringern, zeigt diese Metrik den höchsten Wert der bereitgestellten `ReadCapacityUnits` während dieses Zeitraums an.
- `Average` – Die durchschnittliche bereitgestellte Lesekapazität. Die `ProvisionedReadCapacityUnits`-Metrik wird in Intervallen von fünf Minuten veröffentlicht. Wenn Sie die bereitgestellten Lesekapazitätseinheiten schnell anpassen, spiegelt diese Statistik möglicherweise nicht den tatsächlichen Durchschnitt wider.

ProvisionedWriteCapacityUnits

Die Anzahl der bereitgestellten Schreibkapazitätseinheiten für eine Tabelle oder einen globalen sekundären Index.

Die `TableName`-Dimension gibt `ProvisionedWriteCapacityUnits` für die Tabelle, aber nicht für globale sekundäre Indizes zurück. Zum Anzeigen von `ProvisionedWriteCapacityUnits` für einen globalen sekundären Index, müssen Sie sowohl `TableName` als auch `GlobalSecondaryIndexName`. angeben.

Einheiten: `Count`

Maße: `TableName`, `GlobalSecondaryIndexName`

Gültige Statistiken:

- `Minimum` – Die niedrigste Einstellung für bereitgestellte Schreibkapazität. Wenn Sie `UpdateTable` verwenden, um die Schreibkapazität zu erhöhen, zeigt diese Metrik den niedrigsten Wert der bereitgestellten `WriteCapacityUnits` während dieses Zeitraums an.
- `Maximum` – Die höchste Einstellung für bereitgestellte Schreibkapazität. Wenn Sie `UpdateTable` verwenden, um die Schreibkapazität zu verringern, zeigt diese Metrik den höchsten Wert der bereitgestellten `WriteCapacityUnits` während dieses Zeitraums an.
- `Average` – Die durchschnittliche bereitgestellte Schreibkapazität. Die `ProvisionedWriteCapacityUnits`-Metrik wird in Intervallen von fünf Minuten veröffentlicht. Wenn Sie die bereitgestellten Schreibkapazitätseinheiten schnell anpassen, spiegelt diese Statistik möglicherweise nicht den tatsächlichen Durchschnitt wider.

ReadAccountLimitThrottleEvents

Die Anzahl der Leseanforderungen, die aufgrund von Kontobeschränkungen gedrosselt wurden.

Einheiten: `Count`

Maße: `TableName`, `GlobalSecondaryIndexName`

Gültige Statistiken:

- `Sum` – Gesamtzahl der gedrosselten Ereignisse.
- `SampleCount` – Anzahl der Drosselungen.
- `Minimum` – Mindestanzahl gedrosselter Ereignisse in einer bestimmten Stichprobe.

- `Maximum` – Maximale Anzahl gedrosselter Ereignisse in einer bestimmten Stichprobe.

ReadKeyRangeThroughputThrottleEvents

Die Anzahl der Leseanforderungen, die aufgrund von Partitionsbeschränkungen gedrosselt wurden.

Einheiten: `Count`

Maße: `TableName`, `GlobalSecondaryIndexName`

Gültige Statistiken:

- `Sum` – Gesamtzahl der gedrosselten Ereignisse.
- `SampleCount` – Anzahl der Drosselungen.
- `Minimum` – Mindestanzahl gedrosselter Ereignisse in einer bestimmten Stichprobe.
- `Maximum` – Maximale Anzahl gedrosselter Ereignisse in einer bestimmten Stichprobe.

ReadMaxOnDemandThroughputThrottleEvents

Die Anzahl der Leseanforderungen, die aufgrund des maximalen On-Demand-Durchsatzes gedrosselt wurden.

Einheiten: `Count`

Maße: `TableName`, `GlobalSecondaryIndexName`

Gültige Statistiken:

- `Sum` – Gesamtzahl der gedrosselten Ereignisse.
- `SampleCount` – Anzahl der Drosselungen.
- `Minimum` – Mindestanzahl gedrosselter Ereignisse in einer bestimmten Stichprobe.
- `Maximum` – Maximale Anzahl gedrosselter Ereignisse in einer bestimmten Stichprobe.

ReadProvisionedThroughputThrottleEvents

Die Anzahl der Leseanforderungen, die aufgrund von Beschränkungen des bereitgestellten Durchsatzes gedrosselt wurden.

Einheiten: `Count`

Maße: `TableName`, `GlobalSecondaryIndexName`

Gültige Statistiken:

- `Sum` – Gesamtzahl der gedrosselten Ereignisse.
- `SampleCount` – Anzahl der Drosselungen.
- `Minimum` – Mindestanzahl gedrosselter Ereignisse in einer bestimmten Stichprobe.
- `Maximum` – Maximale Anzahl gedrosselter Ereignisse in einer bestimmten Stichprobe.

ReadThrottleEvents

Anforderungen an DynamoDB, die die bereitgestellten Lesekapazitätseinheiten für eine Tabelle oder einen globalen sekundären Index überschreiten.

Eine einzelne Anforderung kann zu mehreren Ereignissen führen. Ein `BatchGetItem`, das 10 Elemente liest, wird beispielsweise als 10 `GetItem`-Ereignisse verarbeitet. Für jedes Ereignis wird `ReadThrottleEvents` um eins erhöht, wenn dieses Ereignis gedrosselt wird. Die `ThrottledRequests`-Metrik für die gesamte `BatchGetItem` wird nicht erhöht, es sei denn, alle 10 der `GetItem`-Ereignisse werden gedrosselt.

Die `TableName`-Dimension gibt `ReadThrottleEvents` für die Tabelle, aber nicht für globale sekundäre Indizes zurück. Zum Anzeigen von `ReadThrottleEvents` für einen globalen sekundären Index, müssen Sie sowohl `TableName` als auch `GlobalSecondaryIndexName`. angeben.

Einheiten: `Count`

Maße: `TableName`, `GlobalSecondaryIndexName`

Gültige Statistiken:

- `SampleCount`
- `Sum`

ReplicationLatency

(Diese Metrik gilt für globale DynamoDB-Tabellen.) Die verstrichene Zeit zwischen der Anzeige eines aktualisierten Elements im DynamoDB-Stream für eine Replikattabelle und der Anzeige dieses Elements in einem anderen Replikat der globalen Tabelle.

Einheiten: `Milliseconds`

Maße: `TableName, ReceivingRegion`

Gültige Statistiken:

- `Average`
- `Minimum`
- `Maximum`

ReturnedBytes

Die Anzahl der Bytes, die von `GetRecords`-Vorgängen (Amazon DynamoDB Streams) im angegebenen Zeitraum zurückgegeben werden.

Einheiten: `Bytes`

Maße: `Operation, StreamLabel, TableName`

Gültige Statistiken:

- `Minimum`
- `Maximum`
- `Average`
- `SampleCount`
- `Sum`

ReturnedItemCount

Die Anzahl der Elemente, die von `Query`-, `Scan`- oder `ExecuteStatement`-(bestimmten)-Operationen während des angegebenen Zeitraums zurückgegeben werden.

Anzahl der Elemente, die zurückgegeben wurden ist nicht unbedingt dieselbe wie die Anzahl der ausgewerteten Elemente. Angenommen, Sie haben einen `Scan` für eine Tabelle oder einen Index mit 100 Elementen angefordert, aber eine `FilterExpression` angegeben, die die Ergebnisse so eingegrenzt hat, dass nur 15 Elemente zurückgegeben wurden. In diesem Fall wird die Antwort von `Scan` eine `ScanCount` von 100 und eine `Count` von 15 zurückgegebenen Artikeln beinhalten.

Einheiten: `Count`

Maße: `TableName, Operation`

Gültige Statistiken:

- `Minimum`
- `Maximum`
- `Average`
- `SampleCount`
- `Sum`

ReturnedRecordsCount

Die Anzahl der Stream-Datensätze, die von `GetRecords`-Operationen (Amazon DynamoDB Streams) während des angegebenen Zeitraums zurückgegeben wurden.

Einheiten: `Count`

Maße: `Operation, StreamLabel, TableName`

Gültige Statistiken:

- `Minimum`
- `Maximum`
- `Average`
- `SampleCount`
- `Sum`

SuccessfulRequestLatency

Die Latenz erfolgreicher Anforderungen an DynamoDB oder Amazon DynamoDB Streams im angegebenen Zeitraum. `SuccessfulRequestLatency` kann zwei verschiedene Arten von Informationen bereitstellen:

- Die verstrichene Zeit für erfolgreiche Anfragen (`Minimum`, `Maximum`, `Sum`, `Average` oder `Percentile`).

- Die Anzahl erfolgreicher Anforderungen (`SampleCount`).

`SuccessfulRequestLatency` reflektiert Aktivitäten nur innerhalb von DynamoDB- oder Amazon-DynamoDB-Streams und berücksichtigt keine Netzwerklatenz oder clientseitige Aktivität.

> **Note**
>
> Um benutzerdefinierte Perzentilwerte (z. B. p99,9) zu analysieren, können Sie das gewünschte Perzentil (z. B. p99,9) manuell in das Feld für die Metrikstatistik eingeben. CloudWatch So können Sie Latenzverteilungen auswerten, die über die in der Dropdown-Liste aufgeführten Standardperzentilwerte hinausgehen.

Einheiten: `Milliseconds`

Maße: `TableName, Operation, StreamLabel`

Gültige Statistiken:

- `Minimum`
- `Maximum`
- `Sum`
- `Average`
- `Percentile`
- `SampleCount`

SystemErrors

Die Anforderungen an DynamoDB oder Amazon DynamoDB Streams, die während des angegebenen Zeitraums einen HTTP 500-Statuscode generieren. Ein HTTP 500 weist normalerweise auf einen internen Dienstfehler hin.

> **Note**
>
> Wenn DynamoDB einen Systemfehler (HTTP 500) zurückgibt, führen die meisten AWS SDKs automatisch eine konfigurierbare Anzahl von Wiederholungen durch. Wenn das Problem bei einem erneuten Versuchs behoben wird, wird die Anwendung fortgesetzt, ohne dass der

Fehler auftritt, und Sie stellen möglicherweise eine erhöhte, vom Client wahrgenommene Latenz fest. Wenn der Fehler nach allen Wiederholungen weiterhin besteht, überträgt er sich auf den Anwendungscode.

Einheiten: `Count`

Maße: `TableName, Operation`

Gültige Statistiken:

- `Sum`
- `SampleCount`

TimeToLiveDeletedItemCount

Die Anzahl der Elemente, die während des angegebenen Zeitraums von Time to Live (TTL) gelöscht wurden. Diese Metrik hilft Ihnen, die Rate der TTL-Löschungen in Ihrer Tabelle zu überwachen.

Einheiten: `Count`

Abmessungen: TableName

Gültige Statistiken:

- `Sum`

ThrottledPutRecordCount

Die Anzahl der Datensätze, die aufgrund unzureichender Kinesis-Data-Streams-Kapazität von Ihrem Kinesis-Datenstrom gedrosselt wurden.

Einheiten: `Count`

Abmessungen: TableName, DelegatedOperation

Gültige Statistiken:

- `Minimum`
- `Maximum`

- `Average`
- `SampleCount`

ThrottledRequests

Anforderungen an DynamoDB, die die bereitgestellten Durchsatzgrenzen für eine Ressource (z. B. eine Tabelle oder einen Index) überschreiten.

`ThrottledRequests` wird um eins erhöht, wenn ein Ereignis innerhalb einer Anforderung die Grenze eines bereitgestellten Durchsatzes überschreitet. Wenn Sie beispielsweise ein Element in einer Tabelle mit globalen sekundären Indizes aktualisieren, gibt es mehrere Ereignisse – einen Schreibvorgang in die Tabelle und einen Schreibvorgang auf jeden Index. Wenn eines oder mehrere dieser Ereignisse gedrosselt werden, dann wird `ThrottledRequests` um eins erhöht.

> **Note**
>
> In einer Batchanforderung (`BatchGetItem` oder `BatchWriteItem`), wird `ThrottledRequests` nur inkrementiert, wennJEDE-Anforderung im Batch gedrosselt wird. Wenn eine einzelne Anforderung innerhalb des Batches gedrosselt wird, wird eine der folgenden Metriken inkrementiert:
>
> - `ReadThrottleEvents` – Für ein gedrosseltes `GetItem`-Ereignis innerhalb `BatchGetItem`.
> - `WriteThrottleEvents` – Für ein gedrosseltes `PutItem`- oder `DeleteItem`-Ereignis innerhalb `BatchWriteItem`.

Um zu erfahren, durch welches Ereignis eine Anforderung gedrosselt wird, vergleichen Sie `ThrottledRequests` mit den `ReadThrottleEvents` und `WriteThrottleEvents` für die Tabelle und ihre Indizes.

> **Note**
>
> Eine gedrosselte Anforderung führt zu einem HTTP 400-Statuscode. All diese Ereignisse spiegeln sich in der `ThrottledRequests`-Metrik, aber nicht in der `UserErrors`-Metrik

Einheiten: `Count`

Maße: `TableName, Operation`

Gültige Statistiken:

- `Sum`
- `SampleCount`

TransactionConflict

Zurückgewiesene Anforderungen auf Artikelebene aufgrund von Transaktionskonflikten zwischen Hintergrundanforderungen für dieselben Artikel. Weitere Informationen finden Sie unter Handhabung von Transaktionskonflikten in DynamoDB.

Einheiten: `Count`

Maße: `TableName`

Gültige Statistiken:

- `Sum` – Die Anzahl der abgelehnten Anforderungen auf Artikelebene aufgrund von Transaktionskonflikten.

  > **Note**
  >
  > Wenn mehrere Anforderungen auf Elementebene innerhalb eines Aufrufs von `TransactWriteItems` oder `TransactGetItems` abgelehnt wurden, wird `Sum` für jede Elementebene um eins erhöht und `Put`, `Update`, `Delete`, oder `Get` angefordert.

- `SampleCount` – Die Anzahl der abgelehnten Anforderungen aufgrund von Transaktionskonflikten.

  > **Note**
  >
  > Wenn mehrere Anforderungen auf Elementebene innerhalb eines Aufrufs von `TransactWriteItems` oder `TransactGetItems` abgelehnt werden, wird `SampleCount` nur um eins erhöht.

- `Min` – Die Mindestanzahl abgelehnter Anforderungen auf Elementebene innerhalb eines Aufrufs von `TransactWriteItems`, `TransactGetItems`, `PutItem`, `UpdateItem`, oder `DeleteItem`.
- `Max` – Die maximale Anzahl abgelehnter Anforderungen auf Elementebene innerhalb eines Aufrufs von `TransactWriteItems`, `TransactGetItems`, `PutItem`, `UpdateItem`, oder `DeleteItem`.

- `Average` – Die durchschnittliche Anzahl der abgelehnten Anforderungen auf Elementebene innerhalb eines Aufrufs von `TransactWriteItems`, `TransactGetItems`, `PutItem`, `UpdateItem`, oder `DeleteItem`.

UserErrors

Anforderungen an DynamoDB oder Amazon DynamoDB Streams, die während des angegebenen Zeitraums einen HTTP 400-Statuscode generieren. Ein HTTP 400 weist normalerweise auf einen clientseitigen Fehler hin, z. B. eine ungültige Kombination von Parametern, einen Versuch, eine nicht vorhandene Tabelle zu aktualisieren oder eine falsche Anforderungssignatur.

Beispiele für Ausnahmen, die Metriken im Zusammenhang mit `UserErrors` protokollieren:

- `ResourceNotFoundException`
- `ValidationException`
- `TransactionConflict`

All diese Ereignisse spiegeln sich in der `UserErrors`-Metrik, mit Ausnahme der folgenden Elemente:

- ProvisionedThroughputExceededException— Sehen Sie sich die `ThrottledRequests` Metrik in diesem Abschnitt an.
- ConditionalCheckFailedException— Sehen Sie sich die `ConditionalCheckFailedRequests` Metrik in diesem Abschnitt an.

`UserErrors`stellt die Summe der HTTP 400-Fehler für DynamoDB- oder Amazon DynamoDB Streams-Anfragen für die aktuelle AWS Region und das aktuelle Konto dar. AWS

Einheiten: `Count`

Gültige Statistiken:

- `Sum`
- `SampleCount`

WriteAccountLimitThrottleEvents

Die Anzahl der Schreibanforderungen, die aufgrund von Kontobeschränkungen gedrosselt wurden.

Einheiten: `Count`

Maße: `TableName`

Gültige Statistiken:

- `Sum` – Gesamtzahl der gedrosselten Ereignisse.
- `SampleCount` – Anzahl der Drosselungen.
- `Minimum` – Mindestanzahl gedrosselter Ereignisse in einer bestimmten Stichprobe.
- `Maximum` – Maximale Anzahl gedrosselter Ereignisse in einer bestimmten Stichprobe.

WriteKeyRangeThroughputThrottleEvents

Die Anzahl der Schreibanforderungen, die aufgrund von Partitionsbeschränkungen gedrosselt wurden.

Einheiten: `Count`

Maße: `TableName`, `GlobalSecondaryIndexName`

Gültige Statistiken:

- `Sum` – Gesamtzahl der gedrosselten Ereignisse.
- `SampleCount` – Anzahl der Drosselungen.
- `Minimum` – Mindestanzahl gedrosselter Ereignisse in einer bestimmten Stichprobe.
- `Maximum` – Maximale Anzahl gedrosselter Ereignisse in einer bestimmten Stichprobe.

WriteMaxOnDemandThroughputThrottleEvents

Die Anzahl der Schreibanforderungen, die aufgrund des maximalen On-Demand-Durchsatzes gedrosselt wurden.

Einheiten: `Count`

Maße: `TableName`, `GlobalSecondaryIndexName`

Gültige Statistiken:

- `Sum` – Gesamtzahl der gedrosselten Ereignisse.
- `SampleCount` – Anzahl der Drosselungen.
- `Minimum` – Mindestanzahl gedrosselter Ereignisse in einer bestimmten Stichprobe.
- `Maximum` – Maximale Anzahl gedrosselter Ereignisse in einer bestimmten Stichprobe.

WriteProvisionedThroughputThrottleEvents

Die Anzahl der Schreibanforderungen, die aufgrund von Beschränkungen des bereitgestellten Durchsatzes gedrosselt wurden.

Einheiten: `Count`

Maße: `TableName`, `GlobalSecondaryIndexName`

Gültige Statistiken:

- `Sum` – Gesamtzahl der gedrosselten Ereignisse.
- `SampleCount` – Anzahl der Drosselungen.
- `Minimum` – Mindestanzahl gedrosselter Ereignisse in einer bestimmten Stichprobe.
- `Maximum` – Maximale Anzahl gedrosselter Ereignisse in einer bestimmten Stichprobe.

WriteThrottleEvents

Anforderungen an DynamoDB, die die bereitgestellten Schreibkapazitätseinheiten für eine Tabelle oder einen globalen sekundären Index überschreiten.

Eine einzelne Anforderung kann zu mehreren Ereignissen führen. Zum Beispiel, eine `PutItem`-Anforderung für eine Tabelle mit drei globalen sekundären Indexe würde zu vier Ereignissen führen – die Tabelle schreibt und jeder der drei Indexe schreibt. Für jedes Ereignis wird die `WriteThrottleEvents`-Metrik um eins erhöht, wenn dieses Ereignis gedrosselt wird. Für einzelne `PutItem`-Anfragen wird `ThrottledRequests`, wenn eines der Ereignisse gedrosselt wird, ebenfalls um eins erhöht. Für `BatchWriteItem` wird die `ThrottledRequests`-Metrik für die gesamte `BatchWriteItem` nicht inkrementiert, es sei denn, alle `PutItem`- oder `DeleteItem`-Ereignisse werden gedrosselt.

Die `TableName`-Dimension gibt `WriteThrottleEvents` für die Tabelle, aber nicht für globale sekundäre Indizes zurück. Zum Anzeigen von `WriteThrottleEvents` für einen globalen

sekundären Index, müssen Sie sowohl `TableName` als auch `GlobalSecondaryIndexName`. angeben.

Einheiten: `Count`

Maße: `TableName, GlobalSecondaryIndexName`

Gültige Statistiken:

- `Sum`
- `SampleCount`

Nutzungsmetriken

Mit den Nutzungsmetriken in CloudWatch können Sie die Nutzung proaktiv verwalten, indem Sie Metriken in der CloudWatch Konsole visualisieren, benutzerdefinierte Dashboards erstellen, Änderungen in der Aktivität mithilfe von CloudWatch Anomalieerkennung erkennen und Alarme konfigurieren, die Sie benachrichtigen, wenn sich die Nutzung einem Schwellenwert nähert.

DynamoDB integriert diese Nutzungsmetriken auch in Service Quotas. Sie können CloudWatch damit die Nutzung Ihrer Servicekontingenten durch Ihr Konto verwalten. Weitere Informationen finden Sie unter Visualisierung Ihrer Service Quotas und Einstellung von Alarmen.

Liste der verfügbaren Nutzungsmetriken

- AccountProvisionedWriteCapacityUnits
- AccountProvisionedReadCapacityUnits
- TableCount

AccountProvisionedWriteCapacityUnits

Die Summe der bereitgestellten Schreibkapazitätseinheiten für alle Tabellen und globalen sekundären Indizes für ein Konto.

Einheiten: `Count`

Gültige Statistiken:

- `Minimum` – Die niedrigste Anzahl von bereitgestellten Schreibkapazitätseinheiten während eines Zeitraums

- `Maximum` – Die höchste Anzahl von bereitgestellten Schreibkapazitätseinheiten während eines Zeitraums
- `Average` – Die durchschnittliche Anzahl von bereitgestellten Schreibkapazitätseinheiten während eines Zeitraums

Diese Metrik wird in Intervallen von fünf Minuten veröffentlicht. Wenn Sie die bereitgestellten Schreibkapazitätseinheiten schnell anpassen, spiegelt diese Statistik möglicherweise nicht den tatsächlichen Durchschnitt wider.

AccountProvisionedReadCapacityUnits

Die Summe der bereitgestellten Lesekapazitätseinheiten für alle Tabellen und globalen sekundären Indizes für ein Konto.

Einheiten: `Count`

Gültige Statistiken:

- `Minimum` – Die niedrigste Anzahl von bereitgestellten Lesekapazitätseinheiten während eines Zeitraums
- `Maximum` – Die höchste Anzahl von bereitgestellten Lesekapazitätseinheiten während eines Zeitraums
- `Average` – Die durchschnittliche Anzahl von bereitgestellten Lesekapazitätseinheiten während eines Zeitraums

Diese Metrik wird in Intervallen von fünf Minuten veröffentlicht. Wenn Sie die bereitgestellten Lesekapazitätseinheiten schnell anpassen, spiegelt diese Statistik möglicherweise nicht den tatsächlichen Durchschnitt wider.

TableCount

Die Anzahl der aktiven Tabellen eines Kontos

Einheiten: `Count`

Gültige Statistiken:

- `Minimum` – Die niedrigste Anzahl von Tabellen während eines Zeitraums
- `Maximum` – Die höchste Anzahl von Tabellen während eines Zeitraums

- `Average` – Die durchschnittliche Anzahl von Tabellen während eines Zeitraums

FaultInjectionServiceInducedErrors

Die Anfragen an DynamoDB, die während des angegebenen Zeitraums und während des Nachholvorgangs als Ergebnis eines Experiments einen simulierten HTTP 500-Statuscode generieren. AWS FIS

Einheiten: `Count`

Maße: `TableName`, `Operation`

Gültige Statistiken:

- `Sum`
- `SampleCount`

Metriken und Dimensionen für DynamoDB

Die Metriken für DynamoDB qualifizieren sich über die Werte für das Konto, den Tabellennamen, den globalen sekundären Indexnamen oder den Vorgang. Sie können die CloudWatch Konsole verwenden, um DynamoDB-Daten entlang einer der Dimensionen in der Tabelle unten abzurufen.

Liste der verfügbaren Dimensionen

- DelegatedOperation
- GlobalSecondaryIndexName
- Operation
- OperationType
- Verb
- ReceivingRegion
- StreamLabel
- TableName

DelegatedOperation

Diese Dimension schränkt die Daten auf Vorgänge ein, die DynamoDB in Ihrem Auftrag ausführt. Die folgenden Operationen werden erfasst::

- Ändern Sie die Datenerfassung für Kinesis Data Streams

GlobalSecondaryIndexName

Diese Dimension schränkt die Daten auf einen globalen sekundären Index einer Tabelle ein. Wenn Sie `GlobalSecondaryIndexName` angeben, müssen Sie auch `TableName` angeben.

Operation

Diese Dimension schränkt die Daten auf einen der folgenden DynamoDB Vorgänge ein:

- `PutItem`
- `DeleteItem`
- `UpdateItem`
- `GetItem`
- `BatchGetItem`
- `Scan`
- `Query`
- `BatchWriteItem`
- `TransactWriteItems`
- `TransactGetItems`
- `ExecuteTransaction`
- `BatchExecuteStatement`
- `ExecuteStatement`

Darüber hinaus können Sie die Daten auf den folgenden Amazon-DynamoDB-Streams-Vorgang beschränken:

- `GetRecords`

OperationType

Diese Dimension schränkt die Daten auf einen der folgenden Operationstypen:

- `Read`

- `Write`

Diese Dimension wird für `ExecuteTransaction`- und `BatchExecuteStatement`-Anforderungen weggelassen.

Verb

Diese Dimension schränkt die Daten auf eines der folgenden DynamoDB-PartiQL-Verben ein:

- Einfügen: `PartiQLInsert`
- Auswählen: `PartiQLSelect`
- Aktualisieren: `PartiQLUpdate`
- Löschen: `PartiQLDelete`

Diese Dimension wird für die `ExecuteStatement`-Operation verwendet.

ReceivingRegion

Diese Dimension beschränkt die Daten auf eine bestimmte AWS Region. Sie wird mit Metriken verwendet, die aus Replikattabellen in einer globalen DynamoDB-Tabelle stammen.

StreamLabel

Diese Dimension schränkt die Daten auf einen spezifischen Stream-Label ein. Es wird mit Metriken verwendet, die aus Amazon DynamoDB Streams`GetRecords`-Operationen stammen.

TableName

Diese Dimension schränkt die Daten auf einen spezifische Tabelle ein. Dieser Wert kann ein beliebiger Tabellenname in der aktuellen Region und im aktuellen AWS Konto sein.

CloudWatch Alarme in DynamoDB erstellen

Ein CloudWatch Alarm überwacht eine einzelne Metrik über einen bestimmten Zeitraum und führt eine oder mehrere bestimmte Aktionen aus, die auf dem Wert der Metrik im Verhältnis zu einem Schwellenwert im Laufe der Zeit basieren. Die Aktion ist eine Benachrichtigung, die an ein Amazon-SNS-Thema oder eine Auto Scaling-Richtlinie gesendet wird. Sie können auch Alarme zu Dashboards hinzufügen, um Ihre AWS Ressourcen und Anwendungen in mehreren Regionen zu überwachen und Benachrichtigungen zu erhalten. Die Anzahl der Alarme, die Sie erstellen

können, ist unbegrenzt. CloudWatch Alarme rufen keine Aktionen auf, nur weil sie sich in einem bestimmten Zustand befinden. Der Status muss sich geändert haben und für eine bestimmte Anzahl von Zeiträumen beibehalten worden sein. Eine Liste der empfohlenen DynamoDB-Alarme finden Sie unter Empfohlene Alarme.

> **Note**
>
> Sie müssen bei der Erstellung Ihres CloudWatch Alarms alle erforderlichen Dimensionen angeben, da Metriken für eine fehlende Dimension nicht aggregiert CloudWatch werden. Das Erstellen eines CloudWatch Alarms mit einer fehlenden Dimension führt bei der Erstellung des Alarms nicht zu einem Fehler.

Angenommen, Sie haben eine bereitgestellte Tabelle mit fünf Lesekapazitätseinheiten. Sie möchten benachrichtigt werden, bevor Sie die gesamte bereitgestellte Lesekapazität verbrauchen. Daher entscheiden Sie sich dafür, einen CloudWatch Alarm zu erstellen, um benachrichtigt zu werden, wenn die verbrauchte Kapazität 80% der Kapazität erreicht, die Sie für die Tabelle bereitgestellt haben. Sie können Alarme in der CloudWatch Konsole oder mit dem erstellen.AWS CLI

Einen Alarm in der CloudWatch Konsole erstellen

Um einen Alarm in der CloudWatch Konsole zu erstellen

1. Melden Sie sich bei der an AWS-Managementkonsole und öffnen Sie die CloudWatch Konsole unter https://console.aws.amazon.com/cloudwatch/.
2. Wählen Sie im Navigationsbereich Alarms (Alarme) und All alarms (Alle Alarme) aus.
3. Wählen Sie Create alarm (Alarm erstellen) aus.
4. Suchen Sie die Zeile mit der Tabelle, die Sie überwachen möchten, und wählen Sie in der Spalte Metrikname den Wert **`ConsumeReadCapacityUnits`** aus. Aktivieren Sie das Kontrollkästchen neben dieser Zeile, und wählen Sie Metrik auswählen.
5. Wählen Sie unter Metrik und Bedingungen angeben unter Statistik die Option Summe aus. Wählen Sie unter Zeitraum den Wert 1 Minute aus.
6. Geben Sie unter Conditions (Bedingungen) Folgendes an:

    a. Wählen Sie für Threshold type (Schwellenwerttyp) die Option Static (Statisch) aus.

    b. Wählen Sie für Wenn **`ConsumedReadCapacityUnits`** Größer/Gleich aus und geben Sie für den Schwellenwert 240 an.

7. Wählen Sie Weiter aus.
8. Wählen Sie unter Benachrichtigung die Option **`In alarm`** aus und wählen Sie ein SNS-Thema aus, das benachrichtigt werden soll, wenn sich der Alarm im Status `ALARM` befindet.
9. Wenn Sie fertig sind, wählen Sie Weiter.
10. Geben Sie einen Namen und eine Beschreibung für den Alarm ein und wählen Sie Next (Weiter).
11. Bestätigen Sie unter Preview and create (Vorschau und erstellen), dass die Informationen und Bedingungen den Anforderungen entsprechen, und wählen Sie dann Create alarm (Alarm erstellen).

Einen Alarm erstellen in der AWS CLI

```
aws cloudwatch put-metric-alarm \
    -\-alarm-name ReadCapacityUnitsLimitAlarm \
    -\-alarm-description "Alarm when read capacity reaches 80% of my provisioned read
 capacity" \
    -\-namespace AWS/DynamoDB \
    -\-metric-name ConsumedReadCapacityUnits \
    -\-dimensions Name=TableName,Value=myTable \
    -\-statistic Sum \
    -\-threshold 240 \
    -\-comparison-operator GreaterThanOrEqualToThreshold \
    -\-period 60 \
    -\-evaluation-periods 1 \
    -\-alarm-actions arn:aws:sns:us-east-1:123456789012:capacity-alarm
```

Testen Sie den Alarm.

```
aws cloudwatch set-alarm-state -\-alarm-name ReadCapacityUnitsLimitAlarm -\-state-
reason "initializing" -\-state-value OK
```

```
aws cloudwatch set-alarm-state -\-alarm-name ReadCapacityUnitsLimitAlarm -\-state-
reason "initializing" -\-state-value ALARM
```

Weitere AWS CLI Beispiele

Das folgende Verfahren beschreibt, wie Sie benachrichtigt werden, wenn Anforderungen die bereitgestellten Durchsatzquoten einer Tabelle überschreiten.

1. Erstellen Sie ein Amazon SNS SNS-Thema`arn:aws:sns:us-east-1:123456789012:requests-exceeding-throughput`. Weitere Informationen finden Sie unter Einrichten des Amazon Simple Notification Service.
2. Erstellen Sie den Alarm.

```
aws cloudwatch put-metric-alarm \
        -\-alarm-name ReadCapacityUnitsLimitAlarm \
        -\-alarm-description "Alarm when read capacity reaches 80% of my
 provisioned read capacity" \
        -\-namespace AWS/DynamoDB \
        -\-metric-name ConsumedReadCapacityUnits \
        -\-dimensions Name=TableName,Value=myTable \
        -\-statistic Sum \
        -\-threshold 240 \
        -\-comparison-operator GreaterThanOrEqualToThreshold \
        -\-period 60 \
        -\-evaluation-periods 1 \
        -\-alarm-actions arn:aws:sns:us-east-1:123456789012:capacity-alarm
```

3. Testen Sie den Alarm.

```
aws cloudwatch set-alarm-state --alarm-name RequestsExceedingThroughputAlarm --
state-reason "initializing" --state-value OK
```

```
aws cloudwatch set-alarm-state --alarm-name RequestsExceedingThroughputAlarm --
state-reason "initializing" --state-value ALARM
```

Das folgende Verfahren beschreibt, wie Sie bei Systemfehlern benachrichtigt werden.

1. Erstellen Sie ein Amazon SNS SNS-Thema`arn:aws:sns:us-east-1:123456789012:notify-on-system-errors`. Weitere Informationen finden Sie unter Einrichten des Amazon Simple Notification Service.
2. Erstellen Sie den Alarm.

```
aws cloudwatch put-metric-alarm \
    --alarm-name SystemErrorsAlarm \
    --alarm-description "Alarm when system errors occur" \
    --namespace AWS/DynamoDB \
    --metric-name SystemErrors \
```

```
    --dimensions Name=TableName,Value=myTable
 Name=Operation,Value=aDynamoDBOperation \
    --statistic Sum \
    --threshold 0 \
    --comparison-operator GreaterThanThreshold \
    --period 60 \
    --unit Count \
    --evaluation-periods 1 \
    --treat-missing-data breaching \
    --alarm-actions arn:aws:sns:us-east-1:123456789012:notify-on-system-errors
```

3. Testen Sie den Alarm.

```
aws cloudwatch set-alarm-state --alarm-name SystemErrorsAlarm --state-reason
 "initializing" --state-value OK
```

```
aws cloudwatch set-alarm-state --alarm-name SystemErrorsAlarm --state-reason
 "initializing" --state-value ALARM
```

Protokollieren von DynamoDB-Vorgängen mithilfe von AWS CloudTrail

DynamoDB ist in einen Dienst integriert AWS CloudTrail, der eine Aufzeichnung der Aktionen bereitstellt, die von einem Benutzer, einer Rolle oder einem AWS Dienst in DynamoDB ausgeführt wurden. CloudTrail erfasst alle API-Aufrufe für DynamoDB als Ereignisse. Zu den erfassten Aufrufen gehören Aufrufe von der DynamoDB-Konsole und Code-Aufrufe an die DynamoDB-API-Operationen, die sowohl PartiQL als auch die klassische -API verwenden. Wenn Sie einen Trail erstellen, können Sie die kontinuierliche Bereitstellung von CloudTrail Ereignissen an einen Amazon S3 S3-Bucket aktivieren, einschließlich Ereignissen für DynamoDB. Wenn Sie keinen Trail konfigurieren, können Sie die neuesten Ereignisse trotzdem in der CloudTrail Konsole im Ereignisverlauf anzeigen. Anhand der von gesammelten Informationen können Sie die Anfrage CloudTrail, die an DynamoDB gestellt wurde, die IP-Adresse, von der aus die Anfrage gestellt wurde, wer die Anfrage gestellt hat, wann sie gestellt wurde, und weitere Details ermitteln.

Für eine zuverlässige Überwachung und Alarmierung können Sie CloudTrail Ereignisse auch in Amazon CloudWatch Logs integrieren. Um Ihre Analyse der DynamoDB-Serviceaktivitäten zu verbessern und Änderungen der Aktivitäten für ein AWS Konto zu identifizieren, können Sie AWS

CloudTrail Protokolle mit Amazon Athena abfragen. Beispielsweise können Sie mithilfe von Abfragen Trends ermitteln und Vorgänge nach Attributen (z. B. Quell-IP-Adresse oder Benutzer) trennen.

Weitere Informationen darüber CloudTrail, einschließlich der Konfiguration und Aktivierung, finden Sie im AWS CloudTrail Benutzerhandbuch.

Themen

- DynamoDB-Informationen in CloudTrail
- Grundlagen zu DynamoDB-Protokolldateieinträgen

DynamoDB-Informationen in CloudTrail

CloudTrail ist in Ihrem AWS Konto aktiviert, wenn Sie das Konto erstellen. Wenn unterstützte Ereignisaktivitäten in DynamoDB auftreten, wird diese Aktivität zusammen mit anderen AWS Dienstereignissen im CloudTrail Ereignisverlauf in einem Ereignis aufgezeichnet. Sie können aktuelle Ereignisse in Ihrem AWS Konto anzeigen, suchen und herunterladen. Weitere Informationen finden Sie unter Arbeiten mit dem CloudTrail Ereignisverlauf.

Für eine fortlaufende Aufzeichnung von Ereignissen in Ihrem AWS Konto, einschließlich Ereignissen für DynamoDB, erstellen Sie einen Trail. Ein Trail ermöglicht CloudTrail die Übermittlung von Protokolldateien an einen Amazon S3 S3-Bucket. Wenn Sie einen Trail in der Konsole erstellen, gilt der Trail standardmäßig für alle AWS Regionen. Der Trail protokolliert Ereignisse aus allen Regionen der AWS Partition und übermittelt die Protokolldateien an den von Ihnen angegebenen Amazon S3 S3-Bucket. Darüber hinaus können Sie andere AWS Dienste konfigurieren, um die in den CloudTrail Protokollen gesammelten Ereignisdaten weiter zu analysieren und darauf zu reagieren. Weitere Informationen finden Sie hier:

- Übersicht zum Erstellen eines Trails
- CloudTrail unterstützte Dienste und Integrationen
- Konfiguration von Amazon SNS SNS-Benachrichtigungen für CloudTrail
- Empfangen von CloudTrail Protokolldateien aus mehreren Regionen und Empfangen von CloudTrail Protokolldateien von mehreren Konten

Ereignisse auf der Kontrollebene in CloudTrail

Die folgenden API-Aktionen werden standardmäßig als Ereignisse in CloudTrail Dateien protokolliert:

Amazon-DynamoDB

- CreateBackup
- CreateGlobalTable
- CreateTable
- DeleteBackup
- DeleteTable
- DescribeBackup
- DescribeContinuousBackups
- DescribeGlobalTable
- DescribeLimits
- DescribeTable
- DescribeTimeToLive
- ListBackups
- ListTables
- ListTagsOfResource
- ListGlobalTables
- RestoreTableFromBackup
- RestoreTableToPointInTime
- TagResource
- UntagResource
- UpdateGlobalTable
- UpdateTable
- UpdateTimeToLive
- DescribeReservedCapacity
- DescribeReservedCapacityOfferings
- PurchaseReservedCapacityOfferings
- DescribeScalableTargets
- RegisterScalableTarget

DynamoDB-Streams

- DescribeStream
- ListStreams

DynamoDB Accelerator (DAX).

- CreateCluster
- CreateParameterGroup
- CreateSubnetGroup
- DecreaseReplicationFactor
- DeleteCluster
- DeleteParameterGroup
- DeleteSubnetGroup
- DescribeClusters
- DescribeDefaultParameters
- DescribeEvents
- DescribeParameterGroups
- DescribeParameters
- DescribeSubnetGroups
- IncreaseReplicationFactor
- ListTags
- RebootNode
- TagResource
- UntagResource
- UpdateCluster
- UpdateParameterGroup
- UpdateSubnetGroup

DynamoDB-Datenebenenereignisse in CloudTrail

Um die Protokollierung der folgenden API-Aktionen in CloudTrail Dateien zu aktivieren, müssen Sie die Protokollierung der API-Aktivitäten auf der Datenebene in aktivieren. CloudTrail Weitere Informationen finden Sie unter Protokollieren von Datenereignissen für Trails.

Ereignisse auf Datenebene können nach Ressourcentyp gefiltert werden, um detailliert zu steuern, welche DynamoDB-API-Aufrufe Sie selektiv protokollieren und bezahlen möchten. CloudTrail Wenn Sie beispielsweise `AWS::DynamoDB::Stream` als Ressourcentyp angeben, können Sie nur Aufrufe der DynamoDB-Streams protokollieren. APIs Bei Tabellen mit aktivierten Streams enthält das Ressourcenfeld im Datenebenenereignis sowohl `AWS::DynamoDB::Stream` als auch `AWS::DynamoDB::Table`. Wenn Sie `AWS::DynamoDB::Table` als Ressourcentyp angeben, werden standardmäßig sowohl DynamoDB-Tabellen- als auch DynamoDB-Stream-Ereignisse protokolliert. Sie können einen zusätzlichen Filter hinzufügen, um die Stream-Ereignisse auszuschließen, wenn Sie nicht möchten, dass die Stream-Ereignisse protokolliert werden. Weitere Informationen finden Sie unter DataResource in der AWS CloudTrail -API-Referenz.

Amazon-DynamoDB

- BatchExecuteStatement
- BatchGetItem
- BatchWriteItem
- DeleteItem
- ExecuteStatement
- ExecuteTransaction
- GetItem
- PutItem
- Query
- Scan
- TransactGetItems
- TransactWriteItems
- UpdateItem

> **Note**
>
> Aktionen auf der DynamoDB-Datenebene „Time to Live“ werden nicht protokolliert von CloudTrail

DynamoDB-Streams

- GetRecords
- GetShardIterator

Grundlagen zu DynamoDB-Protokolldateieinträgen

Ein Trail ist eine Konfiguration, die die Übertragung von Ereignissen als Protokolldateien an einen von Ihnen angegebenen Amazon S3 S3-Bucket ermöglicht. CloudTrail Protokolldateien enthalten einen oder mehrere Protokolleinträge. Ein Ereignis stellt eine einzelne Anforderung aus einer beliebigen Quelle dar und enthält unter anderem Informationen über die angeforderte Aktion, das Datum und die Uhrzeit der Aktion sowie über die Anforderungsparameter.

Jeder Ereignis- oder Protokolleintrag enthält Informationen zu dem Benutzer, der die Anforderung generiert hat. Die Identitätsinformationen unterstützen Sie bei der Ermittlung der folgenden Punkte:

- Ob die Anforderung mit Root- oder -Benutzeranmeldeinformationen ausgeführt wurde.
- Gibt an, ob die Anforderung mit temporären Sicherheitsanmeldeinformationen für eine Rolle oder einen Verbundbenutzer gesendet wurde.
- Ob die Anfrage von einem anderen AWS Dienst gestellt wurde.

> **Note**
>
> Nicht wichtige Attributwerte werden in den CloudTrail Protokollen von Aktionen, die die PartiQL-API verwenden, geschwärzt und erscheinen nicht in Protokollen von Aktionen, die die klassische API verwenden.

Weitere Informationen finden Sie unter CloudTrail -Element userIdentity.

Die folgenden Beispiele veranschaulichen CloudTrail Protokolle dieser Ereignistypen:

Amazon-DynamoDB

- `UpdateTable`
- `DeleteTable`
- `CreateCluster`
- `PutItem` (erfolgreich)

- UpdateItem (erfolglos)
- TransactWriteItems (erfolgreich)
- TransactWriteItems (mit TransactionCanceledException)
- ExecuteStatement
- BatchExecuteStatement

DynamoDB-Streams

- GetRecords

UpdateTable

```
{
    "Records": [
        {
            "eventVersion": "1.03",
            "userIdentity": {
                "type": "AssumedRole",
                "principalId": "AKIAIOSFODNN7EXAMPLE:bob",
                "arn": "arn:aws:sts::111122223333:assumed-role/users/bob",
                "accountId": "111122223333",
                "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
                "sessionContext": {
                    "attributes": {
                        "mfaAuthenticated": "false",
                        "creationDate": "2015-05-28T18:06:01Z"
                    },
                    "sessionIssuer": {
                        "type": "Role",
                        "principalId": "AKIAI44QH8DHBEXAMPLE",
                        "arn": "arn:aws:iam::444455556666:role/admin-role",
                        "accountId": "444455556666",
                        "userName": "bob"
                    }
                }
            },
            "eventTime": "2015-05-04T02:14:52Z",
            "eventSource": "dynamodb.amazonaws.com",
            "eventName": "UpdateTable",
            "awsRegion": "us-west-2",
```

```
            "sourceIPAddress": "192.0.2.0",
            "userAgent": "console.aws.amazon.com",
            "requestParameters": {
                "provisionedThroughput": {
                    "writeCapacityUnits": 25,
                    "readCapacityUnits": 25
                }
            },
            "responseElements": {
                "tableDescription": {
                    "tableName": "Music",
                    "attributeDefinitions": [
                        {
                            "attributeType": "S",
                            "attributeName": "Artist"
                        },
                        {
                            "attributeType": "S",
                            "attributeName": "SongTitle"
                        }
                    ],
                    "itemCount": 0,
                    "provisionedThroughput": {
                        "writeCapacityUnits": 10,
                        "numberOfDecreasesToday": 0,
                        "readCapacityUnits": 10,
                        "lastIncreaseDateTime": "May 3, 2015 11:34:14 PM"
                    },
                    "creationDateTime": "May 3, 2015 11:34:14 PM",
                    "keySchema": [
                        {
                            "attributeName": "Artist",
                            "keyType": "HASH"
                        },
                        {
                            "attributeName": "SongTitle",
                            "keyType": "RANGE"
                        }
                    ],
                    "tableStatus": "UPDATING",
                    "tableSizeBytes": 0
                }
            },
            "requestID": "AALNP0J2L244N5O15PKISJ1KUFVV4KQNSO5AEMVJF66Q9ASUAAJG",
```

```
            "eventID": "eb834e01-f168-435f-92c0-c36278378b6e",
            "eventType": "AwsApiCall",
            "apiVersion": "2012-08-10",
            "recipientAccountId": "111122223333"
        }
    ]
}
```

DeleteTable

```
{
    "Records": [
        {
            "eventVersion": "1.03",
            "userIdentity": {
                "type": "AssumedRole",
                "principalId": "AKIAIOSFODNN7EXAMPLE:bob",
                "arn": "arn:aws:sts::111122223333:assumed-role/users/bob",
                "accountId": "111122223333",
                "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
                "sessionContext": {
                    "attributes": {
                        "mfaAuthenticated": "false",
                        "creationDate": "2015-05-28T18:06:01Z"
                    },
                    "sessionIssuer": {
                        "type": "Role",
                        "principalId": "AKIAI44QH8DHBEXAMPLE",
                        "arn": "arn:aws:iam::444455556666:role/admin-role",
                        "accountId": "444455556666",
                        "userName": "bob"
                    }
                }
            },
            "eventTime": "2015-05-04T13:38:20Z",
            "eventSource": "dynamodb.amazonaws.com",
            "eventName": "DeleteTable",
            "awsRegion": "us-west-2",
            "sourceIPAddress": "192.0.2.0",
            "userAgent": "console.aws.amazon.com",
            "requestParameters": {
                "tableName": "Music"
            },
```

```
            "responseElements": {
                "tableDescription": {
                    "tableName": "Music",
                    "itemCount": 0,
                    "provisionedThroughput": {
                        "writeCapacityUnits": 25,
                        "numberOfDecreasesToday": 0,
                        "readCapacityUnits": 25
                    },
                    "tableStatus": "DELETING",
                    "tableSizeBytes": 0
                }
            },
            "requestID": "4KBNVRGD25RG1KEO9UT4V3FQDJVV4KQNSO5AEMVJF66Q9ASUAAJG",
            "eventID": "a954451c-c2fc-4561-8aea-7a30ba1fdf52",
            "eventType": "AwsApiCall",
            "apiVersion": "2012-08-10",
            "recipientAccountId": "111122223333"
        }
    ]
}
```


CreateCluster


```
{
    "Records": [
        {
            "eventVersion": "1.05",
            "userIdentity": {
                "type": "IAMUser",
                "principalId": "AKIAIOSFODNN7EXAMPLE:bob",
                "arn": "arn:aws:sts::111122223333:assumed-role/users/bob",
                "accountId": "111122223333",
                "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
                "userName": "bob"
            },
            "eventTime": "2019-12-17T23:17:34Z",
            "eventSource": "dax.amazonaws.com",
            "eventName": "CreateCluster",
            "awsRegion": "us-west-2",
            "sourceIPAddress": "192.0.2.0",
            "userAgent": "aws-cli/1.16.304 Python/3.6.9
 Linux/4.9.184-0.1.ac.235.83.329.metal1.x86_64 botocore/1.13.40",
```

```
            "requestParameters": {
                "sSESpecification": {
                    "enabled": true
                },
                "clusterName": "daxcluster",
                "nodeType": "dax.r4.large",
                "replicationFactor": 3,
                "iamRoleArn": "arn:aws:iam::111122223333:role/
DAXServiceRoleForDynamoDBAccess"
            },
            "responseElements": {
                "cluster": {
                    "securityGroups": [
                        {
                            "securityGroupIdentifier": "sg-1af6e36e",
                            "status": "active"
                        }
                    ],
                    "parameterGroup": {
                        "nodeIdsToReboot": [],
                        "parameterGroupName": "default.dax1.0",
                        "parameterApplyStatus": "in-sync"
                    },
                    "clusterDiscoveryEndpoint": {
                        "port": 8111
                    },
                    "clusterArn": "arn:aws:dax:us-west-2:111122223333:cache/
daxcluster",
                    "status": "creating",
                    "subnetGroup": "default",
                    "sSEDescription": {
                        "status": "ENABLED",
                        "kMSMasterKeyArn": "arn:aws:kms:us-
west-2:111122223333:key/764898e4-adb1-46d6-a762-e2f4225b4fc4"
                    },
                    "iamRoleArn": "arn:aws:iam::111122223333:role/
DAXServiceRoleForDynamoDBAccess",
                    "clusterName": "daxcluster",
                    "activeNodes": 0,
                    "totalNodes": 3,
                    "preferredMaintenanceWindow": "thu:13:00-thu:14:00",
                    "nodeType": "dax.r4.large"
                }
            },
```

```
            "requestID": "585adc5f-ad05-4e27-8804-70ba1315f8fd",
            "eventID": "29158945-28da-4e32-88e1-56d1b90c1a0c",
            "eventType": "AwsApiCall",
            "recipientAccountId": "111122223333"
        }
    ]
}
```

PutItem (erfolgreich)

```
{
    "Records": [
        {
            "eventVersion": "1.06",
            "userIdentity": {
                "type": "AssumedRole",
                "principalId": "AKIAIOSFODNN7EXAMPLE:bob",
                "arn": "arn:aws:sts::111122223333:assumed-role/users/bob",
                "accountId": "111122223333",
                "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
                "sessionContext": {
                    "attributes": {
                        "mfaAuthenticated": "false",
                        "creationDate": "2015-05-28T18:06:01Z"
                    },
                    "sessionIssuer": {
                        "type": "Role",
                        "principalId": "AKIAI44QH8DHBEXAMPLE",
                        "arn": "arn:aws:iam::444455556666:role/admin-role",
                        "accountId": "444455556666",
                        "userName": "bob"
                    }
                }
            },
            "eventTime": "2019-01-19T15:41:54Z",
            "eventSource": "dynamodb.amazonaws.com",
            "eventName": "PutItem",
            "awsRegion": "us-west-2",
            "sourceIPAddress": "192.0.2.0",
            "userAgent": "aws-cli/1.15.64 Python/2.7.16 Darwin/17.7.0
 botocore/1.10.63",
            "requestParameters": {
                "tableName": "Music",
```

```
                "key": {
                    "Artist": "No One You Know",
                    "SongTitle": "Scared of My Shadow"
                },
                "item": [
                    "Artist",
                    "SongTitle",
                    "AlbumTitle"
                ],
                "returnConsumedCapacity": "TOTAL"
            },
            "responseElements": null,
            "requestID": "4KBNVRGD25RG1KEO9UT4V3FQDJVV4KQNSO5AEMVJF66Q9ASUAAJG",
            "eventID": "a954451c-c2fc-4561-8aea-7a30ba1fdf52",
            "readOnly": false,
            "resources": [
                {
                    "accountId": "111122223333",
                    "type": "AWS::DynamoDB::Table",
                    "ARN": "arn:aws:dynamodb:us-west-2:123456789012:table/Music"
                }
            ],
            "eventType": "AwsApiCall",
            "apiVersion": "2012-08-10",
            "managementEvent": false,
            "recipientAccountId": "111122223333",
            "eventCategory": "Data"
        }
    ]
}
```

`UpdateItem` (erfolglos)

```
{
    "Records": [
        {
            "eventVersion": "1.07",
            "userIdentity": {
                "type": "AssumedRole",
                "principalId": "AKIAIOSFODNN7EXAMPLE:bob",
                "arn": "arn:aws:sts::111122223333:assumed-role/users/bob",
                "accountId": "111122223333",
                "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
```

```
                "sessionContext": {
                    "sessionIssuer": {
                        "type": "Role",
                        "principalId": "AKIAI44QH8DHBEXAMPLE",
                        "arn": "arn:aws:iam::444455556666:role/admin-role",
                        "accountId": "444455556666",
                        "userName": "bob"
                    },
                    "attributes": {
                        "creationDate": "2020-09-03T22:14:13Z",
                        "mfaAuthenticated": "false"
                    }
                }
            },
            "eventTime": "2020-09-03T22:27:15Z",
            "eventSource": "dynamodb.amazonaws.com",
            "eventName": "UpdateItem",
            "awsRegion": "us-west-2",
            "sourceIPAddress": "192.0.2.0",
            "userAgent": "aws-cli/1.15.64 Python/2.7.16 Darwin/17.7.0
botocore/1.10.63",
            "errorCode": "ConditionalCheckFailedException",
            "errorMessage": "The conditional request failed",
            "requestParameters": {
                "tableName": "Music",
                "key": {
                    "Artist": "No One You Know",
                    "SongTitle": "Call Me Today"
                },
                "updateExpression": "SET #Y = :y, #AT = :t",
                "expressionAttributeNames": {
                    "#Y": "Year",
                    "#AT": "AlbumTitle"
                },
                "conditionExpression": "attribute_not_exists(#Y)",
                "returnConsumedCapacity": "TOTAL"
            },
            "responseElements": null,
            "requestID": "4KBNVRGD25RG1KEO9UT4V3FQDJVV4KQNSO5AEMVJF66Q9ASUAAJG",
            "eventID": "a954451c-c2fc-4561-8aea-7a30ba1fdf52",
            "readOnly": false,
            "resources": [
                {
                    "accountId": "111122223333",
```

```
                    "type": "AWS::DynamoDB::Table",
                    "ARN": "arn:aws:dynamodb:us-west-2:123456789012:table/Music"
                }
            ],
            "eventType": "AwsApiCall",
            "apiVersion": "2012-08-10",
            "managementEvent": false,
            "recipientAccountId": "111122223333",
            "eventCategory": "Data"
        }
    ]
}
```

`TransactWriteItems` (erfolgreich)

```
{
    "Records": [
        {
            "eventVersion": "1.07",
            "userIdentity": {
                "type": "AssumedRole",
                "principalId": "AKIAIOSFODNN7EXAMPLE:bob",
                "arn": "arn:aws:sts::111122223333:assumed-role/users/bob",
                "accountId": "111122223333",
                "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
                "sessionContext": {
                    "sessionIssuer": {
                        "type": "Role",
                        "principalId": "AKIAI44QH8DHBEXAMPLE",
                        "arn": "arn:aws:iam::444455556666:role/admin-role",
                        "accountId": "444455556666",
                        "userName": "bob"
                    },
                    "attributes": {
                        "creationDate": "2020-09-03T22:14:13Z",
                        "mfaAuthenticated": "false"
                    }
                }
            },
            "eventTime": "2020-09-03T21:48:12Z",
            "eventSource": "dynamodb.amazonaws.com",
            "eventName": "TransactWriteItems",
            "awsRegion": "us-west-1",
```

```
            "sourceIPAddress": "192.0.2.0",
            "userAgent": "aws-cli/1.15.64 Python/2.7.16 Darwin/17.7.0
botocore/1.10.63",
            "requestParameters": {
                "requestItems": [
                    {
                        "operation": "Put",
                        "tableName": "Music",
                        "key": {
                            "Artist": "No One You Know",
                            "SongTitle": "Call Me Today"
                        },
                        "items": [
                            "Artist",
                            "SongTitle",
                            "AlbumTitle"
                        ],
                        "conditionExpression": "#AT = :A",
                        "expressionAttributeNames": {
                            "#AT": "AlbumTitle"
                        },
                        "returnValuesOnConditionCheckFailure": "ALL_OLD"
                    },
                    {
                        "operation": "Update",
                        "tableName": "Music",
                        "key": {
                            "Artist": "No One You Know",
                            "SongTitle": "Call Me Tomorrow"
                        },
                        "updateExpression": "SET #AT = :newval",
                        "ConditionExpression": "attribute_not_exists(Rating)",
                        "ExpressionAttributeNames": {
                            "#AT": "AlbumTitle"
                        },
                        "returnValuesOnConditionCheckFailure": "ALL_OLD"
                    },
                    {
                        "operation": "Delete",
                        "TableName": "Music",
                        "key": {
                            "Artist": "No One You Know",
                            "SongTitle": "Call Me Yesterday"
                        },
```

```
                        "conditionExpression": "#P between :lo and :hi",
                        "expressionAttributeNames": {
                            "#P": "Price"
                        },
                        "ReturnValuesOnConditionCheckFailure": "ALL_OLD"
                    },
                    {
                        "operation": "ConditionCheck",
                        "TableName": "Music",
                        "Key": {
                            "Artist": "No One You Know",
                            "SongTitle": "Call Me Now"
                        },
                        "ConditionExpression": "#P between :lo and :hi",
                        "ExpressionAttributeNames": {
                            "#P": "Price"
                        },
                        "ReturnValuesOnConditionCheckFailure": "ALL_OLD"
                    }
                ],
                "returnConsumedCapacity": "TOTAL",
                "returnItemCollectionMetrics": "SIZE"
            },
            "responseElements": null,
            "requestID": "45EN32OM6TQSMV2MI65O4L5TNFVV4KQNSO5AEMVJF66Q9ASUAAJG",
            "eventID": "4f1cc78b-5c94-4174-a6ad-3ee78605381c",
            "readOnly": false,
            "resources": [
                {
                    "accountId": "111122223333",
                    "type": "AWS::DynamoDB::Table",
                    "ARN": "arn:aws:dynamodb:us-west-2:123456789012:table/Music"
                }
            ],
            "eventType": "AwsApiCall",
            "apiVersion": "2012-08-10",
            "managementEvent": false,
            "recipientAccountId": "111122223333",
            "eventCategory": "Data"
        }
    ]
}
```

TransactWriteItems (mit **TransactionCanceledException**)


```
{
    "Records": [
        {
            "eventVersion": "1.06",
            "userIdentity": {
                "type": "AssumedRole",
                "principalId": "AKIAIOSFODNN7EXAMPLE:bob",
                "arn": "arn:aws:sts::111122223333:assumed-role/users/bob",
                "accountId": "111122223333",
                "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
                "sessionContext": {
                    "sessionIssuer": {
                        "type": "Role",
                        "principalId": "AKIAI44QH8DHBEXAMPLE",
                        "arn": "arn:aws:iam::444455556666:role/admin-role",
                        "accountId": "444455556666",
                        "userName": "bob"
                    },
                    "attributes": {
                        "creationDate": "2020-09-03T22:14:13Z",
                        "mfaAuthenticated": "false"
                    }
                }
            },
            "eventTime": "2019-02-01T00:42:34Z",
            "eventSource": "dynamodb.amazonaws.com",
            "eventName": "TransactWriteItems",
            "awsRegion": "us-west-2",
            "sourceIPAddress": "192.0.2.0",
            "userAgent": "aws-cli/1.16.93 Python/3.4.7
 Linux/4.9.119-0.1.ac.277.71.329.metal1.x86_64 botocore/1.12.83",
            "errorCode": "TransactionCanceledException",
            "errorMessage": "Transaction cancelled, please refer cancellation reasons
 for specific reasons [ConditionalCheckFailed, None]",
            "requestParameters": {
                "requestItems": [
                    {
                        "operation": "Put",
                        "tableName": "Music",
                        "key": {
                            "Artist": "No One You Know",
```

```
                    "SongTitle": "Call Me Today"
                },
                "items": [
                    "Artist",
                    "SongTitle",
                    "AlbumTitle"
                ],
                "conditionExpression": "#AT = :A",
                "expressionAttributeNames": {
                    "#AT": "AlbumTitle"
                },
                "returnValuesOnConditionCheckFailure": "ALL_OLD"
            },
            {
                "operation": "Update",
                "tableName": "Music",
                "key": {
                    "Artist": "No One You Know",
                    "SongTitle": "Call Me Tomorrow"
                },
                "updateExpression": "SET #AT = :newval",
                "ConditionExpression": "attribute_not_exists(Rating)",
                "ExpressionAttributeNames": {
                    "#AT": "AlbumTitle"
                },
                "returnValuesOnConditionCheckFailure": "ALL_OLD"
            },
            {
                "operation": "Delete",
                "TableName": "Music",
                "key": {
                    "Artist": "No One You Know",
                    "SongTitle": "Call Me Yesterday"
                },
                "conditionExpression": "#P between :lo and :hi",
                "expressionAttributeNames": {
                    "#P": "Price"
                },
                "ReturnValuesOnConditionCheckFailure": "ALL_OLD"
            },
            {
                "operation": "ConditionCheck",
                "TableName": "Music",
                "Key": {
```

```
                            "Artist": "No One You Know",
                            "SongTitle": "Call Me Now"
                        },
                        "ConditionExpression": "#P between :lo and :hi",
                        "ExpressionAttributeNames": {
                            "#P": "Price"
                        },
                        "ReturnValuesOnConditionCheckFailure": "ALL_OLD"
                    }
                ],
                "returnConsumedCapacity": "TOTAL",
                "returnItemCollectionMetrics": "SIZE"
            },
            "responseElements": null,
            "requestID": "A0GTQEKLBB9VD8E05REA5A3E1VVV4KQNSO5AEMVJF66Q9ASUAAJG",
            "eventID": "43e437b5-908a-46af-84e6-e27fffb9c5cd",
            "readOnly": false,
            "resources": [
                {
                    "accountId": "111122223333",
                    "type": "AWS::DynamoDB::Table",
                    "ARN": "arn:aws:dynamodb:us-west-2:123456789012:table/Music"
                }
            ],
            "eventType": "AwsApiCall",
            "apiVersion": "2012-08-10",
            "managementEvent": false,
            "recipientAccountId": "111122223333",
            "eventCategory": "Data"
        }
    ]
}
```

ExecuteStatement

```
{
    "Records": [
        {
            "eventVersion": "1.08",
            "userIdentity": {
                "type": "AssumedRole",
                "principalId": "AKIAIOSFODNN7EXAMPLE:bob",
                "arn": "arn:aws:sts::111122223333:assumed-role/users/bob",
```

```
                "accountId": "111122223333",
                "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
                "sessionContext": {
                    "sessionIssuer": {
                        "type": "Role",
                        "principalId": "AKIAI44QH8DHBEXAMPLE",
                        "arn": "arn:aws:iam::444455556666:role/admin-role",
                        "accountId": "444455556666",
                        "userName": "bob"
                    },
                    "attributes": {
                        "creationDate": "2020-09-03T22:14:13Z",
                        "mfaAuthenticated": "false"
                    }
                }
            },
            "eventTime": "2021-03-03T23:06:45Z",
            "eventSource": "dynamodb.amazonaws.com",
            "eventName": "ExecuteStatement",
            "awsRegion": "us-west-2",
            "sourceIPAddress": "192.0.2.0",
            "userAgent": "aws-cli/1.19.7 Python/3.6.13
Linux/4.9.230-0.1.ac.223.84.332.metal1.x86_64 botocore/1.20.7",
            "requestParameters": {
                "statement": "SELECT * FROM Music WHERE Artist = 'No One You Know' AND
SongTitle = 'Call Me Today' AND nonKeyAttr = ***(Redacted)"
            },
            "responseElements": null,
            "requestID": "V7G2KCSFLP83ORB7MMFG6RIAD3VV4KQNSO5AEMVJF66Q9ASUAAJG",
            "eventID": "0b5c4779-e169-4227-a1de-6ed01dd18ac7",
            "readOnly": false,
            "resources": [
                {
                    "accountId": "111122223333",
                    "type": "AWS::DynamoDB::Table",
                    "ARN": "arn:aws:dynamodb:us-west-2:123456789012:table/Music"
                }
            ],
            "eventType": "AwsApiCall",
            "apiVersion": "2012-08-10",
            "managementEvent": false,
            "recipientAccountId": "111122223333",
            "eventCategory": "Data"
        }
```

```
    ]
}
```

BatchExecuteStatement

```
{
    "Records": [
        {
            "eventVersion": "1.08",
            "userIdentity": {
                "type": "AssumedRole",
                "principalId": "AKIAIOSFODNN7EXAMPLE:bob",
                "arn": "arn:aws:sts::111122223333:assumed-role/users/bob",
                "accountId": "111122223333",
                "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
                "sessionContext": {
                    "sessionIssuer": {
                        "type": "Role",
                        "principalId": "AKIAI44QH8DHBEXAMPLE",
                        "arn": "arn:aws:iam::444455556666:role/admin-role",
                        "accountId": "444455556666",
                        "userName": "bob"
                    },
                    "attributes": {
                        "creationDate": "2020-09-03T22:14:13Z",
                        "mfaAuthenticated": "false"
                    }
                }
            },
            "eventTime": "2021-03-03T23:24:48Z",
            "eventSource": "dynamodb.amazonaws.com",
            "eventName": "BatchExecuteStatement",
            "awsRegion": "us-west-2",
            "sourceIPAddress": "192.0.2.0",
            "userAgent": "aws-cli/1.19.7 Python/3.6.13
 Linux/4.9.230-0.1.ac.223.84.332.metal1.x86_64 botocore/1.20.7",
            "requestParameters": {
                "requestItems": [
                    {
                        "statement": "UPDATE Music SET Album = ***(Redacted) WHERE
 Artist = 'No One You Know' AND SongTitle = 'Call Me Today'"
                    },
                    {
```

```
                        "statement": "INSERT INTO Music VALUE {'Artist' :
 ***(Redacted), 'SongTitle' : ***(Redacted), 'Album' : ***(Redacted)}"
                    }
                ]
            },
            "responseElements": null,
            "requestID": "23PE7ED291UD65P9SMS6TISNVBVV4KQNSO5AEMVJF66Q9ASUAAJG",
            "eventID": "f863f966-b741-4c36-b15e-f867e829035a",
            "readOnly": false,
            "resources": [
                {
                    "accountId": "111122223333",
                    "type": "AWS::DynamoDB::Table",
                    "ARN": "arn:aws:dynamodb:us-west-2:123456789012:table/Music"
                }
            ],
            "eventType": "AwsApiCall",
            "apiVersion": "2012-08-10",
            "managementEvent": false,
            "recipientAccountId": "111122223333",
            "eventCategory": "Data"
        }
    ]
}
```

GetRecords

```
{
    "Records": [
        {
            "eventVersion": "1.08",
            "userIdentity": {
                "type": "AssumedRole",
                "principalId": "AKIAIOSFODNN7EXAMPLE:bob",
                "arn": "arn:aws:sts::111122223333:assumed-role/users/bob",
                "accountId": "111122223333",
                "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
                "sessionContext": {
                    "sessionIssuer": {
                        "type": "Role",
                        "principalId": "AKIAI44QH8DHBEXAMPLE",
                        "arn": "arn:aws:iam::444455556666:role/admin-role",
                        "accountId": "444455556666",
```

```
                        "userName": "bob"
                    },
                    "attributes": {
                        "creationDate": "2020-09-03T22:14:13Z",
                        "mfaAuthenticated": "false"
                    }
                }
            },
            "eventTime": "2021-04-15T04:15:02Z",
            "eventSource": "dynamodb.amazonaws.com",
            "eventName": "GetRecords",
            "awsRegion": "us-west-2",
            "sourceIPAddress": "192.0.2.0",
            "userAgent": "aws-cli/1.19.50 Python/3.6.13
 Linux/4.9.230-0.1.ac.224.84.332.metal1.x86_64 botocore/1.20.50",
            "requestParameters": {
                "shardIterator": "arn:aws:dynamodb:us-west-2:123456789012:table/
Music/stream/2021-04-15T04:02:47.428|1|AAAAAAAAAAH7HF3xwDQHBrvk2UBZ1PKh8bX3F
+JeH0rFwHCE7dz4VGV1ZoJ5bMxQwkmerA3wzCTL+zSseGLdSXNJP14EwrjLNvDNoZeRSJ/
n6xc3I4NYOptR4zR8d7VrjMAD6h5nR12NtxGIgJ/
dVsUpluWsHyCW3PPbKsMlJSruVRWoitRhSd3S6slEWEPB0bDC7+
+ISH5mXrCHOnvyezQKlqNshTSPZ5jWwqRj2VNSXCMTGXv9P01/
U0bpOUI2cuRTchgUpPSe3ur2sQrRj3KlbmIyCz7P
+H3CYlugafi8fQ5kipDSkESkIWS6O5ejzibWKg/3izms1eVIm/
zLFdEeihCYJ7G8fpHUSLX5JAk3ab68aUXGSFEZLONntgNIhQkcMo00/
mJlaIgkEdBUyqvZO1vtKUBH5YonIrZqSUhv8Coc+mh24vOg1YI+SPIXlr
+Lnl54BG6AjrmaScjHACVXoPDxPsXSJXC4c9HjoC3YSskCPV7uWi0f65/
n7JAT3cskcX2ISaLHwYzJPaMBSftxOgeRLm3BnisL32nT8uTj2gF/
PUrEjdyoqTX7EerQpcaekXmOgay5Kh8n4T2uPdM83f356vRpar/
DDp8pLFD0ddb6Yvz7zU2zGdAvTod3IScC1GpTqcjRxaMhlBVZy1TnI9Cs
+7fXMdUF6xYScjR2725icFBNLojSFVDmsfHabXaCEpmeuXZsLbp5CjcPAHa66R8mQ5tSoFjrzOEzeB4uconEXAMPLE=="
            },
            "responseElements": null,
            "requestID": "1M0U1Q80P4LDPT7A7N1A758N2VVV4KQNSO5AEMVJF66Q9EXAMPLE",
            "eventID": "09a634f2-da7d-4c9e-a259-54aceexample",
            "readOnly": true,
            "resources": [
                {
                    "accountId": "111122223333",
                    "type": "AWS::DynamoDB::Table",
                    "ARN": "arn:aws:dynamodb:us-west-2:123456789012:table/Music"
                }
            ],
            "eventType": "AwsApiCall",
```

```
            "apiVersion": "2012-08-10",
            "managementEvent": false,
            "recipientAccountId": "111122223333",
            "eventCategory": "Data"
        }
    ]
}
```

Analysieren des Datenzugriffs mithilfe von CloudWatch Contributor Insights für DynamoDB

Amazon CloudWatch Contributor Insights for Amazon DynamoDB ist ein Diagnosetool, mit dem Sie auf einen Blick die am häufigsten aufgerufenen und gedrosselten Schlüssel in Ihrer Tabelle oder Ihrem Index identifizieren können. Dieses Tool verwendet Contributor Insights. CloudWatch

Wenn Sie CloudWatch Contributor Insights for DynamoDB für eine Tabelle oder einen globalen sekundären Index aktivieren, können Sie die am häufigsten aufgerufenen und gedrosselten Elemente in diesen Ressourcen anzeigen.

> **Note**
>
> CloudWatch Für Contributor Insights for DynamoDB fallen Gebühren an. Weitere Informationen zur Preisgestaltung finden Sie unter CloudWatchAmazon-Preise.

Themen

- CloudWatch Einblicke von Mitwirkenden für DynamoDB: So funktioniert es
- Erste Schritte mit CloudWatch Contributor Insights for DynamoDB
- Verwenden von IAM mit CloudWatch Contributor Insights für DynamoDB

CloudWatch Einblicke von Mitwirkenden für DynamoDB: So funktioniert es

Amazon DynamoDB ist in CloudWatch Contributor Insights integriert, um Informationen über die am häufigsten aufgerufenen und gedrosselten Elemente in einer Tabelle oder einem globalen sekundären Index bereitzustellen. DynamoDB stellt Ihnen diese Informationen über Regeln, Berichte und Diagramme mit Berichtsdaten von CloudWatch Contributor Insights zur Verfügung.

CloudWatch Contributor Insights for DynamoDB ist so konzipiert, dass es keine Auswirkungen auf die Leistung Ihrer DynamoDB-Tabelle hat.

Weitere Informationen zu CloudWatch Contributor Insights finden Sie im Amazon-Benutzerhandbuch unter Verwenden von Contributor Insights zur Analyse von Daten mit hoher Kardinalität. CloudWatch

In den folgenden Abschnitten werden die Kernkonzepte und das Verhalten von CloudWatch Contributor Insights for DynamoDB beschrieben.

Themen

- CloudWatch Contributor Insights-Modi für DynamoDB
- CloudWatch Einblicke von Mitwirkenden für DynamoDB-Regeln
- Erkenntnisse von CloudWatch Mitwirkenden für DynamoDB-Diagramme verstehen
- Interaktionen mit anderen DynamoDB-Funktionen
- CloudWatch Einblicke von Mitwirkenden für DynamoDB-Abrechnung

CloudWatch Contributor Insights-Modi für DynamoDB

CloudWatch Contributor Insights for DynamoDB bietet zwei unterschiedliche Modi, um unterschiedlichen Überwachungsanforderungen gerecht zu werden.

Modus für gedrosselte Schlüssel

Dieser Modus konzentriert sich ausschließlich auf gedrosselte Anfragen, indem Ereignisse nur verarbeitet werden, wenn eine Drosselung eintritt. Er bietet Einblicke in Leistungsprobleme, ohne dass alle Zugriffsmuster nachverfolgt werden müssen. In diesem Modus verfolgt DynamoDB nur Folgendes:

- Elemente mit den meisten Drosselungen: Elemente, bei denen die meisten Drosselungsereignisse auftreten

Dieser Modus ist in folgenden Situationen ideal:

- Ihr Hauptanliegen ist die Identifizierung und Lösung von Drosselungsproblemen.
- Sie möchten Contributor Insights kontinuierlich aktiviert lassen, damit Drosselungen in Echtzeit erkannt werden können.
- Sie möchten einen kostenoptimierten Ansatz zur Überwachung von Drosselungsproblemen nutzen.

> **Note**
>
> Modus für gedrosselte Schlüssel: Hierbei werden Ereignisse nur dann verarbeitet, wenn eine Drosselung erfolgt, sodass eine kontinuierliche Überwachung kostengünstig ist. Dieser gezielte Ansatz ermöglicht es Ihnen, die Funktion bei minimalen Auswirkungen auf die Kosten dauerhaft aktiviert zu lassen, während Sie gleichzeitig sofort Einblicke in Drosselungsprobleme erhalten, sobald sie auftreten.

Wenn in Ihrer Tabelle keine Drosselung auftritt, werden in den Contributor-Insights-Diagrammen keine Daten angezeigt, was auf eine gute Leistung hindeutet. Wenn eine Drosselung erkannt wird, helfen Ihnen die generierten Diagramme dabei, bestimmte Zugriffsmuster zu identifizieren, die Leistungsprobleme verursachen. Diese Informationen können Ihnen bei der Implementierung von Strategien zur Behebung ungleichmäßiger Zugriffsmuster helfen.

Für umfassende Überwachungsstrategien können Sie diese Einschränkungsinformationen mit anderen CloudWatch Metriken kombinieren, um einheitliche Dashboards zu erstellen, die Drosselungsereignisse mit der Gesamtleistung der Tabelle korrelieren.

Modus für Schlüssel mit Zugriffen und Drosselungen

Dieser Modus ermöglicht eine umfassende Überwachung sowohl der abgerufenen als auch der gedrosselten Objekte. In diesem Modus verfolgt DynamoDB Folgendes:

- Elemente mit den meisten Zugriffen: Elemente, die am meisten Lese- und Schreibkapazität verbrauchen
- Elemente mit den meisten Drosselungen: Elemente, bei denen die meisten Drosselungsereignisse auftreten

Dieser Modus ist ideal, wenn Sie einen vollständigen Überblick über die Zugriffsmuster Ihrer Tabelle benötigen und sowohl Informationen zu stark frequentierten Elementen als auch zu Drosselungsproblemen erhalten möchten.

Wechseln zwischen den Modi

Sie können jederzeit mit der DynamoDB-Konsole, AWS CLI, oder zwischen den Modi wechseln. APIs Wenn Sie zwischen den Modi wechseln:

- Bestehende CloudWatch Regeln werden aktualisiert, sodass sie dem neuen Modus entsprechen

- Die CloudWatch Regeln für gedrosselte Schlüssel bleiben erhalten, sodass Ihre kontinuierlichen historischen Daten für Drosselungskennzahlen erhalten bleiben:
  - Wenn Sie vom Modus für gedrosselte Schlüssel in den Modus für Schlüssel mit Zugriffen und Drosselungen wechseln, werden die vorhandenen Regeln für gedrosselte Schlüssel beibehalten und es werden neue Regeln für Schlüssel mit Zugriffen erstellt
  - Wenn Sie vom Modus für Schlüssel mit Zugriffen und Drosselungen in den Modus für gedrosselte Schlüssel wechseln, werden nur die vorhandenen Regeln für gedrosselte Schlüssel beibehalten und die Regeln für Schlüssel mit Zugriffen werden entfernt.
- Die Fakturierung wird sofort angepasst, um der Ereignisverarbeitung des neuen Modus zu entsprechen.

CloudWatch Einblicke von Mitwirkenden für DynamoDB-Regeln

Wenn Sie CloudWatch Contributor Insights for DynamoDB für eine Tabelle oder einen globalen sekundären Index aktivieren, erstellt DynamoDB in Ihrem Namen Regeln auf der Grundlage des ausgewählten Modus.

> **Note**
>
> Wenn Sie Contributor Insights in Ihrer DynamoDB-Tabelle aktivieren, gelten die Regelbeschränkungen von Contributor Insights. Weitere Informationen finden Sie unter CloudWatch -Servicekontingente.

Regeln für den Modus für Schlüssel mit Zugriffen und Drosselungen

Im Modus für Schlüssel mit Zugriffen und Drosselungen erstellt DynamoDB die folgenden Regeln:

- Am häufigsten aufgerufene Elemente (Partitionsschlüssel) — Identifiziert die Partitionsschlüssel der Elemente, auf die am häufigsten zugegriffen wird, in Ihrer Tabelle oder Ihrem globalen sekundären Index.

  CloudWatch Format des Regelnamens: `DynamoDBContributorInsights-PKC-[resource_name]-[creationtimestamp]`
- Die meisten gedrosselten Schlüssel (Partitionsschlüssel) — Identifiziert die Partitionsschlüssel der am stärksten gedrosselten Elemente in Ihrer Tabelle oder Ihrem globalen sekundären Index.

CloudWatch Format des Regelnamens: `DynamoDBContributorInsights-PKT-[resource_name]-[creationtimestamp]`

Wenn Ihre Tabelle oder Ihr globaler sekundärer Index Sortierschlüssel enthält, erstellt DynamoDB auch die folgenden Regeln speziell für Sortierschlüssel:

- Schlüssel mit den meisten Zugriffen (Partitions- und Sortierschlüssel) – Partitions- und Sortierschlüssel der Elemente mit den meisten Zugriffen in der Tabelle oder im globalen Sekundärindex.

  CloudWatch Format des Regelnamens: `DynamoDBContributorInsights-SKC-[resource_name]-[creationtimestamp]`
- Schlüssel mit den meisten Drosselungen (Partitions- und Sortierschlüssel) – Partitions- und Sortierschlüssel der Elemente mit den meisten Drosselungen in der Tabelle oder im globalen Sekundärindex.

  CloudWatch Format des Regelnamens: `DynamoDBContributorInsights-SKT-[resource_name]-[creationtimestamp]`

Regeln für den Modus für gedrosselte Schlüssel

Im Modus für gedrosselte Schlüssel erstellt DynamoDB nur die Regeln, die sich auf die Drosselung beziehen:

- Schlüssel mit den meisten Drosselungen (Partitionsschlüssel) – Partitionsschlüssel der Elemente mit den meisten Drosselungen in der Tabelle oder im globalen Sekundärindex.

  CloudWatch Format des Regelnamens: `DynamoDBContributorInsights-PKT-[resource_name]-[creationtimestamp]`

Wenn Ihre Tabelle oder Ihr globaler sekundärer Index Sortierschlüssel hat, erstellt DynamoDB auch:

- Schlüssel mit den meisten Drosselungen (Partitions- und Sortierschlüssel) – Partitions- und Sortierschlüssel der Elemente mit den meisten Drosselungen in der Tabelle oder im globalen Sekundärindex.

CloudWatch Format des Regelnamens: `DynamoDBContributorInsights-SKT-[resource_name]-[creationtimestamp]`

Dieser gezielte Ansatz reduziert die Anzahl der aktiven Regeln und verringert die Menge der verarbeiteten Ereignisse, sodass Sie Ihre Drosselungsereignisse besser diagnostizieren können.

> **Note**
>
> - Wenn Sie die CloudWatch Konsole verwenden oder APIs CloudWatch Contributor Insights for DynamoDB aufrufen, werden Ihnen nur Regeln angezeigt, die Ihrem ausgewählten Modus entsprechen.
> - Sie können die CloudWatch Konsole nicht verwenden oder APIs die von CloudWatch Contributor Insights for DynamoDB erstellten Regeln direkt ändern oder löschen. Wenn CloudWatch Contributor Insights for DynamoDB für eine Tabelle oder einen globalen sekundären Index deaktiviert wird, werden automatisch die Regeln gelöscht, die für diese Tabelle oder den globalen sekundären Index erstellt wurden.
> - Wenn Sie den GetInsightRuleReportVorgang mit CloudWatch Contributor Insights-Regeln verwenden, die von DynamoDB erstellt wurden, nur `MaxContributorValue` und nützliche Statistiken `Maximum` zurückgeben. Die anderen Statistiken in dieser Liste geben keine sinnvollen Werte zurück.
> - CloudWatch Contributor Insights for DynamoDB hat ein Limit von 25 Mitwirkenden. Wenn mehr als 25 Beitragende angefordert werden, wird ein Fehler zurückgegeben.

Sie können CloudWatch Alarme mithilfe der CloudWatch Contributor Insights for DynamoDB-Regeln erstellen. Auf diese Weise werden Sie benachrichtigt, wenn ein Artikel einen bestimmten Schwellenwert für `ConsumedThroughputUnits` oder `ThrottleCount` überschreitet oder erreicht. Weitere Informationen finden Sie unter Festlegen eines Alarms für Contributor-Insights-Metrikdaten.

Erkenntnisse von CloudWatch Mitwirkenden für DynamoDB-Diagramme verstehen

CloudWatch Contributor Insights for DynamoDB zeigt je nach ausgewähltem Modus unterschiedliche Diagrammtypen sowohl auf der DynamoDB als auch auf den CloudWatch Konsolen an.

Diagrammverfügbarkeit nach Modus

Die angezeigten Diagramme hängen vom ausgewählten Contributor-Insights-Modus ab.

- Im Modus für Schlüssel mit Zugriffen und Drosselungen werden sowohl das Diagramm Elemente mit den meisten Zugriffen als auch das Diagramm Elemente mit den meisten Drosselungen angezeigt.
- Im Modus für gedrosselte Schlüssel werden nur die Diagramme zu Elemente mit den meisten Drosselungen angezeigt.

Elemente mit den meisten Zugriffen

Dieses Diagramm ist nur im Modus für Schlüssel mit Zugriffen und Drosselungen verfügbar. Diesem Diagramm können Sie die Elemente mit den meisten Zugriffen in der Tabelle oder im globalen Sekundärindex entnehmen. Das Diagramm zeigt `ConsumedThroughputUnits` auf der y-Achse und die Zeit auf der x-Achse an. Jeder der oberen N Schlüssel wird in einer eigenen Farbe angezeigt. Unter der x-Achse befindet sich eine Legende.

DynamoDB misst die Frequenz der Schlüsselzugriffe mit `ConsumedThroughputUnits`. In diesem Wert werden Lese- und Schreibdatenverkehr aggregiert. `ConsumedThroughputUnits` ist folgendermaßen definiert:

- Bereitgestellt – (3 x verbrauchte Schreibkapazitätseinheiten) + verbrauchte Lesekapazitätseinheiten.
- On-Demand – (3 x Leseanforderungseinheiten) + Schreibanforderungseinheiten

In der DynamoDB-Konsole repräsentiert jeder Datenpunkt im Diagramm das Maximum von `ConsumedThroughputUnits` in einem Zeitraum von einer Minute. Ein Diagrammwert von 180.000 `ConsumedThroughputUnits` gibt beispielsweise an, dass kontinuierlich über 60 Sekunden mit dem Maximaldurchsatz pro Element von 1.000 Schreibanforderungseinheiten oder 3.000 Leseanforderungseinheiten in diesem 1-Minuten-Zeitraum (3.000 x 60 Sekunden) auf das Element zugegriffen wurde. Anders ausgedrückt, die im Diagramm dargestellten Werte repräsentieren die Minute mit dem höchsten Datenverkehrsaufkommen in jedem 1-Minuten-Zeitraum. Sie können die zeitliche Granularität der `ConsumedThroughputUnits` Metrik auf der Konsole ändern (z. B. um 5-Minuten-Metriken statt 1-Minute-Metriken anzuzeigen). CloudWatch

Wenn Sie mehrere Linien ohne offensichtliche Ausreißer als Block sehen, zeigt dies eine weitgehend ausgeglichene Workload über die Elemente im gegebenen Zeitfenster an. Wenn Sie isolierte Punkte anstelle verbundener Linien im Diagramm sehen, zeigt dies ein Element, auf das nur für einen kurzen Zeitraum häufig zugegriffen wurde.

Wenn Ihre Tabelle oder Ihr globaler sekundärer Index Sortierschlüssel enthält, erstellt DynamoDB zwei Diagramme: eines für die am häufigsten aufgerufenen Partitionsschlüssel und eines für die am häufigsten aufgerufenen Partition+Sortierschlüsselpaare. Sie können den Verkehr auf der Ebene der Partitionsschlüssel im Diagramm sehen, das nur für Partitionsschlüssel bestimmt ist. Sie können den Verkehr auf Elementebene in den Diagrammen mit den Sortierschlüsseln „Partition +“ sehen.

Elemente mit den meisten Drosselungen

Dieses Diagramm ist in beiden Modi verfügbar. Diesem Diagramm können Sie die Elemente mit den meisten Drosselungen in der Tabelle oder im globalen Sekundärindex entnehmen. Das Diagramm zeigt `ThrottleCount` auf der y-Achse und die Zeit auf der x-Achse an. Jeder der oberen N Schlüssel wird in einer eigenen Farbe angezeigt. Unter der x-Achse befindet sich eine Legende.

DynamoDB misst die Drosselungsfrequenz mit `ThrottleCount`, also der Anzahl der `ProvisionedThroughputExceededException`-, `ThrottlingException`- und `RequestLimitExceeded`-Fehler.

Schreibeinschränkung, die durch unzureichende Schreibkapazität für einen globalen sekundären Index verursacht wird, wird nicht gemessen. Sie können das Diagramm der am häufigsten zugegriffenen Elemente des globalen sekundären Index verwenden, um unausgeglichene Zugriffsmuster zu identifizieren, die eine Schreibdrosselung verursachen können. Weitere Informationen finden Sie unter Überlegungen im Hinblick auf die bereitgestellte Durchsatzkapazität für globale sekundäre Indizes.

In der DynamoDB-Konsole stellt jeder Datenpunkt im Diagramm die Anzahl der Drosselungsereignisse über einen Zeitraum von einer Minute dar.

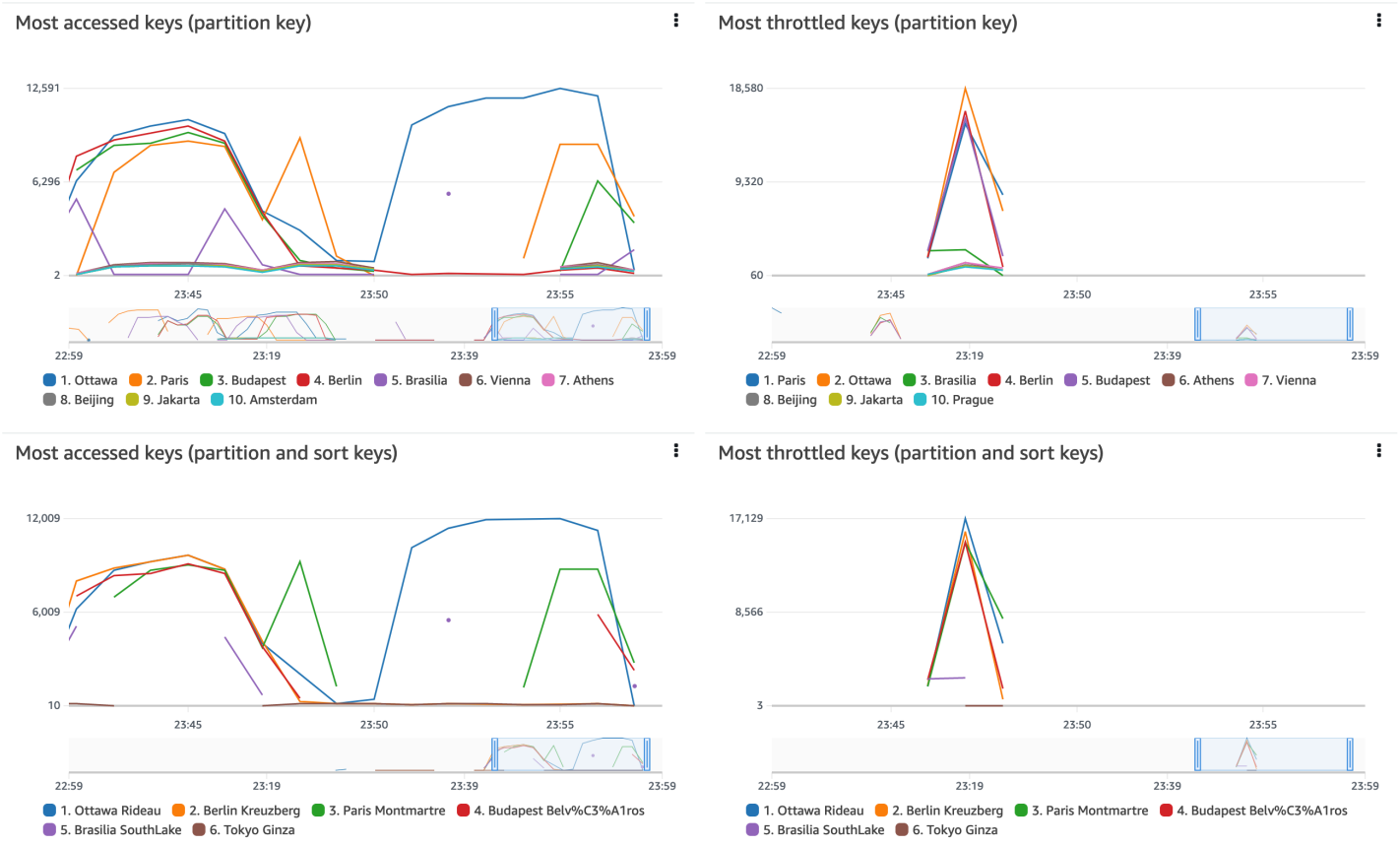
Wenn in diesem Diagramm keine Daten angezeigt werden, wurden die Anforderungen nicht gedrosselt. Wenn Sie isolierte Punkte anstelle verbundener Linien im Diagramm sehen, zeigt dies ein Element, das häufig für einen kurzen Zeitraum gedrosselt wurde.

Wenn Ihre Tabelle oder Ihr globaler sekundärer Index Sortierschlüssel enthält, erstellt DynamoDB zwei Diagramme: eines für die am meisten gedrosselten Partitionsschlüssel und eines für die am meisten gedrosselten Partition+Sortierschlüsselpaare. Sie können die Anzahl der Drosselungen auf der Ebene der Partitionsschlüssel im Diagramm, das nur für die Partitionsschlüssel gilt, und die Anzahl der Drosselungen auf Elementebene in den Diagrammen mit den Sortierschlüsseln „Partition +“ sehen.

> **Note**
>
> Im Modus für gedrosselte Schlüssel ist dies der einzige Diagrammtyp, die Sie sehen. Das Fehlen von Daten in diesen Diagrammen deutet auf eine gute Tabellenleistung hin, ohne dass es zu Drosselungen kommt.

Berichtsbeispiele

Das folgende Beispiel zeigt die Berichte, die für eine Tabelle generiert wurden, die sowohl Partitionsschlüssel als auch Sortierschlüssel im Modus Zugriff und gedrosselte Schlüssel enthält. Im Modus für gedrosselte Schlüssel sehen Sie nur den Teil dieses Berichts, der sich auf die Drosselung bezieht.



Interaktionen mit anderen DynamoDB-Funktionen

In den folgenden Abschnitten wird beschrieben, wie sich CloudWatch Contributor Insights for DynamoDB verhält und mit verschiedenen anderen Funktionen in DynamoDB interagiert. Diese Verhaltensweisen gelten für beide Modi, sofern nicht anders angegeben.

Globale Tabellen

CloudWatch Contributor Insights for DynamoDB überwacht globale Tabellenreplikate als separate Tabellen. Die Contributor Insights-Diagramme für ein Replikat in einer AWS Region zeigen möglicherweise nicht dieselben Muster wie in einer anderen Region. Der Grund besteht darin, dass Schreibdaten über alle Replikate in einer globalen Tabelle repliziert werden, jedes Replikat aber regionsspezifischen Lesedatenverkehr abwickeln kann.

Jedes Replikat kann unabhängig mit einem anderen Contributor-Insights-Modus konfiguriert werden. Beispielsweise können Sie in Ihrer primären Region den Modus für Schlüssel mit Zugriffen und Drosselungen für eine umfassende Überwachung verwenden, während Sie in sekundären Regionen den Modus für gedrosselte Schlüssel verwenden, um den Überblick über Leistungsprobleme zu behalten.

DynamoDB Accelerator (DAX)

CloudWatch Contributor Insights for DynamoDB zeigt keine DAX-Cache-Antworten an. Es zeigt nur Antworten auf den Zugriff auf eine Tabelle oder einen globalen sekundären Index an.

> **Note**
>
> DynamoDB CloudWatch Contributor Insights unterstützt keine PartiQL-Anfragen.

Verschlüsselung im Ruhezustand

CloudWatch Contributor Insights for DynamoDB hat keinen Einfluss darauf, wie die Verschlüsselung in DynamoDB funktioniert. Die Primärschlüsseldaten, die in CloudWatch veröffentlicht werden, werden mit dem verschlüsselt. AWS-eigener Schlüssel DynamoDB unterstützt jedoch auch den Von AWS verwalteter Schlüssel und einen vom Kunden verwalteten Schlüssel.

CloudWatch Contributor Insights for DynamoDB zeigt Partitionsschlüssel und Sortierschlüssel (falls zutreffend) von Objekten an, auf die häufig zugegriffen wird und die gedrosselt werden. CloudWatch Contributor Insights arbeitet zwar mit verschlüsselten DynamoDB-Tabellen, es ist jedoch wichtig zu beachten, dass es seinen eigenen Verschlüsselungskontext verwendet, der von der konfigurierten Verschlüsselung der Tabelle getrennt ist.

Wenn der Primärschlüssel Ihrer DynamoDB-Tabelle vertrauliche Informationen enthält und die Sicherheitsrichtlinien Ihrer Organisation die vollständige Kontrolle über Verschlüsselungsprozesse erfordern, ist die Aktivierung von CloudWatch Contributor Insights möglicherweise nicht geeignet.

Differenzierte Zugriffskontrolle

CloudWatch Contributor Insights for DynamoDB funktioniert bei Tabellen mit feinkörniger Zugriffskontrolle (FGAC) nicht anders. Mit anderen Worten, jeder Benutzer mit den entsprechenden CloudWatch Berechtigungen kann FGAC-geschützte Primärschlüssel in Contributor Insights-Diagrammen anzeigen. CloudWatch

Wenn der Primärschlüssel der Tabelle FGAC-geschützte Daten enthält, in denen Sie nicht veröffentlichen möchten CloudWatch, sollten Sie CloudWatch Contributor Insights for DynamoDB für diese Tabelle nicht aktivieren.

Zugriffskontrolle

Sie steuern den Zugriff auf CloudWatch Contributor Insights for DynamoDB mithilfe von AWS Identity and Access Management (IAM), indem Sie die Berechtigungen der DynamoDB-Steuerungsebene und der Datenebenenberechtigungen einschränken. CloudWatch Weitere Informationen finden Sie unter Verwenden von IAM mit CloudWatch Contributor Insights for DynamoDB.

CloudWatch Einblicke von Mitwirkenden für DynamoDB-Abrechnung

Die Gebühren für CloudWatch Contributor Insights for DynamoDB werden im CloudWatchAbschnitt Ihrer monatlichen Rechnung aufgeführt. Diese Gebühren werden basierend auf der Anzahl der verarbeiten DynamoDB-Ereignisse und dem ausgewählten Modus berechnet.

Fakturierung nach Modus

Die beiden Contributor-Insights-Modi haben unterschiedliche Fakturierungsmerkmale.

- Fakturierung des Modus für Schlüssel mit Zugriffen und Drosselungen: In diesem Modus stellt jedes Element, das über eine Datenebenenoperation geschrieben oder gelesen wird, ein Ereignis dar, unabhängig davon, ob die Anfrage erfolgreich ist oder gedrosselt wird. Wenn eine Tabelle oder ein globaler sekundärer Index Sortierschlüssel enthält, steht jedes gelesene oder geschriebene Element für zwei Ereignisse. Dies liegt daran, dass DynamoDB die wichtigsten Mitwirkenden aus verschiedenen Zeitreihen identifiziert: eine nur für Partitionsschlüssel und eine für Partitions- und Sortierschlüsselpaare.
- Fakturierung des Modus für gedrosselte Schlüssel: In diesem Modus generieren nur gedrosselte Anfragen kostenpflichtige Ereignisse. Ereignisse werden nur generiert, wenn Anfragen zu `ProvisionedThroughputExceededException`-, `ThrottlingException`- oder `RequestLimitExceeded`-Fehlern führen. Wenn eine Tabelle oder ein globaler

sekundärer Index Sortierschlüssel enthält, steht jedes gedrosselte Element für zwei Ereignisse (Partitionsschlüsselverfolgung und Partition+Sortierschlüsselverfolgung).

Abrechnungsbeispiel

Angenommen, Ihre Anwendung führt die folgenden DynamoDB-Operationen aus: `GetItem`, `PutItem`, und `BatchWriteItem`, wobei fünf Elemente eingefügt werden. Gehen Sie außerdem davon aus, dass die `PutItem`-Operation gedrosselt wird, alle anderen Operationen jedoch erfolgreich sind.

- Modus für Schlüssel mit Zugriffen und Drosselungen
  - Wenn Ihre Tabelle oder Ihr globaler sekundärer Index nur Partitionsschlüssel enthält, führt dies zu 7 Ereignissen (1 für die`GetItem`, 1 für die `PutItem` und 5 für die`BatchWriteItem`).
  - Wenn Ihre Tabelle oder Ihr globaler sekundärer Index über Partitionsschlüssel und Sortierschlüssel verfügt, führt dies zu 14 Ereignissen (2 für die`GetItem`, 2 für die `PutItem` und 10 für die`BatchWriteItem`).
- Modus für gedrosselte Schlüssel
  - Wenn Ihre Tabelle oder Ihr globaler sekundärer Index nur Partitionsschlüssel hat, führt dies zu einem Ereignis (nur für gedrosselte`PutItem`).
  - Wenn Ihre Tabelle oder Ihr globaler sekundärer Index über Partitionsschlüssel und Sortierschlüssel verfügt, führt dies zu 2 Ereignissen (2 für `PutItem` gedrosselte Ereignisse).

  Bei erfolgreichen `GetItem`- und `BatchWriteItem`-Operationen werden im Modus mit gedrosselten Schlüsseln keine Ereignisse generiert.

Häufige Abrechnungsfaktoren

Eine `Query`-Operation führt immer zu 1 Ereignis, unabhängig vom Modus oder der Anzahl der zurückgegebenen Elemente.

Im Gegensatz zu anderen DynamoDB-Funktionen variiert die Abrechnung von CloudWatch Contributor Insights for DynamoDB nicht auf der Grundlage der folgenden Faktoren:

- Kapazitätsmodus (Bereitgestellt vs. On-Demand)
- Ob Sie Lese- oder Schreibanforderungen ausführen
- Die Größe (KB) der gelesenen oder geschriebenen Elemente

Erste Schritte mit CloudWatch Contributor Insights for DynamoDB

In diesem Abschnitt wird beschrieben, wie Sie Amazon CloudWatch Contributor Insights in verschiedenen Modi aktivieren und verwenden können, um Ihre Überwachungsanforderungen mithilfe der Amazon DynamoDB DynamoDB-Konsole oder der AWS Command Line Interface () zu erfüllen.AWS CLI

In den folgenden Beispielen verwenden Sie die DynamoDB-Tabelle, die im Erste-Schritte-mit-DynamoDB-Tutorial definiert ist.

Themen

- Auswählen eines Contributor-Insights-Modus
- Verwenden von Contributor Insights (Konsole)
- Verwenden von Contributor Insights (AWS CLI)

Auswählen eines Contributor-Insights-Modus

Bevor Sie Contributor Insights aktivieren, sollten Sie sich mit den beiden verfügbaren Modi vertraut machen. Sehen Sie sich den Vergleich der Modi an, um die Option auszuwählen, die Ihren spezifischen Anforderungen am besten entspricht.

Aspekt	Modus für Schlüssel mit Zugriffen und Drosselungen	Modus für gedrosselte Schlüssel
Monitore	Alle Anforderungen (erfolgreich und gedrosselt)	Nur gedrosselte Anforderungen
Diagramme	Elemente mit den meisten Zugriffen + Elemente mit den meisten Drosselungen	Nur Elemente mit den meisten Drosselungen
Am besten geeignet für	Gezielte Analyse und Optimierung	Überwachung der Drosselung
Anwendungsfall	Sie benötigen einen vollständigen Einblick in die Zugriffsmuster. Sie	Ihr Hauptanliegen ist die Identifizierung und Lösung von Drosselungsproblemen. Sie möchten Contribut

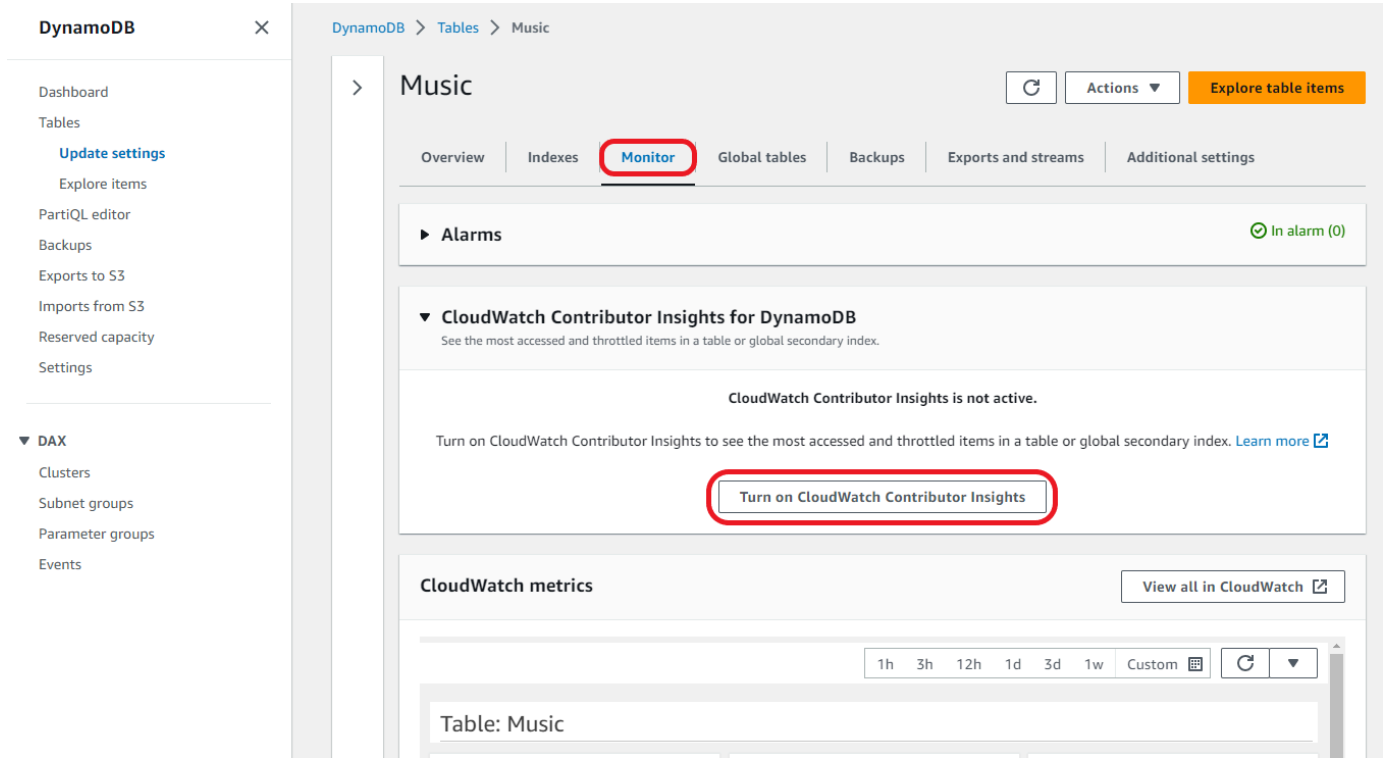
Aspekt	Modus für Schlüssel mit Zugriffen und Drosselungen	Modus für gedrosselte Schlüssel
	führen kurzfristige Analysen oder Debugging durch.	or Insights kontinuierlich aktiviert lassen, um in Echtzeit Warnungen zu Drosselungen zu erhalten.

Verwenden von Contributor Insights (Konsole)

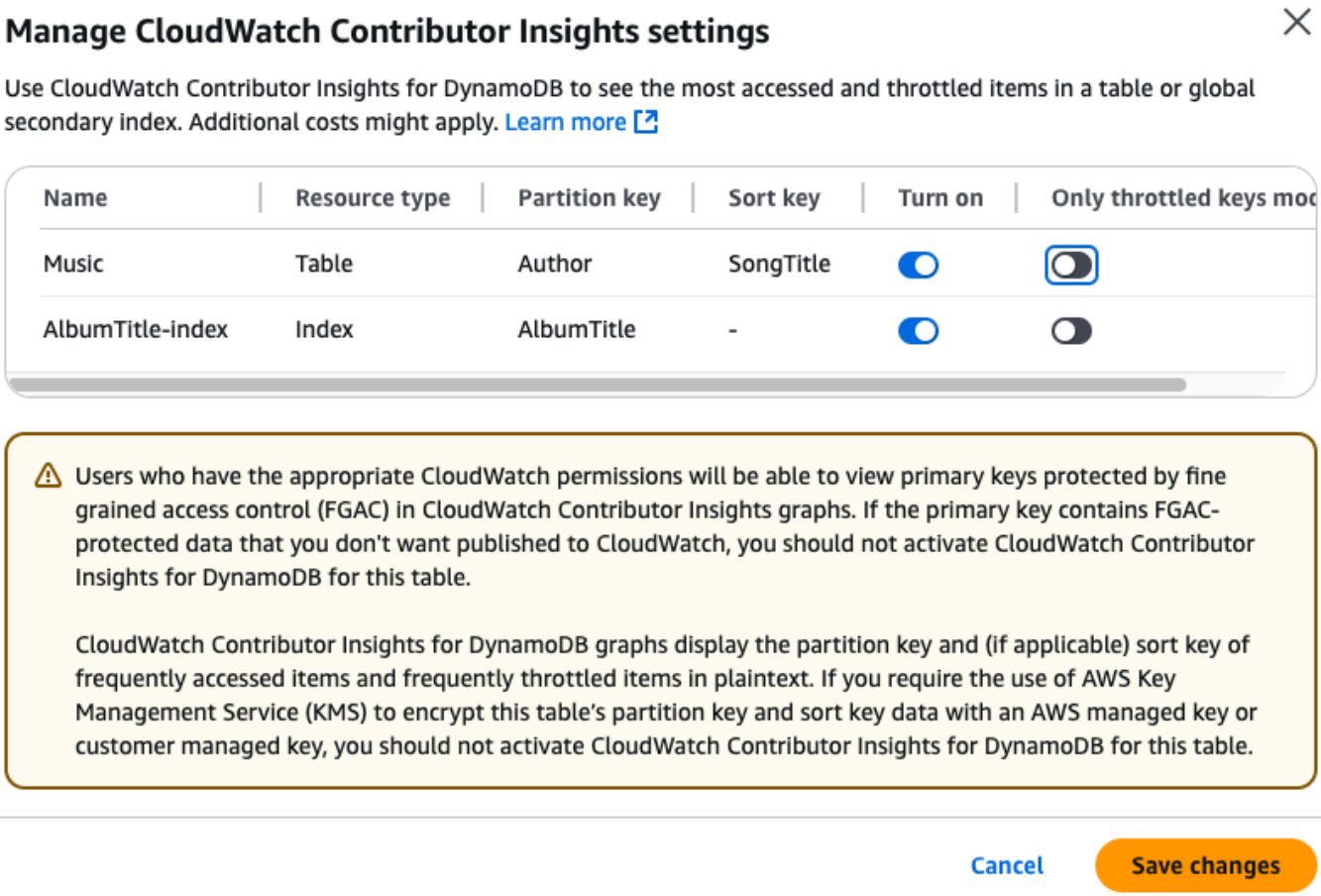
Die Konsole bietet eine intuitive Möglichkeit, Contributor Insights zu aktivieren und den für Ihre Überwachungsanforderungen geeigneten Modus auszuwählen.

So verwenden Sie Contributor Insights in der Konsole

1. Melden Sie sich bei der an AWS-Managementkonsole und öffnen Sie die DynamoDB-Konsole unter. https://console.aws.amazon.com/dynamodb/
2. Klicken Sie im Navigationsbereich auf der linken Seite der Konsole auf Tables (Tabellen).
3. Wählen Sie die `Music` Tabelle aus.
4. Wählen Sie den Tab Überwachung.
5. Wählen Sie CloudWatch Contributor Insights aktivieren aus.



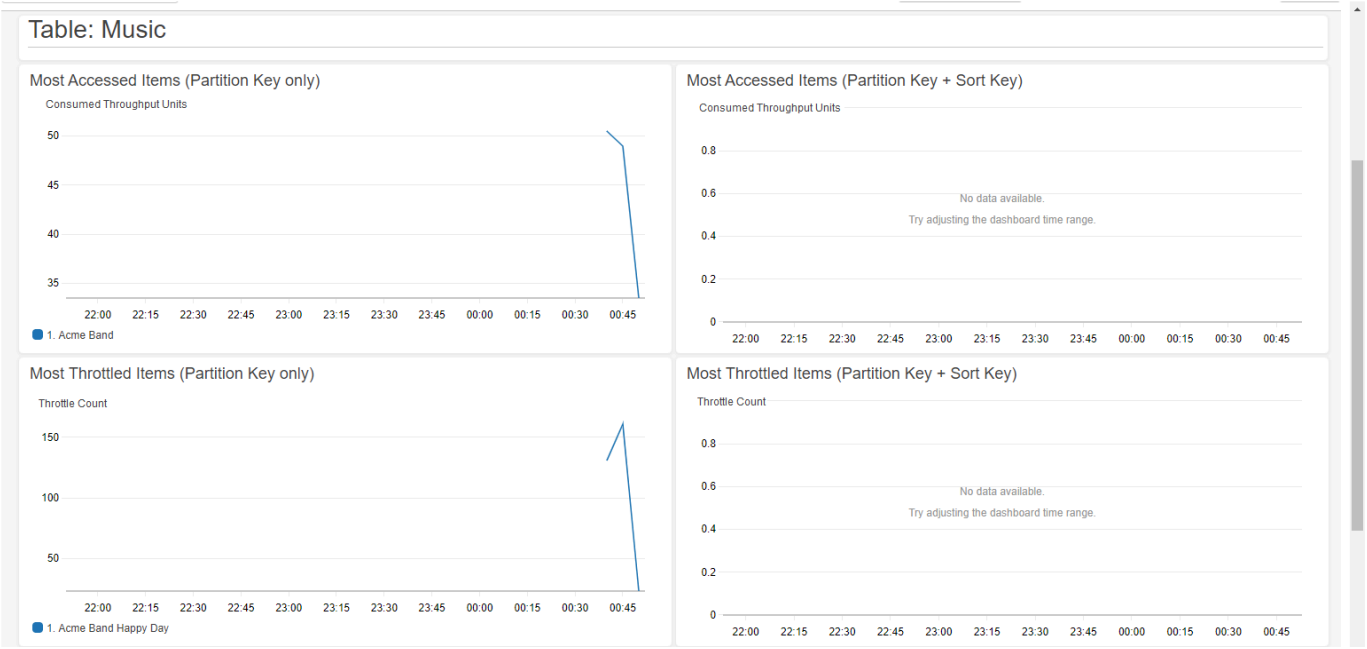
6. Aktivieren Sie im Dialogfeld „ CloudWatch Contributor Insights-Einstellungen verwalten“ die Option Sowohl für die `Music` Basistabelle als auch für den `AlbumTitle-index` globalen Sekundärindex aktivieren.
7. Lassen Sie den Schalter Nur gedrosselter Schlüsselmodus für beide deaktiviert und wählen Sie dann Änderungen speichern aus.



Dadurch wird der Standardmodus für Schlüssel mit Zugriffen und Drosselungen sowohl für die Tabelle als auch für den GSI aktiviert, was eine Überwachung sowohl der aufgerufenen als auch der gedrosselten Elemente ermöglicht. Wenn Sie den Schalter Nur gedrosselter Schlüsselmodus auf „Ein“ festlegen, würde der Modus für gedrosselte Schlüssel aktiviert.

Falls der Vorgang fehlschlägt, finden Sie DescribeContributorInsights FailureExceptionin der Amazon DynamoDB DynamoDB-API-Referenz mögliche Gründe.

8. Die CloudWatch Contributor Insights-Grafiken sind jetzt auf der Registerkarte Monitor für die Tabelle sichtbar. `Music` Da Sie den Modus für Schlüssel mit Zugriffen und Drosselungen aktiviert haben, werden Ihnen sowohl Diagramme für aufgerufene als auch für gedrosselte Elemente angezeigt.



Wechseln zwischen den Modi

Sie können jederzeit zwischen den Modi wechseln, ohne Contributor Insights zu deaktivieren.

So wechseln Sie zwischen Contributor-Insights-Modi

1. Wählen Sie auf der Registerkarte Monitor Ihrer Tabelle die Option CloudWatch Contributor Insights verwalten aus.
2. Wählen Sie im Dialogfeld „Contributor Insights-Einstellungen verwalten“ für jede Basistabelle aus oder: GSIs

   - Schalten Sie Nur gedrosselter Schlüsselmodus ein oder aus, um den Modus für gedrosselte Schlüssel zu aktivieren oder zurück zum Modus für Schlüssel mit Zugriffen und Drosselungen zu wechseln.
   - Aktivieren Sie die Option Einschalten, um CloudWatch Contributor Insight für eine Tabelle oder GSI zu deaktivieren.
3. Wählen Sie Änderungen speichern aus.

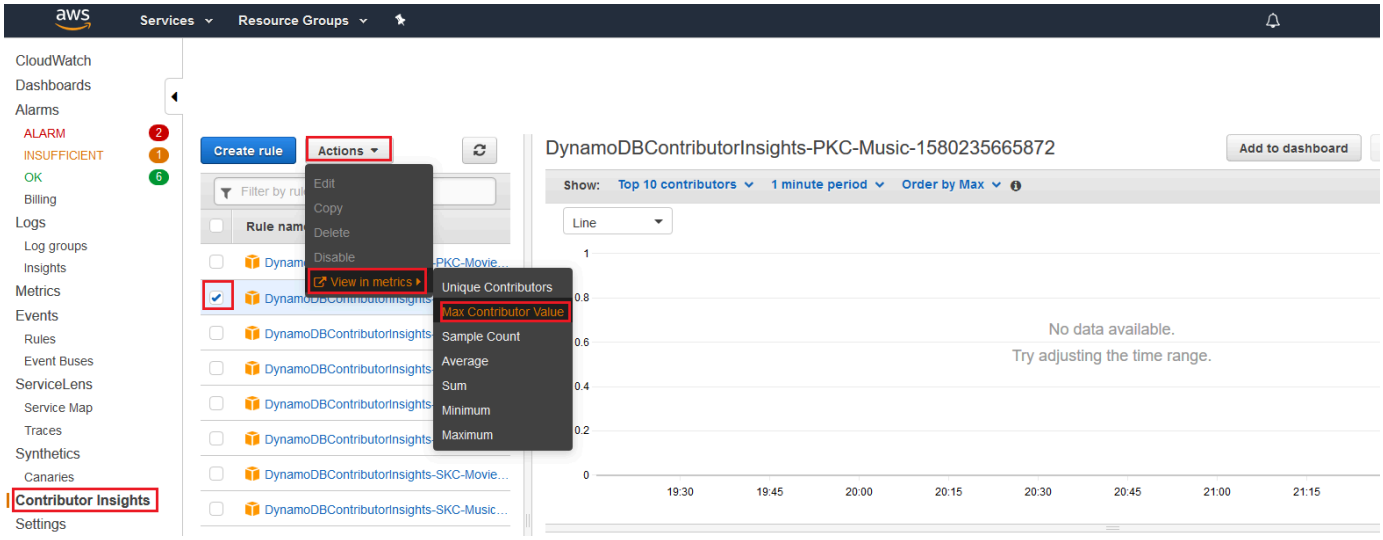
   Sobald der Vorgang abgeschlossen ist, spiegeln die Diagramme den neuen Modus wider.

Alarme erstellen CloudWatch

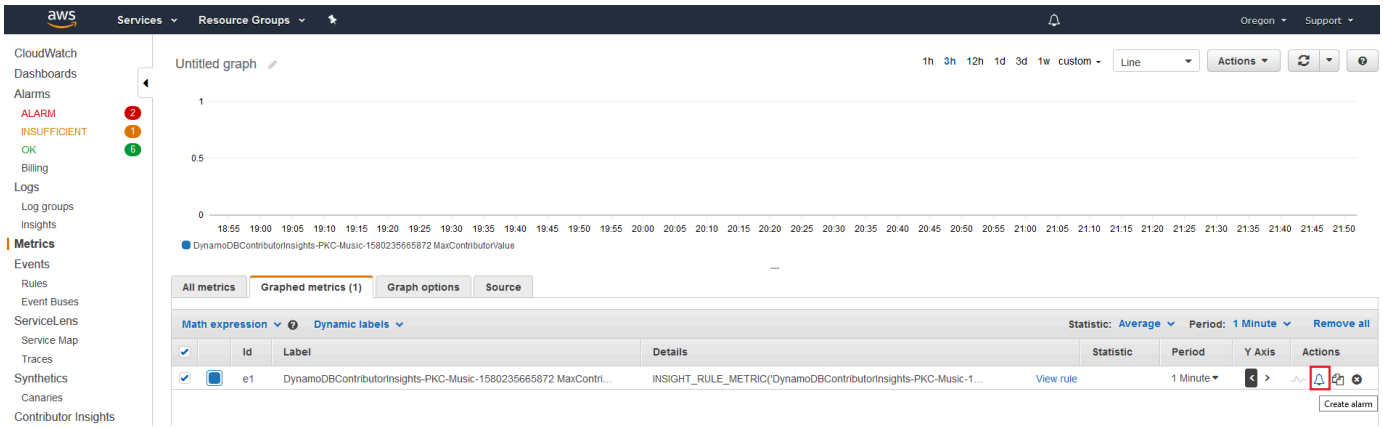
Gehen Sie wie folgt vor, um einen CloudWatch Alarm zu erstellen und benachrichtigt zu werden, wenn ein Partitionsschlüssel mehr als 50.000 verbraucht ConsumedThroughputUnitsoder gedrosselt wird.

1. Melden Sie sich bei der an AWS-Managementkonsole und öffnen Sie die Konsole unter CloudWatch https://console.aws.amazon.com/cloudwatch/
2. Klicken Sie im Navigationsbereich links in der Konsole auf Contributor Insights.
3. Wählen Sie je nach Modus und dem, was Sie überwachen möchten, die entsprechende Regel aus:
   - Für die Überwachung von Objekten, auf die zugegriffen wurde (nur im Modus „Zugriffstasten und gedrosselte Tasten“): Wählen Sie Dynamo Insights-PKC-Music DBContributor
   - Für die Überwachung gedrosselter Objekte (beide Modi): Wählen Sie Dynamo Insights-PKT-Music DBContributor
4. Wählen Sie die Dropdown-Liste Actions (Aktionen).
5. Wählen Sie View in metrics (In Metriken anzeigen).
6. Wählen Sie Max Contributor Value.

> **Note**
>
> Nur `Max Contributor Value` und `Maximum` geben nützliche Statistiken zurück. Die anderen Statistiken in dieser Liste geben keine sinnvollen Werte zurück.

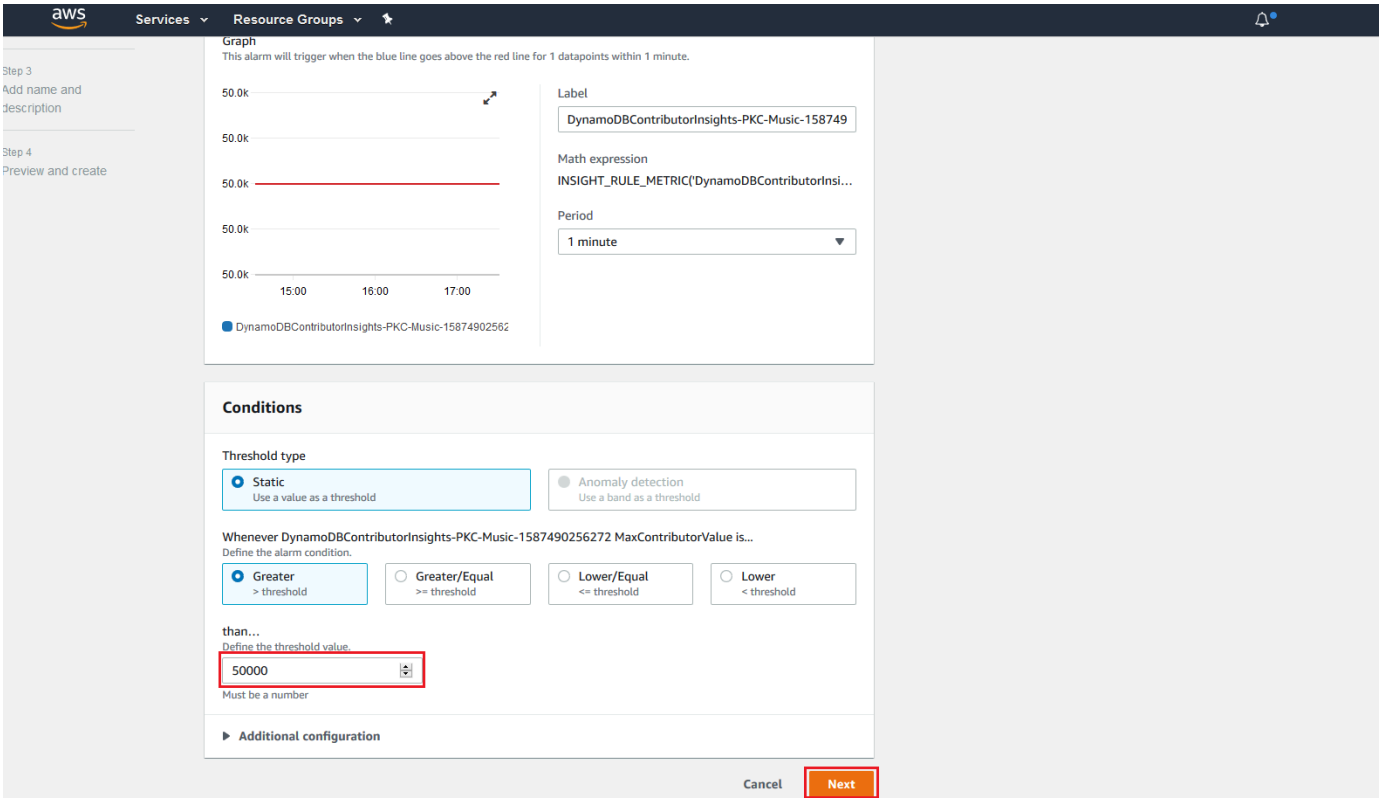


7. Wählen Sie in der Spalte Actions (Aktionen)die Option Create Alarm (Alarm erstellen).



8. Geben Sie einen geeigneten Schwellenwert ein und klicken Sie auf Weiter:

   - Elemente mit Zugriffen (PKC-Regeln): Geben Sie 50 000 für `ConsumedThroughputUnits` ein.
   - Gedrosselte Elemente (PKT-Regeln): Geben Sie 1 für `ThrottleCount` ein, um bei jeder Drosselung gewarnt zu werden.



9. Einzelheiten zur Konfiguration der Benachrichtigung für den CloudWatch Alarm finden Sie unter Amazon-Alarme verwenden.

Verwenden von Contributor Insights (AWS CLI)

Das AWS CLI bietet programmatischen Zugriff auf Contributor Insights mit voller Unterstützung für beide Modi. Sie können den Modus angeben, wenn Sie Contributor Insights aktivieren, oder später zwischen den Modi wechseln.

Grundlegende Operationen im Standardmodus

So verwenden Sie Contributor Insights mit Standardeinstellungen

1. Aktivieren Sie CloudWatch Contributor Insights for DynamoDB in der `Music` Basistabelle im Modus Zugriffstasten und gedrosselte Tasten. Da `ACCESSED_AND_THROTTLED_KEYS` der Standardmodus ist, können Sie den `--contributor-insights-mode=ACCESSED_AND_THROTTLED_KEYS`-Parameter weglassen.

```
aws dynamodb update-contributor-insights \
```

```
                    --table-name Music \
                    --contributor-insights-action=ENABLE
```

2. Aktivieren Sie Contributor Insights for DynamoDB auf dem `AlbumTitle-index` globalen sekundären Index.

```
aws dynamodb update-contributor-insights \
                    --table-name Music \
                    --index-name AlbumTitle-index \
                    --contributor-insights-action=ENABLE
```

3. Rufen Sie den Status und die Regeln für die `Music`-Tabelle und alle seine Indizes auf.

```
aws dynamodb describe-contributor-insights
                    --table-name Music
```

   Die Antwort enthält das `ContributorInsightsMode`-Feld mit `ACCESSED_AND_THROTTLED_KEYS`.

4. Listen Sie den Status der `Music`-Tabelle und aller ihrer Indizes auf.

```
aws dynamodb list-contributor-insights --table-name Music
```

Aktivieren des Modus für gedrosselte Schlüssel

So aktivieren Sie Contributor Insights im Modus für gedrosselte Schlüssel

1. Aktivieren Sie CloudWatch Contributor Insights for DynamoDB in der `Music` Basistabelle mit gedrosseltem Tastenmodus.

```
aws dynamodb update-contributor-insights \
    --table-name Music \
    --contributor-insights-action=ENABLE \
    --contributor-insights-mode=THROTTLED_KEYS
```

2. Aktivieren Sie Contributor Insights im Modus für gedrosselte Schlüssel für den globalen sekundären `AlbumTitle-index`-Index.

```
aws dynamodb update-contributor-insights \
    --table-name Music \
    --index-name AlbumTitle-index \
```

```
    --contributor-insights-action=ENABLE \
    --contributor-insights-mode=THROTTLED_KEYS
```

3. Überprüfen Sie den Modus, indem Sie die Contributor-Insights-Konfiguration beschreiben.

```
aws dynamodb describe-contributor-insights --table-name Music
```

In der Antwort werden `ContributorInsightsMode` als `THROTTLED_KEYS` und weniger Regeln als im Standardmodus angezeigt.

Wechseln zwischen den Modi

So wechseln Sie zwischen Contributor-Insights-Modi

1. Wechseln Sie vom Modus für gedrosselte Schlüssel in den Modus für Schlüssel mit Zugriffen und Drosselungen.

```
aws dynamodb update-contributor-insights \
    --table-name Music \
    --contributor-insights-action=ENABLE \
    --contributor-insights-mode=ACCESSED_AND_THROTTLED_KEYS
```

2. Wechseln Sie vom Modus für Schlüssel mit Zugriffen und Drosselungen in den Modus für gedrosselte Schlüssel.

```
aws dynamodb update-contributor-insights \
    --table-name Music \
    --contributor-insights-action=ENABLE \
    --contributor-insights-mode=THROTTLED_KEYS
```

3. Überprüfen Sie den Status während des Übergangs.

```
aws dynamodb describe-contributor-insights --table-name Music
```

Während des Moduswechsels wird `ContributorInsightsStatus` als `ENABLING` angezeigt. Sobald der Vorgang abgeschlossen ist, wird dies als `ENABLED` mit dem neuen Modus angezeigt.

Verwalten von Contributor Insights

So verwalten Sie Contributor-Insights-Einstellungen

1. Deaktivieren Sie CloudWatch Contributor Insights for DynamoDB im `AlbumTitle-index` globalen sekundären Index.

```
aws dynamodb update-contributor-insights \
                    --table-name Music --index-name AlbumTitle-index \
                    --contributor-insights-action=DISABLE
```

2. Listen Sie alle Contributor-Insights-Konfigurationen in Ihrem Konto auf.

```
aws dynamodb list-contributor-insights
```

   Hiermit werden alle Tabellen und Indizes, für die Contributor Insights aktiviert ist, zusammen mit ihren Modi angezeigt.

3. Rufen Sie detaillierte Informationen zu einer bestimmten Konfiguration ab.

```
aws dynamodb describe-contributor-insights \
                    --table-name Music \
                    --index-name AlbumTitle-index
```

Beispielantworten

Hier finden Sie Beispielantworten, die die Unterschiede zwischen den Modi zeigen:

Reaktion im Modus für Schlüssel mit Zugriffen und Drosselungen

```
{
    "TableName": "Music",
    "ContributorInsightsRuleList": [
        "DynamoDBContributorInsights-PKC-Music-1234567890123",
        "DynamoDBContributorInsights-PKT-Music-1234567890123",
        "DynamoDBContributorInsights-SKC-Music-1234567890123",
        "DynamoDBContributorInsights-SKT-Music-1234567890123"
    ],
    "ContributorInsightsStatus": "ENABLED",
    "ContributorInsightsMode": "ACCESSED_AND_THROTTLED_KEYS",
    "LastUpdateDateTime": "2024-01-15T10:30:00.000Z"
```

```
}
```

Reaktion im Modus für gedrosselte Schlüssel

```
{
    "TableName": "Music",
    "ContributorInsightsRuleList": [
        "DynamoDBContributorInsights-PKT-Music-1234567890123",
        "DynamoDBContributorInsights-SKT-Music-1234567890123"
    ],
    "ContributorInsightsStatus": "ENABLED",
    "ContributorInsightsMode": "THROTTLED_KEYS",
    "LastUpdateDateTime": "2024-01-15T10:35:00.000Z"
}
```

Beachten Sie, dass der Modus für gedrosselte Schlüssel weniger Regeln aufweist (nur PKT und SKT), was einer gezielteren Überwachung entspricht.

Verwenden von IAM mit CloudWatch Contributor Insights für DynamoDB

Wenn Sie Amazon CloudWatch Contributor Insights für Amazon DynamoDB zum ersten Mal aktivieren, erstellt DynamoDB automatisch eine AWS Identity and Access Management (IAM) -Serviceverknüpfte Rolle für Sie. Diese Rolle ermöglicht es `DynamoDBAWSServiceRoleForDynamoDBCloudWatchContributorInsights`, CloudWatch Contributor Insights-Regeln in Ihrem Namen zu verwalten. Löschen Sie diese serviceverknüpfte Rolle nicht. Wenn Sie sie löschen, werden die verwalteten Regeln beim Löschen der Tabelle oder des globalen Sekundärindex nicht mehr bereinigt.

Weitere Informationen zu serviceverknüpften Rollen finden Sie unter Verwenden serviceverknüpfter Rollen im IAM-Benutzerhandbuch.

Die folgenden Berechtigungen sind erforderlich:

- Um CloudWatch Contributor Insights for DynamoDB zu aktivieren oder zu deaktivieren, benötigen Sie `dynamodb:UpdateContributorInsights` Berechtigungen für die Tabelle oder den Index.
- Um CloudWatch Contributor Insights for DynamoDB-Grafiken anzeigen zu können, benötigen Sie eine entsprechende Berechtigung. `cloudwatch:GetInsightRuleReport`
- Um CloudWatch Contributor Insights for DynamoDB für eine bestimmte DynamoDB-Tabelle oder einen bestimmten DynamoDB-Index zu beschreiben, benötigen Sie die entsprechende Berechtigung. `dynamodb:DescribeContributorInsights`

- Um den Status von CloudWatch Contributor Insights for DynamoDB für jede Tabelle und jeden globalen sekundären Index aufzulisten, benötigen Sie die entsprechende Berechtigung. `dynamodb:ListContributorInsights`

Beispiel: CloudWatch Contributor Insights für DynamoDB aktivieren oder deaktivieren

Die folgende IAM-Richtlinie gewährt Berechtigungen zum Aktivieren oder Deaktivieren von CloudWatch Contributor Insights for DynamoDB.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "iam:CreateServiceLinkedRole",
            "Resource": "arn:aws:iam::*:role/aws-
service-role/contributorinsights.dynamodb.amazonaws.com/
AWSServiceRoleForDynamoDBCloudWatchContributorInsights",
            "Condition": {"StringLike": {"iam:AWSServiceName":
 "contributorinsights.dynamodb.amazonaws.com"}}
        },
        {
            "Effect": "Allow",
            "Action": [
                "dynamodb:UpdateContributorInsights"
            ],
            "Resource": "arn:aws:dynamodb:*:*:table/*"
        }
    ]
}
```

Für Tabellen, die mit einem KMS-Schlüssel verschlüsselt wurden, benötigt der Benutzer kms:Decrypt-Berechtigungen, um Contributor Insights aktualisieren zu können.

JSON

```
{
```

```
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "iam:CreateServiceLinkedRole",
            "Resource": "arn:aws:iam::*:role/aws-
service-role/contributorinsights.dynamodb.amazonaws.com/
AWSServiceRoleForDynamoDBCloudWatchContributorInsights",
            "Condition": {
                "StringLike": {
                    "iam:AWSServiceName":
 "contributorinsights.dynamodb.amazonaws.com"
                }
            }
        },
        {
            "Effect": "Allow",
            "Action": [
                "dynamodb:UpdateContributorInsights"
            ],
            "Resource": "arn:aws:dynamodb:*:*:table/*"
        },
        {
            "Effect": "Allow",
            "Resource": "arn:aws:kms:*:*:key/*",
            "Action": [
                "kms:Decrypt"
            ]
        }
    ]
}
```

Beispiel: Rufe den CloudWatch Contributor Insights-Regelbericht ab

Die folgende IAM-Richtlinie gewährt Berechtigungen zum Abrufen des CloudWatch Contributor Insights-Regelberichts.

JSON

```
{
    "Version":"2012-10-17",
```

```
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "cloudwatch:GetInsightRuleReport"
            ],
            "Resource": "arn:aws:cloudwatch:*:*:insight-rule/
DynamoDBContributorInsights*"
        }
    ]
}
```

Beispiel: Selektive Anwendung von CloudWatch Contributor Insights für DynamoDB-Berechtigungen basierend auf der Ressource

Mit der folgenden IAM-Richtlinie werden Rechte, die `ListContributorInsights` und die Aktionen `DescribeContributorInsights` gewährt und die`UpdateContributorInsights` Aktion für einen bestimmten globalen Sekundärindex verweigert.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "dynamodb:ListContributorInsights",
                "dynamodb:DescribeContributorInsights"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Deny",
            "Action": [
                "dynamodb:UpdateContributorInsights"
            ],
            "Resource": "arn:aws:dynamodb:us-west-2:123456789012:table/Books/
index/Author-index"
        }
    ]
```

```
}
```

Verwenden von serviceverknüpften Rollen für CloudWatch Contributor Insights for DynamoDB

CloudWatch Contributor Insights for DynamoDB verwendet AWS Identity and Access Management (IAM) service-verknüpfte Rollen. Eine serviceverknüpfte Rolle ist eine einzigartige Art von IAM-Rolle, die direkt mit CloudWatch Contributor Insights for DynamoDB verknüpft ist. Dienstbezogene Rollen sind von CloudWatch Contributor Insights für DynamoDB vordefiniert und beinhalten alle Berechtigungen, die der Dienst benötigt, um andere AWS Dienste in Ihrem Namen aufzurufen.

Eine dienstverknüpfte Rolle erleichtert die Einrichtung von CloudWatch Contributor Insights für DynamoDB, da Sie die erforderlichen Berechtigungen nicht manuell hinzufügen müssen. CloudWatch Contributor Insights for DynamoDB definiert die Berechtigungen seiner dienstbezogenen Rollen, und sofern nicht anders definiert, kann nur CloudWatch Contributor Insights for DynamoDB seine Rollen übernehmen. Die definierten Berechtigungen umfassen die Vertrauens- und Berechtigungsrichtlinie. Diese Berechtigungsrichtlinie kann keinen anderen IAM-Entitäten zugewiesen werden.

Informationen zu anderen Services, die serviceverknüpften Rollen unterstützen, finden Sie unter AWS -Services, die mit IAM funktionieren. Suchen Sie nach den Services, für die Ja in der Spalte Serviceverknüpfte Rolle angegeben ist. Wählen Sie über einen Link Ja aus, um die Dokumentation zu einer serviceverknüpften Rolle für diesen Service anzuzeigen.

Dienstbezogene Rollenberechtigungen für CloudWatch Contributor Insights for DynamoDB

CloudWatch Contributor Insights for DynamoDB verwendet die mit dem Dienst verknüpfte Rolle namens. AWSServiceRoleForDynamoDBCloudWatchContributorInsights Der Zweck der serviceverknüpften Rolle besteht darin, Amazon DynamoDB zu ermöglichen, Amazon CloudWatch Contributor Insights-Regeln, die für DynamoDB-Tabellen und globale Sekundärindizes erstellt wurden, in Ihrem Namen zu verwalten.

Die serviceverknüpfte Rolle `AWSServiceRoleForDynamoDBCloudWatchContributorInsights` vertraut darauf, dass die folgenden Services die Rolle annehmen:

- `contributorinsights.dynamodb.amazonaws.com`

Die Rollenberechtigungsrichtlinie ermöglicht es CloudWatch Contributor Insights for DynamoDB, die folgenden Aktionen für die angegebenen Ressourcen durchzuführen:

- Aktion: `Create and manage Insight Rules` für `DynamoDBContributorInsights`

Sie müssen Berechtigungen konfigurieren, damit eine juristische Stelle von IAM (z. B. Benutzer, Gruppe oder Rolle) eine serviceverknüpfte Rolle erstellen, bearbeiten oder löschen kann. Weitere Informationen finden Sie unter serviceverknüpfte Rollenberechtigungen im IAM-Benutzerhandbuch.

Eine serviceverknüpfte Rolle für CloudWatch Contributor Insights for DynamoDB erstellen

Sie müssen eine serviceverknüpfte Rolle nicht manuell erstellen. Wenn Sie Contributor Insights in der AWS-Managementkonsole, der oder der AWS API aktivieren AWS CLI, erstellt CloudWatch Contributor Insights for DynamoDB die serviceverknüpfte Rolle für Sie.

Wenn Sie diese serviceverknüpfte Rolle löschen und sie dann erneut erstellen müssen, können Sie dasselbe Verfahren anwenden, um die Rolle in Ihrem Konto neu anzulegen. Wenn Sie Contributor Insights aktivieren, erstellt CloudWatch Contributor Insights for DynamoDB die serviceverknüpfte Rolle erneut für Sie.

Bearbeiten einer serviceverknüpften Rolle für CloudWatch Contributor Insights for DynamoDB

CloudWatch Contributor Insights for DynamoDB ermöglicht es Ihnen nicht, die serviceverknüpfte Rolle zu bearbeiten. `AWSServiceRoleForDynamoDBCloudWatchContributorInsights` Da möglicherweise verschiedene Entitäten auf die Rolle verweisen, kann der Rollenname nach dem Erstellen einer serviceverknüpften Rolle nicht mehr geändert werden. Sie können jedoch die Beschreibung der Rolle mit IAM bearbeiten. Weitere Informationen finden Sie unter Bearbeiten einer serviceverknüpften Rolle im IAM-Benutzerhandbuch.

Löschen einer serviceverknüpften Rolle für CloudWatch Contributor Insights for DynamoDB

Sie müssen die Rolle `AWSServiceRoleForDynamoDBCloudWatchContributorInsights` nicht manuell löschen. Wenn Sie Contributor Insights in der AWS-Managementkonsole, der oder der AWS API deaktivieren AWS CLI, bereinigt CloudWatch Contributor Insights for DynamoDB die Ressourcen.

Sie können auch die IAM-Konsole, die AWS CLI oder die AWS API verwenden, um die mit dem Service verknüpfte Rolle manuell zu löschen. Sie müssen jedoch die Ressourcen für Ihre serviceverknüpfte Rolle zuerst manuell bereinigen, bevor Sie diese manuell löschen können.

> **Note**
>
> Wenn der Dienst CloudWatch Contributor Insights for DynamoDB die Rolle verwendet, wenn Sie versuchen, die Ressourcen zu löschen, schlägt das Löschen möglicherweise fehl. Wenn dies passiert, warten Sie einige Minuten und versuchen Sie es erneut.

So löschen Sie die serviceverknüpfte Rolle mit IAM

Verwenden Sie die IAM-Konsole, die oder die AWS API AWS CLI, um die dienstverknüpfte Rolle zu löschen. `AWSServiceRoleForDynamoDBCloudWatchContributorInsights` Weitere Informationen finden Sie unter Löschen einer serviceverknüpften Rolle im IAM-Benutzerhandbuch.

Bewährte Methoden für Design und Architektur mit DynamoDB

In diesem Abschnitt finden Sie schnell Empfehlungen für die Maximierung der Leistung und die Minimierung der Durchsatzkosten bei der Arbeit mit DynamoDB.

Themen

- NoSQL-Design für DynamoDB
- Verwenden der DynamoDB Well-Architected Lens zur Optimierung Ihres DynamoDB-Workloads
- Bewährte Methoden für das Entwerfen und effektive Verwenden von Partitionsschlüsseln in DynamoDB
- Bewährte Methoden für die Verwendung von Sortierschlüsseln zur Organisation von Daten in DynamoDB
- Bewährte Methoden für die Verwendung sekundärer Indexe in DynamoDB
- Bewährte Methoden für das Speichern großer Elemente und Attribute in DynamoDB
- Bewährte Methoden für die Verarbeitung von Zeitreihendaten in DynamoDB
- Bewährte Methoden für die Verwaltung von many-to-many Beziehungen in DynamoDB-Tabellen
- Bewährte Methoden für das Abfragen und Scannen von Daten in DynamoDB
- Bewährte Methoden für das DynamoDB-Tabellendesign
- Verwenden von globalen Tabellen in DynamoDB
- Bewährte Methoden für die Verwaltung der Steuerebene in DynamoDB
- Bewährte Methoden für die Verwendung von Massendatenoperationen in DynamoDB
- Bewährte Methoden für den Umgang mit gleichzeitigen Updates in DynamoDB
- Bewährte Methoden zum Verständnis Ihrer AWS Abrechnungs- und Nutzungsberichte in DynamoDB
- Migrieren einer DynamoDB-Tabelle zwischen zwei Konten
- Vorgeschriebene Anleitung zur Integration von DAX in DynamoDB-Anwendungen
- Überlegungen bei der Verwendung AWS PrivateLink für Amazon DynamoDB

NoSQL-Design für DynamoDB

NoSQL-Datenbanksysteme wie Amazon DynamoDB verwenden alternative Modelle für die Verwaltung von Daten, wie Schlüssel-Wert-Paare oder Dokumentspeicher. Wenn Sie von einem relationalen Datenbankmanagementsystem zu einem NoSQL-Datenbanksystem wie DynamoDB wechseln, ist es wichtig, die wesentlichen Unterschiede und die spezifischen Designansätze zu verstehen.

Themen

- Unterschiede zwischen einem relationalen Datendesign und NoSQL
- Zwei Schlüsselkonzepte für das NoSQL-Design
- Ansatz für ein NoSQL-Design
- NoSQL-Workbench für DynamoDB

Unterschiede zwischen einem relationalen Datendesign und NoSQL

Relationale Datenbankmanagementsysteme (RDBMS) und nNoSQL-Datenbanken besitzen unterschiedliche Vor- und Nachteile:

- In RDBMS können Daten flexibel abgerufen werden, Abfragen sind jedoch vergleichsweise kostspielig und können bei einer großen Zahl von Zugriffen nicht gut skaliert werden (siehe Erste Schritte für die Modellierung relationaler Daten in DynamoDB).
- In einer NoSQL-Datenbank wie DynamoDB können Daten effizient mithilfe einer begrenzten Anzahl von Möglichkeiten abgerufen werden, die ansonsten möglicherweise kostspielig und langsam sind.

Diese Unterschiede führen dazu, dass das Datenbankdesign der beiden Systeme verschieden ist:

- In RDBMS entwerfen Sie das Design im Hinblick auf Flexibilität, ohne sich um Implementierungsdetails oder Leistung zu kümmern. Die Optimierung von Abfragen wirkt sich im Allgemeinen nicht auf das Schemadesign aus, eine Standardisierung ist jedoch wichtig.
- In DynamoDB entwerfen Sie das Schema im Hinblick darauf, die häufigsten und wichtigsten Abfragen so schnell und kostengünstig wie möglich ausführen zu können. Die Datenstrukturen sind an die spezifischen Anforderungen Ihrer geschäftlichen Anwendungsfälle angepasst.

Zwei Schlüsselkonzepte für das NoSQL-Design

Das NoSQL-Design erfordert einen anderen Ansatz als das RDBMS-Design. Sie können für ein RDBMS ein standardisiertes Datenmodell entwickeln, ohne sich Gedanken über Zugriffsmuster machen zu müssen. Anschließend können Sie es erweitern, wenn neue Fragen und Abfrageanforderungen entstehen. Sie können jeden einzelnen Typ von Daten in einer eigenen Tabelle organisieren.

Wie sich das NoSQL-Design unterscheidet

- Im Fall von DynamoDB sollten Sie nicht mit der Entwicklung des Schemas beginnen, bis Sie die Fragen kennen, die es beantworten soll. Es ist äußerst wichtig, die geschäftlichen Probleme und Anwendungsfälle vor der Entwicklung des Schemas zu kennen.
- Sie sollten in einer DynamoDB-Anwendung so wenig Tabellen wie möglich verwenden. Weniger Tabellen sorgen dafür, dass die Dinge besser skalierbar sind, weniger Berechtigungsmanagement erforderlich sind und der Overhead für Ihre DynamoDB-Anwendung reduziert wird. Dies kann auch dazu beitragen, die Backup-Kosten insgesamt niedrig zu halten.

Ansatz für ein NoSQL-Design

Der erste Schritt beim Design der DynamoDB-Anwendung besteht in der Identifizierung der spezifischen Abfragemuster, die das System unterstützen muss.

Insbesondere ist es wichtig, drei Basiseigenschaften der Zugriffsmuster Ihrer Anwendung zu kennen, bevor Sie mit dem Design beginnen:

- Datenmenge: Wenn Sie wissen, wie viele Daten gleichzeitig gespeichert und angefordert werden, kann dies helfen, die effektivste Partitionierung der Daten zu ermitteln.
- Datenform: Statt Daten neu zu gestalten, wenn eine Abfrage verarbeitet wird (wie im Fall von RDBMS-Systemen), organisieren NoSQL-Datenbanken Daten so, dass ihre Form in der Datenbank dem entspricht, was abgefragt werden wird. Dies ist ein entscheidender Faktor, um Geschwindigkeit und Skalierbarkeit zu verbessern.
- Datengeschwindigkeit: DynamoDB wird durch die Erhöhung der Anzahl der physischen Partitionen skaliert, die für die Verarbeitung von Abfragen verfügbar sind, sowie durch die effiziente Verteilung von Daten über diese Partitionen. Wenn Sie im Voraus wissen, wie hoch die maximale Abfragelast sein wird, kann es hilfreich sein, zu bestimmen, wie Daten partitioniert werden, um die Kapazität optimal zu nutzen. I/O

Nach der Identifizierung spezifischer Abfrageanforderungen können Sie die Daten nach allgemeinen Grundsätzen organisieren, denen die Leistung unterliegt:

- Speichern Sie verwandte Daten zusammen. Studien haben gezeigt, dass das Prinzip der „Referenzlokalität“, d. h. die Zusammenführung verwandter Daten an einem Ort, ein Schlüsselfaktor für die Verbesserung der Leistung und der Reaktionszeiten in NoSQL-Systemen ist, genauso wie es sich vor vielen Jahren als wichtig für die Optimierung von Routing-Tabellen erwiesen hat.

  Allgemein sollten Sie in einer DynamoDB-Anwendung so wenig Tabellen wie möglich verwenden.

  Ausnahmen hiervon sind Fälle, in denen große Mengen von Zeitreihendaten oder Datensätze mit sehr unterschiedlichen Zugriffsmustern vorhanden sind. Eine einzelne Tabelle mit umgekehrten Indizes kann in der Regel einfache Abfragen unterstützen, um die komplexen hierarchischen Datenstrukturen zu erstellen und abzurufen, die Ihre Anwendung benötigt.
- Verwenden Sie eine Sortierreihenfolge. Verwandte Elemente können zusammen gruppiert und effizient abgefragt werden, wenn ihr Schlüsseldesign dazu führt, dass sie gemeinsam sortiert werden. Dies stellt eine wichtige Strategie für das NoSQL-Design dar.
- Verteilen Sie Abfragen. Es ist auch wichtig, dass eine große Anzahl von Abfragen nicht auf einen Teil der Datenbank konzentriert wird, wo sie die I/O Kapazität überschreiten könnten. Sie sollten stattdessen Datenschlüssel entwerfen, um die Zugriffe so gleichmäßig wie möglich über die Partitionen zu verteilen und Konzentrationspunkte zu vermeiden.
- Verwenden Sie globale sekundäre Indizes. Durch die Erstellung spezifischer globaler sekundärer Indizes können Sie eine größere Zahl unterschiedlicher Abfragen unterstützen, als Ihre Haupttabelle unterstützen kann. Gleichzeitig sind diese nach wie vor schnell und vergleichsweise kostengünstig.

Diese allgemeinen Grundsätze führen zu einigen häufigen Designmustern, die Sie verwenden können, um Daten in DynamoDB effizient zu modellieren.

NoSQL-Workbench für DynamoDB

NoSQL-Workbench für DynamoDB ist eine plattformübergreifende clientseitige GUI-Anwendung für moderne Datenbankentwicklung und -operationen. Sie ist für Windows, macOS und Linux verfügbar. NoSQL Workbench ist ein visuelles Entwicklungstool, das Funktionen zur Datenmodellierung, Datenvisualisierung, Beispieldatengenerierung und Abfrageentwicklung bereitstellt, mit denen Sie DynamoDB-Tabellen entwerfen, erstellen, abfragen und verwalten können. Mit NoSQL-Workbench

für DynamoDB können Sie neue Datenmodelle aus vorhandenen Datenmodellen erstellen oder Modelle basierend auf vorhandenen Datenmodellen entwerfen, die den Datenzugriffsmustern Ihrer Anwendung entsprechen. Sie können das gestaltete Datenmodell am Ende des Prozesses auch importieren und exportieren. Weitere Informationen finden Sie unter Erstellen von Datenmodellen mit NoSQL Workbench.

Verwenden der DynamoDB Well-Architected Lens zur Optimierung Ihres DynamoDB-Workloads

Dieser Abschnitt behandelt die Amazon DynamoDB Well-Architected Lens, eine Sammlung von Gestaltungsprinzipien und Leitlinien für die Gestaltung gut strukturierter DynamoDB-Workloads.

Optimierung der Kosten für DynamoDB-Tabellen

In diesem Abschnitt werden bewährte Methoden zur Kostenoptimierung für Ihre vorhandenen DynamoDB-Tabellen vorgestellt. Sie sollten sich die folgenden Strategien zur Kostenoptimierung ansehen, um herauszufinden, welche davon Ihren Anforderungen am besten entspricht, und sie iterativ angehen. Jede Strategie bietet einen Überblick darüber, was sich auf Ihre Kosten auswirken könnte und auf welche Anzeichen Sie achten sollten. Darüber hinaus finden Sie ausführliche Anleitungen, wie Sie die Kosten reduzieren können.

Themen

- Auswerten Ihrer Kosten auf Tabellenebene
- Auswerten des Kapazitätsmodus Ihrer DynamoDB-Tabelle
- Auswerten der Auto-Scaling-Einstellungen Ihrer DynamoDB-Tabelle
- Auswerten Ihrer DynamoDB-Tabellenklassenauswahl
- Ermitteln von ungenutzten Ressourcen in DynamoDB
- Auswerten Ihrer DynamoDB-Tabellennutzungsmuster
- Evaluieren Ihrer Nutzung von DynamoDB Streams
- Auswerten Ihrer bereitgestellten Kapazität, um eine angemessene Bereitstellung in Ihrer DynamoDB-Tabelle zu erzielen

Auswerten Ihrer Kosten auf Tabellenebene

Mit dem Cost Explorer Explorer-Tool im AWS-Managementkonsole können Sie die Kosten nach Typ aufgeschlüsselt sehen, z. B. Lese-, Schreib-, Speicher- und Backup-Gebühren. Sie können diese Kosten auch nach Zeitraum zusammengefasst sehen, beispielsweise die Kosten für einen Monat oder einen Tag.

Administratoren können vor dem Problem stehen, dass sie nur die Kosten für eine bestimmte Tabelle prüfen müssen. Einige dieser Daten sind über die DynamoDB-Konsole oder über Aufrufe der `DescribeTable`-API verfügbar. In Cost Explorer können Sie jedoch standardmäßig nicht nach den mit einer bestimmten Tabelle verbundenen Kosten filtern oder gruppieren. In diesem Abschnitt erfahren Sie, wie Sie mithilfe von Tagging eine Kostenanalyse für eine einzelne Tabelle in Cost Explorer durchführen.

Themen

- So zeigen Sie die Kosten einer einzelnen DynamoDB-Tabelle an
- Standardansicht von Cost Explorer
- So verwenden Sie Tabellen-Tags in Cost Explorer

So zeigen Sie die Kosten einer einzelnen DynamoDB-Tabelle an

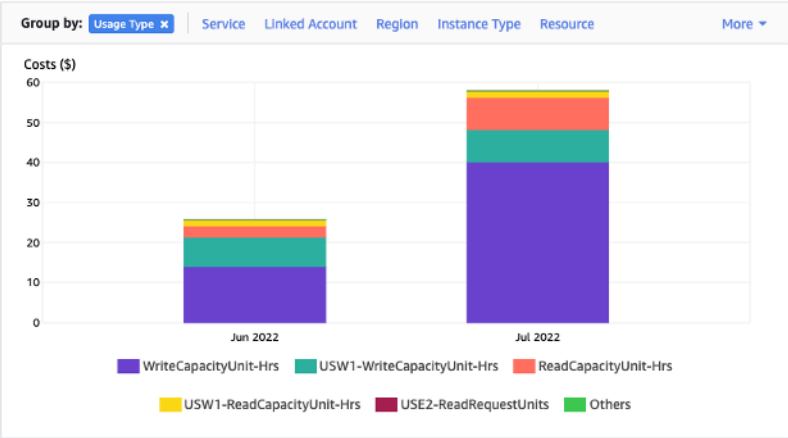
Sowohl Amazon DynamoDB AWS-Managementkonsole als auch die `DescribeTable` API zeigen Ihnen Informationen zu einer einzelnen Tabelle, einschließlich des Primärschlüsselschemas, aller Indizes in der Tabelle sowie der Größe und Anzahl der Elemente der Tabelle und aller Indizes. Anhand der Größe der Tabelle sowie der Größe der Indizes können die monatlichen Speicherkosten für Ihre Tabelle berechnet werden. Beispiel: 0,25 USD pro GB in der Region us-east-1.

Wenn sich die Tabelle im Modus bereitgestellter Kapazität befindet, werden auch die aktuellen RCU- und WCU-Einstellungen zurückgegeben. Mithilfe dieser Werte könnten die aktuellen Lese- und Schreibkosten für die Tabelle berechnet werden, doch diese Kosten können sich ändern, insbesondere wenn die Tabelle mit Auto Scaling konfiguriert wurde.

> **Note**
>
> Wenn sich die Tabelle im On-Demand-Kapazitätsmodus befindet, ist `DescribeTable` nicht zur Schätzung der Durchsatzkosten geeignet, da diese auf der Grundlage der tatsächlichen Nutzung und nicht nach der bereitgestellten Nutzung in einem Zeitraum abgerechnet werden.

Standardansicht von Cost Explorer

Die Standardansicht von Cost Explorer enthält Diagramme, in denen die Kosten verbrauchter Ressourcen wie Durchsatz und Speicher dargestellt sind. Sie können die Kosten nach Zeitraum gruppieren, z. B. die Gesamtkosten pro Monat oder Tag. Die Kosten für Speicher, Lese- und Schreibvorgänge sowie andere Funktionen können ebenfalls aufgeschlüsselt und verglichen werden.



So verwenden Sie Tabellen-Tags in Cost Explorer

Standardmäßig bietet Cost Explorer keine Zusammenfassung der Kosten für eine bestimmte Tabelle, da die Kosten mehrerer Tabellen zu einer Summe zusammengefasst werden. Sie können jedoch AWS -Ressourcen-Tagging verwenden, um jede Tabelle mit einem Metadaten-Tag zu identifizieren. Tags sind Schlüssel-Wert-Paare, die Sie für verschiedenste Zwecke verwenden können, z. B. um alle zu einem Projekt oder einer Abteilung gehörigen Ressourcen zu identifizieren. In diesem Beispiel gehen wir davon aus, dass Sie eine Tabelle mit dem Namen haben. MyTable

1. Setze ein Tag mit dem Schlüssel von table_name und dem Wert von. MyTable
2. Aktivieren Sie das Tag in Cost Explorer und filtern Sie dann nach dem Tag-Wert, um einen besseren Einblick in die Kosten der einzelnen Tabellen zu erhalten.

> **Note**
>
> Es kann ein oder zwei Tage dauern, bis das Tag in Cost Explorer angezeigt wird.

Sie können Metadaten-Tags selbst in der Konsole oder über Automatisierung wie die AWS CLI oder das AWS SDK festlegen. Überlegen Sie sich, im Rahmen des neuen Tabellenerstellungsprozesses Ihrer Organisation die Festlegung eines table_name-Tags zu erfordern. Für vorhandene Tabellen

steht ein Python-Dienstprogramm zur Verfügung, das diese Tags findet und auf alle vorhandenen Tabellen in einer bestimmten Region in Ihrem Konto anwendet. Weitere Informationen finden Sie unter Epononymous Table Tagger on GitHub.

Auswerten des Kapazitätsmodus Ihrer DynamoDB-Tabelle

In diesem Abschnitt erhalten Sie einen Überblick darüber, wie Sie den geeigneten Kapazitätsmodus für Ihre DynamoDB-Tabelle auswählen. Jeder Modus ist auf die Anforderungen einer anderen Workload abgestimmt, sowohl was die Reaktionsfähigkeit auf Durchsatzänderungen, als auch was die Abrechnung der Nutzung anbelangt. Sie müssen diese Faktoren bei Ihrer Entscheidung abwägen.

Themen

- Verfügbare Tabellenkapazitätsmodi
- Fälle, in denen der On-Demand-Modus ausgewählt werden sollte
- Fälle, in denen der Modus bereitgestellter Kapazität ausgewählt werden sollte
- Zusätzliche Faktoren, die bei der Auswahl eines Tabellenkapazitätsmodus zu berücksichtigen sind

Verfügbare Tabellenkapazitätsmodi

Beim Erstellen einer DynamoDB-Tabelle müssen Sie entweder den On-Demand-Kapazitätsmodus oder den Modus bereitgestellter Kapazität auswählen.

Sie können Tabellen innerhalb eines fortlaufenden 24-stündigen Zeitfensters bis zu viermal vom Modus mit bereitgestellter Kapazität auf den On-Demand-Modus umstellen. Sie können Tabellen jederzeit vom On-Demand-Modus auf den Modus mit bereitgestellter Kapazität umstellen.

On-Demand-Kapazitätsmodus

Der On-Demand-Kapazitätsmodus ist so konzipiert, dass die Kapazität Ihrer DynamoDB-Tabelle nicht mehr geplant oder bereitgestellt werden muss. In diesem Modus bewältigt Ihre Tabelle Anforderungen an die Tabelle sofort, ohne dass Ressourcen hoch- oder herunterskaliert werden müssen (bis zum Zweifachen des vorherigen Spitzendurchsatzes der Tabelle).

DynamoDB On-Demand bietet pay-per-request Preise für Lese- und Schreibanforderungen, sodass Sie nur für das bezahlen, was Sie tatsächlich nutzen.

Modus bereitgestellter Kapazität

Beim Modus bereitgestellter Kapazität handelt es sich um ein traditionelleres Modell, bei dem Sie entweder direkt oder mithilfe von Auto Scaling festlegen können, wie viel Kapazität die Tabelle für

Anforderungen zur Verfügung hat. Da zu jedem Zeitpunkt eine bestimmte Kapazität für die Tabelle bereitgestellt wird, basiert die Abrechnung auf der bereitgestellten Gesamtkapazität und nicht auf der Anzahl der verbrauchten Anforderungen. Wenn die zugewiesene Kapazität überschritten wird, kann dies zudem dazu führen, dass die Tabelle Anforderungen ablehnt und die Erfahrung Ihrer Anwendungsbenutzer beeinträchtigt wird.

Der Modus bereitgestellter Kapazität erfordert eine konstante Überwachung, um eine Balance zwischen einer Über- und Unterbereitstellung von Kapazität für die Tabelle zu finden, damit es nur zu einer geringen Drosselung kommt und die Kosten optimiert werden können.

Fälle, in denen der On-Demand-Modus ausgewählt werden sollte

Im Hinblick auf die Kostenoptimierung ist der On-Demand-Modus die beste Wahl, wenn Sie eine ähnliche Workload wie in den folgenden Diagrammen haben.

Die folgenden Faktoren bestimmen diese Art von Workload:

- Datenverkehrsmuster, das sich im Laufe der Zeit weiterentwickelt
- Variables Volumen der Anforderungen (aufgrund von Batch-Workloads)
- Unvorhersehbares Timing der Anforderungen (dadurch Datenverkehrsspitzen)
- Rückgang auf Null oder unter 30 % des Spitzenwerts zu einer bestimmten Stunde

Wenn bei Workloads mit den oben genannten Faktoren Auto Scaling verwendet wird, um eine ausreichende Kapazität der Tabelle für Datenverkehrsspitzen aufrechtzuerhalten, führt dies wahrscheinlich zu einer Überbereitstellung und unnötig hohen Kosten oder zu einer Unterbereitstellung und unnötigen Drosselung von Anforderungen. Der On-Demand-Kapazitätsmodus ist die bessere Wahl, da er Datenverkehrsschwankungen bewältigen kann, ohne dass Sie die Kapazität vorhersehen oder anpassen müssen.

Mit dem pay-per-request Preismodell des On-Demand-Modus müssen Sie sich keine Gedanken über ungenutzte Kapazitäten machen, da Sie nur für den Durchsatz zahlen, den Sie tatsächlich nutzen. Die Abrechnung erfolgt pro verbrauchter Lese- oder Schreibanforderung, sodass Ihre Kosten direkt Ihre tatsächliche Nutzung widerspiegeln. So können Sie Kosten und Leistung auf einfache Weise ausgleichen. Optional können Sie auch den maximalen Lese- oder Schreibdurchsatz (oder beides) pro Sekunde für einzelne On-Demand-Tabellen und globale sekundäre Indizes konfigurieren, um Kosten und Nutzen zu begrenzen. Weitere Informationen finden Sie unter Maximaler Durchsatz für On-Demand-Tabellen.

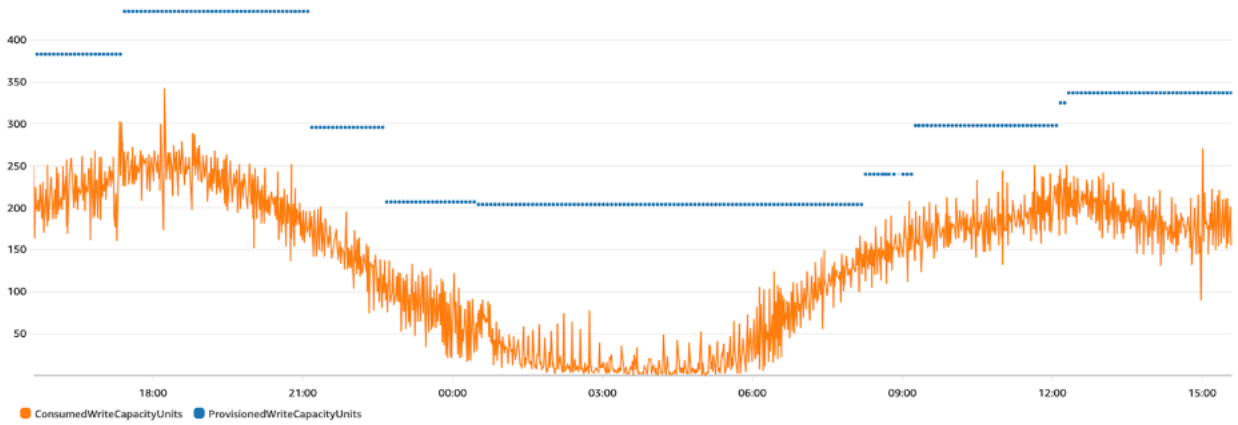
Fälle, in denen der Modus bereitgestellter Kapazität ausgewählt werden sollte

Ein idealer Workload für den Modus bereitgestellter Kapazität ist ein Workload mit einem stabileren und besser vorhersehbaren Nutzungsmuster, wie im folgenden Diagramm dargestellt.

> **Note**
>
> Wir empfehlen, die Kennzahlen in einem bestimmten Zeitraum, z. B. 14 Tagen, zu überprüfen, bevor Sie Maßnahmen in Bezug auf Ihre bereitgestellte Kapazität ergreifen.

Die folgenden Faktoren bestimmen diese Art von Workload:

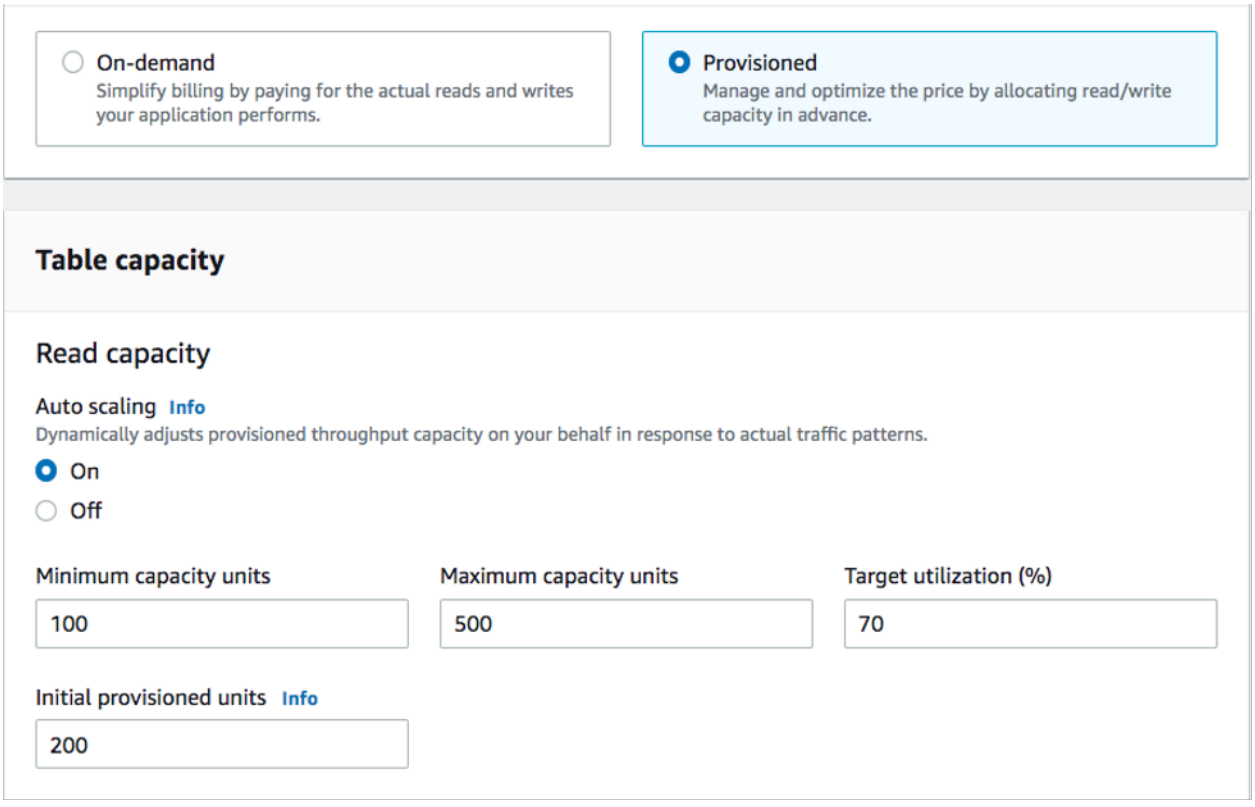
- Stetiger, vorhersehbarer und zyklischer Datenverkehr für eine bestimmte Stunde oder einen bestimmten Tag
- Begrenzte kurzfristige Datenverkehrsspitzen



Da das Datenverkehrsvolumen zu einer bestimmten Stunde oder an einem bestimmten Tag stabiler ist, können Sie die bereitgestellte Kapazität der Tabelle relativ nah an der tatsächlich verbrauchten

Kapazität der Tabelle festlegen. Für die Kostenoptimierung einer Tabelle mit bereitgestellter Kapazität geht es letztendlich darum, die bereitgestellte Kapazität (blaue Linie) so nah wie möglich an die verbrauchte Kapazität (orange Linie) heranzuführen, ohne `ThrottledRequests` für die Tabelle zu erhöhen. Der Abstand zwischen den beiden Linien ist einerseits verschwendete Kapazität, andererseits eine Versicherung gegen eine schlechte Benutzererfahrung aufgrund von Drosselung. Wenn Sie die Durchsatzanforderungen Ihrer Anwendung vorhersehen können und die Kostenvorhersehbarkeit des Kontrollierens der Lese- und Schreibkapazität bevorzugen, sollten Sie weiterhin bereitgestellte Tabellen verwenden.

DynamoDB bietet eine automatische Skalierung für Tabellen mit bereitgestellter Kapazität, die dies automatisch für Sie ausgleicht. Auf diese Weise können Sie Ihre verbrauchte Kapazität im Laufe des Tages verfolgen und die Kapazität der Tabelle anhand einiger Variablen festlegen. Wenn Sie Auto Scaling verwenden, wird Ihre Tabelle überdimensioniert sein und Sie müssen das Verhältnis zwischen der Anzahl der Drosselungen und den überdimensionierten Kapazitätseinheiten optimieren, um Ihre Workload-Anforderungen zu erfüllen.



Einheiten für minimale Kapazität

Sie können die Mindestkapazität einer Tabelle festlegen, um die Drosselung zu begrenzen. Die Kosten der Tabelle werden dadurch allerdings nicht reduziert. Wenn es bei Ihrer Tabelle Zeiten geringer Auslastung gibt, gefolgt von einer plötzlichen hohen Auslastung, kann durch Festlegung

des Minimums verhindert werden, dass die Tabellenkapazität durch die automatische Skalierung zu niedrig eingestellt wird.

Einheiten für maximale Kapazität

Sie können die maximale Kapazität einer Tabelle festlegen, um zu verhindern, dass eine Tabelle stärker als beabsichtigt skaliert wird. Sie sollten die Anwendung eines Maximums für Entwicklungs- oder Testtabellen in Betracht ziehen, bei denen umfangreiche Lasttests nicht erwünscht sind. Sie können für jede Tabelle ein Maximum festlegen, sollten diese Einstellung jedoch regelmäßig anhand der Tabellen-Baseline bewerten, wenn Sie sie in der Produktion verwenden, um eine versehentliche Drosselung zu verhindern.

Zielauslastung

Die Festlegung einer Zielauslastung der Tabelle ist das primäre Mittel zur Kostenoptimierung für eine Tabelle mit bereitgestellter Kapazität. Wenn Sie hier einen niedrigeren Prozentwert festlegen, kommt es zu einer stärkeren Überbereitstellung für die Tabelle. Dies führt zu höheren Kosten, verringert jedoch das Risiko einer Drosselung. Wenn Sie hier einen höheren Prozentwert festlegen, kommt es zu einer weniger starken Überbereitstellung für die Tabelle. Allerdings erhöht sich das Risiko einer Drosselung.

Zusätzliche Faktoren, die bei der Auswahl eines Tabellenkapazitätsmodus zu berücksichtigen sind

Bei der Entscheidung zwischen den beiden Modi sollten einige zusätzliche Faktoren berücksichtigt werden.

Auslastung bereitgestellter Kapazität

Um festzustellen, wann der On-Demand-Modus weniger kosten würde als die bereitgestellte Kapazität, ist es hilfreich, einen Blick auf die Auslastung Ihrer bereitgestellten Kapazität zu werfen. Diese bezieht sich darauf, wie effizient die zugewiesenen (oder „bereitgestellten“) Ressourcen genutzt werden. Der On-Demand-Modus kostet weniger für Workloads, bei denen die durchschnittliche Auslastung der bereitgestellten Kapazität unter 35 % liegt. In vielen Fällen kann es sogar bei Workloads mit einer Auslastung der bereitgestellten Kapazität von mehr als 35 % kostengünstiger sein, den On-Demand-Modus zu verwenden, insbesondere wenn der Workload Phasen geringer Aktivität und gelegentliche Spitzen aufweist.

Reservierte Kapazität

Für Tabellen mit bereitgestellter Kapazität bietet DynamoDB die Möglichkeit, reservierte Kapazität für Ihre Lese- und Schreibkapazität zu erwerben (replicated write capacity units (rWCU, replizierte

Schreibkapazitätseinheiten) und Standard-IA-Tabellen sind zurzeit nicht zulässig). Reservierte Kapazität bietet erhebliche Rabatte gegenüber den Standardpreisen für bereitgestellte Kapazität.

Berücksichtigen Sie bei der Entscheidung zwischen den beiden Tabellenmodi die Auswirkungen dieses zusätzlichen Rabatts auf die Tabellenkosten. In einigen Fällen kann es bei einem relativ unvorhersehbaren Workload günstiger sein, ihr in einer überdimensionierten Tabelle mit reservierter Kapazität auszuführen.

Verbessern der Vorhersehbarkeit Ihrer Workload

In einigen Situationen kann eine Workload sowohl ein vorhersehbares als auch ein unvorhersehbares Muster aufweisen. Dies kann zwar leicht mit einer On-Demand-Tabelle unterstützt werden, die Kosten werden jedoch wahrscheinlich niedriger sein, wenn die unvorhersehbaren Muster in der Workload verbessert werden können.

Eine der häufigsten Ursachen für diese Muster sind Batch-Importe. Bei dieser Art von Datenverkehr kann die Basiskapazität der Tabelle oft so weit überschritten werden, dass es bei der Ausführung zu einer Drosselung kommen würde. Um eine Workload wie diese in einer Tabelle mit bereitgestellter Kapazität auszuführen, sollten Sie die folgenden Optionen in Betracht ziehen:

- Wenn die Batch-Ausführung zu festgelegten Zeiten erfolgt, können Sie vor der Ausführung eine Erhöhung Ihrer Auto-Scaling-Kapazität planen.
- Wenn die Batch-Ausführung zufällig erfolgt, sollten Sie versuchen, die Ausführungszeit zu verlängern, anstatt eine möglichst schnelle Ausführung anzustreben.
- Fügen Sie dem Import eine Anlaufzeit hinzu, wenn der Import langsam beginnt, aber über einige Minuten allmählich schneller wird, bis Auto Scaling die Möglichkeit hatte, mit der Anpassung der Tabellenkapazität zu beginnen.

Auswerten der Auto-Scaling-Einstellungen Ihrer DynamoDB-Tabelle

In diesem Abschnitt erfahren Sie, wie Sie die Auto-Scaling-Einstellungen in Ihren DynamoDB-Tabellen auswerten können. Amazon DynamoDB Auto Scaling ist eine Funktion, die den Durchsatz von Tabellen und globalen sekundären Indizes (GSI) auf der Grundlage Ihres Anwendungsdatenverkehrs und Ihrer Zielauslastungsmetrik verwaltet. Dadurch GSIs wird sichergestellt, dass Ihre Tabellen oder Tabellen über die für Ihre Anwendungsmuster erforderliche Kapazität verfügen.

Der AWS Auto Scaling-Dienst überwacht Ihre aktuelle Tabellenauslastung und vergleicht sie mit dem Zielnutzungswert:`TargetValue`. Wenn es an der Zeit ist, die zugewiesene Kapazität zu erhöhen oder zu verringern, werden Sie benachrichtigt.

Themen

- Grundlegende Informationen zu Ihren Auto-Scaling-Einstellungen
- So ermitteln Sie Tabellen mit geringer Zielauslastung (<= 50 %)
- So bewältigen Sie Workloads mit saisonalen Schwankungen
- So bewältigen Sie stark schwankende Workloads mit unbekannten Mustern
- So bewältigen Sie Workloads mit verknüpften Anwendungen

Grundlegende Informationen zu Ihren Auto-Scaling-Einstellungen

Die Festlegung des richtigen Werts für die Zielauslastung, den ersten Schritt und die Endwerte erfordert die Beteiligung Ihres Operations-Teams. So können Sie die Werte, die zur Auslösung der AWS -Auto-Scaling-Richtlinien verwendet werden, auf der Grundlage der Anwendungsnutzung in der Vergangenheit angemessen definieren. Das Auslastungsziel ist der Prozentsatz Ihrer Gesamtkapazität, der in einem bestimmten Zeitraum erreicht werden muss, bevor die Auto-Scaling-Regeln angewendet werden.

Wenn Sie ein hohes Auslastungsziel (ein Ziel von rund 90 %) festlegen, muss der Datenverkehr einige Zeit einen Wert von über 90 % erreichen, bevor Auto Scaling aktiviert wird. Ein hohes Auslastungsziel sollten Sie nur verwenden, wenn Ihre Anwendung sehr konstant arbeitet und keine Datenverkehrsspitzen verzeichnet.

Wenn Sie eine sehr niedrige Auslastung (ein Ziel von weniger als 50 %) festlegen, wird eine Auto-Scaling-Richtlinie erst ausgelöst, wenn Ihre Anwendung 50 % der bereitgestellten Kapazität erreicht. Sofern Ihr Anwendungsdatenverkehr nicht extrem schnell zunimmt, führt dies in der Regel zu ungenutzter Kapazität und einer Ressourcenverschwendung.

So ermitteln Sie Tabellen mit geringer Zielauslastung (<= 50 %)

Sie können entweder das AWS CLI oder verwenden AWS-Managementkonsole , um die `TargetValues` für Ihre Auto Scaling-Richtlinien in Ihren DynamoDB-Ressourcen geltenden Richtlinien zu überwachen und zu identifizieren:

AWS CLI

1. Verwenden Sie den folgenden Befehl, um die gesamte Ressourcenliste zurückzugeben:

```
aws application-autoscaling describe-scaling-policies --service-namespace
 dynamodb
```

Dieser Befehl gibt die gesamte Liste der Auto-Scaling-Richtlinien zurück, die für jede DynamoDB-Ressource ausgegeben werden. Wenn Sie nur die Ressourcen aus einer bestimmten Tabelle abrufen möchten, können Sie den `–resource-id parameter` hinzufügen. Beispiel:

```
aws application-autoscaling describe-scaling-policies --service-namespace
 dynamodb --resource-id "table/<table-name>”
```

2. Verwenden Sie den folgenden Befehl, um nur die Auto-Scaling-Richtlinien für einen bestimmten GSI zurückzugeben:

```
aws application-autoscaling describe-scaling-policies --service-namespace
 dynamodb --resource-id "table/<table-name>/index/<gsi-name>”
```

Die für die Auto-Scaling-Richtlinien interessanten Werte sind unten hervorgehoben. Wir möchten einen Zielwert von über 50 % sicherstellen, um eine übermäßige Bereitstellung zu vermeiden. Das Ergebnis sollte etwa wie folgt aussehen:

```
{
    "ScalingPolicies": [
        {
            "PolicyARN": "arn:aws:autoscaling:<region>:<account-
id>:scalingPolicy:<uuid>:resource/dynamodb/table/<table-name>/index/<index-
name>:policyName/$<full-gsi-name>-scaling-policy",
            "PolicyName": $<full-gsi-name>”,
            "ServiceNamespace": "dynamodb",
            "ResourceId": "table/<table-name>/index/<index-name>",
            "ScalableDimension": "dynamodb:index:WriteCapacityUnits",
            "PolicyType": "TargetTrackingScaling",
            "TargetTrackingScalingPolicyConfiguration": {
                "TargetValue": 70.0,
                "PredefinedMetricSpecification": {
                    "PredefinedMetricType": "DynamoDBWriteCapacityUtilization"
```

```
                }
            },
            "Alarms": [
                ...
            ],
            "CreationTime": "2022-03-04T16:23:48.641000+10:00"
        },
        {
            "PolicyARN": "arn:aws:autoscaling:<region>:<account-
id>:scalingPolicy:<uuid>:resource/dynamodb/table/<table-name>/index/<index-
name>:policyName/$<full-gsi-name>-scaling-policy",
            "PolicyName":$<full-gsi-name>”,
            "ServiceNamespace": "dynamodb",
            "ResourceId": "table/<table-name>/index/<index-name>",
            "ScalableDimension": "dynamodb:index:ReadCapacityUnits",
            "PolicyType": "TargetTrackingScaling",
            "TargetTrackingScalingPolicyConfiguration": {
                "TargetValue": 70.0,
                "PredefinedMetricSpecification": {
                    "PredefinedMetricType": "DynamoDBReadCapacityUtilization"
                }
            },
            "Alarms": [
                ...
            ],
            "CreationTime": "2022-03-04T16:23:47.820000+10:00"
        }
    ]
}
```

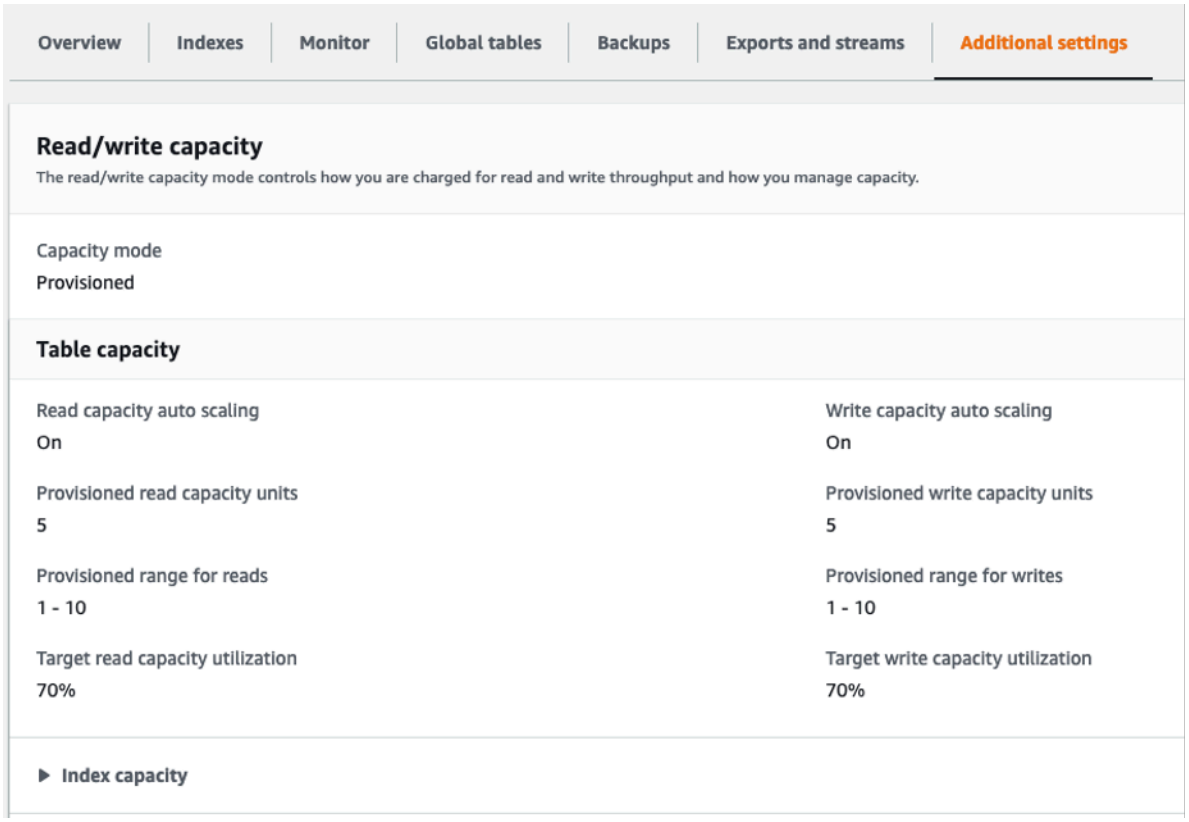
AWS-Managementkonsole

1. Melden Sie sich bei der an AWS-Managementkonsole und öffnen Sie die DynamoDB-Konsole unter. https://console.aws.amazon.com/dynamodb/

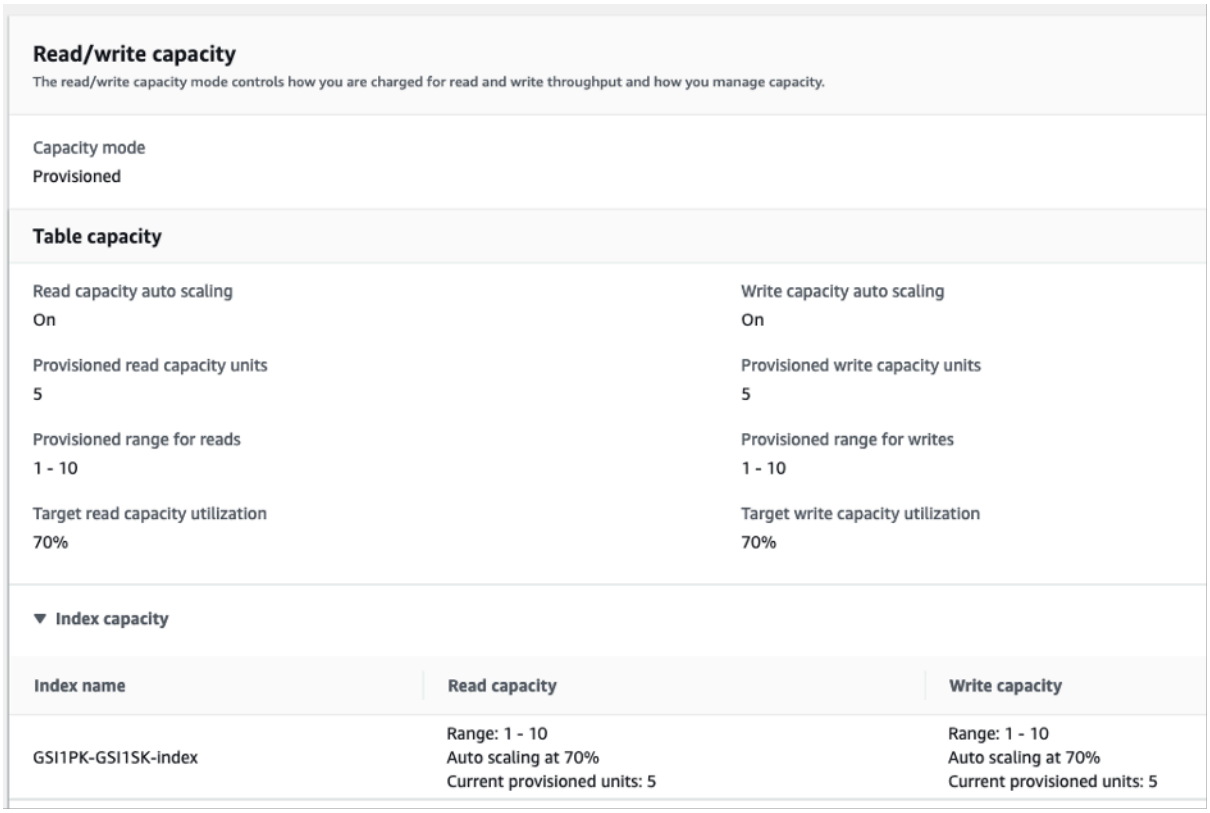
   Wählen Sie bei AWS-Region Bedarf eine geeignete aus.

2. Wählen Sie in der linken Navigationsleiste die Option Tables (Tabellen) aus. Wählen Sie auf der Seite Tables (Tabellen) den Namen der Tabelle aus.

3. Überprüfen Sie auf der Seite Tabellendetails auf der Registerkarte Zusätzliche Einstellungen die Auto-Scaling-Einstellungen Ihrer Tabelle.



Erweitern Sie für Indizes die Registerkarte Indexkapazität, um die Auto-Scaling-Einstellungen des Index zu überprüfen.



Bei Zielauslastungswerten von maximal 50 % sollten Sie Ihre Tabellenauslastungsmetriken prüfen, um festzustellen, ob eine zu geringe oder eine übermäßige Bereitstellung vorliegt.

So bewältigen Sie Workloads mit saisonalen Schwankungen

Stellen Sie sich folgendes Szenario vor: Ihre Anwendung läuft die meiste Zeit unter einem minimalen Durchschnittswert, doch das Auslastungsziel ist niedrig. Somit kann Ihre Anwendung schnell auf Ereignisse reagieren, die zu bestimmten Tageszeiten auftreten, Sie verfügen über genügend Kapazität und es kommt nicht zu Drosselungen. Ein solches Szenario kommt häufig vor, wenn eine Anwendung während der normalen Bürozeiten (9 bis 17 Uhr) sehr stark ausgelastet ist, außerhalb der Geschäftszeiten jedoch auf niedrigem Niveau läuft. Da einige Benutzer vor 9 Uhr beginnen, eine Verbindung herzustellen, verwendet die Anwendung diesen niedrigen Schwellenwert, um schnell hochzufahren und zu Spitzenzeiten die erforderliche Kapazität zu erreichen.

Dieses Szenario könnte wie folgt aussehen:

- Zwischen 17 Uhr und 9 Uhr bleiben die `ConsumedWriteCapacity`-Einheiten zwischen 90 und 100.

- Benutzer beginnen vor 9 Uhr, eine Verbindung zu der Anwendung herzustellen, und die Kapazitätseinheiten steigen erheblich an (der maximale Wert, den Sie gesehen haben, beträgt 1 500 WCU).
- Im Durchschnitt variiert Ihre Anwendungsnutzung während der Arbeitszeit zwischen 800 und 1 200.

Wenn das vorherige Szenario auch bei Ihnen anzutreffen ist, sollten Sie geplantes Auto Scaling in Betracht ziehen. Dabei könnte für Ihre Tabelle immer noch eine Auto-Scaling-Regel für die Anwendung konfiguriert sein, jedoch mit einer weniger aggressiven Zielauslastung, wobei die zusätzliche Kapazität nur in den von Ihnen benötigten Intervallen bereitgestellt wird.

Sie können AWS CLI die folgenden Schritte ausführen, um eine geplante Auto Scaling-Regel zu erstellen, die je nach Tageszeit und Wochentag ausgeführt wird.

1. Registrieren Sie Ihre DynamoDB-Tabelle oder Ihren GSI als skalierbares Ziel bei Application Auto Scaling. Ein skalierbares Ziel ist eine Ressource, die Application Auto Scaling auf- und abskalieren kann.

   ```
   aws application-autoscaling register-scalable-target \
       --service-namespace dynamodb \
       --scalable-dimension dynamodb:table:WriteCapacityUnits \
       --resource-id table/<table-name> \
       --min-capacity 90 \
       --max-capacity 1500
   ```

2. Richten Sie geplante Aktionen entsprechend Ihren Anforderungen ein.

   Für dieses Szenario werden wir zwei Regeln benötigen: eine zum Hochskalieren und eine zum Herunterskalieren. Die erste Regel zum Hochskalieren der geplanten Aktion:

   ```
   aws application-autoscaling put-scheduled-action \
       --service-namespace dynamodb \
       --scalable-dimension dynamodb:table:WriteCapacityUnits \
       --resource-id table/<table-name> \
       --scheduled-action-name my-8-5-scheduled-action \
       --scalable-target-action MinCapacity=800,MaxCapacity=1500 \
       --schedule "cron(45 8 ? * MON-FRI *)" \
       --timezone "Australia/Brisbane"
   ```

   Die zweite Regel zum Herunterskalieren der geplanten Aktion:

```
aws application-autoscaling put-scheduled-action \
    --service-namespace dynamodb \
    --scalable-dimension dynamodb:table:WriteCapacityUnits \
    --resource-id table/<table-name> \
    --scheduled-action-name my-5-8-scheduled-down-action \
    --scalable-target-action MinCapacity=90,MaxCapacity=1500 \
    --schedule "cron(15 17 ? * MON-FRI *)" \
    --timezone "Australia/Brisbane"
```

3. Führen Sie den folgenden Befehl aus, um zu bestätigen, dass beide Regeln aktiviert wurden:

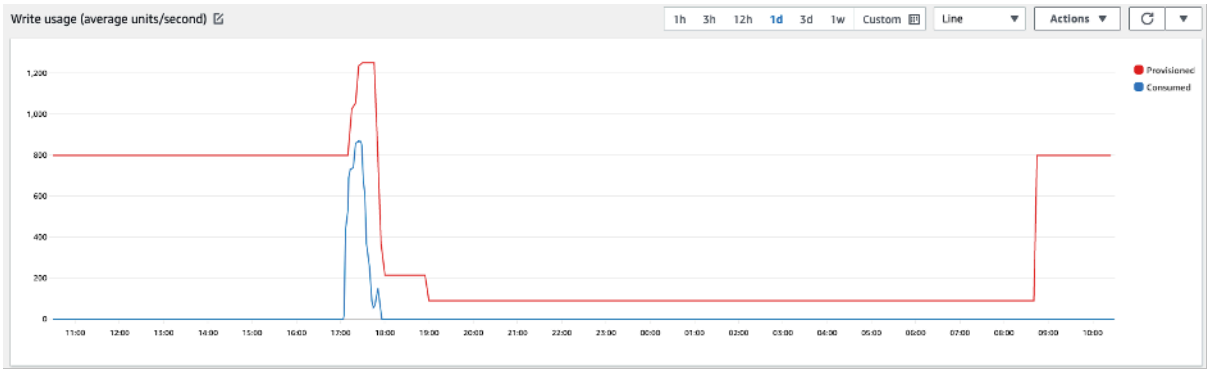
```
aws application-autoscaling describe-scheduled-actions --service-namespace dynamodb
```

Das Ergebnis sollte ungefähr wie folgt aussehen:

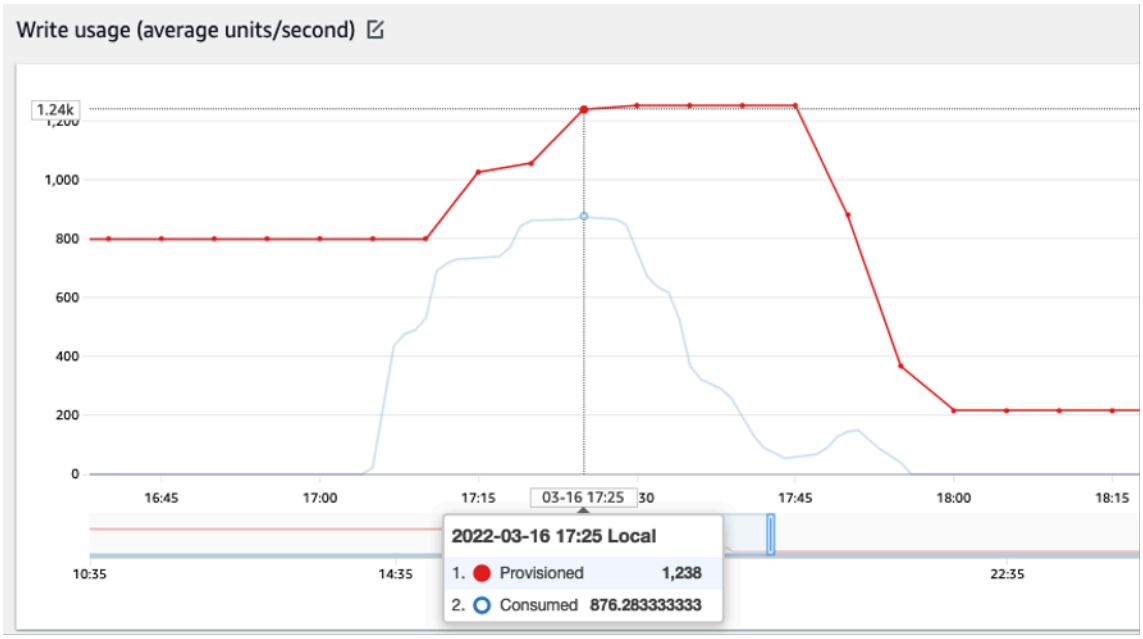
```
{
    "ScheduledActions": [
        {
            "ScheduledActionName": "my-5-8-scheduled-down-action",
            "ScheduledActionARN":
 "arn:aws:autoscaling:<region>:<account>:scheduledAction:<uuid>:resource/dynamodb/
table/<table-name>:scheduledActionName/my-5-8-scheduled-down-action",
            "ServiceNamespace": "dynamodb",
            "Schedule": "cron(15 17 ? * MON-FRI *)",
            "Timezone": "Australia/Brisbane",
            "ResourceId": "table/<table-name>",
            "ScalableDimension": "dynamodb:table:WriteCapacityUnits",
            "ScalableTargetAction": {
                "MinCapacity": 90,
                "MaxCapacity": 1500
            },
            "CreationTime": "2022-03-15T17:30:25.100000+10:00"
        },
        {
            "ScheduledActionName": "my-8-5-scheduled-action",
            "ScheduledActionARN":
 "arn:aws:autoscaling:<region>:<account>:scheduledAction:<uuid>:resource/dynamodb/
table/<table-name>:scheduledActionName/my-8-5-scheduled-action",
            "ServiceNamespace": "dynamodb",
            "Schedule": "cron(45 8 ? * MON-FRI *)",
            "Timezone": "Australia/Brisbane",
            "ResourceId": "table/<table-name>",
```

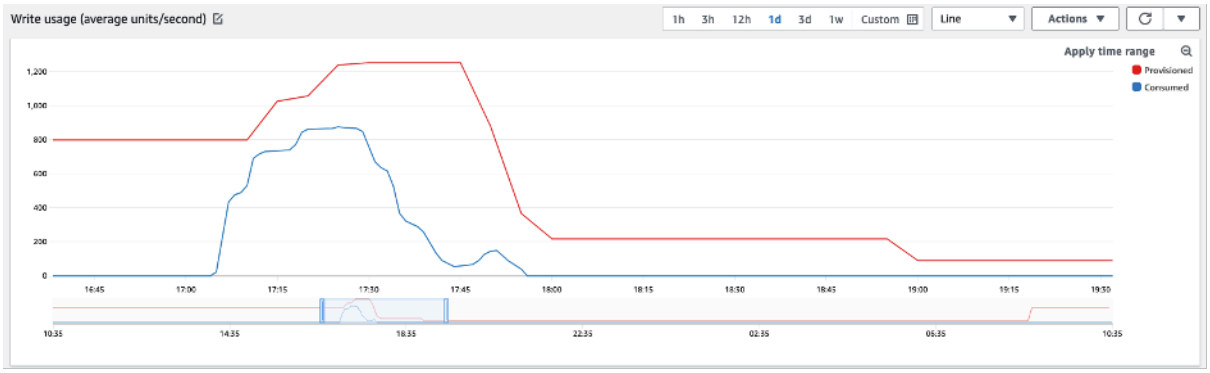
```
            "ScalableDimension": "dynamodb:table:WriteCapacityUnits",
            "ScalableTargetAction": {
                "MinCapacity": 800,
                "MaxCapacity": 1500
            },
            "CreationTime": "2022-03-15T17:28:57.816000+10:00"
        }
    ]
}
```

Im folgenden Bild ist ein Workload zu sehen, bei dem immer die Zielauslastung von 70 % beibehalten wird. Beachten Sie, dass die Auto-Scaling-Regeln weiterhin gelten und der Durchsatz nicht reduziert wird.



Wenn wir die Ansicht vergrößern, sehen wir, dass es in der Anwendung einen Spitzenwert gegeben hat, der den Auto-Scaling-Schwellenwert von 70 % ausgelöst hat, sodass Auto Scaling aktiviert wurde und die für die Tabelle benötigte zusätzliche Kapazität bereitgestellt hat. Das geplante Auto Scaling wirkt sich auf die Maximal- und Minimalwerte aus. Für ihre Einrichtung sind Sie zuständig.





So bewältigen Sie stark schwankende Workloads mit unbekannten Mustern

In diesem Szenario verwendet die Anwendung ein sehr niedriges Auslastungsziel, da Sie die Anwendungsmuster noch nicht kennen und sicherstellen möchten, dass Ihr Workload nicht gedrosselt wird.

Sie sollten stattdessen die Verwendung des On-Demand-Kapazitätsmodus in Betracht ziehen. On-Demand-Tabellen eignen sich perfekt für stark schwankende Workloads, deren Datenverkehrsmuster Sie nicht kennen. Im On-Demand-Kapazitätsmodus zahlen Sie pro Anforderung für die Lese- und Schreibvorgänge, die Ihre Anwendung in Ihren Tabellen ausführt. Sie müssen nicht angeben, wie viel Lese- und Schreibdurchsatz Sie von Ihrer Anwendung erwarten, da DynamoDB Ihre Workloads umgehend anpasst, wenn diese größer oder kleiner werden.

So bewältigen Sie Workloads mit verknüpften Anwendungen

In diesem Szenario hängt die Anwendung von anderen Systemen ab. Dies kann beispielsweise in Batchverarbeitungsszenarien der Fall sein, bei denen es abhängig von Ereignissen in der Anwendungslogik zu starken Datenverkehrsspitzen kommen kann.

Möglicherweise möchten Sie eine benutzerdefinierte Auto-Scaling-Logik entwickeln, die auf diese Ereignisse reagiert und mit der Sie die Tabellenkapazität und `TargetValues` Ihren spezifischen Anforderungen entsprechend erhöhen können. Sie könnten von einer Kombination von AWS Diensten wie Lambda Amazon EventBridge und Step Functions profitieren und diese nutzen, um auf Ihre spezifischen Anwendungsanforderungen zu reagieren.

Auswerten Ihrer DynamoDB-Tabellenklassenauswahl

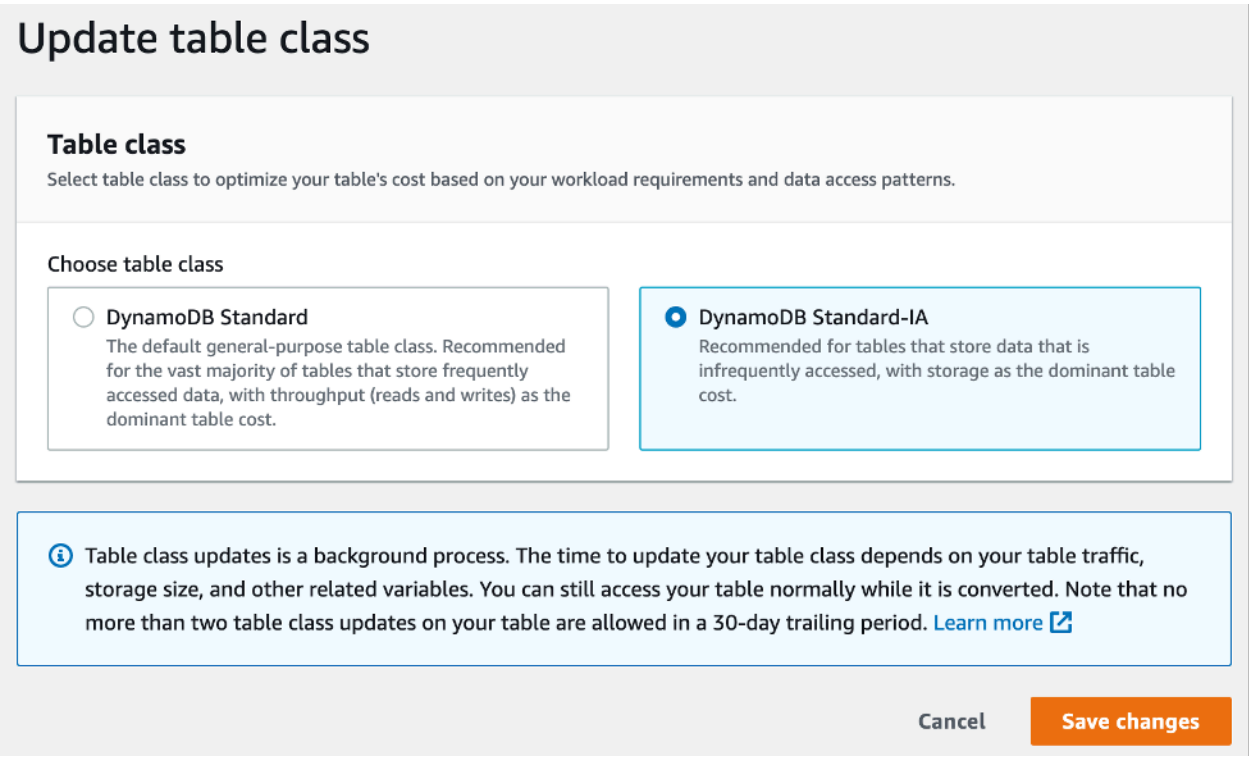
In diesem Abschnitt erhalten Sie einen Überblick darüber, wie Sie die geeignete Tabellenklasse für Ihre DynamoDB-Tabelle auswählen. Mit der Einführung der Tabellenklasse Standard Infrequent-Access (Standard-IA) haben Sie jetzt die Möglichkeit, eine Tabelle entweder für niedrigere Speicherkosten oder für niedrigere Durchsatzkosten zu optimieren.

Themen

- Welche Tabellenklassen sind verfügbar?
- Wann sollte die DynamoDB-Tabellenklasse Standard ausgewählt werden?
- Wann sollte die DynamoDB-Tabellenklasse Standard-IA ausgewählt werden?
- Zusätzliche Faktoren, die bei der Auswahl einer Tabellenklasse zu berücksichtigen sind

Welche Tabellenklassen sind verfügbar?

Wenn Sie eine DynamoDB-Tabelle erstellen, müssen Sie entweder die Tabellenklasse DynamoDB Standard oder DynamoDB Standard-IA auswählen. Die Tabellenklasse kann innerhalb von 30 Tagen zweimal geändert werden, sodass Sie sie zukünftig jederzeit ändern können. Die Auswahl einer der Tabellenklassen hat keinerlei Auswirkungen auf die Leistung, Verfügbarkeit, Zuverlässigkeit oder Beständigkeit der Tabelle.



Tabellenklasse Standard

Die Tabellenklasse Standard ist die Standardoption für neue Tabellen. Bei dieser Option wird der ursprüngliche Abrechnungssaldo von DynamoDB beibehalten, der ein ausgewogenes Verhältnis zwischen Durchsatz- und Speicherkosten für Tabellen mit häufig aufgerufenen Daten bietet.

Tabellenklasse Standard-IA

Die Tabellenklasse Standard-IA bietet niedrigere Speicherkosten (~60 % niedriger) für Workloads, die eine langfristige Speicherung von Daten mit seltenen Aktualisierungen oder Lesevorgängen erfordern. Da die Klasse für den seltenen Zugriff optimiert ist, werden Lese- und Schreibvorgänge hier zu etwas höheren Kosten (~25 % höher) abgerechnet als bei der Tabellenklasse Standard.

Wann sollte die DynamoDB-Tabellenklasse Standard ausgewählt werden?

Die Tabellenklasse DynamoDB Standard eignet sich am besten für Tabellen, deren Speicherkosten etwa 50 % oder weniger der monatlichen Gesamtkosten der Tabelle betragen. Dieser Kostensaldo weist auf eine Workload hin, die regelmäßig Elemente aufruft oder aktualisiert, die bereits in DynamoDB gespeichert sind.

Wann sollte die DynamoDB-Tabellenklasse Standard-IA ausgewählt werden?

Die Tabellenklasse DynamoDB Standard-IA eignet sich am besten für Tabellen, deren Speicherkosten etwa 50 % oder mehr der monatlichen Gesamtkosten der Tabelle betragen. Dieser Kostensaldo weist auf eine Workload hin, bei der pro Monat weniger Elemente erstellt oder gelesen werden als gespeichert sind.

Die Tabellenklasse Standard-IA wird häufig verwendet, um Daten, auf die weniger häufig zugegriffen wird, in einzelne Standard-IA-Tabellen zu verschieben. Weitere Informationen finden Sie unter Optimieren der Speicherkosten Ihrer Workloads mit der Amazon-DynamoDB-Tabellenklasse Standard-IA.

Zusätzliche Faktoren, die bei der Auswahl einer Tabellenklasse zu berücksichtigen sind

Bei der Wahl zwischen den beiden Tabellenklassen gibt es einige zusätzliche Faktoren, die Sie bei Ihrer Entscheidung berücksichtigen sollten.

Reservierte Kapazität

Der Kauf reservierter Kapazität für Tabellen, die die Tabellenklasse Standard-IA verwenden, wird derzeit nicht unterstützt. Der Wechsel von einer Standard-Tabelle mit reservierter Kapazität zu einer Standard-IA-Tabelle ohne reservierte Kapazität bringt möglicherweise keinen Kostenvorteil mit sich.

Ermitteln von ungenutzten Ressourcen in DynamoDB

Dieser Abschnitt gibt einen Überblick darüber, wie Sie Ihre nicht verwendeten Ressourcen regelmäßig bewerten können. Wenn sich Ihre Anwendungsanforderungen weiterentwickeln, sollten Sie sicherstellen, dass keine Ressourcen ungenutzt bleiben und unnötige Kosten für Amazon DynamoDB verursachen. Die unten beschriebenen Verfahren verwenden CloudWatch Amazon-Metriken, um ungenutzte Ressourcen zu identifizieren und Ihnen zu helfen, diese Ressourcen zu identifizieren und Maßnahmen zur Kostensenkung zu ergreifen.

Sie können DynamoDB mithilfe von DynamoDB überwachen CloudWatch, das Rohdaten aus DynamoDB sammelt und in lesbare Metriken nahezu in Echtzeit verarbeitet. Diese Statistiken werden eine gewisse Zeit aufbewahrt, damit Sie zum besseren Verständnis Ihrer Nutzung Verlaufsdaten zur Verfügung haben. Standardmäßig werden DynamoDB-Metrikdaten automatisch an CloudWatch gesendet. Weitere Informationen finden Sie unter Was ist Amazon CloudWatch? und Aufbewahrung von Kennzahlen im CloudWatch Amazon-Benutzerhandbuch.

Themen

- So ermitteln Sie nicht verwendete Ressourcen

- Ermitteln von nicht verwendeten Tabellenressourcen
- Bereinigen von nicht verwendeten Tabellenressourcen
- Ermitteln von nicht verwendeten GSI-Ressourcen
- Bereinigen von nicht verwendeten GSI-Ressourcen
- Bereinigen von nicht verwendeten globalen Tabellen
- Säuberung ungenutzter Backups oder point-in-time Recovery (PITR)

So ermitteln Sie nicht verwendete Ressourcen

Um ungenutzte Tabellen oder Indizes zu identifizieren, schauen wir uns die folgenden CloudWatch Kennzahlen über einen Zeitraum von 30 Tagen an, um zu ermitteln, ob es aktive Lese- oder Schreibvorgänge in der Tabelle oder Lesevorgänge in den globalen Sekundärindizes gibt ()GSIs:

ConsumedReadCapacityUnits

Die Anzahl der in einem bestimmten Zeitraum verbrauchten Lesekapazitätseinheiten, um nachverfolgen zu können, wie viel Kapazität Sie genutzt haben. Sie können die gesamte verbrauchte Lesekapazität für eine Tabelle und alle ihre globalen sekundären Indizes oder für einen bestimmten globalen sekundären Index abrufen.

ConsumedWriteCapacityUnits

Die Anzahl der in einem bestimmten Zeitraum verbrauchten Schreibkapazitätseinheiten, um nachverfolgen zu können, wie viel Kapazität Sie genutzt haben. Sie können die gesamte verbrauchte Schreibkapazität für eine Tabelle und alle ihre globalen sekundären Indizes oder für einen bestimmten globalen sekundären Index abrufen.

Ermitteln von nicht verwendeten Tabellenressourcen

Amazon CloudWatch ist ein Überwachungs- und Beobachtbarkeitsservice, der die DynamoDB-Tabellenmetriken bereitstellt, anhand derer Sie ungenutzte Ressourcen identifizieren können. CloudWatch Metriken können sowohl über AWS-Managementkonsole als auch über die eingesehen werden. AWS Command Line Interface

AWS Command Line Interface

Um die Metriken Ihrer Tabellen über anzuzeigen AWS Command Line Interface, können Sie die folgenden Befehle verwenden.

1. Werten Sie zunächst die Lesevorgänge Ihrer Tabelle aus:

```
aws cloudwatch get-metric-statistics --metric-name
ConsumedReadCapacityUnits --start-time <start-time> --end-time <end-
time> --period <period> --namespace AWS/DynamoDB --statistics Sum --
dimensions Name=TableName,Value=<table-name>
```

Damit Tabellen nicht fälschlicherweise als nicht verwendet ermittelt werden, sollten Sie die Metriken über einen längeren Zeitraum auswerten. Wählen Sie einen geeigneten Startzeit- und Endzeitbereich, beispielsweise 30 Tage, und einen geeigneten Zeitraum, wie z. B. 86400.

In den zurückgegebenen Daten zeigt eine Summe von mehr als 0 an, dass die auszuwertende Tabelle während dieses Zeitraums Lesedatenverkehr empfangen hat.

Das folgende Ergebnis zeigt eine Tabelle, die im ausgewerteten Zeitraum Lesedatenverkehr empfangen hat:

```
        {
            "Timestamp": "2022-08-25T19:40:00Z",
            "Sum": 36023355.0,
            "Unit": "Count"
        },
        {
            "Timestamp": "2022-08-12T19:40:00Z",
            "Sum": 38025777.5,
            "Unit": "Count"
        },
```

Das folgende Ergebnis zeigt eine Tabelle, die im ausgewerteten Zeitraum keinen Lesedatenverkehr empfangen hat:

```
        {
            "Timestamp": "2022-08-01T19:50:00Z",
            "Sum": 0.0,
            "Unit": "Count"
        },
        {
            "Timestamp": "2022-08-20T19:50:00Z",
            "Sum": 0.0,
            "Unit": "Count"
```

```
        },
```

2. Werten Sie als Nächstes die Schreibvorgänge Ihrer Tabelle aus:

```
aws cloudwatch get-metric-statistics --metric-name
ConsumedWriteCapacityUnits --start-time <start-time> --end-time <end-
time> --period <period> --namespace AWS/DynamoDB --statistics Sum --
dimensions Name=TableName,Value=<table-name>
```

Damit Tabellen nicht fälschlicherweise als nicht verwendet ermittelt werden, sollten Sie die Metriken über einen längeren Zeitraum auswerten. Wählen Sie einen geeigneten Startzeit- und Endzeitbereich, beispielsweise 30 Tage, und einen geeigneten Zeitraum, wie z. B. 86400.

In den zurückgegebenen Daten zeigt eine Summe von mehr als 0 an, dass die auszuwertende Tabelle während dieses Zeitraums Lesedatenverkehr empfangen hat.

Das folgende Ergebnis zeigt eine Tabelle, die im ausgewerteten Zeitraum Schreibdatenverkehr empfangen hat:

```
        {
            "Timestamp": "2022-08-19T20:15:00Z",
            "Sum": 41014457.0,
            "Unit": "Count"
        },
        {
            "Timestamp": "2022-08-18T20:15:00Z",
            "Sum": 40048531.0,
            "Unit": "Count"
        },
```

Das folgende Ergebnis zeigt eine Tabelle, die im ausgewerteten Zeitraum keinen Schreibdatenverkehr empfangen hat:

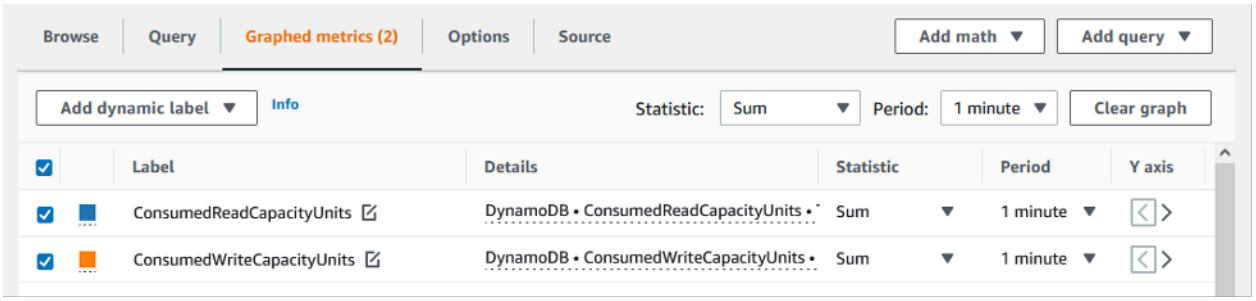
```
        {
            "Timestamp": "2022-07-31T20:15:00Z",
            "Sum": 0.0,
            "Unit": "Count"
        },
        {
            "Timestamp": "2022-08-19T20:15:00Z",
```

```
            "Sum": 0.0,
            "Unit": "Count"
        },
```

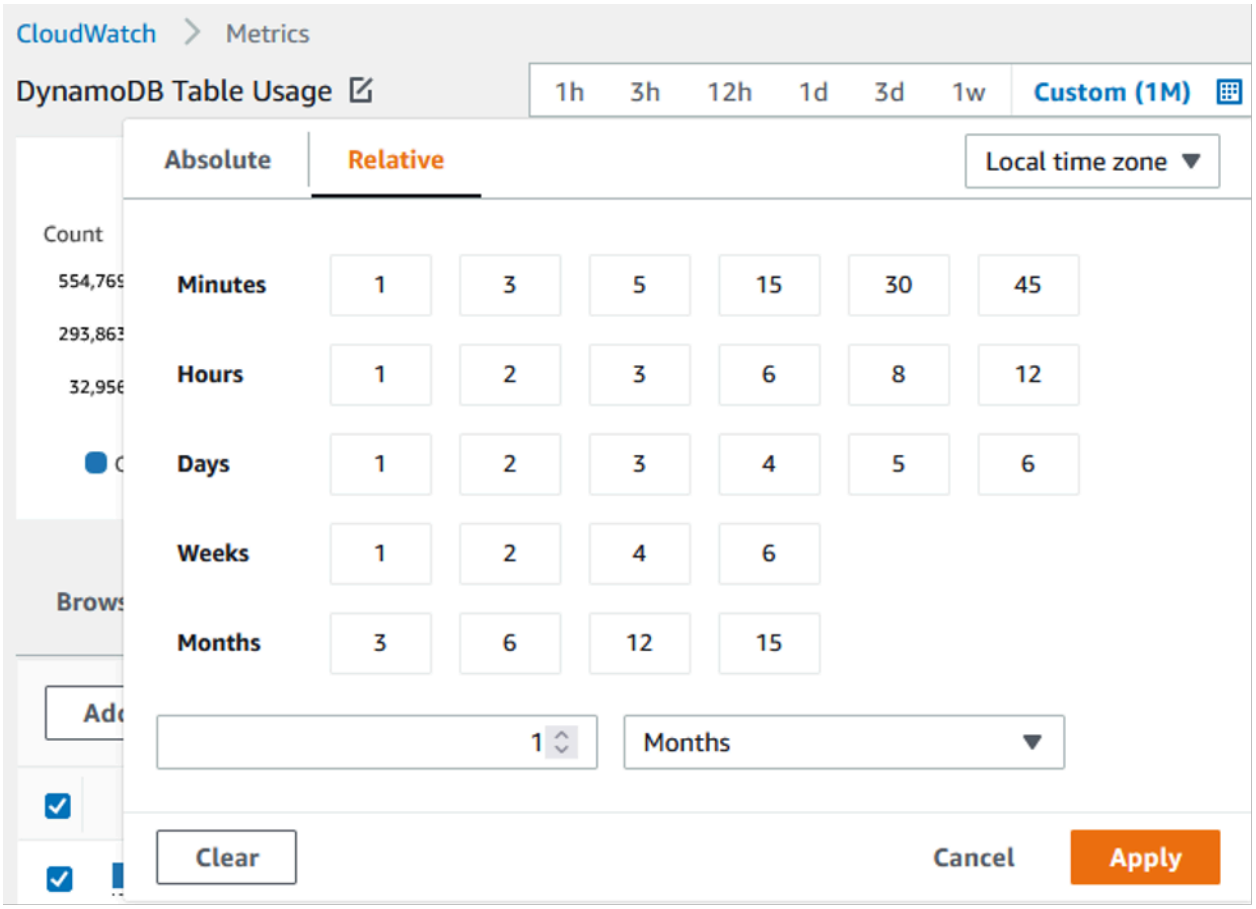
AWS-Managementkonsole

Mit den folgenden Schritten können Sie Ihre Ressourcennutzung über die AWS-Managementkonsole bewerten.

1. Melden Sie sich bei der AWS Konsole an und navigieren Sie zur CloudWatch Serviceseite unter https://console.aws.amazon.com/cloudwatch/. Wählen Sie bei Bedarf die entsprechende AWS Region oben rechts in der Konsole aus.
2. Suchen Sie in der linken Navigationsleiste den Metrikabschnitt und wählen Sie dann Alle Metriken aus.
3. Daraufhin wird ein Dashboard mit zwei Bereichen geöffnet. Im oberen Bereich sehen Sie die aktuell grafisch dargestellten Metriken. Im unteren Bereich wählen Sie die Metriken aus, die Sie grafisch darstellen möchten. Wählen Sie im unteren Bereich DynamoDB aus.
4. Wählen Sie im Auswahlbereich für DynamoDB-Metriken die Kategorie Tabellenmetriken aus, um die Metriken für Ihre Tabellen in der aktuellen Region anzuzeigen.
5. Ermitteln Sie Ihren Tabellennamen, indem Sie im Menü nach unten scrollen, und wählen Sie dann die Metriken `ConsumedReadCapacityUnits` und `ConsumedWriteCapacityUnits` für Ihre Tabelle aus.
6. Wählen Sie die Registerkarte Grafisch dargestellte Metriken (2) aus und stellen Sie die Spalte Statistik auf Summe ein.

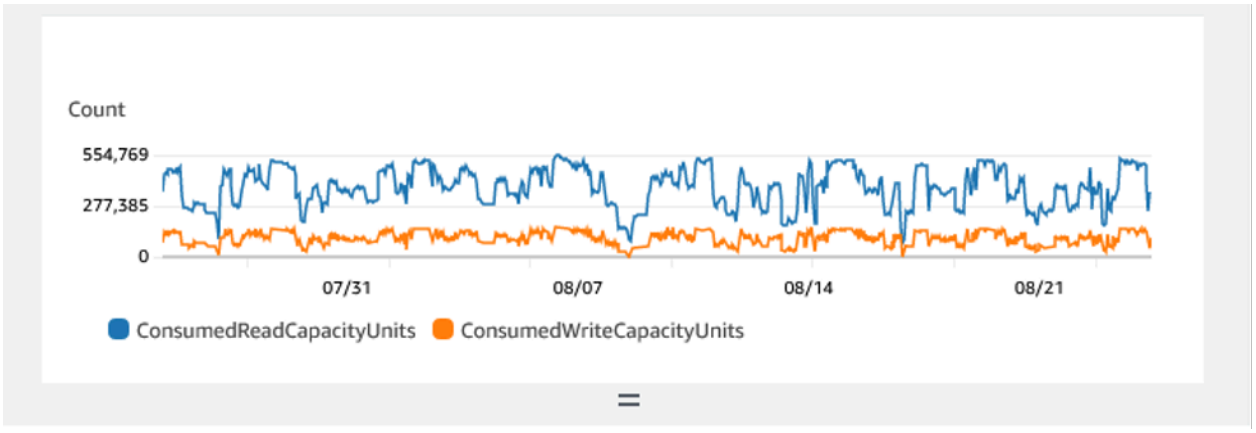


7. Damit Tabellen nicht fälschlicherweise als nicht genutzt ermittelt werden, sollten Sie die Metriken über einen längeren Zeitraum auswerten. Wählen Sie oben im Diagrammfenster einen geeigneten Zeitrahmen, beispielsweise 1 Monat, für die Auswertung Ihrer Tabelle aus. Wählen Sie Benutzerdefiniert, dann in den Dropdown-Listen 1 Monate und schließlich Anwenden aus.

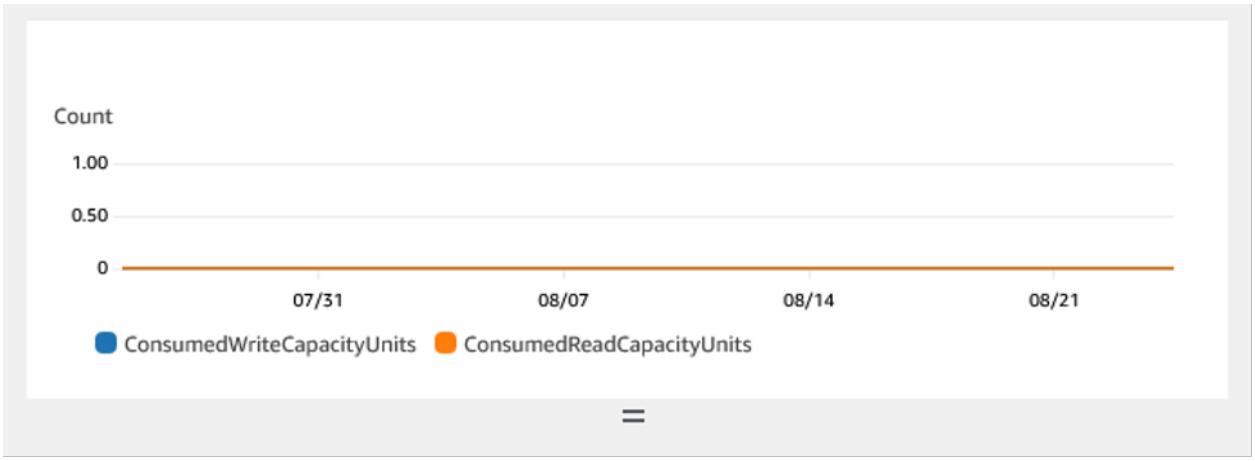


8. Werten Sie die grafisch dargestellten Metriken für Ihre Tabelle aus, um festzustellen, ob die Tabelle genutzt wird. Metriken, die über 0 hinausgehen, weisen darauf hin, dass eine Tabelle während des ausgewerteten Zeitraums genutzt wurde. Ein flaches Diagramm bei 0 sowohl für Lese- als auch für Schreibvorgänge zeigt an, dass die betreffende Tabelle nicht verwendet wird.

   Das folgende Bild zeigt eine Tabelle mit Lesedatenverkehr:



Das folgende Bild zeigt eine Tabelle ohne Lesedatenverkehr:



Bereinigen von nicht verwendeten Tabellenressourcen

Wenn Sie nicht verwendete Tabellenressourcen ermittelt haben, können Sie die laufenden Kosten für diese Ressourcen auf folgende Weise reduzieren.

> **Note**
>
> Wenn Sie eine nicht verwendete Tabelle ermittelt haben, die jedoch verfügbar bleiben soll, falls in Zukunft darauf zugegriffen werden muss, sollten Sie eine Umstellung auf den On-Demand-Modus in Betracht ziehen. Andernfalls können Sie sich überlegen, die Tabelle vollständig zu sichern und zu löschen.

Kapazitätsmodi

DynamoDB berechnet Gebühren für das Lesen, Schreiben und Speichern von Daten in Ihren DynamoDB-Tabellen.

DynamoDB bietet zwei Kapazitätsmodi mit spezifischen Abrechnungsoptionen für die Verarbeitung von Lese- und Schreibvorgängen für Ihre Tabellen: On-Demand und Bereitgestellt. Der read/write Kapazitätsmodus steuert, wie Ihnen der Lese- und Schreibdurchsatz in Rechnung gestellt wird und wie Sie die Kapazität verwalten.

Für On-Demand-Modustabellen müssen Sie nicht angeben, wie viel Lese- und Schreibdurchsatz Sie von Ihrer Anwendung erwarten. DynamoDB berechnet Ihnen die Lese- und Schreibvorgänge, die Ihre Anwendung bei Ihren Tabellen durchführt, als Leseanforderungseinheiten und als

Schreibanforderungseinheiten. Wenn auf Ihrem Computer keine Aktivität stattfindet, zahlen table/index Sie nicht für den Durchsatz, aber es fallen trotzdem Speichergebühren an.

Klasse „Tabelle“

DynamoDB bietet zwei Tabellenklassen, mit deren Hilfe Sie Ihre Kosten optimieren können. Die DynamoDB-Standard-Tabellenklasse ist die Standardeinstellung und wird für die meisten Workloads empfohlen. Die Tabellenklasse DynamoDB Standard-Infrequent Access (DynamoDB Standard-IA) ist für Tabellen optimiert, in denen Speicher die dominierenden Kosten darstellt.

Wenn in Ihrer Tabelle oder Ihrem Index keine Aktivität stattfindet, stellt Speicher wahrscheinlich die dominierenden Kosten dar und eine Änderung der Tabellenklasse führt zu erheblichen Einsparungen.

Löschen von Tabellen

Wenn Sie eine nicht verwendete Tabelle entdeckt haben und diese löschen möchten, sollten Sie zuerst eine Sicherungskopie erstellen oder die Daten exportieren.

Mit AWS Backup erstellte Backups können Cold Storage Tiering nutzen und so die Kosten weiter senken. In der Verwendung von AWS Backup mit DynamoDB Dokumentation finden Sie Informationen darüber, wie Sie Backups über AWS Backup aktivieren, sowie in der Dokumentation Backuppläne verwalten finden Sie Informationen dazu, wie Sie den Lebenszyklus verwenden können, um Ihr Backup in einen Cold Storage zu verschieben.

Alternativ können Sie die Daten Ihrer Tabelle zu S3 exportieren. Weitere Informationen finden Sie in der Dokumentation Export zu Amazon S3. Wenn Sie nach dem Export Ihrer Daten S3 Glacier Instant Retrieval, S3 Glacier Flexible Retrieval oder S3 Glacier Deep Archive nutzen möchten, um die Kosten weiter zu senken, beachten Sie die Informationen unter Verwalten Ihres Speicher-Lebenszyklus.

Nachdem Ihre Tabelle gesichert wurde, können Sie sie entweder über die AWS-Managementkonsole oder über die AWS Command Line Interface löschen.

Ermitteln von nicht verwendeten GSI-Ressourcen

Die Schritte zum Ermitteln eines nicht verwendeten globalen sekundären Indizes sind ähnlich wie bei der Ermittlung einer nicht verwendeten Tabelle. Da DynamoDB in Ihre Basistabelle geschriebene Elemente in Ihre GSI repliziert, wenn sie das Attribut enthalten, das als Partitionsschlüssel der GSI verwendet wird, hat ein nicht verwendeter GSI wahrscheinlich immer noch einen `ConsumedWriteCapacityUnits`-Wert über 0, wenn seine Basistabelle verwendet wird. Daher werten Sie nur die `ConsumedReadCapacityUnits`-Metrik aus, um festzustellen, ob Ihr GSI nicht verwendet wird.

Um Ihre GSI-Metriken über anzuzeigen AWS AWS CLI, können Sie die folgenden Befehle verwenden, um die Lesevorgänge Ihrer Tabelle auszuwerten:

```
aws cloudwatch get-metric-statistics --metric-name
ConsumedReadCapacityUnits --start-time <start-time> --end-time <end-
time> --period <period> --namespace AWS/DynamoDB --statistics Sum --
dimensions Name=TableName,Value=<table-name>
Name=GlobalSecondaryIndexName,Value=<index-name>
```

Damit Tabellen nicht fälschlicherweise als nicht verwendet ermittelt werden, sollten Sie die Metriken über einen längeren Zeitraum auswerten. Wählen Sie einen geeigneten Startzeit- und Endzeitbereich, beispielsweise 30 Tage, und einen geeigneten Zeitraum, wie z. B. 86400.

In den zurückgegebenen Daten zeigt eine Summe von mehr als 0 an, dass die auszuwertende Tabelle während dieses Zeitraums Lesedatenverkehr empfangen hat.

Das folgende Ergebnis zeigt einen GSI, der im ausgewerteten Zeitraum Lesedatenverkehr empfangen hat:

```
        {
          "Timestamp": "2022-08-17T21:20:00Z",
          "Sum": 36319167.0,
          "Unit": "Count"
        },
        {
          "Timestamp": "2022-08-11T21:20:00Z",
          "Sum": 1869136.0,
          "Unit": "Count"
        },
```

Das folgende Ergebnis zeigt einen GSI, der im ausgewerteten Zeitraum minimalen Lesedatenverkehr empfangen hat:

```
        {
          "Timestamp": "2022-08-28T21:20:00Z",
          "Sum": 0.0,
          "Unit": "Count"
        },
        {
          "Timestamp": "2022-08-15T21:20:00Z",
          "Sum": 2.0,
```

```
      "Unit": "Count"
    },
```

Das folgende Ergebnis zeigt einen GSI, der im ausgewerteten Zeitraum keinen Lesedatenverkehr empfangen hat:

```
    {
      "Timestamp": "2022-08-17T21:20:00Z",
      "Sum": 0.0,
      "Unit": "Count"
    },
    {
      "Timestamp": "2022-08-11T21:20:00Z",
      "Sum": 0.0,
      "Unit": "Count"
    },
```

Bereinigen von nicht verwendeten GSI-Ressourcen

Wenn Sie einen nicht verwendeten GSI ermittelt haben, können Sie diesen löschen. Da alle in einem GSI vorhandenen Daten auch in der Basistabelle vorhanden sind, ist vor dem Löschen eines GSI keine zusätzliche Sicherung erforderlich. Wenn der GSI in Zukunft erneut benötigt wird, kann er wieder in die Tabelle aufgenommen werden.

Wenn Sie einen selten verwendeten GSI ermittelt haben, sollten Sie Designänderungen in Ihrer Anwendung in Betracht ziehen, damit Sie diesen löschen oder seine Kosten senken können. Während DynamoDB-Scans beispielsweise sparsam eingesetzt werden sollten, da sie große Mengen an Systemressourcen verbrauchen können, sind sie möglicherweise kostengünstiger als ein GSI, wenn das von diesem unterstützte Zugriffsmuster nur sehr selten verwendet wird.

Wenn ein GSI erforderlich ist, um ein seltenes Zugriffsmuster zu unterstützen, sollten Sie außerdem erwägen, einen begrenzteren Satz von Attributen zu projizieren. Dies kann zwar nachfolgende Abfragen der Basistabelle erfordern, um Ihre seltenen Zugriffsmuster zu unterstützen, kann jedoch möglicherweise zu einer erheblichen Reduzierung der Speicher- und Schreibkosten führen.

Bereinigen von nicht verwendeten globalen Tabellen

Globale Amazon-DynamoDB-Tabellen bieten eine vollständig verwaltete Lösung für die Bereitstellung einer multiregionalen, multiaktiven Datenbank, ohne dass eine eigene Replikationslösung erstellt und gepflegt werden muss.

Globale Tabellen eignen sich ideal für den Zugriff mit geringer Latenz auf Daten in der Nähe von Benutzern und als sekundäre Region für die Notfallwiederherstellung.

Wenn die Option für globale Tabellen für eine Ressource aktiviert ist, um den Zugriff auf Daten mit geringer Latenz zu ermöglichen, aber nicht Teil Ihrer Disaster-Recovery-Strategie ist, überprüfen Sie, ob beide Replikate aktiv Lesetraffic verarbeiten, indem Sie ihre Metriken auswerten. CloudWatch Wenn ein Replikat keinen Lesedatenverkehr bereitstellt, kann es sich um eine nicht verwendete Ressource handeln.

Wenn globale Tabellen Teil Ihrer Disaster-Recovery-Strategie sind, kann davon ausgegangen werden, dass ein Replikat nach einem bestimmten Muster keinen Leseverkehr empfängt. active/ standby

Säuberung ungenutzter Backups oder point-in-time Recovery (PITR)

DynamoDB bietet zwei Arten von Backups. Point-in-timeRecovery bietet kontinuierliche Backups für bis zu 35 Tage, um Sie vor versehentlichen Schreib- oder Löschvorgängen zu schützen. Ein On-Demand-Backup ermöglicht die Erstellung von Snapshots, die langfristig gespeichert werden können. Der Wiederherstellungszeitraum kann auf einen Wert zwischen 1 und 35 Tagen festgelegt werden. Beide Arten von Sicherungen sind mit Kosten verbunden.

Stellen Sie mithilfe der Dokumentation zu Backup und Wiederherstellung für DynamoDB und Zeitpunktbezogene Backups für DynamoDB fest, ob für Ihre Tabellen Sicherungen aktiviert sind, die möglicherweise nicht mehr benötigt werden.

Auswerten Ihrer DynamoDB-Tabellennutzungsmuster

In diesem Abschnitt erfahren Sie, wie Sie prüfen können, ob Sie Ihre DynamoDB-Tabellen effizient nutzen. Gewisse Nutzungsmuster sind für DynamoDB nicht optimal und bieten sowohl mit Blick auf die Leistung als auch auf die Kosten Raum für Optimierungen.

Themen

- Ausführen von weniger strikt konsistenten Lesevorgängen
- Ausführen von weniger Transaktionen für Lesevorgänge
- Ausführen von weniger Scans
- Verkürzen von Attributnamen
- Aktivieren von Time to Live (TTL)
- Ersetzen von globalen Tabellen durch regionsübergreifende Sicherungen

Ausführen von weniger strikt konsistenten Lesevorgängen

In DynamoDB können Sie Lesekonsistenz für jede Anfrage konfigurieren. Leseanforderungen sind standardmäßig letztendlich konsistent. Letztendlich konsistente Lesevorgänge werden mit 0,5 RCU für bis zu 4 KB Daten berechnet.

Die meisten Teile von verteilten Workloads sind flexibel und können letztendliche Konsistenz tolerieren. Es kann jedoch Zugriffsmuster geben, die strikt konsistente Lesevorgänge erfordern. Strikt konsistente Lesevorgänge werden mit 1 RCU für bis zu 4 KB Daten berechnet, sodass sich Ihre Lesekosten im Prinzip verdoppeln. DynamoDB bietet Ihnen die Flexibilität, beide Konsistenzmodelle in derselben Tabelle zu verwenden.

Durch Auswertung Ihres Workloads und Anwendungscodes können Sie prüfen, ob strikt konsistente Lesevorgänge nur bei Bedarf verwendet werden.

Ausführen von weniger Transaktionen für Lesevorgänge

DynamoDB ermöglicht es Ihnen, bestimmte Aktionen auf eine all-or-nothing Weise zu gruppieren, was bedeutet, dass Sie ACID-Transaktionen mit DynamoDB ausführen können. Wie bei relationalen Datenbanken ist es jedoch nicht notwendig, dieses Vorgehen für jede Aktion zu befolgen.

Ein transaktionaler Lesevorgang von bis zu 4 KB verbraucht 2 KB im Gegensatz zu den RCUs Standardwerten von 0,5, wenn dieselbe Datenmenge RCUs letztendlich konsistent gelesen werden soll. Bei Schreibvorgängen verdoppeln sich die Kosten, was bedeutet, dass ein transaktionaler Schreibvorgang von bis zu 1 KB 2 entspricht. WCUs

Um festzustellen, ob es sich bei allen Vorgängen in Ihren Tabellen um Transaktionen handelt, können die CloudWatch Metriken für die Tabelle bis zur Transaktion gefiltert werden. APIs Wenn Transaktionen die einzigen Grafiken APIs sind, die unter den `SuccessfulRequestLatency` Metriken für die Tabelle verfügbar sind, würde dies bestätigen, dass es sich bei jeder Operation um eine Transaktion für diese Tabelle handelt. Wenn der allgemeine Trend der Kapazitätsauslastung mit dem Trend der Transaktions-API übereinstimmt, sollten Sie alternativ erwägen, das Anwendungsdesign zu überdenken, da es offenbar von Transaktionen APIs dominiert wird.

Ausführen von weniger Scans

Die häufige Verwendung von `Scan`-Operationen ist im Allgemeinen auf die Notwendigkeit von analytischen Abfragen der DynamoDB-Daten zurückzuführen. Wenn häufig `Scan`-Operationen in großen Tabellen ausgeführt werden, kann dies ineffizient und teuer sein.

Eine bessere Alternative besteht darin, die Funktion Nach S3 exportieren zu verwenden und einen Zeitpunkt für den Export des Tabellenstatus nach S3 auszuwählen. AWS bietet Dienste wie Athena, mit denen dann analytische Abfragen der Daten ausgeführt werden können, ohne Kapazität aus der Tabelle zu verbrauchen.

Die Häufigkeit von `Scan`-Operationen kann mithilfe der `SampleCount`-Statistik unter der Metrik `SuccessfulRequestLatency` für `Scan` bestimmt werden. Wenn tatsächlich sehr häufig `Scan`-Operationen stattfinden, sollten die Zugriffsmuster und das Datenmodell neu bewertet werden.

Verkürzen von Attributnamen

Die Gesamtgröße eines Elements in DynamoDB ist die Summe seiner Attributnamenlängen und Werte. Lange Attributnamen tragen nicht nur zu den Speicherkosten bei, sondern können auch zu einem höheren RCU/WCU Verbrauch führen. Wir empfehlen, dass Sie eher kürzere Attributnamen als längere wählen. Kürzere Attributnamen können dazu beitragen, die Elementgröße innerhalb der nächsten Grenze von 4 KB/1 KB zu begrenzen. Danach würden Sie zusätzliche RCU/WCU Daten verbrauchen, um darauf zuzugreifen.

Aktivieren von Time to Live (TTL)

Time to Live (TTL) kann Elemente ermitteln, die älter als die von Ihnen für ein Element festgelegte Ablaufzeit sind, und diese aus der Tabelle entfernen. Wenn Ihre Datenmenge im Laufe der Zeit wächst und ältere Daten irrelevant werden, können Sie durch die Aktivierung von TTL in der Tabelle die Datenmenge reduzieren und Speicherkosten sparen.

Ein weiterer nützlicher Aspekt von TTL ist, dass sich die abgelaufenen Elemente in Ihren DynamoDB-Streams befinden. Anstatt die Daten einfach aus Ihren Daten zu entfernen, ist es möglich, diese Elemente aus dem Datenstrom zu verwenden und in einer kostengünstigeren Speicherebene zu archivieren. Darüber hinaus fallen für das Löschen von Objekten über TTL keine zusätzlichen Kosten an – es wird keine Kapazität verbraucht und es entsteht kein Aufwand für die Entwicklung einer Bereinigungsanwendung.

Ersetzen von globalen Tabellen durch regionsübergreifende Sicherungen

Durch die Verwendung von globalen Tabellen können Sie mehrere aktive Replikattabellen in verschiedenen Regionen unterhalten. Sie alle können Schreibvorgänge akzeptieren und Daten untereinander replizieren. Replikate lassen sich einfach einrichten und die Synchronisation wird für Sie verwaltet. Die Replikate nähern sich unter Verwendung einer Strategie „Wer zuletzt schreibt, gewinnt“ einem konsistenten Zustand an.

Wenn Sie globale Tabellen ausschließlich im Rahmen einer Failover- oder Notfallwiederherstellungsstrategie verwenden, können Sie sie durch eine regionsübergreifende Sicherungskopie ersetzen, wenn die Anforderungen an die Recovery Point Objectives und Recovery Time Objectives relativ gering sind. Wenn Sie keinen schnellen lokalen Zugriff und eine hohe Verfügbarkeit von 99,999 % benötigen, ist die Unterhaltung eines Replikats einer globalen Tabelle möglicherweise nicht das beste Failover-Konzept.

Als Alternative sollten Sie die Verwendung von AWS Backup zur Verwaltung von DynamoDB-Backups in Betracht ziehen. Sie können regelmäßige Sicherungen planen und diese regionsübergreifend kopieren, um die Anforderungen an die Notfallwiederherstellung auf kostengünstigere Weise zu erfüllen als bei der Verwendung von globalen Tabellen.

Evaluieren Ihrer Nutzung von DynamoDB Streams

In diesem Abschnitt erfahren Sie, wie Sie Ihre Nutzung von DynamoDB-Streams auswerten können. Gewisse Nutzungsmuster sind für DynamoDB nicht optimal und bieten sowohl mit Blick auf die Leistung als auch auf die Kosten Raum für Optimierungen.

Es gibt zwei native Streaming-Integrationen für streaming- und ereignisgesteuerte Anwendungsfälle:

- Amazon DynamoDB Streams
- Amazon Kinesis Data Streams

Auf dieser Seite geht es um die Strategien zur Kostenoptimierung für diese Optionen. Wenn Sie mehr über die Auswahl zwischen den beiden Optionen erfahren möchten, beachten Sie den Abschnitt Streaming-Optionen für die Erfassung der Änderung von Daten.

Themen

- Optimieren der Kosten für DynamoDB-Streams
- Optimieren der Kosten für Kinesis Data Streams
- Strategien zur Kostenoptimierung für beide Arten der Nutzung von Streams

Optimieren der Kosten für DynamoDB-Streams

Wie auf der Seite zu den Preisen für DynamoDB-Streams erwähnt, berechnet DynamoDB unabhängig vom Durchsatzkapazitätsmodus der Tabelle Gebühren für die Anzahl der Leseanforderungen, die an den DynamoDB-Stream der Tabelle gestellt werden. Es gibt Unterschiede

zwischen Leseanforderungen an einen DynamoDB-Stream und Leseanforderungen an eine DynamoDB-Tabelle.

Jede Leseanforderung an den Datenstrom erfolgt in Form eines `GetRecords`-API-Aufrufs, der bis zu 1 000 Datensätze bzw. 1 MB an Datensätzen in der Antwort zurückgeben kann, wobei der zuerst erreichte Wert maßgeblich ist. Keiner der anderen DynamoDB Streams wird in Rechnung gestellt, und DynamoDB-Streams APIs werden nicht berechnet, wenn sie sich im Leerlauf befinden. Mit anderen Worten, wenn keine Leseanforderungen an einen DynamoDB-Stream gestellt werden, entstehen allein durch die Tatsache, dass ein DynamoDB-Stream für eine Tabelle aktiviert ist, keine Kosten.

Einige Konsumentenanwendungen für DynamoDB Streams:

- AWS Lambda Funktion (en)
- Amazon Kinesis Data Streams-basierte Anwendungen
- Kundenanwendungen für Privatanwender, die mit einem AWS SDK erstellt wurden

Leseanfragen von AWS Lambda-basierten Benutzern von DynamoDB Streams sind kostenlos, wohingegen Anrufe von Verbrauchern anderer Art kostenpflichtig sind. Auch sind jeden Monat die ersten 2 500 000 Leseanforderungen von Nicht-Lambda-Konsumenten kostenlos. Dies gilt für alle Leseanfragen, die an DynamoDB Streams in einem AWS Konto für jede AWS Region gestellt werden.

Überwachen Ihrer Nutzung von DynamoDB-Streams

Die Gebühren für DynamoDB Streams auf der Abrechnungskonsole werden für alle DynamoDB Streams in der gesamten AWS Region in einem Konto zusammengefasst. AWS Zurzeit wird das Markieren von DynamoDB-Streams nicht unterstützt. Somit ist es nicht möglich, mithilfe von Kostenzuordnungs-Tags die detaillierten Kosten für DynamoDB-Streams zu ermitteln. Die Anzahl der `GetRecords`-Aufrufe kann auf DynamoDB-Stream-Ebene abgerufen werden, um die Gebühren pro Stream zu berechnen. Das Volumen wird durch die CloudWatch Metrik `SuccessfulRequestLatency` und die zugehörige `SampleCount` Statistik des DynamoDB-Streams dargestellt. Diese Metrik umfasst auch `GetRecords` Aufrufe von globalen Tabellen zur Durchführung einer laufenden Replikation sowie Aufrufe von AWS Lambda-Verbrauchern, für die beide keine Gebühren anfallen. Informationen zu anderen von DynamoDB Streams veröffentlichten CloudWatch Metriken finden Sie unter. DynamoDB-Metriken und -Dimensionen

AWS Lambda als Verbraucher verwenden

Prüfen Sie, ob die Verwendung von AWS Lambda-Funktionen als Verbraucher für DynamoDB Streams praktikabel ist, da dadurch die mit dem Lesen aus dem DynamoDB-Stream verbundenen Kosten entfallen können. Andererseits erfolgt die Abrechnung für DynamoDB-Streams-Kinesis-Adapter- oder SDK-basierte Konsumentenanwendungen nach der Anzahl der `GetRecords`-Aufrufe an den DynamoDB-Stream.

Aufrufe von Lambda-Funktionen werden unter Berücksichtigung der Lambda-Standardpreise berechnet, für DynamoDB-Streams fallen jedoch keine Gebühren an. Lambda fragt Shards in Ihrem DynamoDB-Stream mit einer Basisrate von viermal pro Sekunde nach Datensätzen ab. Sind Datensätze verfügbar, ruft Lambda Ihre Funktion auf und wartet auf das Ergebnis. Ist die Verarbeitung erfolgreich, setzt Lambda die Abrufe fort, bis es weitere Datensätze erhält.

Optimieren von DynamoDB-Streams-Kinesis-Adapter-basierten Konsumentenanwendungen

Da Leseanforderungen von nicht-Lambda-basierten Konsumenten für DynamoDB-Streams in Rechnung gestellt werden, ist es wichtig, ein Gleichgewicht zu finden zwischen der Anforderung einer Verarbeitung nahezu in Echtzeit und der Häufigkeit, mit der die Konsumentenanwendung den DynamoDB-Stream abfragen muss.

Die Häufigkeit der Abfrage von DynamoDB-Streams mithilfe einer DynamoDB-Streams-Kinesis-Adapter-basierten Anwendung wird durch den konfigurierten `idleTimeBetweenReadsInMillis`-Wert bestimmt. Dieser Parameter legt fest, wie viele Millisekunden der Konsument warten sollte, bevor er einen Shard verarbeitet, falls der vorherige `GetRecords`-Aufruf an denselben Shard keine Datensätze zurückgegeben hat. Der Standardwert für diesen Parameter sind 1 000 ms. Wenn keine echtzeitnahe Verarbeitung notwendig ist, kann der Wert für diesen Parameter erhöht werden, damit die Konsumentenanwendung weniger `GetRecords`-Aufrufe tätigt und die DynamoDB-Streams-Aufrufe optimiert werden.

Optimieren der Kosten für Kinesis Data Streams

Wenn ein Kinesis Data Stream als Ziel für die Bereitstellung von Änderungsdatenerfassungsereignissen für eine DynamoDB-Tabelle festgelegt ist, muss die Dimensionierung des Kinesis Data Streams möglicherweise separat erfolgen. Dies wirkt sich auf die Gesamtkosten aus. DynamoDB berechnet Gebühren in Form von Change Data Capture Units (CDUs), wobei jede Einheit aus einer DynamoDB-Elementgröße von bis zu 1 KB besteht, die der DynamoDB-Dienst versucht, den Ziel-Kinesis Data Stream zu erreichen.

Zusätzlich zu den Gebühren des DynamoDB-Service fallen die Kinesis-Data-Stream-Standardgebühren an. Wie auf der Seite zu den Preisen erwähnt, unterscheiden sich die

Servicepreise je nach Kapazitätsmodus – Modus bereitgestellter Kapazität und On-Demand-Modus. Diese Kapazitätsmodi unterscheiden sich von den Kapazitätsmodi für DynamoDB-Tabellen und sind benutzerdefiniert. Allgemein berechnet Kinesis Data Streams einen Stundensatz, der auf dem Kapazitätsmodus sowie auf den Daten basiert, die von dem DynamoDB-Service in den Strom aufgenommen werden. Je nach Benutzerkonfiguration für den Kinesis Data Stream können zusätzliche Gebühren, beispielsweise für den Datenabruf (beim On-Demand-Modus), für eine verlängerte Datenaufbewahrung (über die standardmäßigen 24 Stunden hinaus) und für erweiterte Fan-Out-Konsumentenabrufe anfallen.

Überwachen Ihrer Nutzung von Kinesis Data Streams

Kinesis Data Streams for DynamoDB veröffentlicht zusätzlich zu den standardmäßigen Kinesis Data Stream-Metriken auch Metriken von DynamoDB. CloudWatch Es ist möglich, dass ein `Put` Versuch des DynamoDB-Dienstes vom Kinesis-Dienst aufgrund unzureichender Kinesis Data Streams-Kapazität oder durch abhängige Komponenten wie einen AWS KMS Dienst, der für die Verschlüsselung der inaktiven Kinesis Data Stream-Daten konfiguriert sein kann, gedrosselt wird.

Weitere Informationen zu den vom DynamoDB-Dienst für den Kinesis Data Stream veröffentlichten CloudWatch Metriken finden Sie unter. Änderung der Datenerfassung für Kinesis Data Streams überwachen Um zusätzliche Kosten für Servicewiederholungen aufgrund von Drosselungen zu vermeiden, ist die richtige Dimensionierung des Kinesis Data Streams im Modus bereitgestellter Kapazität wichtig.

Wählen des richtigen Kapazitätsmodus für Kinesis Data Streams

Kinesis Data Streams werden in zwei Kapazitätsmodi unterstützt: im Modus bereitgestellter Kapazität und im On-Demand-Modus.

- Wenn der den Kinesis Data Stream beinhaltende Workload einen vorhersehbaren Anwendungsdatenverkehr, einen konsistenten oder allmählich ansteigenden Datenverkehr oder einen präzise prognostizierbaren Datenverkehr aufweist, ist der Modus bereitgestellter Kapazität von Kinesis Data Streams geeignet und kosteneffizienter.
- Bei einem neuen Workload, einem Workload mit unvorhersehbarem Anwendungsdatenverkehr oder falls Sie die Kapazität nicht verwalten möchten, ist der On-Demand-Modus von Kinesis Data Streams geeignet und kosteneffizienter.

Eine bewährte Methode zur Kostenoptimierung besteht darin, zu prüfen, ob die dem Kinesis Data Stream zugeordnete DynamoDB-Tabelle ein vorhersehbares Datenverkehrsmuster aufweist, sodass

der Modus bereitgestellter Kapazität von Kinesis Data Streams genutzt werden kann. Wenn es sich um einen neuen Workload handelt, können Sie in den ersten Wochen den On-Demand-Modus für die Kinesis Data Streams verwenden, die CloudWatch Metriken überprüfen, um die Verkehrsmuster zu verstehen, und dann denselben Stream je nach Art der Arbeitslast in den Bereitstellungsmodus umschalten. Im Modus bereitgestellter Kapazität kann die Anzahl der Shards unter Berücksichtigung der Überlegungen zur Shard-Verwaltung für Kinesis Data Streams geschätzt werden.

Auswerten Ihrer Konsumentenanwendungen unter Verwendung von Kinesis Data Streams for DynamoDB

Da Kinesis Data Streams nicht wie DynamoDB-Streams nach der Anzahl der `GetRecords`-Aufrufe abgerechnet werden, könnten Konsumentenanwendungen so viele Aufrufe wie möglich tätigen, sofern die Frequenz unter den Drosselungsgrenzen für `GetRecords` liegt. Beim On-Demand-Modus für Kinesis Data Streams werden Datenlesevorgänge pro GB berechnet. Beim Modus bereitgestellter Kapazität für Kinesis Data Streams werden Lesevorgänge nicht berechnet, wenn die Daten weniger als 7 Tage alt sind. Wenn es sich bei den Kinesis-Data-Streams-Konsumenten um Lambda-Funktionen handelt, fragt Lambda jeden Shard in Ihrem Kinesis-Stream mit einer Basisrate von einmal pro Sekunde nach Datensätzen ab.

Strategien zur Kostenoptimierung für beide Arten der Nutzung von Streams

Ereignisfilterung für AWS Lambda-Verbraucher

Mit der Lambda-Ereignisfilterung können Sie Ereignisse auf der Grundlage eines Filterkriteriums aus dem Stapel der Lambda-Funktionsaufrufe ausschließen. Dadurch werden die Lambda-Kosten für die Verarbeitung oder das Verwerfen unerwünschter Datenstrom-Datensätze innerhalb der Konsumenten-Funktionslogik optimiert. Weitere Informationen zum Konfigurieren der Ereignisfilterung und zum Schreiben von Filterkriterien finden Sie unter Lambda-Ereignisfilterung.

AWS Lambda-Verbraucher einstellen

Durch eine bessere Einstellung der Lambda-Konfigurationsparameter, beispielsweise die Erhöhung von `BatchSize`, um pro Aufruf mehr verarbeiten zu können, die Aktivierung von `BisectBatchOnFunctionError`, um die Verarbeitung von Duplikaten zu verhindern (die zusätzliche Kosten verursacht) und die Einstellung von `MaximumRetryAttempts`, um zu viele Wiederholungen zu vermeiden, können die Kosten weiter optimiert werden. Standardmäßig werden fehlgeschlagene Lambda-Konsumentenaufrufe unendlich oft wiederholt, bis der Datensatz im Datenstrom abläuft, was bei DynamoDB-Streams nach etwa 24 Stunden der Fall ist und bei Kinesis Data Streams zwischen 24 Stunden bis zu einem Jahr konfigurierbar ist. Weitere verfügbare Lambda-

Konfigurationsoptionen, einschließlich der oben für DynamoDB-Stream-Konsumenten genannten Optionen, finden Sie im AWS -Lambda-Entwicklerhandbuch.

Auswerten Ihrer bereitgestellten Kapazität, um eine angemessene Bereitstellung in Ihrer DynamoDB-Tabelle zu erzielen

In diesem Abschnitt erfahren Sie, wie Sie prüfen können, ob die Kapazitätsbereitstellung für Ihre DynamoDB-Tabellen angemessen ist. Im Zuge der Weiterentwicklung Ihres Workloads sollten Sie Ihre Betriebsverfahren entsprechend ändern, insbesondere wenn Ihre DynamoDB-Tabelle im Modus bereitgestellter Kapazität konfiguriert ist und die Gefahr einer zu geringen oder übermäßigen Kapazitätsbereitstellung für Ihre Tabellen besteht.

Für die unten beschriebenen Verfahren werden statistische Informationen benötigt. Diese sollten aus den DynamoDB-Tabellen erfasst werden, die Ihre Produktionsanwendung unterstützen. Um Ihr Anwendungsverhalten zu verstehen, sollten Sie einen Zeitraum definieren, der signifikant genug ist, um die Saisonalität der Daten aus Ihrer Anwendung zu erfassen. Wenn Ihre Anwendung beispielsweise wöchentliche Muster aufweist, sollte ein Zeitraum von drei Wochen ausreichen, um die Anforderungen an den Anwendungsdurchsatz zu analysieren.

Wenn Sie nicht wissen, wo Sie anfangen sollen, verwenden Sie für die folgenden Berechnungen die Datennutzung von mindestens einem Monat.

Bei der Bewertung der Kapazität können DynamoDB-Tabellen Lesekapazitätseinheiten (RCUs) und Schreibkapazitätseinheiten (WCU) unabhängig voneinander konfigurieren. Wenn in Ihren Tabellen Global Secondary Indexes (GSI) konfiguriert sind, müssen Sie den Durchsatz angeben, den diese Indizes verbrauchen, der auch unabhängig von der Basistabelle RCUs und WCUs von der Basistabelle ist.

> **Note**
>
> Lokale sekundäre Indizes (LSI) verbrauchen Kapazität aus der Basistabelle.

Themen

- So rufen Sie Verbrauchsmetriken aus Ihren DynamoDB-Tabellen ab
- So ermitteln Sie DynamoDB-Tabellen mit zu geringer bereitgestellter Kapazität
- So ermitteln Sie DynamoDB-Tabellen mit zu viel bereitgestellter Kapazität

So rufen Sie Verbrauchsmetriken aus Ihren DynamoDB-Tabellen ab

Um die Tabelle und die GSI-Kapazität zu bewerten, überwachen Sie die folgenden CloudWatch Messwerte und wählen Sie die entsprechende Dimension aus, um entweder Tabellen- oder GSI-Informationen abzurufen:

Lesekapazitätseinheiten	Schreibkapazitätseinheiten
`ConsumedReadCapacityUnits`	`ConsumedWriteCapacityUnits`
`ProvisionedReadCapacityUnits`	`ProvisionedWriteCapacityUnits`
`ReadThrottleEvents`	`WriteThrottleEvents`

Sie können dies entweder über die AWS CLI oder die tun. AWS-Managementkonsole

AWS CLI

Bevor wir die Kennzahlen zum Tabellenverbrauch abrufen, müssen wir zunächst einige historische Datenpunkte mithilfe der CloudWatch API erfassen.

Erstellen Sie zunächst zwei Dateien: `write-calc.json` und `read-calc.json`. Diese Dateien stellen die Berechnungen für eine Tabelle oder einen GSI dar. Sie müssen einige der Felder, die in der folgenden Tabelle dargestellt sind, Ihrer Umgebung entsprechend aktualisieren.

Feldname	Definition	Beispiel
`<table-name>`	Der Name der Tabelle, die Sie analysieren	SampleTable
`<period>`	Der Zeitraum, den Sie zum Auswerten des Auslastungsziels verwenden werden, in Sekunden	Für einen Zeitraum von 1 Stunde müssen Sie Folgendes angeben: 3600
`<start-time>`	Der Beginn Ihres Bewertungsintervalls, angegeben im ISO8601 Format	2022-02-21T23:00:00

Feldname	Definition	Beispiel
<end-time>	Das Ende Ihres Bewertung sintervalls, angegeben im ISO8601 Format	2022-02-22T06:00:00

Die Datei mit den Schreibberechnungen ruft die Anzahl der in dem Zeitraum bereitgestellten und verbrauchten WCU für den angegebenen Datumsbereich ab. Zudem generiert sie einen Prozentsatz für die Auslastung, der für die Analyse verwendet wird. Der vollständige Inhalt der Datei `write-calc.json` sollte wie folgt aussehen:

```
{
  "MetricDataQueries": [
    {
      "Id": "provisionedWCU",
      "MetricStat": {
        "Metric": {
          "Namespace": "AWS/DynamoDB",
          "MetricName": "ProvisionedWriteCapacityUnits",
          "Dimensions": [
            {
              "Name": "TableName",
              "Value": "<table-name>"
            }
          ]
        },
        "Period": <period>,
        "Stat": "Average"
      },
      "Label": "Provisioned",
      "ReturnData": false
    },
    {
      "Id": "consumedWCU",
      "MetricStat": {
        "Metric": {
          "Namespace": "AWS/DynamoDB",
          "MetricName": "ConsumedWriteCapacityUnits",
          "Dimensions": [
            {
              "Name": "TableName",
```

```
              "Value": "<table-name>""
            }
          ]
        },
        "Period": <period>,
        "Stat": "Sum"
      },
      "Label": "",
      "ReturnData": false
    },
    {
      "Id": "m1",
      "Expression": "consumedWCU/PERIOD(consumedWCU)",
      "Label": "Consumed WCUs",
      "ReturnData": false
    },
    {
      "Id": "utilizationPercentage",
      "Expression": "100*(m1/provisionedWCU)",
      "Label": "Utilization Percentage",
      "ReturnData": true
    }
  ],
  "StartTime": "<start-time>",
  "EndTime": "<ent-time>",
  "ScanBy": "TimestampDescending",
  "MaxDatapoints": 24
}
```

Die Datei mit den Leseberechnungen ist ähnlich. In dieser Datei wird abgerufen, wie viele während des angegebenen Zeitraums bereitgestellt und genutzt RCUs wurden. Der Inhalt der Datei `read-calc.json` sollte wie folgt aussehen:

```
{
  "MetricDataQueries": [
    {
      "Id": "provisionedRCU",
      "MetricStat": {
        "Metric": {
          "Namespace": "AWS/DynamoDB",
          "MetricName": "ProvisionedReadCapacityUnits",
          "Dimensions": [
            {
```

```
            "Name": "TableName",
            "Value": "<table-name>"
          }
        ]
      },
      "Period": <period>,
      "Stat": "Average"
    },
    "Label": "Provisioned",
    "ReturnData": false
  },
  {
    "Id": "consumedRCU",
    "MetricStat": {
      "Metric": {
        "Namespace": "AWS/DynamoDB",
        "MetricName": "ConsumedReadCapacityUnits",
        "Dimensions": [
          {
            "Name": "TableName",
            "Value": "<table-name>"
          }
        ]
      },
      "Period": <period>,
      "Stat": "Sum"
    },
    "Label": "",
    "ReturnData": false
  },
  {
    "Id": "m1",
    "Expression": "consumedRCU/PERIOD(consumedRCU)",
    "Label": "Consumed RCUs",
    "ReturnData": false
  },
  {
    "Id": "utilizationPercentage",
    "Expression": "100*(m1/provisionedRCU)",
    "Label": "Utilization Percentage",
    "ReturnData": true
  }
],
"StartTime": "<start-time>",
```

```
  "EndTime": "<end-time>",
  "ScanBy": "TimestampDescending",
  "MaxDatapoints": 24
}
```

Wenn Sie die Dateien erstellt haben, können Sie mit dem Abrufen von Auslastungsdaten beginnen.

1. Geben Sie den folgenden Befehl aus, um die Daten zur Schreibauslastung abzurufen:

   ```
   aws cloudwatch get-metric-data --cli-input-json file://write-calc.json
   ```

2. Geben Sie den folgenden Befehl aus, um die Daten zur Leseauslastung abzurufen:

   ```
   aws cloudwatch get-metric-data --cli-input-json file://read-calc.json
   ```

Das Ergebnis beider Abfragen ist eine Reihe von Datenpunkten im JSON-Format, die für die Analyse verwendet werden. Ihr Ergebnis hängt von der Anzahl der angegebenen Datenpunkte, dem Zeitraum und Ihren spezifischen Workload-Daten ab. Es könnte etwa wie folgt aussehen:

```
{
    "MetricDataResults": [
        {
            "Id": "utilizationPercentage",
            "Label": "Utilization Percentage",
            "Timestamps": [
                "2022-02-22T05:00:00+00:00",
                "2022-02-22T04:00:00+00:00",
                "2022-02-22T03:00:00+00:00",
                "2022-02-22T02:00:00+00:00",
                "2022-02-22T01:00:00+00:00",
                "2022-02-22T00:00:00+00:00",
                "2022-02-21T23:00:00+00:00"
            ],
            "Values": [
                91.55364583333333,
                55.066631944444445,
                2.6114930555555556,
                24.9496875,
                40.94725694444445,
                25.61819444444444,
```

```
                0.0
            ],
            "StatusCode": "Complete"
        }
    ],
    "Messages": []
}
```

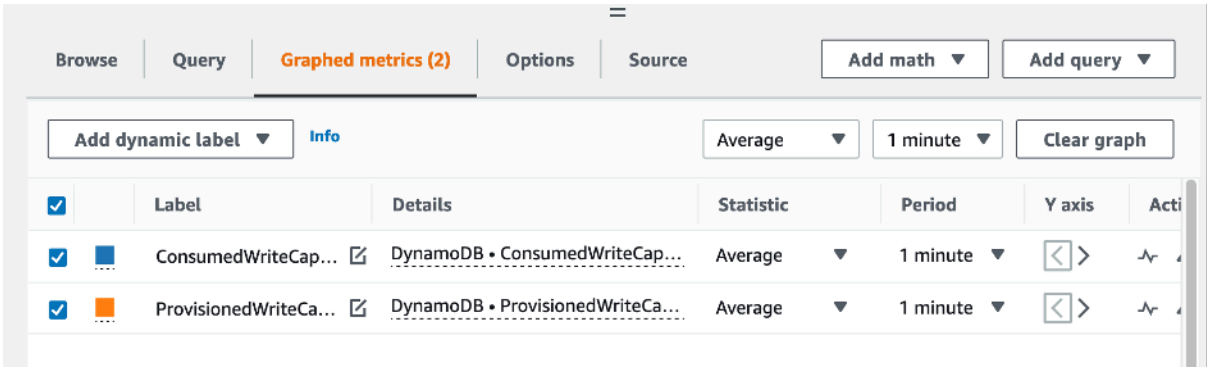
> **Note**
>
> Wenn Sie einen kurzen Zeitraum und einen langen Zeitbereich angeben, müssen Sie möglicherweise den Wert für `MaxDatapoints` ändern. Dieser ist im Skript standardmäßig auf 24 gesetzt. Dies entspricht einem Datenpunkt pro Stunde und 24 pro Tag.

AWS-Managementkonsole

1. Melden Sie sich bei an AWS-Managementkonsole und navigieren Sie zur CloudWatch Serviceseite. Wählen Sie bei AWS-Region Bedarf eine geeignete aus.
2. Suchen Sie in der linken Navigationsleiste nach dem Abschnitt Metriken und wählen Sie Alle Metriken aus.
3. Daraufhin wird ein Dashboard mit zwei Bereichen geöffnet. Im oberen Bereich wird die Grafik angezeigt und im unteren Bereich sind die Metriken aufgeführt, die Sie grafisch darstellen möchten. Wählen Sie DynamoDB aus.
4. Wählen Sie Tabellenmetriken aus. Daraufhin werden die Tabellen in Ihrer aktuellen Region angezeigt.
5. Verwenden Sie das Suchfeld, um nach dem Namen Ihrer Tabelle zu suchen, und wählen Sie die Metriken für Schreibvorgänge `ConsumedWriteCapacityUnits` und `ProvisionedWriteCapacityUnits` aus.

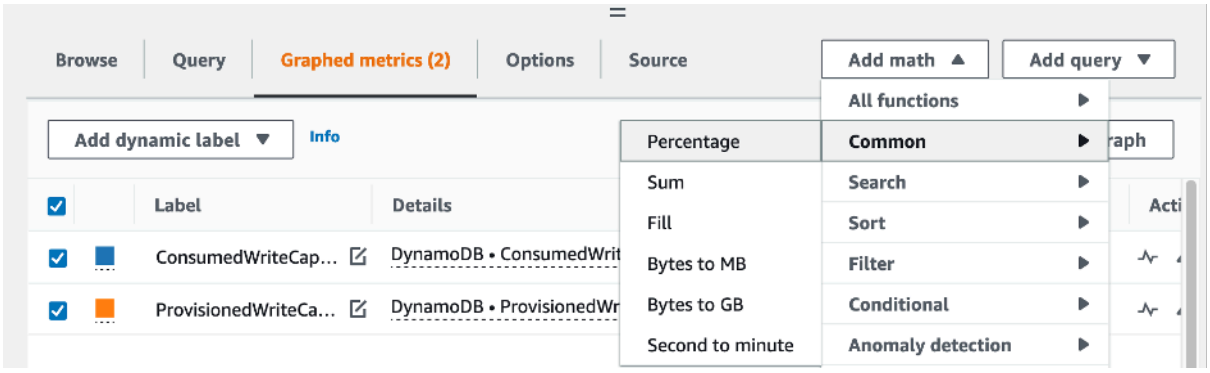
   > **Note**
   >
   > In diesem Beispiel geht es um Metriken für Schreibvorgänge, Sie können diese Schritte jedoch auch verwenden, um die Metriken für Lesevorgänge grafisch darzustellen.

6. Wählen Sie die Registerkarte Grafisch dargestellte Metriken (2) aus, um die Formeln zu ändern. CloudWatch Wählt standardmäßig die Statistikfunktion Durchschnitt für die Grafiken aus.

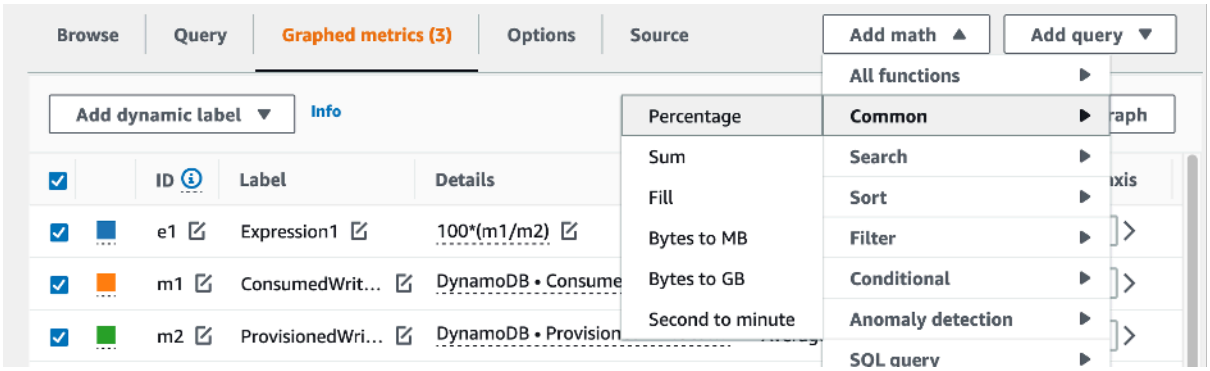


7. Wenn beide grafisch dargestellten Metriken ausgewählt sind (Kontrollkästchen auf der linken Seite), wählen Sie das Menü Add math (Math. hinzufügen), dann Common (Allgemein) und schließlich die Funktion Percentage (Prozent) aus. Wiederholen Sie den Vorgang zweimal.

   Erstes Auswählen der Funktion Percentage (Prozent):

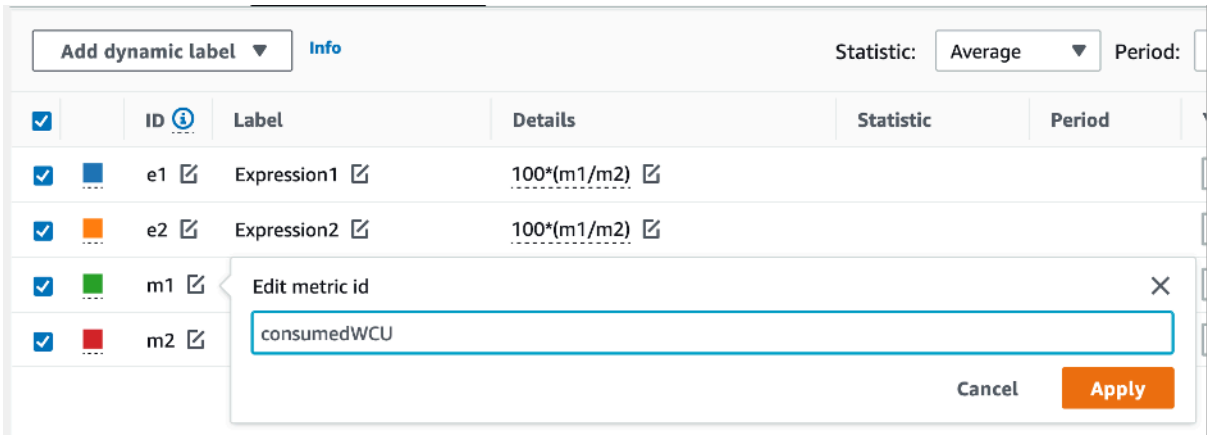


   Zweites Auswählen der Funktion Percentage (Prozent):

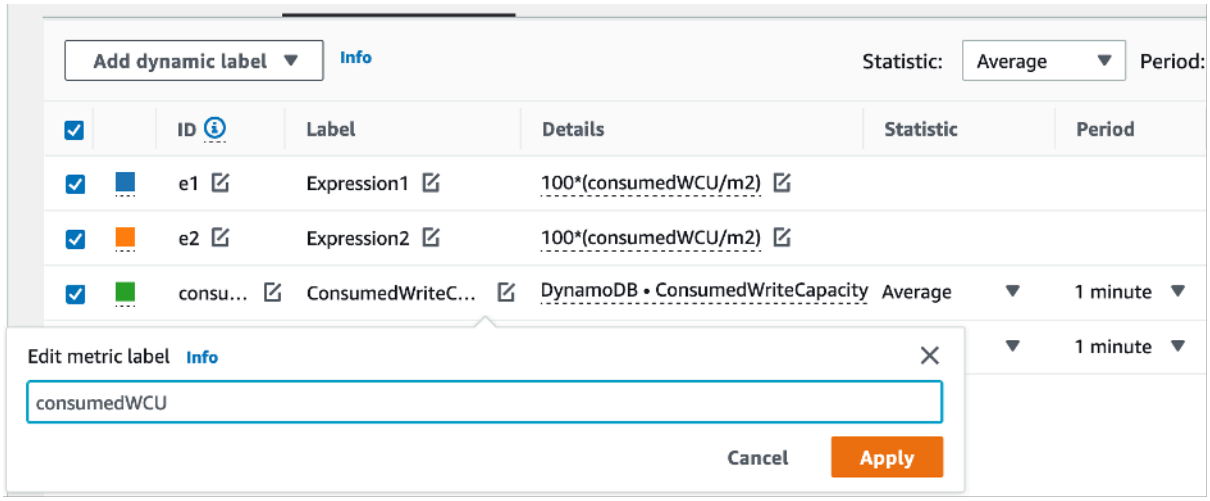


8. Zu diesem Zeitpunkt sollten vier Metriken im unteren Menü aufgeführt sein. Befassen wir uns nun mit der Berechnung von `ConsumedWriteCapacityUnits`. Um konsistent zu sein,

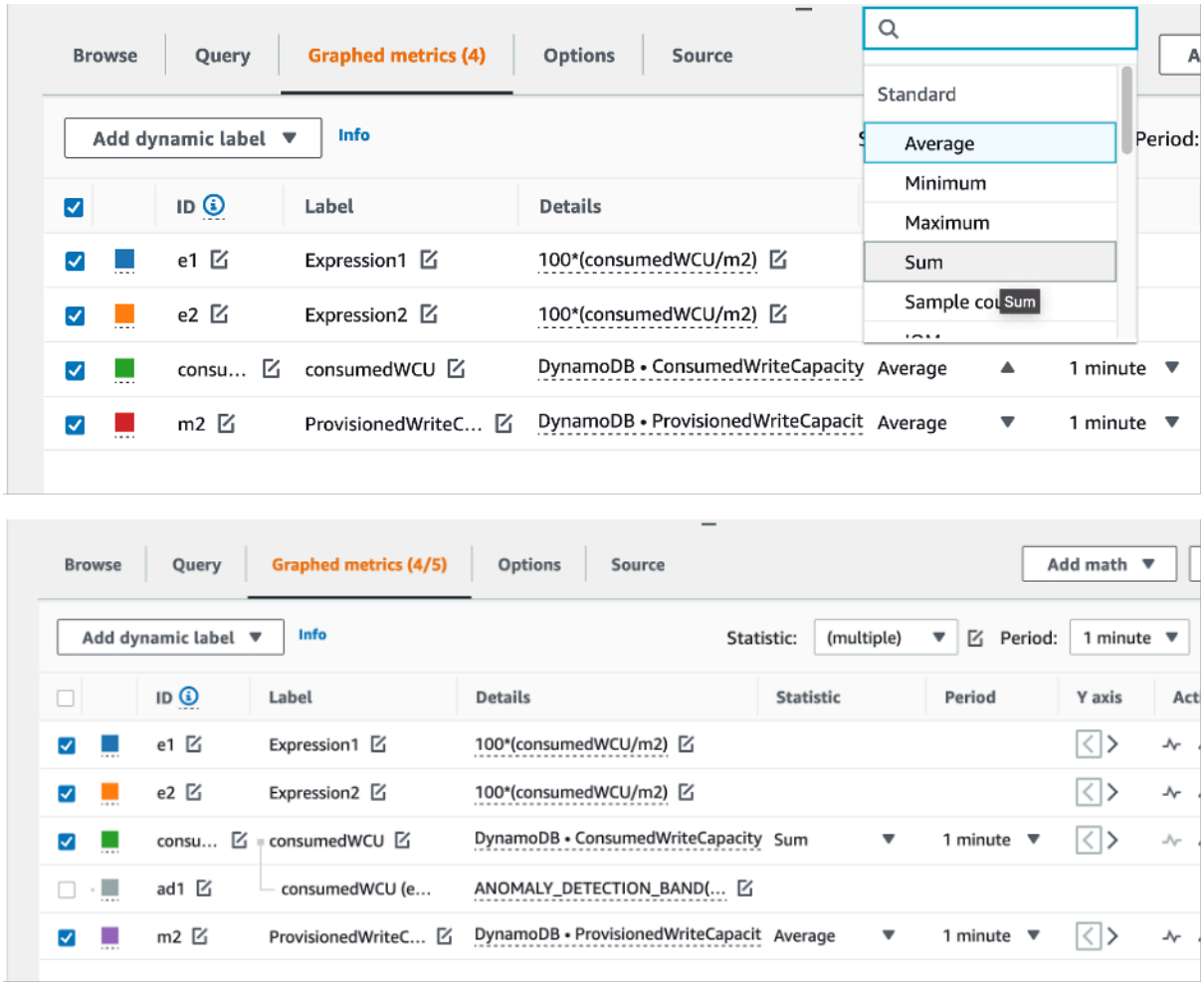
müssen wir die Namen mit denen abgleichen, die wir in dem AWS CLI Abschnitt verwendet haben. Klicken Sie auf die m1 ID und ändern Sie diesen Wert in consumedWCU.



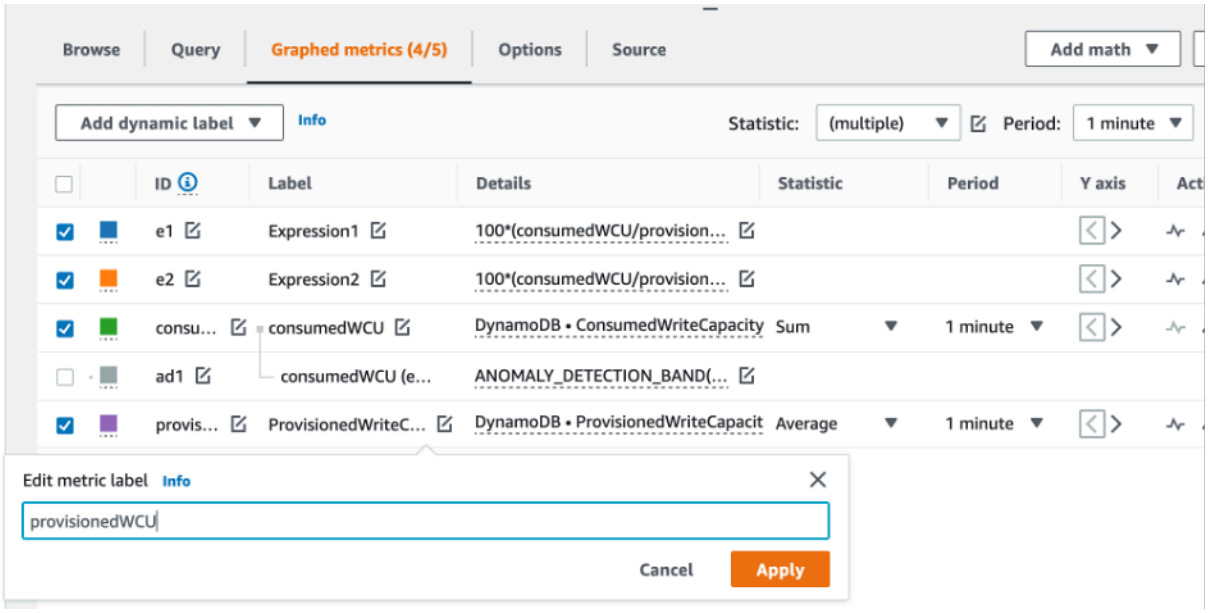
Benennen Sie das ConsumedWriteCapacityUnitLabel um als**`consumedWCU`**.



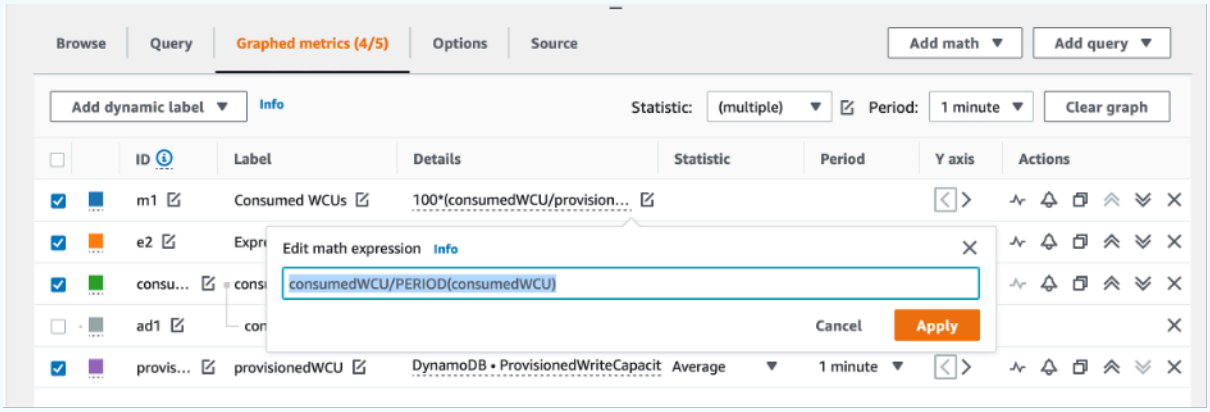
9. Ändern Sie die Statistik von Average (Durchschnitt) in Sum (Summe). Bei dieser Aktion wird automatisch eine weitere Metrik namens ANOMALY_DETECTION_BAND erstellt. Für die Zwecke dieses Verfahrens können wir diese ignorieren, indem wir das Kontrollkästchen für die neu generierte Metrik ad1 entfernen.



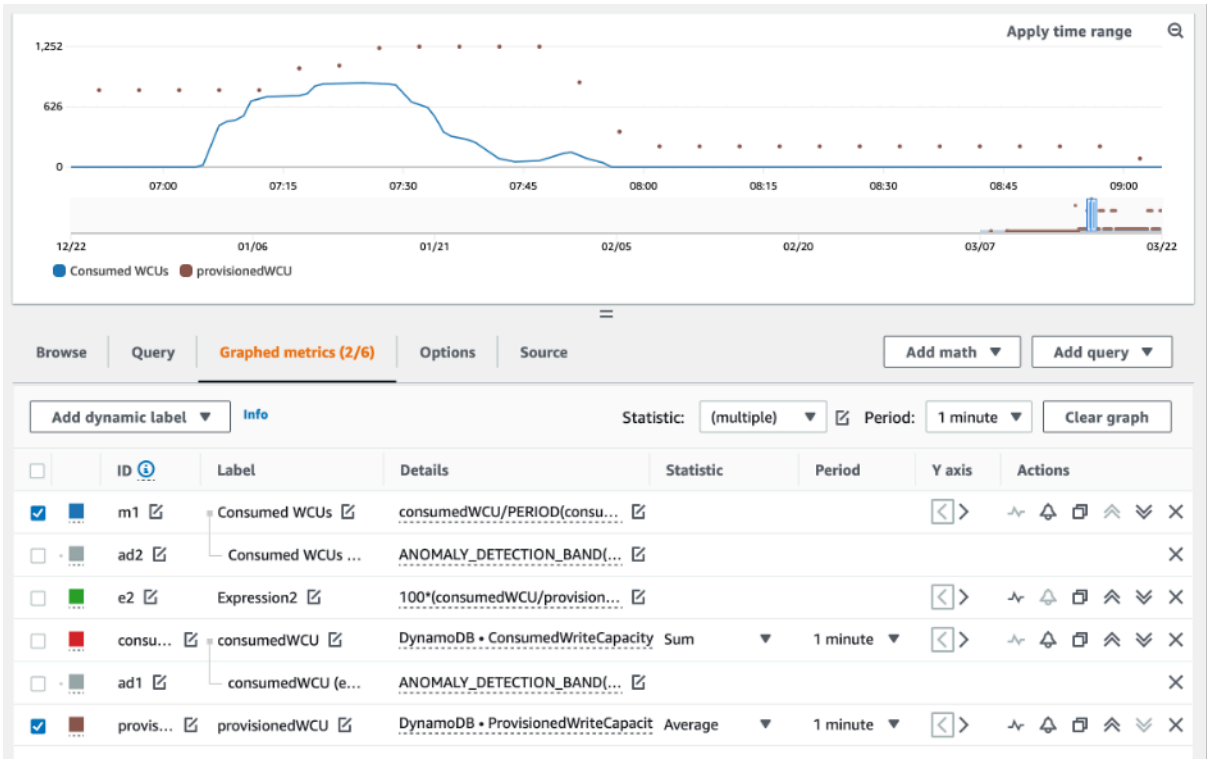
10. Wiederholen Sie Schritt 8, um die m2 ID in provisionedWCU umzubenennen. Lassen Sie die Statistik auf Average (Durchschnitt) eingestellt.



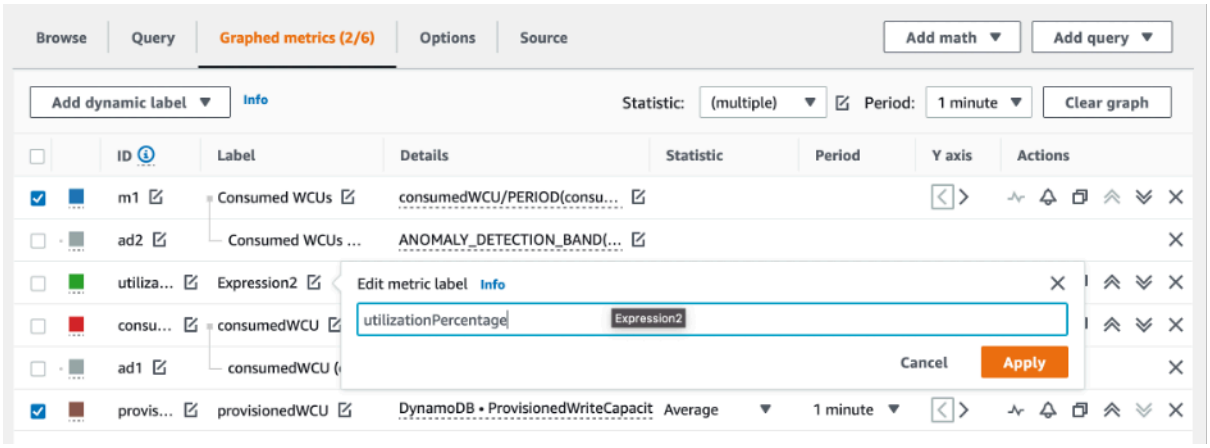
11. Wählen Sie das Label Expression1 aus und aktualisieren Sie den Wert auf m1 und das Label auf Consumed. WCUs

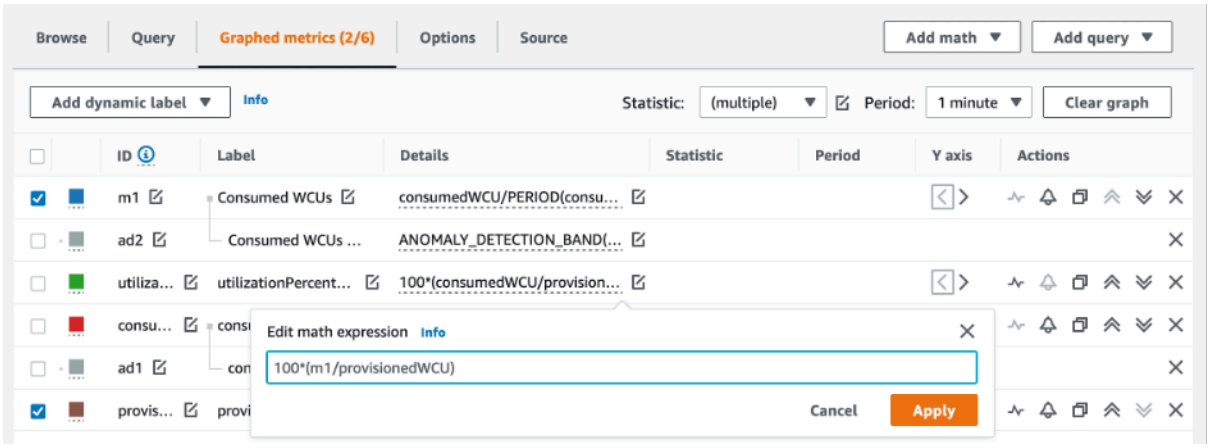
> **Note**
>
> Stellen Sie sicher, dass Sie nur m1 (Kontrollkästchen links) und provisionedWCU ausgewählt haben, damit die Daten richtig angezeigt werden. Aktualisieren Sie die Formel, indem Sie auf Details klicken und die Formel in consumedWCU/PERIOD(consumedWCU) ändern. In diesem Schritt könnte eine weitere Metrik ANOMALY_DETECTION_BAND generiert werden, für die Zwecke dieses Verfahrens können wir diese jedoch ignorieren.
>
> 
>

12. Sie sollten nun über zwei Grafiken verfügen: eine, die Ihre bereitgestellten WCUs Daten in der Tabelle anzeigt, und eine andere, die den Verbrauch anzeigt. WCUs Die Grafik kann sich in ihrer Form von der unten dargestellten Grafik unterscheiden, Sie können sie jedoch als Referenz verwenden:



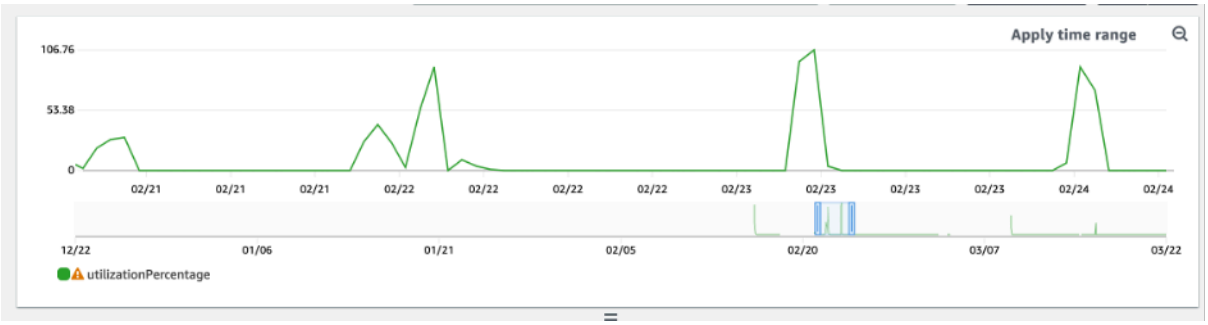
13. Aktualisieren Sie die Prozentformel, indem Sie die Grafik für Expression2 (e2) auswählen. Benennen Sie die Beschriftungen und in UtilizationPercentage IDs um. Benennen Sie die Formel so um, dass sie 100*(m1/provisionedWCU) entspricht.





14. Entfernen Sie das Kontrollkästchen für alle Metriken außer utilizationPercentage, um Ihre Nutzungsmuster anzuzeigen. Als Standardintervall ist 1 Minute eingestellt, Sie können dies jedoch nach Bedarf ändern.

Hier sehen Sie die Ansicht eines längeren Zeitraums sowie eines größeren Zeitbereichs von 1 Stunde. Wie Sie sehen, gibt es einige Intervalle mit einer Auslastung von über 100 %, doch bei diesem speziellen Workload sind längere Intervalle mit einer Auslastung von null zu finden.



Hier unterscheiden sich Ihre Ergebnisse möglicherweise von den hier dargestellten Ergebnissen. Es hängt alles von den Daten aus Ihrem Workload ab. Wenn es Intervalle mit einer Auslastung von mehr als 100 % gibt, besteht die Gefahr einer Drosselung. DynamoDB bietet Burst-Kapazität, doch sobald die Burst-Kapazität erschöpft ist, wird alles über 100 % gedrosselt.

So ermitteln Sie DynamoDB-Tabellen mit zu geringer bereitgestellter Kapazität

Bei den meisten Workloads gilt die bereitgestellte Kapazität einer Tabelle als zu gering, wenn die Tabelle ständig mehr als 80 % ihrer bereitgestellten Kapazität beansprucht.

Burst-Kapazität ist eine DynamoDB-Funktion, mit der Kunden vorübergehend mehr RCUs/WCUs als ursprünglich bereitgestellt nutzen können (mehr als den bereitgestellten Durchsatz pro Sekunde, der in der Tabelle definiert wurde). Die Burst-Kapazität soll plötzliche Zunahmen des Datenverkehrs aufgrund von besonderen Ereignissen oder Auslastungsspitzen auffangen. Doch die Burst-Kapazität ist irgendwann erschöpft. Sobald die ungenutzten RCUs Daten aufgebraucht WCUs sind, werden Sie gedrosselt, wenn Sie versuchen, mehr Kapazität als die bereitgestellte Kapazität zu verbrauchen. Wenn sich Ihr Anwendungsdatenverkehr der Auslastungsrate von 80 % nähert, ist das Risiko einer Drosselung deutlich höher.

Die Regel der Auslastungsrate von 80 % hängt von der Saisonalität Ihrer Daten und dem Wachstums Ihres Datenverkehrs ab. Betrachten Sie folgende Szenarien:

- Wenn Ihr Datenverkehr in den letzten 12 Monaten bei einer Auslastung von ~ 90 % stabil war, hat Ihre Tabelle genau die richtige Kapazität.
- Wenn Ihr Anwendungsdatenverkehr monatlich um 8 % wächst, werden Sie in weniger als 3 Monaten 100 % erreichen.
- Auch wenn Ihr Anwendungsdatenverkehr monatlich um 5 % wächst, werden Sie in etwas mehr als 4 Monaten 100 % erreichen.

Die Ergebnisse der obigen Abfragen vermitteln ein Bild Ihrer Auslastungsrate. Verwenden Sie sie als Orientierungshilfe für die Auswertung weiterer Metriken, die bei der Entscheidung, Ihre Tabellenkapazität nach Bedarf zu erhöhen, hilfreich sein können (z. B. monatliche oder wöchentliche Wachstumsrate). Legen Sie gemeinsam mit Ihrem Operations-Team fest, welcher Prozentsatz für Ihren Workload und Ihre Tabellen geeignet ist.

Es gibt gewisse Situationen, in denen die Daten verzerrt sind, wenn wir sie täglich oder wöchentlich analysieren. So könnte beispielsweise bei saisonalen Anwendungen, deren Auslastung während der Arbeitszeiten stark ansteigt (außerhalb der Arbeitszeiten aber nahezu auf Null fällt), geplantes Auto Scaling von Vorteil sein, wobei Sie angeben, zu welchen Stunden des Tages (und an welchen Wochentagen) die bereitgestellte Kapazität erhöht und wann sie reduziert werden soll. Anstatt eine höhere Kapazität anzustreben, um die Stoßzeiten abzudecken, können Sie auch von Auto-Scaling-Konfigurationen für DynamoDB-Tabellen profitieren, wenn die Saisonalität weniger ausgeprägt ist.

> **Note**
>
> Denken Sie beim Erstellen einer DynamoDB-Auto-Scaling-Konfiguration für Ihre Basistabelle daran, eine weitere Konfiguration für jeden GSI hinzuzufügen, der der Tabelle zugeordnet ist.

So ermitteln Sie DynamoDB-Tabellen mit zu viel bereitgestellter Kapazität

Die mit den obigen Skripts erhaltenen Abfrageergebnisse liefern die für eine erste Analyse erforderlichen Datenpunkte. Wenn Ihr Datensatz für mehrere Intervalle Auslastungswerte von weniger als 20 % aufweist, wurde für Ihre Tabelle möglicherweise zu viel Kapazität bereitgestellt. Um genauer zu definieren, ob Sie die Anzahl der WCUs und RCUS reduzieren müssen, sollten Sie die anderen Messwerte in den Intervallen erneut überprüfen.

Wenn Ihre Tabellen mehrere Intervalle mit geringer Auslastung aufweisen, können Sie von der Verwendung von Auto-Scaling-Richtlinien profitieren, indem Sie entweder Auto Scaling planen oder einfach standardmäßige Auto-Scaling-Richtlinien für die Tabelle konfigurieren, die auf der Auslastung basieren.

Wenn Sie einen Workload mit einem Verhältnis von geringer Auslastung zu hohem Drosselungsverhältnis (Max (ThrottleEvents) /Min (ThrottleEvents) im Intervall) haben, kann dies passieren, wenn Sie eine sehr hohe Arbeitslast haben, bei der der Verkehr an einigen Tagen (oder Stunden) stark zunimmt, der Verkehr aber im Allgemeinen konstant gering ist. In solchen Situationen kann es von Vorteil sein, geplantes Auto Scaling zu verwenden.

Das AWS Well-Architected Framework unterstützt Cloud-Architekten beim Aufbau einer sicheren, leistungsstarken, belastbaren und effizienten Infrastruktur für eine Vielzahl von Anwendungen und Workloads.AWS Well-Architected basiert auf sechs Säulen — betriebliche Exzellenz, Sicherheit, Zuverlässigkeit, Leistungseffizienz, Kostenoptimierung und Nachhaltigkeit — und bietet Kunden und Partnern einen konsistenten Ansatz zur Bewertung von Architekturen und zur Implementierung skalierbarer Designs.

Die AWS Well-Architected Lenses erweitern das Beratungsangebot von AWS Well-Architected auf spezifische Branchen- und Technologiebereiche. Die Amazon DynamoDB Well-Architected Lens bezieht sich auf DynamoDB-Workloads. Sie umfasst bewährte Methoden, Gestaltungsprinzipien und Fragen zur Bewertung und Überprüfung eines DynamoDB-Workloads. Bei Durchführung einer Überprüfung unter Verwendung der Amazon DynamoDB Well-Architected Lens erhalten Sie Informationen und Anleitungen zu den empfohlenen Gestaltungsprinzipien für die einzelnen

AWS -Well-Architected-Säulen. Diese Anleitungen stützen sich auf unsere Erfahrung in der Zusammenarbeit mit Kunden unterschiedlicher Größe aus verschiedenen Branchen, Segmenten und Regionen.

Als direktes Ergebnis der Überprüfung unter Verwendung der Well-Architected Lens erhalten Sie eine Zusammenfassung umsetzbarer Empfehlungen zur Optimierung und Verbesserung Ihres DynamoDB-Workloads.

Durchführung der Überprüfung unter Verwendung der Amazon DynamoDB Well-Architected Lens

Die Überprüfung von DynamoDB Well-Architected Lens wird in der Regel von einem AWS Solutions Architect zusammen mit dem Kunden durchgeführt, kann aber auch vom Kunden als Self-Service durchgeführt werden. Wir empfehlen zwar, im Rahmen der Amazon DynamoDB Well-Architected Lens alle sechs Säulen des Well-Architected Frameworks zu überprüfen, Sie können sich aber auch zunächst auf eine oder mehrere Säulen konzentrieren.

Zusätzliche Informationen und Anweisungen zur Durchführung einer Überprüfung von Amazon DynamoDB Well-Architected Lens finden Sie in diesem Video und auf der Seite DynamoDB Well-Architected Lens. GitHub

Die Säulen der Amazon DynamoDB Well-Architected Lens

Die Amazon DynamoDB Well-Architected Lens beruht auf sechs Säulen:

Säule der Leistungseffizienz

Die Säule der Leistungseffizienz betrifft die Fähigkeit zur effizienten Nutzung von Computerressourcen, um die Systemanforderungen zu erfüllen, sowie die Möglichkeit zur Aufrechterhaltung dieser Effizienz bei Nachfrageänderungen und einer Weiterentwicklung der Technologien.

Die wichtigsten Gestaltungsprinzipien von DynamoDB für diese Säule beziehen sich auf die Modellierung der Daten, die Auswahl von Partitionsschlüsseln und Sortierschlüsseln sowie die Definition sekundärer Indizes auf der Grundlage der Anwendungszugriffsmuster. Zu den weiteren Überlegungen gehören die Auswahl des optimalen Durchsatzmodus für den Workload, die AWS SDK-Optimierung und gegebenenfalls die Verwendung einer optimalen Caching-Strategie. Wenn Sie mehr über diese Gestaltungsprinzipien erfahren möchten, sehen Sie sich dieses vertiefende Video über die Säule der Leistungseffizienz der DynamoDB Well-Architected Lens an.

Säule der Kostenoptimierung

Die Säule der Kostenoptimierung konzentriert sich auf die Vermeidung unnötiger Kosten.

Zu den wichtigsten Themen gehören das Verständnis und die Kontrolle darüber, wofür Geld ausgegeben wird, die Auswahl der am besten geeigneten Ressourcentypen in der richtigen Anzahl, die Analyse der Ausgaben im Laufe der Zeit, die Gestaltung Ihrer Datenmodelle zur Optimierung der Kosten für anwendungsspezifische Zugriffsmuster und eine den Geschäftsanforderungen entsprechende Skalierung ohne zu hohe Ausgaben.

Die wichtigsten Gestaltungsprinzipien in Bezug auf die Kostenoptimierung für DynamoDB betreffen die Auswahl des optimalen Kapazitätsmodus und der am besten geeigneten Tabellenklasse für Ihre Tabellen sowie die Vermeidung einer übermäßigen Kapazitätsbereitstellung durch Verwendung des On-Demand-Kapazitätsmodus oder des Modus mit bereitgestellter Kapazität bei gleichzeitiger Nutzung einer automatischen Skalierung. Weitere Aspekte sind eine effiziente Datenmodellierung und -abfrage zur Reduzierung der verbrauchten Kapazität, die Reservierung von Teilen der verbrauchten Kapazität zu einem reduzierten Preis, die Minimierung der Elementgröße, die Ermittlung und Entfernung von ungenutzten Ressourcen sowie die Verwendung von TTL, um veraltete Daten automatisch und kostenlos zu löschen. Wenn Sie mehr über diese Gestaltungsprinzipien erfahren möchten, sehen Sie sich dieses vertiefende Video über die Säule der Kostenoptimierung der DynamoDB Well-Architected Lens an.

Zusätzliche Informationen zu bewährten Methoden zur Kostenoptimierung für DynamoDB finden Sie unter Optimierung der Kosten.

Säule „Operational Excellence“

Die Säule „Operational Excellence“ konzentriert sich auf den Betrieb und die Überwachung von Systemen, um einen Mehrwert für das Unternehmen zu schaffen, sowie auf die kontinuierliche Verbesserung von Prozessen und Verfahren. Zu den wichtigsten Themen gehören die Automatisierung von Änderungen, die Reaktion auf Ereignisse und die Definition von Standards für die Verwaltung des täglichen Betriebs.

Zu den wichtigsten Designprinzipien für Operational Excellence für DynamoDB gehören die Überwachung von DynamoDB-Metriken über Amazon CloudWatch und die automatische Warnung AWS Config und Behebung von Abhilfemaßnahmen, wenn vordefinierte Schwellenwerte überschritten oder nicht konforme Regeln erkannt werden. Weitere Aspekte sind die Definition von DynamoDB-Ressourcen über Infrastruktur als Code und die Verwendung von Tags, um eine bessere Organisation, Ermittlung und Kostenabrechnung Ihrer DynamoDB-Ressourcen zu ermöglichen. Wenn

Sie mehr über diese Gestaltungsprinzipien erfahren möchten, sehen Sie sich dieses vertiefende Video über die Säule „Operational Excellence“ der DynamoDB Well-Architected Lens an.

Säule der Zuverlässigkeit

Die Säule der Zuverlässigkeit soll sicherstellen, dass ein Workload seine beabsichtigte Funktion durchgängig korrekt erfüllt, wenn dies erwartet wird. Ein belastbarer Workload erholt sich schnell von Ausfällen und kann so den Geschäfts- und Kundenanforderungen gerecht werden. Zu den wichtigsten Themen gehören das Design verteilter Systeme, die Wiederherstellungsplanung und der Umgang mit Veränderungen.

Die grundlegenden Prinzipien des Zuverlässigkeitsdesigns für DynamoDB basieren auf der Auswahl der Backup-Strategie und Aufbewahrung auf der Grundlage Ihrer RPO- und RTO-Anforderungen, der Verwendung globaler DynamoDB-Tabellen für multiregionale Workloads oder regionsübergreifende Disaster Recovery-Szenarien mit niedrigem RTO, der Implementierung von Wiederholungslogik mit exponentiellem Backoff in der Anwendung durch Konfiguration und Verwendung dieser Funktionen im AWS SDK und der Überwachung von DynamoDB-Metriken über Amazon sowie automatische Warnmeldungen und Problembehebungen, wenn vordefinierte Schwellenwerte überschritten werden. CloudWatch Wenn Sie mehr über diese Gestaltungsprinzipien erfahren möchten, sehen Sie sich dieses vertiefende Video über die Säule der Zuverlässigkeit der DynamoDB Well-Architected Lens an.

Säule der Sicherheit

Die Säule der Sicherheit konzentriert sich auf den Schutz von Informationen und Systemen. Zu den wichtigsten Themen gehören die Vertraulichkeit und Integrität von Daten, die Ermittlung und Verwaltung von Berechtigungen mithilfe der Rechteverwaltung, der Schutz von Systemen und die Einrichtung von Kontrollen zur Erkennung von Sicherheitsereignissen.

Die wichtigsten Gestaltungsprinzipien in Bezug auf Sicherheit für DynamoDB sind die Verschlüsselung von Daten während der Übertragung mit HTTPS, die Auswahl des Schlüsseltyps für die Verschlüsselung von Daten im Ruhezustand und die Definition der IAM-Rollen und -Richtlinien zur Authentifizierung und Autorisierung sowie zur Bereitstellung eines differenzierten Zugriffs auf DynamoDB-Ressourcen. Zu den weiteren Überlegungen gehört die Prüfung der DynamoDB-Steuerungsebene und der Datenebenenoperationen durch. AWS CloudTrail Wenn Sie mehr über diese Gestaltungsprinzipien erfahren möchten, sehen Sie sich dieses vertiefende Video über die Säule der Sicherheit der DynamoDB Well-Architected Lens an.

Zusätzliche Informationen zur Sicherheit für DynamoDB finden Sie unter Sicherheit.

Säule der Nachhaltigkeit

Die Säule der Nachhaltigkeit konzentriert sich auf die Minimierung der Umweltauswirkungen der Ausführung von Cloud-Workloads. Zu den wichtigsten Themen gehören das Modell der gemeinsamen Verantwortung für Nachhaltigkeit, das Verständnis der Auswirkungen und die Maximierung der Nutzung, um möglichst wenig Ressourcen zu benötigen und die Auswirkungen auf nachgelagerte Bereiche zu reduzieren.

Zu den wichtigsten Gestaltungsprinzipien in Bezug auf Nachhaltigkeit für DynamoDB gehören die Ermittlung und Entfernung ungenutzter DynamoDB-Ressourcen, die Vermeidung einer übermäßigen Bereitstellung durch Verwendung des On-Demand-Kapazitätsmodus oder des Modus mit bereitgestellter Kapazität bei gleichzeitiger Verwendung der automatischen Skalierung, effiziente Abfragen zur Verringerung des Kapazitätsverbrauchs und Reduzierung des Speicherbedarfs durch Komprimieren von Daten und Löschen veralteter Daten mithilfe von TTL. Wenn Sie mehr über diese Gestaltungsprinzipien erfahren möchten, sehen Sie sich dieses vertiefende Video über die Säule der Nachhaltigkeit der DynamoDB Well-Architected Lens an.

Bewährte Methoden für das Entwerfen und effektive Verwenden von Partitionsschlüsseln in DynamoDB

Der Primärschlüssel, der jedes Element in einer Amazon-DynamoDB-Tabelle eindeutig identifiziert, kann einfach (nur Partitionsschlüssel) oder zusammengesetzt (Partitionsschlüssel in Kombination mit Sortierschlüssel) sein.

Sie sollten die Anwendung für eine gleichmäßige Aktivität über alle Partitionsschlüssel in der Tabelle und ihre sekundären Indizes hinweg entwerfen. Sie können ermitteln, welche Zugriffsmuster Ihre Anwendung braucht und wie viele Lese- und Schreibeinheiten jede Tabelle und jeder sekundäre Index insgesamt benötigen.

> **Note**
>
> Adaptive Kapazität gilt für den On-Demand-Modus und die bereitgestellte Kapazität.

Jede Partition in einer DynamoDB-Tabelle ist so konzipiert, dass sie eine maximale Kapazität von 3 000 Leseeinheiten pro Sekunde und 1 000 Schreibeinheiten pro Sekunde bietet. Eine Leseeinheit entspricht einem strikt konsistenten Lesevorgang pro Sekunde oder zwei letztendlich konsistenten

Lesevorgängen pro Sekunde für ein Element mit einer Größe von bis zu 4 KB. Eine Schreibeinheit entspricht einem Schreibvorgang pro Sekunde für ein Element mit einer Größe von bis zu 1 KB.

Bei der Auswertung der Durchsatzgrenzen für die Partitionen Ihrer Tabelle müssen Sie die Größe der Elemente berücksichtigen. Wenn die Tabelle beispielsweise Elemente mit einer Größe von 20 KB enthält, verbraucht ein einziger konsistenter Lesevorgang 5 Leseeinheiten. Das bedeutet, dass Sie für dieses einzelne Element 600 konsistente Lesevorgänge pro Sekunde gleichzeitig ausführen können, bevor Sie die Partitionsgrenzen erreichen. Der Gesamtdurchsatz über alle Partitionen in der Tabelle hinweg kann durch den bereitgestellten Durchsatz im Modus bereitgestellter Kapazität oder durch die Durchsatzgrenze auf Tabellenebene im On-Demand-Modus eingeschränkt sein. Weitere Informationen finden Sie unter Servicekontingente.

Themen

- Entwerfen von Partitionsschlüsseln zum Verteilen Ihres Workloads in DynamoDB
- Verwenden von Schreib-Sharding zur gleichmäßigen Verteilung von Workloads in Ihrer DynamoDB-Tabelle
- Effiziente Verteilung der Schreibaktivität beim Hochladen von Daten in DynamoDB

Entwerfen von Partitionsschlüsseln zum Verteilen Ihres Workloads in DynamoDB

Der Partitionsschlüsselabschnitt des primären Schlüssels einer Tabelle legt die logischen Partitionen fest, in denen die Daten einer Tabelle gespeichert werden. Dies wirkt sich wiederum auf die zugrunde liegenden physischen Partitionen aus. Ein Partitionsschlüsseldesign, das I/O Anfragen nicht effektiv verteilt, kann zu „heißen“ Partitionen führen, die zu Drosselung führen und Ihre bereitgestellte Kapazität ineffizient nutzen. I/O

Die optimale Nutzung des bereitgestellten Durchsatzes einer Tabelle ist nicht nur von den Workload-Mustern einzelner Elemente abhängig, sondern auch vom Design des Partitionsschlüssels. Dies bedeutet nicht, dass Sie auf alle Partitionsschlüsselwerte zugreifen müssen, um einen effizienten Durchsatz zu erzielen. Es bedeutet noch nicht einmal, dass der Prozentsatz der Partitionsschlüsselwerte, auf die zugegriffen wird, hoch sein muss. Es bedeutet jedoch, dass die Anforderungen umso weiter über den partitionierten Bereich verteilt werden, je mehr unterschiedliche Partitionsschlüsselwerte dem Zugriff durch Ihren Workload unterliegen. Im Allgemeinen nutzen Sie den bereitgestellten Durchsatz effizienter, wenn das Verhältnis zwischen den Partitionsschlüsselwerten, die einem Zugriff unterliegen, und der Gesamtzahl der Partitionsschlüsselwerte zunimmt.

Im Folgenden finden Sie einen Vergleich der Effizienz des bereitgestellten Durchsatzes für einige häufige Partitionsschlüsselschemas.

Partitionsschlüsselwert	Gleichmäßigkeit
Benutzer-ID, in der die Anwendung viele Benutzer hat.	Gut
Statuscode, in dem es nur einige wenige mögliche Statuscodes gibt.	Schlecht
Erstellungsdatum eines Elements, gerundet auf den nächsten Zeitraum (z. B. Tag, Stunde oder Minute).	Schlecht
Geräte-ID, in der jedes Gerät mit relativ ähnlichen Intervallen auf Daten zugreift.	Gut
Geräte-ID, wobei eine bei weitem beliebter als alle anderen ist, auch wenn es viele überwachte Geräte gibt.	Schlecht

Wenn eine einzelne Tabelle nur eine kleine Anzahl von Partitionsschlüsselwerten besitzt, sollten Sie in Betracht ziehen, die Schreibvorgänge auf mehrere unterschiedliche Partitionsschlüsselwerte zu verteilen. Mit anderen Worten, strukturieren Sie die Primärschlüsselelemente, um einen "heißen" (stark angeforderten) Partitions-Schlüsselwert, der die Gesamtleistung verlangsamt, zu vermeiden.

Nehmen wir als Beispiel eine Tabelle mit einem zusammengesetzten Primärschlüssel. Der Partitionsschlüssel repräsentiert das Erstelldatum des Elements, gerundet auf den nächsten Tag. Der Sortierschlüssel ist eine Element-ID. An einem bestimmten Tag, beispielsweise `2014-07-09`, werden alle neuen Elemente zu diesem bestimmten Partitionsschlüsselwert (und der entsprechenden physischen Partition) geschrieben.

Wenn die Tabelle vollständig in eine einzelne Partition passt (unter Berücksichtigung des Wachstums Ihrer Daten über die Zeit) und die Lese- und Schreibdurchsatzanforderungen Ihrer Anwendung die Lese- und Schreibkapazitäten einer einzelnen Partition nicht überschreiten, erfährt Ihre Anwendung keine unerwartete Drosselung durch eine Partitionierung.

Informationen zur Verwendung von NoSQL Workbench für DynamoDB zur Visualisierung Ihres Partitionsschlüsseldesigns finden Sie unter Erstellen von Datenmodellen mit NoSQL Workbench.

Verwenden von Schreib-Sharding zur gleichmäßigen Verteilung von Workloads in Ihrer DynamoDB-Tabelle

Eine Möglichkeit, Schreibvorgänge besser auf einen Partitionsschlüsselbereich in Amazon DynamoDB zu verteilen, besteht darin, den Speicherplatz zu erweitern. Dies kann auf verschiedene Arten geschehen. Sie können den Partitionsschlüsselwerten eine Zufallszahl hinzufügen, um die Elemente auf Partitionen zu verteilen. Oder Sie können eine Zahl verwenden, die basierend auf einer Abfrage berechnet wird.

Sharding unter Verwendung zufälliger Suffixe

Eine Strategie zur gleichmäßigeren Verteilung von Workloads auf einen Partitions-Schlüsselraum besteht darin, am Ende der Partitions-Schlüsselwerte eine zufällige Zahl hinzuzufügen. Anschließend werden die Schreibvorgänge zufällig auf den größeren Raum verteilt.

Für einen Partitionsschlüssel beispielsweise, der das Datum des aktuellen Tages repräsentiert, könnten Sie eine zufällige Zahl zwischen 1 und `200` auswählen, und diese als Suffix mit dem Datum verketten. So ergeben sich Partitions-Schlüsselwerte wie `2014-07-09.1`, `2014-07-09.2` usw. bis `2014-07-09.200`. Durch die Randomisierung des Partitionsschlüssels werden die Schreibvorgänge in die Tabelle jeden Tag gleichmäßig auf mehrere Partitionen verteilt. Dies führt zu einer besseren Parallelverarbeitung und einem höheren Gesamtdurchsatz.

Um alle Elemente für einen bestimmten Tag zu lesen, müssten Sie allerdings die Elemente für alle Suffixe abfragen und dann die Ergebnisse zusammenführen. Beispielsweise würden Sie zunächst eine `Query`-Anforderung für den Partitionsschlüsselwert `2014-07-09.1` ausgeben. Dann geben Sie eine andere `Query` für `2014-07-09.2`, und so weiter bis `2014-07-09.200`, aus. Schließlich müsste Ihre Anwendung die Ergebnisse von allen diesen `Query`-Anfragen zusammenführen.

Sharding unter Verwendung berechneter Suffixe

Mit einer Randomisierungsstrategie kann der Schreibdurchsatz erheblich verbessert werden. Allerdings ist es schwierig, ein bestimmtes Element zu lesen, da Sie nicht wissen, welcher Suffixwert beim Schreiben des Elements verwendet wurde. Um einzelne Elemente leichter lesen zu können, können Sie eine andere Strategie verwenden. Anstatt eine zufällige Zahl zu verwenden, um die Elemente auf die Partitionen zu verteilen, verwenden Sie eine Zahl, die Sie auf der Grundlage von etwas, für das Sie eine Abfrage durchführen möchten, berechnen können.

Denken Sie an das vorherige Beispiel, in dem eine Tabelle das Datum des aktuellen Tages im Partitionsschlüssel verwendet. Nehmen Sie nun an, dass jedes Element über ein zugängliches `OrderId`-Attribut verfügt und dass Sie Elemente meistens neben dem Datum anhand der Order-ID suchen müssen. Bevor Ihre Anwendung das Element in die Tabelle schreibt, könnte sie basierend auf der Order-ID ein Hash-Suffix berechnen und dem Partitionsschlüsseldatum anhängen. Die Berechnung könnte eine Zahl zwischen 1 und 200 ergeben, die ähnlich wie bei der Zufallsstrategie relativ gleichmäßig verteilt ist.

Eine einfache Berechnung würde wahrscheinlich genügen, beispielsweise das Produkt der UTF-8-Codepunktwerte für die Zeichen in der Order-ID, Modulo 200, +1. Der Partitions-Schlüsselwert wäre dann das Datum, verkettet mit dem Berechnungsergebnis.

Mit dieser Strategie werden die Schreibvorgänge gleichmäßig auf die Partitions-Schlüsselwerte und somit auch über die physischen Partitionen verteilt. Sie können problemlos eine `GetItem`-Operation für ein bestimmtes Element und Datum durchführen, da Sie den Partitions-Schlüsselwert für einen bestimmten `OrderId`-Wert berechnen können.

Um alle Elemente für einen bestimmten Tag zu lesen, müssen Sie immer noch alle `Query` von den `2014-07-09.N`-Schlüsseln (wobei `N` 1-200 ist) abfragen und anschließend muss die Anwendung alle Ergebnisse zusammenführen. Der Vorteil dabei ist, dass Sie so vermeiden, dass ein einzelner „Hot Partition”-Schlüsselwert die gesamte Workload übernimmt.

> **Note**
>
> Eine noch effizientere Strategie speziell für die Verarbeitung von großen Zeitreihendatenvolumen finden Sie im Abschnitt Zeitreihendaten.

Effiziente Verteilung der Schreibaktivität beim Hochladen von Daten in DynamoDB

Normalerweise partitioniert Amazon DynamoDB Ihre Tabellendaten auf mehreren Servern, wenn Sie Daten aus anderen Datenquellen laden. Sie erhalten eine bessere Leistung, wenn Sie Daten gleichzeitig auf alle zugewiesenen Server hochladen.

Angenommen, Sie möchten Benutzermitteilungen in eine DynamoDB-Tabelle hochladen, die einen zusammengesetzten Primärschlüssel mit `UserID` als Partitionsschlüssel und `MessageID` als Sortierschlüssel verwenden.

Wenn Sie die Daten hochladen, können Sie einen Ansatz verwenden, um alle Nachrichtenelemente für jeden Benutzer, einen Benutzer nach dem anderen, hochzuladen:

BenutzerID	MitteilungsID
B1	1
B1	2
B1	...
B1	... bis zu 100
B2	1
B2	2
B2	...
B2	... bis zu 200

Das Problem in diesem Fall besteht darin, dass Sie Ihre Schreibanforderungen nicht mittels Ihrer Partitionsschlüsselwerte in DynamoDB verteilen. Sie nehmen einen Partitionsschlüsselwert nach dem anderen und laden alle seine Elemente hoch, bevor Sie zum nächsten Partitionsschlüsselwert gehen und dasselbe tun.

Im Hintergrund partitioniert DynamoDB die Daten in Ihren Tabellen auf mehreren Servern. Um die gesamte Durchsatzkapazität, die für die Tabelle bereitgestellt wird, vollständig zu nutzen, müssen Sie Ihre Workload auf Ihre Partitionsschlüsselwerte verteilen. Indem Sie eine ungleichmäßige Menge der Upload-Arbeit auf Elemente lenken, die alle denselben Partitionsschlüsselwert haben, nutzen Sie nicht alle Ressourcen, die DynamoDB für Ihre Tabelle bereitgestellt hat.

Sie können Ihre Upload-Arbeit verteilen, indem Sie den Sortierschlüssel verwenden, um ein Element aus jedem Partitionsschlüsselwert und dann ein anderes Element aus jedem Partitionsschlüsselwert usw. zu laden:

BenutzerID	MitteilungsID
B1	1

BenutzerID	MitteilungsID
B2	1
B3	1
...	...
B1	2
B2	2
B3	2
...	...

Jeder Upload in dieser Sequenz verwendet einen anderen Partitionsschlüsselwert, was mehrere DynamoDB-Server gleichzeitig beschäftigt und Ihre Durchsatzleistung verbessert.

Bewährte Methoden für die Verwendung von Sortierschlüsseln zur Organisation von Daten in DynamoDB

In einer Amazon-DynamoDB-Tabelle besteht der Primärschlüssel, der jedes Element in der Tabelle eindeutig identifiziert, aus einem Partitionsschlüssel und einem Sortierschlüssel.

Gut konzipierte Sortierschlüssel haben zwei wichtige Vorteile:

- Sie bringen verwandte Informationen an einem Ort zusammen, an dem sie effizient abgefragt werden können. Wenn der Sortierschlüssel sorgfältig entworfen ist, können Sie allgemein benötigte Gruppen verwandter Elemente mithilfe von Bereichsabfragen mit Operatoren wie `begins_with`, `between`, >, < usw. abrufen.
- Mit zusammengesetzten Sortierschlüsseln können Sie hierarchische (one-to-many) Beziehungen in Ihren Daten definieren, die Sie auf jeder Hierarchieebene abfragen können.

  In einer Tabelle, in der geografische Standorte aufgeführt sind, können Sie den Sortierschlüssel beispielsweise wie folgt strukturieren.

  ```
  [country]#[region]#[state]#[county]#[city]#[neighborhood]
  ```

Sie könnten dann effiziente Bereichsabfragen für eine Liste von Standorten auf einer dieser Aggregationsebenen durchführen, von `country` bis hin zu `neighborhood` und auf allen Ebenen dazwischen.

Verwenden von Sortierschlüsseln für die Versionskontrolle

Viele Anwendungen müssen einen Versionsverlauf auf Elementebene für Prüfungs- oder Compliance-Zwecke pflegen und die neueste Version problemlos abrufen können. Es gibt ein effizientes Entwurfsmuster, das zu diesem Zweck Sortierschlüsselpräfixe verwendet:

- Erstellen Sie von jedem neuen Element zwei Kopien: Eine Kopie sollte das Versionsnummernpräfix null (z. B. `v0_`) zu Beginn des Sortierschlüssels haben, die andere das Versionsnummernpräfix eins (z. B. `v1_`).
- Verwenden Sie bei jeder Aktualisierung des Elements das nächsthöhere Versionspräfix im Sortierschlüssel der aktualisierten Version und kopieren Sie die aktualisierten Inhalte in das Element mit dem Versionspräfix null. So kann die neueste Version eines Elements leicht anhand des Präfixes null gefunden werden.

Ein Teilehersteller beispielsweise könnte ein wie im Folgenden dargestelltes Schema verwenden.

Primary Key		Data-Item Attributes...				
Partition Key *Equipment_ID*	Sort Key *(varies)*	Attribute 1	Attribute 2	Attribute 3	Attribute 4	...
Equipment_1	Details	Name: Biphasic Cardiometer *(equipment name)*	Factory_ID: S14_Tukwilla *(factory where manufactured)*	Line_ID R_7 *(assembly-line ID)*		
	v0_Audit	Auditor: Padma *(name of the auditor)*	Latest: 3 *(most recent audit version)*	Time: 2018-04-15T11:00 *(audit date and time)*	Result Passed *(audit result)*	...etc.
	v1_Audit	Auditor: Rick *(name of the auditor)*	Time: 2018-03-14T11:00 *(audit date and time)*	Result Open *(audit result)*	Report: 0943922EKG14 *(detailed problem report in S3)*	...etc.
	v2_Audit	Auditor: George *(name of the auditor)*	Time: 2018-03-18T11:00 *(audit date and time)*	Result Open *(audit result)*	Report: 0943923EKG15 *(detailed problem report in S3)*	...etc.
	v3_Audit	Auditor: Padma *(name of the auditor)*	Time: 2018-04-15T11:00 *(audit date and time)*	Result Passed *(audit result)*	Report: x792 *(pass confirmation report)*	...etc.

Das Element `Equipment_1` durchläuft eine Reihe von Prüfungen durch verschiedene Prüfer. Die Ergebnisse jeder neuen Prüfung werden in einem neuen Element in der Tabelle erfasst, dabei wird mit Versionsnummer eins begonnen und die Zahl wird bei jeder nachfolgenden Überarbeitung schrittweise erhöht.

Immer wenn eine neue Überarbeitung hinzugefügt wird, werden die Inhalte des Elements der Version null (mit dem Sortierschlüssel `v0_Audit`) von der Anwendungsebene durch die Inhalte der neuen Überarbeitung ersetzt.

Wenn die Anwendung den Status der aktuellsten Prüfung abrufen muss, kann sie das Sortierschlüsselpräfix `v0_` abfragen.

Wenn die Anwendung den gesamten Revisionsverlauf abrufen muss, kann sie alle Elemente unter dem Partitionsschlüssel des Elements abfragen und das Element `v0_` herausfiltern.

Dieses Design eignet sich auch für Prüfungen mehrerer Teile eines Geräts, wenn Sie das einzelne Teil- IDs in den Sortierschlüssel nach dem Sortierschlüsselpräfix aufnehmen.

Bewährte Methoden für die Verwendung sekundärer Indexe in DynamoDB

Sekundäre Indexe sind häufig unverzichtbar, um die von Ihrer Anwendung benötigten Abfragemuster zu unterstützen. Gleichzeitig kann eine übermäßige oder ineffiziente Verwendung sekundärer Indizes die Kosten unnötig erhöhen und die Leistung unnötig reduzieren.

Inhalt

- Allgemeine Richtlinien für sekundäre Indizes in DynamoDB
  - Effiziente Verwendung von Indizes
  - Sorgfältige Auswahl von Projektionen
  - Optimierung häufiger Abfragen zur Vermeidung von Abrufen
  - Berücksichtigung der Größenbegrenzungen von Elementauflistungen beim Erstellen lokaler sekundärer Indizes
- Verwendung von Sparse Indexes
  - Beispiele für Sparse Indexes in DynamoDB
- Verwenden globaler sekundärer Indizes für materialisierte Aggregationsabfragen in DynamoDB
  - Beispielszenario und Zugriffsmuster
  - Warum Aggregationen vorab berechnen
  - Tabellendesign
  - Aggregationspipeline mit Streams und AWS Lambda
  - Sparsamer GSI-Entwurf

  - Den GSI abfragen
  - Überlegungen
- Überladen globaler sekundärer Indizes in DynamoDB
- Verwenden des globalen sekundären Index mit Schreib-Sharding für selektive Tabellenabfragen in DynamoDB
  - Musterdesign
  - Sharding-Strategie
  - Abfragen der GSI-Shards
  - Überlegungen zur parallelen Abfrageausführung
  - Codebeispiel
- Verwenden globaler sekundärer Indizes zum Erstellen eines letztendlich konsistenten Replikats in DynamoDB

Allgemeine Richtlinien für sekundäre Indizes in DynamoDB

Amazon DynamoDB unterstützt zwei Arten sekundärer Indexe:

- Globaler sekundärer Index (GSI) – Dieser Index verfügt über einen Partitionsschlüssel und einen Sortierschlüssel, die nicht mit denen der Basistabelle übereinstimmen müssen. Ein globaler sekundärer Index wird als „global“ betrachtet, da Indexabfragen partitionsübergreifend alle Daten in der Basistabelle umfassen können. Ein globaler sekundärer Index unterliegt hinsichtlich seiner Größe keinen Einschränkungen und besitzt eigene Durchsatzeinstellungen für Lese- und Schreibaktivitäten, die von denen der übergeordneten Tabelle unabhängig sind.
- Lokaler sekundärer Index (LSI) – Dieser Index weist denselben Partitionsschlüssel wie die Basistabelle auf, hat aber einen anderen Sortierschlüssel. Ein lokaler sekundärer Index wird als „lokal“ betrachtet, da jede Partition eines lokalen sekundären Index auf die Basistabellenpartition bezogen ist, die denselben Partitionsschlüsselwert besitzt. Daher kann die Gesamtgröße der indizierten Elemente für jeden einzelnen Partitionsschlüsselwert 10 GB nicht überschreiten. Darüber hinaus besitzen lokale sekundäre Indizes die gleichen Durchsatzeinstellungen für Lese- und Schreibaktivitäten wie die Tabelle, die indiziert wird.

Jede Tabelle in DynamoDB kann bis zu 20 globale sekundäre Indizes (Standardkontingent) und 5 lokale sekundäre Indizes enthalten.

Globale Sekundärindizes sind oft nützlicher als lokale Sekundärindizes. Die Entscheidung, welcher Indextyp verwendet werden soll, hängt auch von den Anforderungen Ihrer Anwendung ab. Einen Vergleich der globalen sekundären Indizes und der lokalen sekundären Indizes sowie weitere Informationen zur Auswahl zwischen diesen Indizes finden Sie unter the section called “Arbeiten mit Indizes”.

Im Folgenden werden einige allgemeine Grundsätze und Designmuster beschrieben, die Sie beim Erstellen von Indizes in DynamoDB beachten sollten:

Themen

- Effiziente Verwendung von Indizes
- Sorgfältige Auswahl von Projektionen
- Optimierung häufiger Abfragen zur Vermeidung von Abrufen
- Berücksichtigung der Größenbegrenzungen von Elementauflistungen beim Erstellen lokaler sekundärer Indizes

Effiziente Verwendung von Indizes

Begrenzen Sie die Zahl der Indizes auf ein Minimum. Erstellen Sie keine sekundären Indizes für Attribute, die Sie nicht oft abfragen. Selten verwendete Indizes tragen zu mehr Speicherplatz und höheren I/O Kosten bei, ohne die Anwendungsleistung zu verbessern.

Sorgfältige Auswahl von Projektionen

Da sekundäre Indizes Speicher- und Durchsatzkapazitäten verbrauchen, sollten Sie dafür sorgen, dass die Indizes möglichst klein sind. Je kleiner der Index ist, desto größer ist auch der Leistungsvorteil im Vergleich zur Abfrage der vollständigen Tabelle. Wenn Ihre Abfragen in der Regel nur einen kleinen Teil der Attribute zurückgeben und die Gesamtgröße dieser Attribute sehr viel kleiner als das gesamte Element ist, sollten Sie nur die Attribute projizieren, die Sie regelmäßig anfordern.

Wenn Sie von davon ausgehen, dass eine Tabelle sehr viel mehr Schreibaktivitäten als Leseaktivitäten unterliegt, befolgen Sie diese bewährten Methoden:

- Ziehen Sie die Projizierung einer kleineren Anzahl von Attributen in Betracht, um die Größe der Elemente, die in den Index geschrieben werden, zu minimieren. Dies gilt jedoch nur, wenn die Größe der projizierten Attribute andernfalls die Größe einer einzelnen Schreibkapazitätseinheit (1 KB) überschreiten würde. Wenn z. B. die Größe eines Indexeintrags nur 200 Byte beträgt, rundet

DynamoDB diesen Wert auf 1 KB ab. Mit anderen Worten: Solange die Indexelemente klein sind, können Sie mehr Attribute ohne Zusatzkosten projizieren.

- Vermeiden Sie die Projizierung von Attributen, von denen Sie wissen, dass sie selten in Abfragen benötigt werden. Bei jeder Aktualisierung eines Attributs, das in einem Index projiziert ist, fallen auch zusätzliche Kosten für die Aktualisierung des Index an. Sie können nach wie vor die nicht projizierten Attribute in einer `Query` zu höheren Durchsatzkosten abrufen. Die Kosten für die Abfrage sind jedoch möglicherweise deutlich niedriger als die Kosten für die häufige Aktualisierung des Index.
- Geben Sie `ALL` nur an, wenn Ihre Abfragen das gesamte Tabellenelement sortiert nach einem anderen Sortierschlüssel zurückgeben sollen. Die Projizierung aller Attribute beseitigt die Notwendigkeit des Abrufens von Tabellen, verdoppelt jedoch in den meisten Fällen die Kosten für Speicherung und Schreibaktivitäten.

Stellen Sie ein Gleichgewicht zwischen der Notwendigkeit her, die Indizes so klein wie möglich zu halten, und der Notwendigkeit, die Zahl der Abrufe auf ein Minimum zu begrenzen.

Optimierung häufiger Abfragen zur Vermeidung von Abrufen

Um Abfragen so schnell wie möglich mit der geringstmöglichen Latenz auszuführen, sollten Sie alle Attribute projizieren, die von diesen Abfragen voraussichtlich zurückgegeben werden. Wenn Sie einen lokalen sekundären Index nach nicht projizierten Attributen abfragen, ruft DynamoDB diese Attribute automatisch aus der Tabelle ab. Dies erfordert, dass das gesamte Element aus der Tabelle gelesen wird. Dies führt zu Latenz und zusätzlichen I/O Vorgängen, die Sie vermeiden können.

Beachten Sie, dass in vielen Fällen aus „gelegentlichen“ Abfragen „wesentliche“ Abfragen werden. Wenn es Attribute gibt, die Sie nicht projizieren möchten, da Sie davon ausgehen, dass sie nur gelegentlich abgefragt werden, sollten Sie sich überlegen, ob sich die Umstände vielleicht ändern könnten und Sie es daher bedauern könnten, dass Sie diese Attribute nicht projiziert haben.

Weitere Informationen zum Abrufen von Tabellen finden Sie unter Überlegungen im Hinblick auf die bereitgestellte Durchsatzkapazität für lokale sekundäre Indizes.

Berücksichtigung der Größenbegrenzungen von Elementauflistungen beim Erstellen lokaler sekundärer Indizes

Eine Elementauflistung besteht aus allen Elementen in einer Tabelle und ihren lokalen sekundären Indizes, die denselben Partitionsschlüssel haben. Elementauflistungen dürfen 10 GB nicht

überschreiten. Daher ist es möglich, dass es für einen bestimmten Partitionsschlüsselwert keinen Platz mehr gibt.

Wenn Sie ein Tabellenelement hinzufügen oder aktualisieren, aktualisiert DynamoDB alle betroffenen lokalen sekundären Indizes. Wenn die indizierten Attribute in der Tabelle definiert sind, nimmt die Größe der lokalen sekundären Indizes ebenfalls zu.

Überlegen Sie beim Erstellen eines lokalen sekundären Index, wie viele Daten in den Index geschrieben werden und wie viele dieser Datenelemente denselben Partitionsschlüsselwert haben werden. Wenn die Summe der Tabellen- und Indexelemente für einen bestimmten Partitionsschlüsselwert 10 GB überschreiten könnte, sollten Sie überlegen, ob Sie die Erstellung des Index vermeiden sollten.

Wenn Sie die Erstellung des lokalen sekundären Index nicht vermeiden können, müssen Sie geeignete Maßnahmen ergreifen, bevor die Größeneinschränkung für die Elementauflistung überschritten wird. Als bewährte Methode sollten Sie den Parameter `ReturnItemCollectionMetrics` beim Schreiben von Elementen verwenden, um die Größe von Elementsammlungen, die sich der Größenbeschränkung von 10 GB nähern, zu überwachen und entsprechende Warnmeldungen zu erhalten. Eine Überschreitung der maximalen Größe für die Elementsammlung führt zu fehlgeschlagenen Schreibversuchen. Sie können die Probleme mit der Größe der Elementsammlung verringern, indem Sie die Größe der Elementsammlungen überwachen und entsprechende Warnmeldungen erhalten, bevor Ihre Anwendung beeinträchtigt wird.

> **Note**
>
> Sie können einen lokalen sekundären Index nach dem Erstellen nicht löschen.

Strategien zum Arbeiten innerhalb des Grenzwerts und zur Ergreifung von Korrekturmaßnahmen finden Sie unter Größenlimit der Elementauflistung.

Verwendung von Sparse Indexes

Für jedes Element in einer Tabelle schreibt DynamoDB nur dann einen entsprechenden Indexeintrag, wenn die Indexschlüsselattribute im Element vorhanden sind. Für einen globalen sekundären Index bedeutet dies, dass der Indexpartitionsschlüssel für das Element definiert sein muss, und wenn der Index auch einen Sortierschlüssel hat, muss dieses Attribut ebenfalls vorhanden sein. Wenn eines der identifizierenden Attribute in einem Element fehlt, erscheint dieses Element nicht im Index. Ein

Index, in dem nur eine Teilmenge der Elemente aus der Basistabelle erscheint, wird als Sparse-Index bezeichnet.

Sparse Indexes sind für Abfragen nützlich, die für einen kleinen Unterabschnitt einer Tabelle ausgeführt werden. Angenommen, Sie verfügen über eine Tabelle, in der Sie alle Kundenbestellungen speichern und die die folgenden Schlüsselattribute besitzt:

- Partitionsschlüssel: `CustomerId`
- Sortierschlüssel: `OrderId`

Um offene Bestellungen nachzuverfolgen, können Sie ein Attribut mit dem Namen `isOpen` in Bestellelemente einfügen, die noch nicht versendet wurden. Wenn die Bestellung versendet wird, können Sie das Attribut löschen. Wenn Sie anschließend einen Index für `CustomerId` (Partitionsschlüssel) und `isOpen` (Sortierschlüssel) erstellen, enthält dieser nur Bestellungen, für die `isOpen` definiert ist. Wenn es Tausende von Bestellungen gibt, von denen nur eine kleine Zahl offen ist, ist es schneller und kostengünstiger, diesen Index für offene Bestellungen abzufragen, statt die ganze Tabelle zu scannen.

Anstelle eines Attributs wie `isOpen` könnten Sie ein Attribut mit einem Wert verwenden, der eine nützliche Sortierreihenfolge im Index ergibt. Sie könnten beispielsweise das Attribut `OrderOpenDate` verwenden, das auf das Datum festgelegt ist, an dem die einzelnen Bestellungen platziert wurden, und dieses löschen, nachdem die Bestellung ausgeführt wurde. Auf diese Weise werden die Elemente sortiert nach dem Datum zurückgegeben, an dem die einzelnen Bestellungen platziert wurden, wenn Sie den Sparse Index abfragen.

Beispiele für Sparse Indexes in DynamoDB

Globale sekundäre Indizes sind standardmäßig Sparse Indexes. Wenn Sie einen globalen sekundären Index erstellen, geben Sie einen Partitionsschlüssel und optional einen Sortierschlüssel an. Nur Elemente in der Basistabelle, die die erforderlichen Schlüsselattribute enthalten, werden im Index angezeigt. Wenn bei einem Element der Indexpartitionsschlüssel — oder der Sortierschlüssel, sofern einer definiert ist — fehlt, wird dieses Element aus dem Index ausgeschlossen.

Wenn Sie einen globalen sekundären Index als Sparse Index festlegen, können Sie ihn mit einem Schreibdurchsatz bereitstellen, der unter dem der Basistabelle liegt, und dennoch eine herausragende Leistung erzielen.

Eine Gaming-Anwendung könnte beispielsweise zwar alle Punktzahlen sämtlicher Benutzer nachverfolgen, muss jedoch in der Regel nur einige hohe Punktzahlen abfragen. Im folgenden Design wird dieses Szenario effizient behandelt:

Table

Primary Key		Data Attributes...		
Partition Key	*Sort Key*			
Player_ID	Game_ID	Attribute 1	Attribute 2	Attribute 3
Rick	Game_1	Score: 36,750 *(game score)*	Date: 2017-11-14 *(date of game)*	
	Game_2	Score: 69,450 *(game score)*	Date: 2017-12-31 *(date of game)*	
	Game_3	Score: 135,900 *(game score)*	Date: 2018-01-19 *(date of game)*	Award: Champ *(type of award)*
Padma	Game_4	Score: 25,350 *(game score)*	Date: 2018-01-27 *(date of game)*	
	Game_5	Score: 69,450 *(game score)*	Date: 2028-01-19 *(date of game)*	
	Game_6	Score: 147,300 *(game score)*	Date: 2018-02-02 *(date of game)*	Award: Champ *(type of award)*
	Game_7	Score: 169,100 *(game score)*	Date: 2018-03-10 *(date of game)*	Award: Champ *(type of award)*

Hier hat Rick drei Spiele gespielt und in einem der Spiele den Status `Champ` erreicht. Padma hat vier Spiele gespielt und in zwei der Spiele den Status `Champ` erreicht. Beachten Sie, dass das Attribut `Award` nur in Elementen vorhanden ist, in denen der Benutzer eine Auszeichnung erhalten hat. Der verknüpfte globale sekundäre Index sieht wie folgt aus:

GSI

Primary Key	Projected Attributes...			
Partition Key				
Award	Player_ID	Game_ID	Score	Date
Champ	Rick	Game_3	135,900	2018-01-19
	Padma	Game_6	147,300	2018-02-02
	Padma	Game_7	169,100	2018-03-10

Der globale sekundäre Index enthält nur die hohen Punktzahlen, die häufig abgefragt werden. Dies ist ein kleiner Teilsatz der Elemente in der Basistabelle.

Verwenden globaler sekundärer Indizes für materialisierte Aggregationsabfragen in DynamoDB

Die Wartung von Aggregationen, die beinahe in Echtzeit ausgeführt werden, und Schlüsselmetriken für Daten, die sich schnell verändern, wird zunehmend wichtiger für Unternehmen, um schnell Entscheidungen treffen zu können. Beispielsweise möchte eine Musikbibliothek ihre am häufigsten heruntergeladenen Songs nahezu in Echtzeit präsentieren, oder eine E-Commerce-Plattform muss möglicherweise Trendprodukte nach Kategorien anzeigen.

Da DynamoDB keine systemeigenen Aggregationsoperationen wie `SUM` oder `COUNT` elementübergreifend unterstützt, würde die Berechnung dieser Werte zum Zeitpunkt des Lesens das Scannen einer großen Anzahl von Elementen erfordern — was langsam und teuer sein kann. Stattdessen können Sie Aggregationen vorab berechnen, wenn sich Daten ändern, und die Ergebnisse als reguläre Elemente in Ihrer Tabelle speichern. Dieses Muster wird als materialisierte Aggregation bezeichnet.

Themen

- Beispielszenario und Zugriffsmuster
- Warum Aggregationen vorab berechnen
- Tabellendesign
- Aggregationspipeline mit Streams und AWS Lambda
- Sparsamer GSI-Entwurf
- Den GSI abfragen
- Überlegungen

Beispielszenario und Zugriffsmuster

Stellen Sie sich eine Musikbibliotheksanwendung mit den folgenden Anforderungen vor:

- Die Anwendung zeichnet einzelne Lied-Downloads mit hoher Lautstärke (Tausende pro Sekunde) auf.
- Benutzer müssen die am häufigsten heruntergeladenen Songs für einen bestimmten Monat mit einer Latenz im einstelligen Millisekundenbereich sehen.
- Die Anwendung muss auch Abfragen wie „Die 10 besten Songs dieses Monats“ und „Alle Songs, die in einem bestimmten Monat heruntergeladen wurden“ unterstützen.

Bei dieser Größenordnung kann es teuer sein, die Anzahl der Downloads zum Zeitpunkt des Lesens zu berechnen, indem alle Download-Datensätze gescannt werden. Stattdessen können Sie eine fortlaufende Zählung beibehalten, die bei jedem Download aktualisiert wird, und diese so speichern, dass effiziente Abfragen unterstützt werden.

Warum Aggregationen vorab berechnen

Es gibt verschiedene Ansätze zur Berechnung von Aggregationen. In der folgenden Tabelle werden gängige Alternativen verglichen und erklärt, warum die materialisierte Aggregation in DynamoDB für diese Art von Anwendungsfall oft am besten geeignet ist.

Ansatz	Kompromisse	Wann sollte dies verwendet werden?
Zum Zeitpunkt des Lesens scannen und zählen	Erfordert das Lesen aller Download-Datensätze für jede Abfrage. Die Latenz wächst mit dem Datenvolumen und verbraucht erhebliche Lesekapazität.	Nur für sehr kleine Datensätze geeignet, bei denen die Latenz kein Problem darstellt.
Externer Aggregationsspeicher (z. B. Amazon ElastiCache)	Erhöht die betriebliche Komplexität durch einen separaten Service, der verwaltet werden muss. Erfordert Synchronisationslogik zwischen DynamoDB und dem Cache.	Wenn Sie Lesevorgänge unter einer Millisekunde oder eine komplexe Aggregationslogik benötigen, die über einfache Zählungen hinausgeht.
Aggregation auf Anwendungsebene beim Schreiben	Koppelt die Aggregationslogik an den Schreibpfad. Wenn die Anwendung nach der Aufzeichnung des Downloads, aber vor der Aktualisierung der Anzahl fehlschlägt, wird die Aggregation inkonsistent.	Wenn Sie eine synchrone, stark konsistente Aggregation benötigen und zusätzliche Schreiblatenz tolerieren können.
Materialisierte Aggregation mit Streams und Lambda	Entkoppelt die Aggregation vom Schreibpfad. Die Aggregation ist letztendlich konsistent (normalerweise Sekunden später). Fügt Lambda-Aufrufkosten hinzu.	Wenn Sie Aggregationen nahezu in Echtzeit mit geringer Leselatenz benötigen und eine eventuelle Konsistenz tolerieren können. Dieser Ansatz wird auf dieser Seite beschrieben.

Der Ansatz der materialisierten Aggregation hält den Schreibpfad einfach (zeichnet einfach den Download auf), verlagert die Aggregation in einen asynchronen Prozess und speichert das Ergebnis in DynamoDB, wo es mit einer Latenz im einstelligen Millisekundenbereich abgefragt werden kann.

Tabellendesign

Dieser Entwurf verwendet eine einzelne Tabelle mit zwei Elementtypen, die denselben Partitionsschlüssel () verwenden, aber unterschiedliche Sortierschlüsselmuster verwenden, um zwischen ihnen zu unterscheiden: `songID`

- Datensätze herunterladen — Einzelne Download-Ereignisse. Der Sortierschlüssel ist der `DownloadID` (eine eindeutige Kennung für jeden Download).
- Monatliche Aggregationselemente — Vorab berechnete Download-Zahlen pro Song und Monat. Der Sortierschlüssel gibt das `YYYY-MM` Format des Monats an (z. B.). `2018-01` Diese Elemente enthalten auch ein `DownloadCount` Attribut mit der laufenden Summe.

Nur die monatlichen Aggregationselemente enthalten das `Month` Attribut. Diese Unterscheidung ist wichtig für das später beschriebene, spärliche GSI-Design.

Das folgende Diagramm zeigt das Tabellenlayout mit beiden Elementtypen:

Music Library Table

Primary Key		Data-Item Attributes...			
Partition Key	Sort Key	Attribute 1	Attribute 2	Attribute 3	
Song-129 *(song ID)*	Details	Title: Wild Love *(song title)*	Artist: Argyboots *(artist or band name)*	Downloads: 15,314,822 *(lifetime total downloads)*	...etc.
	Month-2018-01	GSI Primary Key Month: 2018-01 *(download month)*	GSI Secondary Key MonthTotal: 1,746,992 *(month total downloads)*		
	Dld-9349823681	Time: 2018-01-01T00:00:07 *(download timestamp)*			
	Dld-9349823682	Time: 2018-01-01T00:00:07 *(download timestamp)*			
	Dld-9349823683	Time: 2018-01-01T00:00:07 *(download timestamp)*			

Elementart	Partitionsschlüssel (SongID)	Sortierschlüssel	Zusätzliche Attribute
Datensatz herunterladen	`song1`	`download-abc123`	`UserID`, `Timestamp`
Monatliche Aggregation	`song1`	`2018-01`	`Month=2018-01`, `DownloadCount =1,746,992`

Aggregationspipeline mit Streams und AWS Lambda

Die Aggregationspipeline funktioniert wie folgt:

1. Wenn ein Song heruntergeladen wird, schreibt die Anwendung mit `Partition-Key=songID` und `Sort-Key=DownloadID` ein neues Element in die Tabelle.
2. DynamoDB Streams erfasst diesen Schreibvorgang als Stream-Record.
3. Eine Lambda-Funktion, die an den Stream angehängt ist, verarbeitet den neuen Datensatz. Sie identifiziert den `songID` und den aktuellen Monat und aktualisiert dann das entsprechende monatliche Aggregationselement, indem das Attribut inkrementiert wird. `DownloadCount`
4. Das aktualisierte Aggregationselement ist dann für Abfragen über den globalen Index mit geringer Dichte verfügbar.

Die Lambda-Funktion verwendet einen `UpdateItem` Aufruf mit einem `ADD` Ausdruck, um die Anzahl der Downloads atomar zu erhöhen. Dadurch werden Rennbedingungen vermieden read-modify-write:

```
import boto3

dynamodb = boto3.resource('dynamodb')
table = dynamodb.Table('MusicLibrary')

def handler(event, context):
    for record in event['Records']:
        if record['eventName'] == 'INSERT':
            new_image = record['dynamodb']['NewImage']
            song_id = new_image['songID']['S']
            # Derive the month from the download timestamp
```

```
            timestamp = new_image['Timestamp']['S']
            month = timestamp[:7]  # Extract YYYY-MM

            table.update_item(
                Key={
                    'songID': song_id,
                    'SK': month
                },
                UpdateExpression='ADD DownloadCount :inc SET #m = :month',
                ExpressionAttributeNames={
                    '#m': 'Month'
                },
                ExpressionAttributeValues={
                    ':inc': 1,
                    ':month': month
                }
            )
```

> **Note**
>
> Wenn eine Lambda-Ausführung nach dem Schreiben des aktualisierten Aggregationswerts fehlschlägt, kann der Stream-Datensatz erneut versucht werden. Da der *ADD* Vorgang die Anzahl bei jeder Ausführung erhöht, würde ein erneuter Versuch die Anzahl für denselben Download mehrmals erhöhen, sodass Sie einen ungefähren Wert erhalten. Für die meisten Anwendungsfälle in den Bereichen Analytik und Bestenlisten ist diese geringe Fehlerquote akzeptabel. Wenn Sie genaue Zahlen benötigen, sollten Sie erwägen, eine Idempotenzlogik hinzuzufügen, z. B. indem Sie einen Bedingungsausdruck verwenden, der überprüft, ob die spezifische Zahl bereits verarbeitet wurde. `DownloadID`

Sparsamer GSI-Entwurf

Um die aggregierten Ergebnisse effizient abzufragen, erstellen Sie einen globalen sekundären Index mit dem folgenden Schlüsselschema:

- GSI-Partitionsschlüssel: **`Month`** (Zeichenfolge)
- GSI-Sortierschlüssel: **`DownloadCount`** (Zahl)

Dieser globale Index ist spärlich, da das Attribut nur in den monatlichen Aggregationselementen enthalten ist. `Month` Die einzelnen Download-Datensätze haben dieses Attribut nicht und werden

daher automatisch aus dem Index ausgeschlossen. Das bedeutet, dass der globale Index nur die vorberechneten Aggregationselemente enthält — einen kleinen Teil der gesamten Elemente in der Tabelle.

Ein spärlicher globaler Index bietet zwei wichtige Vorteile:

- Niedrigere Kosten — Da nur Aggregationselemente in den Index repliziert werden, verbrauchen Sie im Vergleich zu einem Index, der jedes Element in der Tabelle umfasst, deutlich weniger Schreibkapazität und Speicherplatz.
- Schnellere Abfragen — Der Index enthält nur die Daten, die Sie abfragen müssen, sodass Lesevorgänge effizient sind und Ergebnisse mit einer Latenz im einstelligen Millisekundenbereich zurückgegeben werden.

Weitere Hinweise zur Funktionsweise von Indizes mit geringer Dichte finden Sie unter. Verwendung von Sparse Indexes

Den GSI abfragen

Wenn der globale Index mit geringer Dichte eingerichtet ist, können Sie verschiedene Arten von Abfragen effizient beantworten:

Holen Sie sich den Song, der in einem bestimmten Monat am häufigsten heruntergeladen wurde:

```
aws dynamodb query \
    --table-name "MusicLibrary" \
    --index-name "MonthDownloadsIndex" \
    --key-condition-expression "#m = :month" \
    --expression-attribute-names '{"#m": "Month"}' \
    --expression-attribute-values '{":month": {"S": "2018-01"}}' \
    --scan-index-forward false \
    --limit 1
```

Bei Einstellung `ScanIndexForward` auf werden die Ergebnisse `DownloadCount` in absteigender Reihenfolge `false` sortiert und es wird nur der Titel `Limit=1` zurückgegeben, der am häufigsten gespielt wird.

Ruft die 10 besten Songs für einen bestimmten Monat ab:

```
aws dynamodb query \
```

```
    --table-name "MusicLibrary" \
    --index-name "MonthDownloadsIndex" \
    --key-condition-expression "#m = :month" \
    --expression-attribute-names '{"#m": "Month"}' \
    --expression-attribute-values '{":month": {"S": "2018-01"}}' \
    --scan-index-forward false \
    --limit 10
```

Erhalte alle Songs, die in einem bestimmten Monat heruntergeladen wurden (sortiert nach Anzahl der Downloads):

```
aws dynamodb query \
    --table-name "MusicLibrary" \
    --index-name "MonthDownloadsIndex" \
    --key-condition-expression "#m = :month" \
    --expression-attribute-names '{"#m": "Month"}' \
    --expression-attribute-values '{":month": {"S": "2018-01"}}' \
    --scan-index-forward false
```

Überlegungen

Beachten Sie bei der Implementierung dieses Musters Folgendes:

- Eventuelle Konsistenz — Die Aggregationswerte werden asynchron über DynamoDB Streams und Lambda aktualisiert. Zwischen der Aufzeichnung eines Downloads und der Aktualisierung der Aggregation liegt in der Regel eine Verzögerung von einigen Sekunden. Das bedeutet, dass der GSI Daten nahezu in Echtzeit wiedergibt, keine Echtzeitdaten.
- Lambda-Parallelität — Wenn Ihre Tabelle ein hohes Schreibvolumen hat, können mehrere Lambda-Aufrufe versuchen, dasselbe Aggregationselement gleichzeitig zu aktualisieren. Die atomare `ADD` Operation handhabt dies sicher, aber Sie sollten die Lambda-Parallelität und die Drosselungsmetriken überwachen, um sicherzustellen, dass Ihre Funktion mit dem Stream Schritt halten kann.
- GSI-Schreibkapazität — Da die dünn besetzte GSI nur Aggregationselemente enthält, benötigt sie deutlich weniger Schreibkapazität als die Basistabelle. Sie sollten jedoch dennoch genügend Kapazität bereitstellen (oder den On-Demand-Modus verwenden), um die Geschwindigkeit der Aggregationsaktualisierungen bewältigen zu können.
- Ungefähre Anzahl — Wie bereits erwähnt, können Lambda-Wiederholungen dazu führen, dass die Anzahl leicht überzählt wird. Implementieren Sie für Anwendungsfälle, die genaue Zählungen erfordern, Idempotenzprüfungen in der Lambda-Funktion.

Überladen globaler sekundärer Indizes in DynamoDB

Auch wenn Amazon DynamoDB ein Standardkontingent von globalen sekundären Indexen pro Tabelle aufweist, können Sie in der Praxis sehr viel mehr als 20 Datenfelder indizieren. Im Gegensatz zu Tabellen in relationalen Datenbankmanagementsystemen (RDBMS), in denen das Schema gleichförmig ist, können Tabellen in DynamoDB viele verschiedene Arten von Datenelementen gleichzeitig enthalten. Zusätzlich kann das gleiche Attribut in verschiedenen Elementen völlig andere Arten von Informationen enthalten.

Betrachten Sie das folgende Beispiel eines Layouts für eine DynamoDB-Tabelle, die eine Vielzahl unterschiedlicher Arten von Daten speichert.

Primary Key		Data-Item Attributes...		
Partition Key	Sort Key	Attribute 1	Attribute 2	...
HR-974 *(employee ID)*	Employee_Name	Data: Murphy, John *(employee name)*	Start: 2008-11-08 *(start date)*	...etc.
	YYYY-Q1	Data: $5,477 *(order totals in USD)*	Name: Murphy, John *(employee name)*	
	HR_confidential	Data: 2008-11-08 *(hire date)*	Name: Murphy, John *(employee name)*	...etc.
	Warehouse_01	Data: Murphy, John *(employee name)*		
	v0_Job_title	Data: Operator-1 *(job title)*	Start: 2008-11-08 *(start date)*	...etc.
	v1_Job_title	Data: Operator-2 *(job title)*	Start: 2016-11-04 *(start date)*	...etc.
	v2_Job_title	Data: Supervisor-1 *(job title)*	Start: 2017-11-01 *(start date)*	...etc.

Das Attribut `Data`, das allen Elementen gemeinsam ist, hat abhängig vom übergeordneten Element unterschiedliche Inhalte. Wenn Sie einen globalen sekundären Index für die Tabelle erstellen, die den Sortierschlüssel der Tabelle als Partitionsschlüssel und das Attribut `Data` als Sortierschlüssel

verwendet, können Sie verschiedene Abfragen ausführen, die alle diesen globalen sekundären Index verwenden. Zu diesen Abfragen können gehören:

- Nachschlagen des Namens eines Mitarbeiters im globalen sekundären Index mit `Employee_Name` als Partitionsschlüsselwert und den Namen des Mitarbeiters (z. B. `Murphy, John`) als Sortierschlüsselwert.
- Verwendung des globalen sekundären Index für die Suche nach allen Mitarbeitern, die in einem bestimmten Lager arbeiten, durch Suchen anhand einer Lager-ID (wie `Warehouse_01`)
- Rufen Sie eine Liste der letzten Einstellungen ab, fragen Sie den globalen sekundären Index für `HR_confidential` als Partitionsschlüsselwert ab und verwenden Sie einen Datumsbereich im Sortierschlüsselwert.

Verwenden des globalen sekundären Index mit Schreib-Sharding für selektive Tabellenabfragen in DynamoDB

Wenn Sie aktuelle Daten innerhalb eines bestimmten Zeitfensters abfragen müssen, kann die Anforderung von DynamoDB, für die meisten Lesevorgänge einen Partitionsschlüssel bereitzustellen, eine Herausforderung darstellen. Um dieses Szenario zu bewältigen, können Sie mithilfe einer Kombination aus Schreib-Sharding und einem globalen sekundären Index (GSI) ein effektives Abfragemuster implementieren.

Mit diesem Ansatz können Sie zeitkritische Daten effizient abrufen und analysieren, ohne vollständige Tabellenscans durchführen zu müssen, die ressourcenintensiv und kostspielig sein können. Durch die strategische Gestaltung Ihrer Tabellenstruktur und Indizierung können Sie eine flexible Lösung entwickeln, die den zeitbasierten Datenabruf unterstützt und gleichzeitig eine optimale Leistung beibehält.

Themen

- Musterdesign
- Sharding-Strategie
- Abfragen der GSI-Shards
- Überlegungen zur parallelen Abfrageausführung
- Codebeispiel

Musterdesign

Wenn Sie mit DynamoDB arbeiten, können Sie Probleme beim zeitbasierten Datenabruf bewältigen, indem Sie ein komplexes Muster implementieren, das Schreib-Sharding und globale sekundäre Indizes kombiniert, um flexible und effiziente Abfragen in aktuellen Datenfenstern zu ermöglichen.

Struktur der Tabelle

- Partitionsschlüssel (PK): „Benutzername“

Struktur des GSI

- GSI-Partitionsschlüssel (PK_GSI): "#“ ShardNumber
- GSI-Sortierschlüssel (SK_GSI): ISO 8601-Zeitstempel (z. B. „2030-04-01T12:00:00Z“)

[TABLE] - TimeBounded Metadata Actions

PK	SK_GSI	Address	PK_GSI
Trenton69	2023-12-05T06:40:31.312Z	117 Hudson Divide	ShardNumber#0
Scot_Langworth-Prosacco	2024-10-09T14:44:45.819Z	84021 Herzog Skyway	ShardNumber#6
Juliet.Morissette	2025-03-28T07:20:56.538Z	4871 N Broadway	ShardNumber#4
Kay.Von71	2024-11-23T16:19:26.763Z	14554 Odell Throughway	ShardNumber#2
Kiarra43	2023-11-15T16:21:57.078Z	8471 London Road	ShardNumber#1
Marcella91	2024-12-17T09:02:31.623Z	10271 Nick Crescent	ShardNumber#6
Bernard.Lemke36	2025-09-09T00:18:34.482Z	5438 Douglas Groves	ShardNumber#2
Daisha83	2024-01-24T06:38:08.482Z	8786 Armstrong Radial	ShardNumber#3
Marcos_Schmitt58	2024-04-06T17:38:58.551Z	510 S Center Street	ShardNumber#3
Jeremie_VonRueden-Mills	2025-04-06T04:24:23.701Z	157 Koch Drives	ShardNumber#1
Verna28	2025-10-10T18:42:21.059Z	70040 Baylee Trafficway	ShardNumber#1
Trycia.Doyle	2024-01-10T17:00:08.360Z	9511 Tillman Extensions	ShardNumber#2

Sharding-Strategie

Wenn Sie sich für die Verwendung von 10 Shards entscheiden, könnten Ihre Shard-Nummern zwischen 0 und 9 liegen. Wenn Sie eine Aktivität protokollieren, berechnen Sie die Shard-Nummer (z. B. indem Sie eine Hash-Funktion für die Benutzer-ID verwenden und dann den Modulo der Anzahl

der Shards ermitteln) und stellen diese dem GSI-Partitionsschlüssel voran. Bei dieser Methode werden die Einträge auf verschiedene Shards verteilt, wodurch das Risiko von Hot-Partitionen verringert wird.

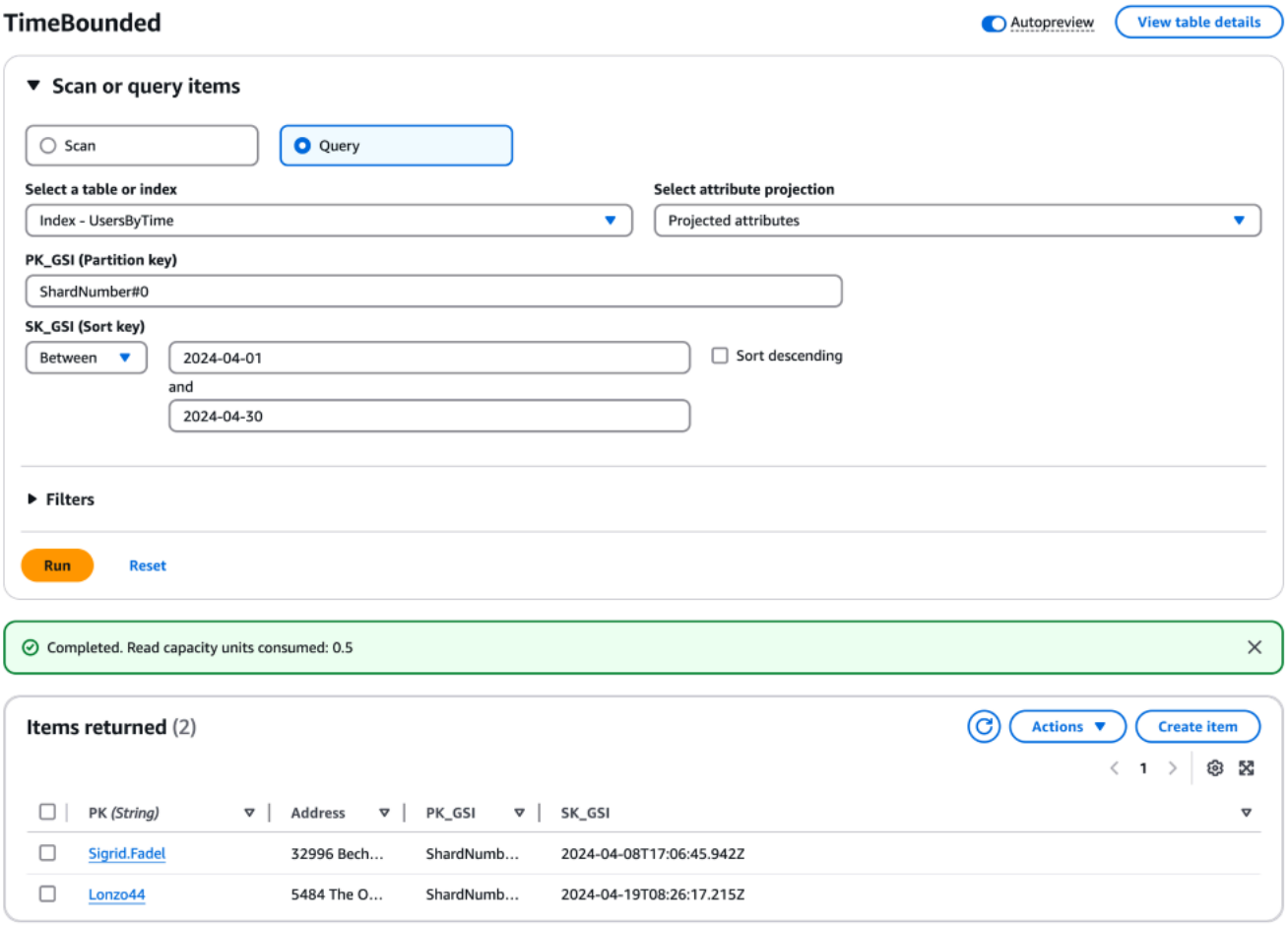
Abfragen der GSI-Shards

Für das Abfragen aller Shards nach Elementen innerhalb eines bestimmten Zeitbereichs in einer DynamoDB-Tabelle, in der Daten auf mehrere Partitionsschlüssel aufgeteilt werden, erfordert einen anderen Ansatz als die Abfrage einer einzelnen Partition. Da DynamoDB-Abfragen jeweils auf einen einzelnen Partitionsschlüssel beschränkt sind, können Sie nicht direkt mehrere Shards mit einem einzigen Abfragevorgang abfragen. Sie können jedoch mithilfe von Logik auf Anwendungsebene das gewünschte Ergebnis erzielen, indem Sie mehrere Abfragen ausführen, von denen jede auf einen bestimmten Shard abzielt, und dann die Ergebnisse aggregieren. Wie Sie dazu vorgehen müssen, ist im unten stehenden Verfahren erläutert.

So fragen Sie Shards ab und aggregieren die Ergebnisse

1. Identifizieren Sie den Bereich der Shard-Nummern, die in Ihrer Sharding-Strategie verwendet werden. Wenn Sie beispielsweise 10 Shards haben, liegen Ihre Shard-Nummern zwischen 0 und 9.
2. Erstellen Sie für jeden Shard eine Abfrage und führen Sie sie aus, um Elemente innerhalb des gewünschten Zeitraums abzurufen. Diese Abfragen können parallel ausgeführt werden, um die Effizienz zu verbessern. Verwenden Sie für diese Abfragen den Partitionsschlüssel mit der Shard-Nummer und den Sortierschlüssel mit Ihrem Zeitraum. Hier ist ein Beispiel für eine Abfrage für einen einzelnen Shard:

```
aws dynamodb query \
    --table-name "YourTableName" \
    --index-name "YourIndexName" \
    --key-condition-expression "PK_GSI = :pk_val AND SK_GSI BETWEEN :start_date
 AND :end_date" \
    --expression-attribute-values '{
        ":pk_val": {"S": "ShardNumber#0"},
        ":start_date": {"S": "2024-04-01"},
        ":end_date": {"S": "2024-04-30"}
    }'
```



Sie würden diese Abfrage für jeden Shard replizieren und den Partitionsschlüssel entsprechend anpassen (z. B. "ShardNumber#1 „," ShardNumber #2 „,..." #9 „). ShardNumber

3. Nachdem alle Abfragen abgeschlossen sind, aggregieren Sie die Ergebnisse jeder Abfrage. Führen Sie diese Aggregation in Ihrem Anwendungscode durch und kombinieren Sie die Ergebnisse zu einem einzigen Datensatz, der die Elemente aus allen Shards innerhalb des angegebenen Zeitraums darstellt.

Überlegungen zur parallelen Abfrageausführung

Jede Abfrage verbraucht Lesekapazität aus Ihrer Tabelle oder Ihrem Index. Wenn Sie den bereitgestellten Durchsatz verwenden, stellen Sie sicher, dass Ihre Tabelle über genügend Kapazität verfügt, um den Anstieg paralleler Abfragen zu verarbeiten. Wenn Sie On-Demand-Kapazität verwenden, sollten Sie die möglichen Auswirkungen auf die Kosten berücksichtigen.

Codebeispiel

Um parallel Abfragen zwischen Shards in DynamoDB mit Python auszuführen, können Sie die boto3-Bibliothek verwenden, d. h. das Amazon Web Services SDK for Python. In diesem Beispiel wird davon ausgegangen, dass Sie boto3 installiert und mit den entsprechenden Anmeldeinformationen konfiguriert haben. AWS

Der folgende Python-Code zeigt, wie parallele Abfragen über mehrere Shards für einen bestimmten Zeitraum durchgeführt werden. Er verwendet gleichzeitige Futures, um Abfragen parallel auszuführen, wodurch die Gesamtausführungszeit im Vergleich zur sequenziellen Ausführung reduziert wird.

```
import boto3
from concurrent.futures import ThreadPoolExecutor, as_completed

# Initialize a DynamoDB client
dynamodb = boto3.client('dynamodb')

# Define your table name and the total number of shards
table_name = 'YourTableName'
total_shards = 10  # Example: 10 shards numbered 0 to 9
time_start = "2030-03-15T09:00:00Z"
time_end = "2030-03-15T10:00:00Z"

def query_shard(shard_number):
    """
    Query items in a specific shard for the given time range.
    """
    response = dynamodb.query(
        TableName=table_name,
        IndexName='YourGSIName',  # Replace with your GSI name
        KeyConditionExpression="PK_GSI = :pk_val AND SK_GSI BETWEEN :date_start
 AND :date_end",
        ExpressionAttributeValues={
            ":pk_val": {"S": f"ShardNumber#{shard_number}"},
            ":date_start": {"S": time_start},
            ":date_end": {"S": time_end},
        }
    )
    return response['Items']

# Use ThreadPoolExecutor to query across shards in parallel
```

```
with ThreadPoolExecutor(max_workers=total_shards) as executor:
    # Submit a future for each shard query
    futures = {executor.submit(query_shard, shard_number): shard_number for
 shard_number in range(total_shards)}

    # Collect and aggregate results from all shards
    all_items = []
    for future in as_completed(futures):
        shard_number = futures[future]
        try:
            shard_items = future.result()
            all_items.extend(shard_items)
            print(f"Shard {shard_number} returned {len(shard_items)} items")
        except Exception as exc:
            print(f"Shard {shard_number} generated an exception: {exc}")

# Process the aggregated results (e.g., sorting, filtering) as needed
# For example, simply printing the count of all retrieved items
print(f"Total items retrieved from all shards: {len(all_items)}")
```

Bevor Sie diesen Code ausführen, stellen Sie sicher, dass Sie `YourTableName` und `YourGSIName` durch die tatsächlichen Tabellen- und GSI-Namen aus Ihrem DynamoDB-Setup ersetzen. Passen Sie außerdem die Variablen `total_shards`, `time_start` und `time_end` an Ihre spezifischen Anforderungen an.

Dieses Skript fragt jeden Shard nach Elementen innerhalb des angegebenen Zeitraums ab und aggregiert die Ergebnisse.

Verwenden globaler sekundärer Indizes zum Erstellen eines letztendlich konsistenten Replikats in DynamoDB

Sie können einen globalen sekundären Index verwenden, um ein schließlich konsistentes Replikat einer Tabelle zu erstellen. Wenn Sie ein Replikat erstellen, können Sie Folgendes tun:

- Legen Sie unterschiedliche bereitgestellte Lesekapazität für verschiedene Leser fest. So könnten Sie z. B. über zwei Anwendungen verfügen: Eine Anwendung verarbeitet Abfragen mit hoher Priorität und benötigt die höchste Leseleistung, während die andere Abfragen mit niedriger Priorität verarbeitet, die eine Drosselung der Leseaktivität tolerieren können.

Wenn beide Anwendungen aus derselben Tabelle lesen, könnte eine hohe Leselast von der Anwendung mit niedriger Priorität die gesamte verfügbare Lesekapazität für die Tabelle verbrauchen. Dies würde die Leseaktivität der Anwendung mit hoher Priorität drosseln.

Stattdessen können Sie ein Replikat über einen globalen sekundären Index erstellen, dessen Lesekapazität Sie getrennt von der Tabelle selbst festlegen können. Sie können dann die App mit niedriger Priorität anstelle der Tabelle das Replikat abfragen lassen.

- Eliminieren Sie Lesevorgänge aus einer Tabelle vollständig. So könnten Sie z. B. über eine Anwendung verfügen, die ein hohes Volumen an Clickstream-Aktivitäten von einer Website erfasst und Sie möchten nicht riskieren, dass Lesevorgänge dies stören. Sie können diese Tabelle isolieren und Lesevorgänge durch andere Anwendungen verhindern (siehe Verwenden von IAM-Richtlinienbedingungen für die differenzierte Zugriffskontrolle), während andere Anwendungen ein Replikat lesen können, das mit einem globalen sekundären Index erstellt wurde.

Um ein Replikat zu erstellen, richten Sie einen globalen sekundären Index ein, der dasselbe Schlüsselschema wie die übergeordnete Tabelle hat, in den einige oder alle Nicht-Schlüsselattribute projiziert werden. In Anwendungen können Sie einige oder alle Leseaktivitäten an diesen globalen sekundären Index statt an die übergeordnete Tabelle richten. Anschließend können Sie die bereitgestellte Lesekapazität des globalen sekundären Indexes anpassen, um diese Lesevorgänge zu verarbeiten, ohne die bereitgestellte Lesekapazität der übergeordneten Tabelle zu ändern.

Es gibt immer eine kurze Verzögerung zwischen dem Schreiben in die übergeordnete Tabelle und dem Anzeigen der geschriebenen Daten im Index zu einer kurzen Verzögerung. Mit anderen Worten, Ihre Anwendungen sollten berücksichtigen, dass das globale sekundäre Indexreplikat mit der übergeordneten Tabelle eventuell konsistent ist.

Sie können mehrere globale sekundäre Indexreplikate erstellen, um verschiedene Lesemuster zu unterstützen. Wenn Sie die Replikate erstellen, projizieren Sie nur die Attribute, die jedes Lesemuster tatsächlich benötigt. Eine Anwendung kann dann weniger bereitgestellte Lesekapazität verbrauchen, um nur die benötigten Daten zu erhalten, anstatt das Element aus der übergeordneten Tabelle lesen zu müssen. Diese Optimierung kann langfristig zu erheblichen Kosteneinsparungen führen.

Bewährte Methoden für das Speichern großer Elemente und Attribute in DynamoDB

Amazon DynamoDB begrenzt die Größe der einzelnen Elemente, die Sie in einer Tabelle speichern können, auf 400 KB (siehe Elementgröße). Wenn Ihre Anwendung mehr Daten in einem Element speichern muss, als die Größeneinschränkung von DynamoDB zulässt, können Sie versuchen, einzelne oder mehrere große Attribute zu komprimieren oder das Element in mehrere Elemente zu teilen (durch Sortierschlüssel effizient indiziert). Sie können das Element auch als Objekt in Amazon Simple Storage Service (Amazon S3) speichern und die Amazon S3-Objekt-ID in Ihrem DynamoDB-Element speichern.

Als bewährte Methode sollten Sie den Parameter `ReturnConsumedCapacity` beim Schreiben von Elementen verwenden, um die Größe von Elementen, die sich der maximalen Elementgröße von 400 KB nähern, zu überwachen und entsprechende Warnmeldungen zu erhalten. Eine Überschreitung der maximalen Elementgröße führt zu fehlgeschlagenen Schreibversuchen. DynamoDB gibt einen ValidationException Fehler zurück. Durch die Überwachung der Elementgröße und entsprechende Warnmeldungen können Sie Probleme mit der Elementgröße beheben, bevor Ihre Anwendung beeinträchtigt wird.

Komprimierung großer Attributwerte

Wenn Sie große Attributwerte komprimieren, liegen sie möglicherweise innerhalb der Elementeinschränkungen in DynamoDB und senken Ihre Speicherkosten. Komprimierungsalgorithmen wie GZIP oder LZO erzeugen binäre Ausgaben, die Sie anschließend in einem `Binary`-Attributtyp innerhalb des Elements speichern können.

Stellen Sie sich als Beispiel eine Tabelle vor, in der Nachrichten von Forenbenutzern gespeichert werden. Solche Nachrichten enthalten oft lange Textfolgen, die für eine Komprimierung in Frage kommen. Durch die Komprimierung können zwar die Elementgrößen reduziert werden, der Nachteil ist jedoch, dass die komprimierten Attributwerte nicht zum Filtern geeignet sind.

Einen Beispielcode, der die Komprimierung solcher Nachrichten in DynamoDB zeigt, finden Sie unter:

- Beispiel: Umgang mit binären Typattributen mithilfe der AWS SDK für Java Dokument-API
- Beispiel: Umgang mit binären Attributen mithilfe der AWS SDK für .NET Low-Level-API

Vertikale Partitionierung

Eine alternative Lösung zum Umgang mit großen Elementen besteht darin, sie in kleinere Datenblöcke aufzuteilen und alle relevanten Elemente anhand des Partitionsschlüsselwerts zuzuordnen. Sie können dann eine Sortierschlüsselzeichenfolge verwenden, um die zugehörigen Informationen zu identifizieren, die neben der Zeichenfolge gespeichert sind. Auf diese Weise und wenn Sie mehrere Elemente nach demselben Partitionsschlüsselwert gruppieren, erstellen Sie eine Elementsammlung.

Weitere Informationen zu diesem Ansatz finden Sie unter:

- Verwenden der vertikalen Partitionierung für eine effiziente Skalierung von Daten in Amazon DynamoDB
- Implementieren Sie vertikale Partitionierung in Amazon DynamoDB mit AWS Glue

Speichern großer Attributwerte in Amazon S3

Wie bereits erwähnt, können Sie mit Amazon S3 auch große Attributwerte speichern, die nicht in ein DynamoDB-Element passen können. Sie können sie als Objekt in Amazon S3 speichern und dann die Objekt-ID in Ihrem DynamoDB-Element speichern.

Sie können auch die Objektmetadatenunterstützung in Amazon S3 verwenden, um eine Verknüpfung zum übergeordneten Element in DynamoDB bereitzustellen. Speichern Sie den Primärschlüsselwert des Elements als Amazon S3-Metadaten des Objekts in Amazon S3. Dies hilft häufig bei der Wartung der Amazon S3-Objekte.

Betrachten Sie beispielsweise die Tabelle `ProductCatalog`. Die Elemente in dieser Tabelle speichern Informationen über Artikelpreise, Beschreibungen, Buchautoren und Abmessungen für andere Produkte. Wenn Sie für die einzelnen Produkte ein Image speichern möchten, das zu groß für die Elemente ist, könnten Sie die Images in Amazon S3 statt in DynamoDB speichern.

Beachten Sie Folgendes, wenn Sie diese Strategie implementieren:

- DynamoDB unterstützt keine Transaktionen, die über Amazon S3 und DynamoDB ausgeführt werden. Daher muss Ihre Anwendung Fehler behandeln, was auch die Bereinigung verwaister Amazon S3-Objekte umfassen könnte.
- Amazon S3 begrenzt die Länge von Objekt-IDs. Daher müssen Sie Ihre Daten so organisieren, dass keine übermäßig langen Objektbezeichner generiert werden oder gegen andere Amazon S3-Einschränkungen verstoßen wird.

Weitere Informationen zur Verwendung von Amazon S3 finden Sie im Benutzerhandbuch für den Amazon Simple Storage Service.

Bewährte Methoden für die Verarbeitung von Zeitreihendaten in DynamoDB

Laut den allgemeinen Designgrundsätzen in Amazon DynamoDB empfiehlt es sich, die Anzahl der verwendeten Tabellen auf ein Minimum zu beschränken. Für die meisten Anwendungen benötigen Sie nur eine einzige Tabelle. Beachten Sie jedoch, dass mit Zeitreihendaten oft am besten umgegangen wird, indem eine Tabelle pro Anwendung pro Zeitraum verwendet wird.

Entwurfsmuster für Zeitreihendaten

Stellen Sie sich ein typisches Zeitserienszenario vor, in dem Sie große Volumen an Ereignissen nachverfolgen möchten. Laut Ihrem Schreibzugriffsmuster haben alle Ereignisse, die aufgezeichnet werden, das Datum des aktuellen Tages. Ihr Lesezugriffsmuster könnte so aussehen, dass die Ereignisse des aktuellen Tages am häufigsten, die Ereignisse des Tags zuvor viel weniger häufig und die noch älteren Ereignisse überhaupt nur sehr wenig gelesen werden. Dies kann z. B. durch Integration des aktuellen Datums und der aktuellen Uhrzeit in den Primärschlüssel verwirklicht werden.

Diese Art von Szenario kann oft mit dem folgenden Entwurfsmuster effektiv bewältigt werden:

- Erstellen Sie eine Tabelle pro Zeitraum, die mit der benötigten Lese- und Schreibkapazität und den erforderlichen Indizes bereitgestellt wird.
- Erstellen Sie vor dem Ende jedes Zeitraums die Tabelle für den nächsten Zeitraum. Sobald der aktuelle Zeitraum endet, leiten Sie den Ereignisdatenverkehr zu der neuen Tabelle um. Sie können diesen Tabellen Namen zuweisen, die angeben, welche Zeiträume aufgezeichnet wurden.
- Sobald nicht mehr in eine Tabelle geschrieben wird, reduzieren Sie die für sie bereitgestellte Schreibkapazität auf einen geringeren Wert (z. B. 1 WCU) und stellen Sie eine angemessene Lesekapazität bereit. Reduzieren Sie die bereitgestellte Lesekapazität mit dem Altern früherer Tabellen. Es empfiehlt sich, die die Tabellen zu archivieren oder zu löschen, deren Inhalt selten oder nie benötigt wird.

Auf diese Weise soll sichergestellt werden, dass die erforderlichen Ressourcen für den aktuellen Zeitraum mit dem höchsten Datenverkehrsvolumen zugeordnet werden und die Bereitstellung für ältere Tabellen, die nicht aktiv verwendet werden, herunterskaliert wird, wodurch Kosten

gesenkt werden. Je nach Ihren geschäftlichen Anforderungen können Sie die Nutzung von Schreib-Sharding in Betracht ziehen, um den Datenverkehr gleichmäßig an den logischen Partitionsschlüssel zu verteilen. Weitere Informationen finden Sie unter Verwenden von Schreib-Sharding zur gleichmäßigen Verteilung von Workloads in Ihrer DynamoDB-Tabelle.

Beispiele für Zeitreihentabellen

Im Folgenden finden Sie ein Beispiel für Zeitreihendaten, in dem die aktuelle Tabelle mit einer höheren read/write Kapazität bereitgestellt wird und die älteren Tabellen herunterskaliert werden, da nur selten auf sie zugegriffen wird.

Current table — Provisioned at: WCU=750 and RCU=300

Primary Key		Attributes		
Partition Key	Sort Key	Radiant Intensity	Wavelength	...
2018-03-15	00:00:00.002	17.372 W/Sr	713 nm	...
2018-03-15	00:00:00.004	17.385 W/Sr	712 nm	...
2018-03-15	00:00:00.005	17.478 W/Sr	708 nm	...
2018-03-15	00:00:00.007	19.172 W/Sr	674 nm	...
• • •	• • •	• • •	• • •	...

Previous table — Provisioned at: WCU=1 and RCU=100

Primary Key		Attributes		
Partition Key	Sort Key	Radiant Intensity	Wavelength	...
2018-03-14	00:00:00.001	16.473 W/Sr	512	...
2018-03-14	00:00:00.003	16.489 W/Sr	519	...
2018-03-14	00:00:00.004	16.814 W/Sr	522	...
2018-03-14	00:00:00.006	16.719 W/Sr	506	...
• • •	• • •	• • •	• • •	...

Older table — Provisioned at: WCU=1 and RCU=1

Primary Key		Attributes		
Partition Key	Sort Key	Radiant Intensity	Wavelength	...
2018-03-10	00:00:00.001	13.669 W/Sr	456	...
2018-03-10	00:00:00.002	13.522 W/Sr	459	...
2018-03-10	00:00:00.004	13.596 W/Sr	457	...
2018-03-10	00:00:00.005	15.721 W/Sr	425	...
• • •	• • •	• • •	• • •	...

Bewährte Methoden für die Verwaltung von many-to-many Beziehungen in DynamoDB-Tabellen

Adjazenzlisten sind ein Entwurfsmuster, das für die Modellierung von many-to-many Beziehungen in Amazon DynamoDB nützlich ist. Allgemein ausgedrückt, stellen sie eine Möglichkeit für die Darstellung von Diagrammdaten (Knoten und Edges) in DynamoDB dar.

Adjazenzlisten-Designmuster

Wenn zwischen verschiedenen Entitäten einer Anwendung eine many-to-many Beziehung besteht, kann die Beziehung als Adjazenzliste modelliert werden. In diesem Muster werden alle Entitäten auf der obersten Ebene (synonym mit den Knoten im Diagrammmodell) mithilfe des Partitionsschlüssels dargestellt. Beziehungen mit anderen Entitäten (Edges in einem Diagramm) werden als Elemente innerhalb der Partition dargestellt, indem der Wert des Sortierschlüssels auf die ID der Zielentität (des Zielknotens) eingestellt wird.

Zu den Vorteilen dieses Musters gehören eine minimale Datenduplizierung und vereinfachte Abfragemuster, um alle Entitäten (Knoten) zu finden, die sich auf eine Zielentität beziehen (d. h., ein Edge zu einem Zielknoten besitzen).

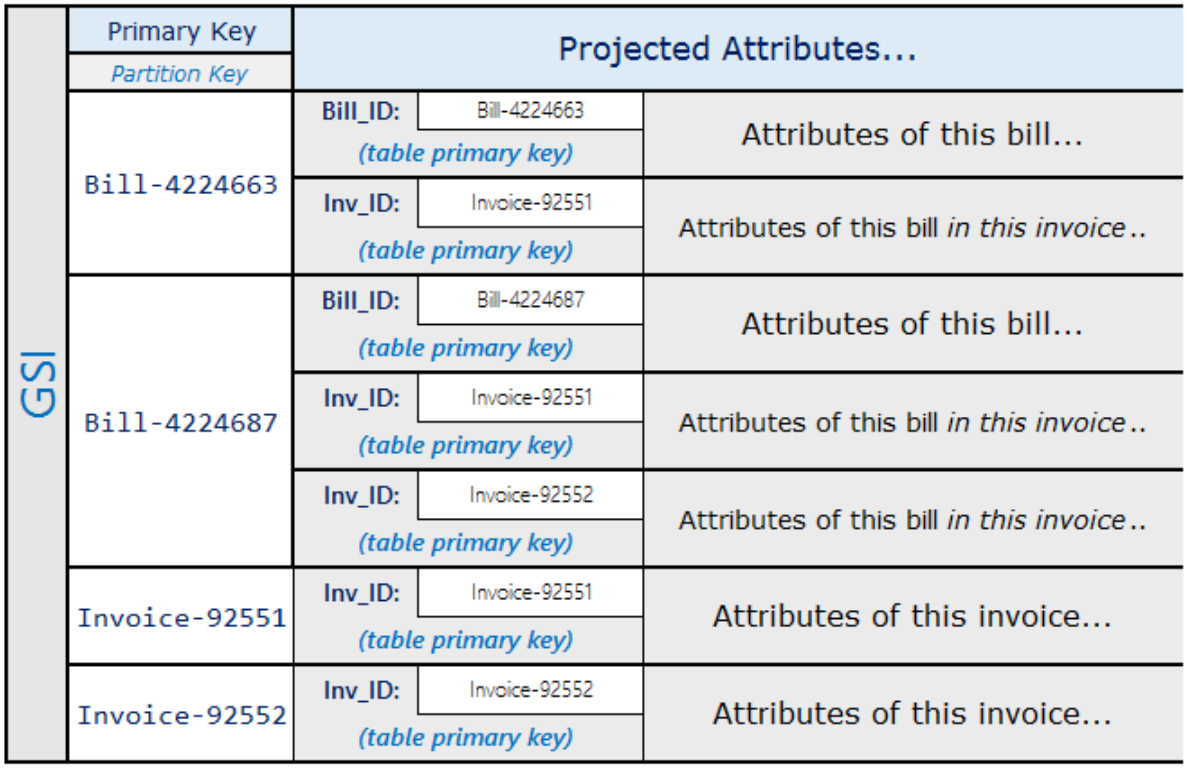
Ein Beispiel aus der Praxis, für das dieses Muster nützlich ist, ist ein Rechnungssystem, bei dem Rechnungen mehrere Abrechnungen umfassen können. Eine Abrechnung kann zu mehreren Rechnungen gehören. Der Partitionsschlüssel in diesem Beispiel ist entweder eine `InvoiceID` oder eine `BillID`. `BillID`-Partitionen besitzen alle für Rechnungen spezifischen Attribute. `InvoiceID`-Partitionen enthalten ein Element zum Speichern rechnungsspezifischer Attribute und ein Element für jede `BillID`, die zu dieser Rechnung führt.

Das Schema sieht wie folgt aus.

Table	Primary Key		Data Attributes...	
	Partition Key	*Sort Key (and GSI PK)*		
	Invoice-92551	**Inv_ID:** Invoice-92551 *(invoice ID)*	**Dated:** 2018-02-07 *(date created)*	More attributes of this invoice...
		Bill_ID: Bill-4224663 *(bill ID)*	**Dated:** 2017-12-03 *(date created)*	Attributes of this bill *in this invoice*..
		Bill_ID: Bill-4224687 *(bill ID)*	**Dated:** 2018-01-09 *(date created)*	Attributes of this bill *in this invoice*..
	Invoice-92552	**Inv_ID:** Invoice-92552 *(invoice ID)*	**Dated:** 2018-03-04 *(date created)*	More attributes of this invoice...
		Bill_ID: Bill-4224687 *(bill ID)*	**Dated:** 2018-01-09 *(date created)*	Attributes of this bill *in this invoice*..
	Bill-4224663	**Bill_ID:** Bill-4224663 *(bill ID)*	**Dated:** 2017-12-03 *(date created)*	More attributes of this bill...
	Bill-4224687	**Bill_ID:** Bill-4224687 *(bill ID)*	**Dated:** 2018-01-09 *(date created)*	More attributes of this bill...

Anhand des vorangehenden Schemas können Sie sehen, dass alle Abrechnungen für eine Rechnung mittels des primären Schlüssels für die Tabelle abgefragt werden können. Um alle Rechnungen nachzuschlagen, die einen Teil einer Abrechnung enthalten, erstellen Sie einen globalen sekundären Index für den Sortierschlüssel der Tabelle.

Die Projektionen für den globalen sekundären Index sehen wie folgt aus.



Materialisiertes Diagrammmuster

Viele Anwendungen werden auf der Basis der Kenntnis von Peer-Rangstufen, der allgemeinen Beziehungen zwischen Entitäten, des Status benachbarter Entitäten und anderer Arten von Diagramm-Workflows erstellt. Für diese Arten von Anwendungen sollten Sie die folgenden Schemadesignmuster berücksichtigen.

TABLE	Primary Key		Attributes		
	PK (NodeId)	SK (TypeTarget, GSI 2 SK)	Data	GSI PK	Graph Projections
	1	DATE|2|BIRTH	1980-12-19	Hash(Person.Data)	
		PERSON|1	Data (GSI1 SK)	GSI PK	
			John Doe	Hash(Person.Data)	
		PERSON|5|FRIEND	Data	GSI PK	
			Jane Smith	Hash(Person.Data)	
		PLACE|4|BIRTH	Data	GSI PK	
			USA|Texas|Austin	Hash(Person.Data)	
		SKILL|6	Data	GSI PK	
			Java Developer|Senior	Hash(Person.Data)	
	2	DATE|2	Data	GSI PK	
			1980-12-19	0	
	3	PLACE|3	Data	GSI PK	
			UK|England|London	0	
	4	PLACE|4	Data	GSI PK	
			USA|Texas|Austin	0	...
	5	DATE|2|BIRTH	Data	GSI PK	
			1980-12-19	Hash(Person.Data)	
		PERSON|5	Data	GSI PK	
			Jane Smith	Hash(Person.Data)	
		PERSON|1|FRIEND	Data	GSI PK	
			John Doe	Hash(Person.Data)	
		PLACE|3|BIRTH	Data	GSI PK	
			UK|England|London	Hash(Person.Data)	
		SKILL|7	Data	GSI PK	
			Guitar|Advanced	Hash(Person.Data)	
	6	SKILL|6	Data	GSI PK	
			Java Developer	0	
	7	SKILL|7	Data	GSI PK	
			Guitar	0	

	Primary Key		Attributes		
	GSI PK	GSI 1 SK (Data)			
GSI 1	O-N	1980-12-19	NodeID	TypeTarget	Graph Projections
			2	DATE|2	...
			NodeID	TypeTarget	
			1	DATE|2|BIRTH	
			NodeID		
			5		
		Guitar	NodeID	TypeTarget	
			7	SKILL|7	
		Guitar|Advanced	NodeID		
			5		
		Jane Smith	NodeID	TypeTarget	
			5	Person|5	
			NodeID	TypeTarget	
			1	Person|5|FRIEND	
		Java Developer	NodeID	TypeTarget	
			6	SKILL|6	
		Java Developer|Senior	NodeID		
			1		
		John Doe	NodeID	TypeTarget	
			1	Person|1	
			NodeID	TypeTarget	
			5	Person|1|FRIEND	
		UK|England|London	NodeID	TypeTarget	
			3	PLACE|3	
			NodeID	TypeTarget	
			1	PLACE|3|BIRTH	
		USA|Texas|Austin	NodeID	TypeTarget	
			4	PLACE|4	
			NodeID	TypeTarget	
			5	PLACE|4|BIRTH	

GSI 2	Primary Key		Attributes		
	GSI PK	GSI 2 SK (TypeTarget)			
	O-N	DATE|2	NodeID	Data	Graph Projections
			2	1980-12-19	...
		DATE|2|BIRTH	NodeID		
			1		
			NodeID		
			5		
		PERSON|1	NodeID	Data	
			1	John Doe	
		PERSON|1|FRIEND	NodeID		
			5		
		PERSON|5	NodeID	Data	
			5	Jane Smith	
		PERSON|5|FRIEND	NodeID		
			1		
		PLACE|3	NodeID	Data	
			3	UK|England|London	
		PLACE|3|BIRTH	NodeID		
			5		
		PLACE|4	NodeID	Data	
			4	USA|texas|Austin	
		PLACE|4|BIRTH	NodeID		
			1		
		SKILL|6	NodeID	Data	
			6	Java Developer	
			NodeID	Data	
			1	Java Developer|Senior	
		SKILL|7	NodeID	Data	
			7	Guitar	
			NodeID	Data	
			5	Guitar|Advanced	

Das vorangehende Schema zeigt eine Diagrammdatenstruktur, die durch einen Satz von Datenpartitionen definiert wird, die die Elemente enthalten, welche die Edges und Knoten des Diagramms definieren. Die Edge-Elemente enthalten ein `Target`- und ein `Type`-Attribut. Diese Attribute werden als Teil eines zusammengesetzten Schlüsselnamens "TypeTarget" verwendet, um das Element in einer Partition in der Primärtabelle oder in einem zweiten globalen Sekundärindex zu identifizieren.

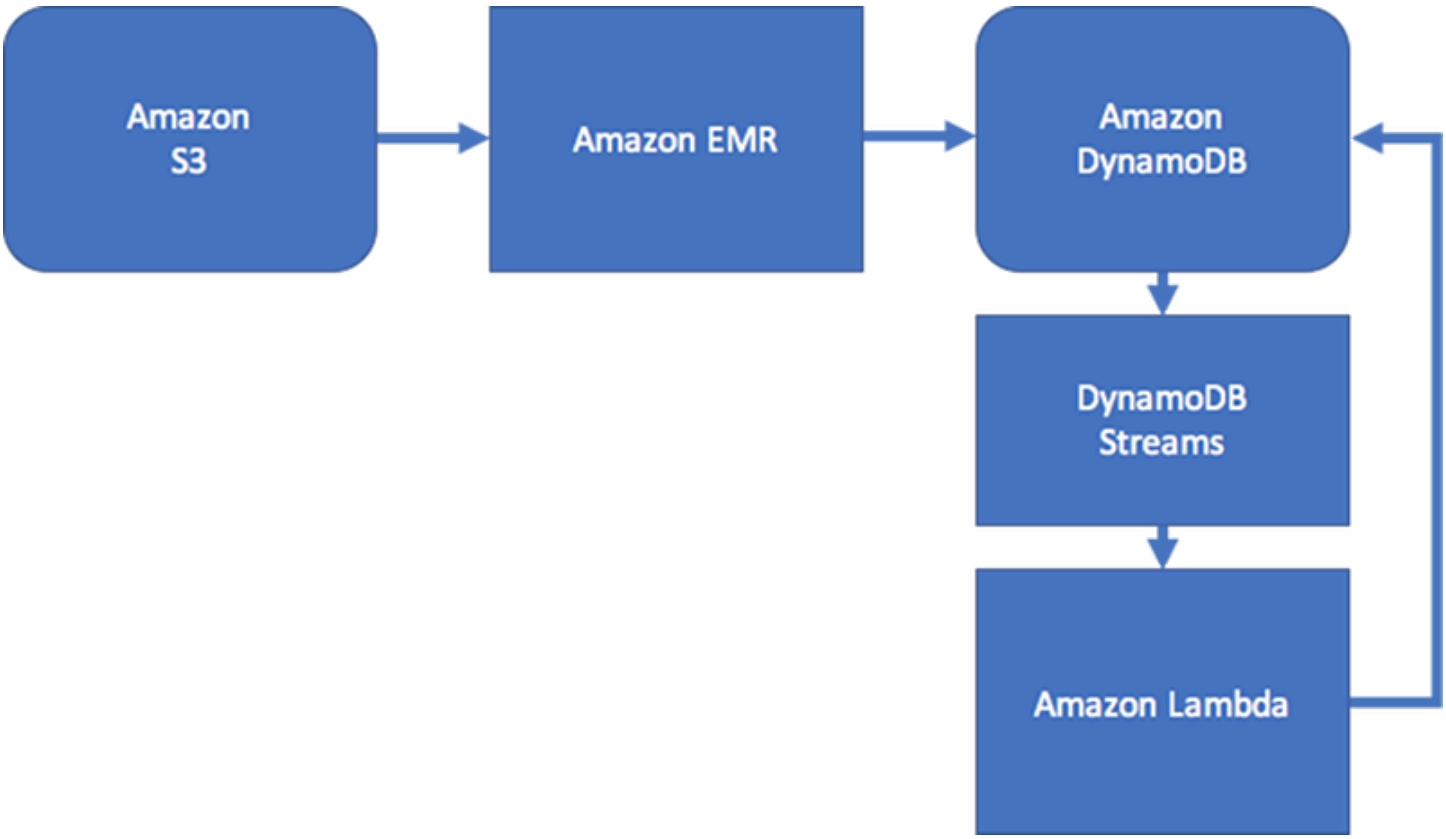
Der erste globale sekundäre Index basiert auf dem Attribut `Data`. Dieses Attribut verwendet Überladungen des globalen sekundären Index wie zuvor beschrieben, um verschiedene Attributtypen zu indizieren, d. h. `Dates`, `Names`, `Places` und `Skills`. Hier indiziert ein einziger globaler sekundärer Index effektiv vier verschiedene Attribute.

Wenn Sie in die Tabelle Elemente einfügen, können Sie eine intelligente Sharding-Strategie verwenden, um Elementsätze mit großen Aggregationen (Geburtsdaten, Qualifikationen) über so viele logische Partitionen in den globalen sekundären Indizes wie nötig zu verteilen, um Probleme mit Hot-Lese-/Schreibvorgängen zu vermeiden.

Das Ergebnis dieser Kombination von Designmustern ist ein robuster Datenspeicher für hoch effiziente Diagramm-Workflows, die in Echtzeit ausgeführt werden. Diese Workflows können hoch leistungsfähige Abfragen zum Status benachbarter Entitäten und zur Edge-Aggregation für Empfehlungsmodule, soziale Netzwerke, Knoteneinstufungen, Unterstrukturaggregationen und andere häufige Anwendungsfälle für Diagramme bereitstellen.

Wenn Ihr Anwendungsfall unempfindlich gegenüber der Konsistenz von Echtzeitdaten ist, können Sie einen geplanten Amazon-EMR-Prozess verwenden, um Edges mit relevanten Übersichtsaggregationen für Diagramme für Ihre Workflows aufzufüllen. Wenn Ihre Anwendung nicht sofort wissen muss, wenn dem Diagramm ein Edge hinzugefügt wird, können Sie einen geplanten Prozess verwenden, um die Ergebnisse zu aggregieren.

Um einen gewissen Grad an Konsistenz zu wahren, kann das Design Amazon DynamoDB Streams und AWS Lambda für die Verarbeitung von Edge-Aktualisierungen umfassen. Sie könnte auch einen Amazon-EMR-Auftrag verwenden, um die Ergebnisse in regelmäßigen Abständen zu validieren. Dieser Ansatz wird im folgenden Diagramm gezeigt. Er wird häufig für soziale Netzwerke verwendet, wenn die Kosten für Echtzeitabfragen hoch sind und die Notwendigkeit, einzelne Benutzerupdates sofort zu erkennen, gering ist.



IT-Servicemanagement (ITSM)- und Sicherheitsanwendungen müssen im Allgemeinen in Echtzeit auf Änderungen des Zustands von Entitäten reagieren, die aus komplexen Edge-Aggregationen bestehen. Diese Anwendungen benötigen ein System, das in Echtzeit Aggregationen mehrerer Knoten aus Beziehungen auf der zweiten und dritten Ebene oder komplexe Edge-Traversals unterstützen. Wenn Ihr Anwendungsfall diese Arten von Workflows für Diagrammabfragen in Echtzeit erfordert, empfehlen wir Ihnen, für die Verwaltung dieser Workflows die Verwendung von Amazon Neptune in Betracht zu ziehen.

> **Note**
>
> Wenn Sie stark vernetzte Datensätze abfragen oder Abfragen ausführen müssen, die mehrere Knoten (auch Multi-Hop-Abfragen genannt) mit Millisekunden-Latenz durchlaufen müssen, sollten Sie die Verwendung von Amazon Neptune in Betracht ziehen. Amazon Neptune ist eine speziell entwickelte, hochleistungsfähige Graphdatenbank-Engine, die für die Speicherung von Milliarden von Beziehungen und die Abfrage des Graphen bzw. des Diagramms mit einer Latenzzeit von Millisekunden optimiert ist.

Bewährte Methoden für das Abfragen und Scannen von Daten in DynamoDB

In diesem Abschnitt werden einige bewährte Methoden für `Query` und `Scan`-Vorgänge in Amazon DynamoDB behandelt.

Leistungsüberlegungen für Scans

Im Allgemeinen sind `Scan`-Vorgänge weniger effizient als andere Operationen in DynamoDB. Ein `Scan`-Vorgang scannt immer die gesamte Tabelle oder den Sekundärindex. Anschließend werden die Werte herausgefiltert, um das gewünschte Ergebnis zu erhalten, wobei im Wesentlichen ein zusätzlicher Schritt zur Entfernung von Daten aus dem Ergebnissatz erfolgt.

Vermeiden Sie, falls möglich, die Verwendung eines `Scan`-Vorgangs für eine große Tabelle oder einen Index mit einem Filter, der viele Ergebnisse entfernt. Außerdem verlangsamt sich der `Scan`-Vorgang, während eine Tabelle oder ein Index wächst. Die `Scan`-Operation untersucht jedes Element auf die angeforderten Werte und kann den bereitgestellten Durchsatz für eine große Tabelle oder Index in einer einzigen Operation verbrauchen. Für schnellere Reaktionszeiten sollten Sie Ihre Tabellen und Indizes so entwerfen, dass Ihre Anwendungen `Query` anstatt `Scan` verwenden

können. (Für Tabellen können Sie auch die Verwendung von `GetItem` und in Betracht ziehen `BatchGetItem` APIs.)

Alternativ können Sie eine Anwendung so entwickeln, dass bei der Nutzung der `Scan`-Operationen die Auswirkung auf Ihre Anforderungsrate minimiert wird. Dies kann auch Modellierungen beinhalten, bei denen es möglicherweise effizienter ist, einen globalen sekundären Index anstelle eines `Scan`-Vorgangs zu verwenden. Weitere Informationen zu diesem Vorgang finden Sie im folgenden Video.

Modellierung von Zugriffsmustern mit niedriger Geschwindigkeit

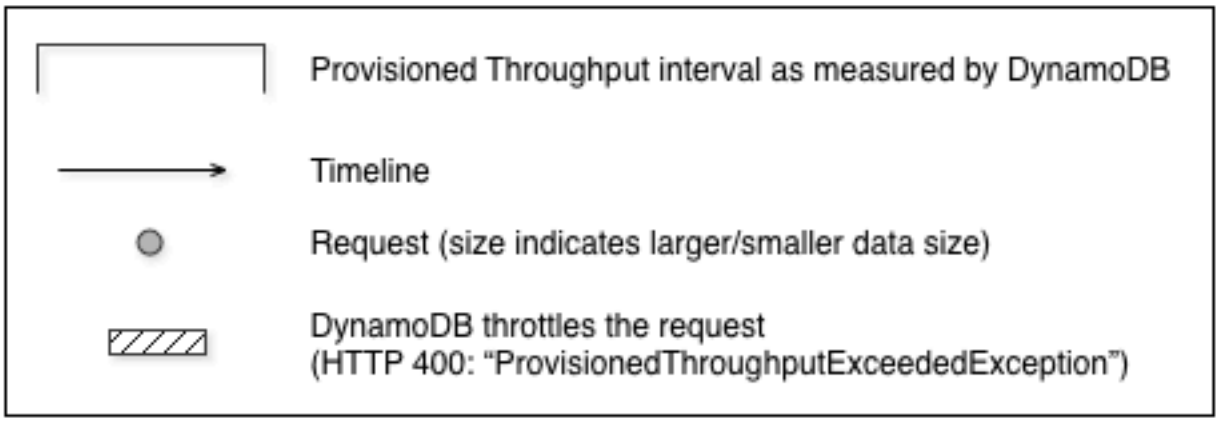
Vermeiden Sie plötzliche Spitzen in der Leseaktivität

Wenn Sie eine Tabelle erstellen, legen Sie ihre Anforderungen der Lese- und Schreibkapazitätseinheiten fest. Die Kapazitätseinheiten werden für Lesevorgänge als die Anzahl der Strongly-Consistent-4-KB-Datenleseanforderungen pro Sekunde ausgedrückt. Für Eventually-Consistent-Lesevorgänge besteht eine Kapazitätseinheit aus zwei 4-KB-Leseanforderungen pro Sekunde. Ein `Scan`-Vorgang führt standardmäßig letztendliche-Lesekonsistenz-Lesevorgänge durch und kann bis zu 1 MB (eine Seite) an Daten zurückgeben. Daher kann eine einzelne -`Scan`Anforderung (1 MB Seitengröße/ 4 KB Elementgröße)/2 (Eventually-Consistent-Lesevorgänge) = 128 Leseoperationen verbrauchen. Wenn Sie stattdessen Strongly-Consistent-Lesevorgänge anfordern würden, würde die `Scan`-Operation doppelt so viel bereitgestellten Durchsatz verbrauchen – 256 Leseoperationen.

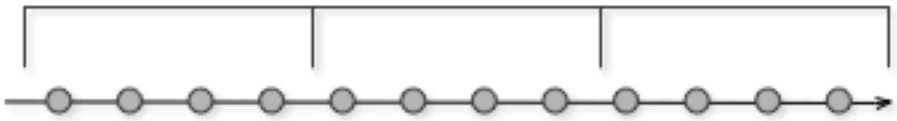
Das stellt eine plötzliche Nutzungsspitze im Vergleich zu der konfigurierten Lesekapazität für die Tabelle dar. Diese Nutzung von Kapazitätseinheiten durch einen Scan verhindert, dass andere potenziell wichtigere Anforderungen für dieselbe Tabelle die verfügbaren Kapazitätseinheiten verwenden. Daher erhalten Sie wahrscheinlich eine `ProvisionedThroughputExceeded`-Ausnahme für diese Anforderungen.

Beachten Sie, dass nicht nur die plötzliche Erhöhung der Kapazitätseinheiten, die `Scan` verwendet, ein Problem darstellt. Der Scan wird wahrscheinlich auch alle seine Kapazitätseinheiten von derselben Partition verbrauchen, da er Leseelemente anfordert, die auf der Partition nebeneinander liegen. Dies bedeutet, dass die Anforderung auf dieselbe Partition trifft, sodass alle Kapazitätseinheiten verbraucht werden und andere Anforderungen an die Partition gedrosselt werden. Wenn die Anforderung zum Lesen von Daten über mehrere Partitionen verteilt ist, dann würde die Operation eine bestimmte Partition nicht drosseln.

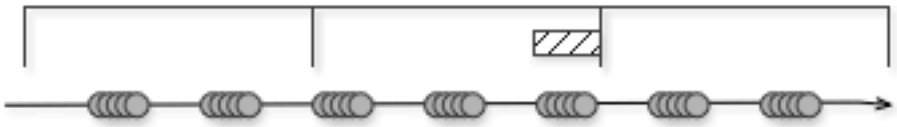
Das folgende Diagramm veranschaulicht die Auswirkung einer plötzlichen Nutzungsspitze der Kapazitätseinheiten durch `Query` und `Scan`-Operationen und deren Auswirkung auf andere Anforderungen für dieselbe Tabelle.



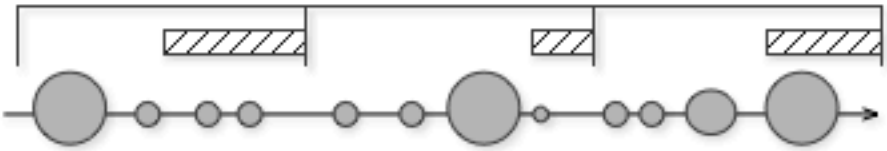
1. Good: Even distribution of requests and size



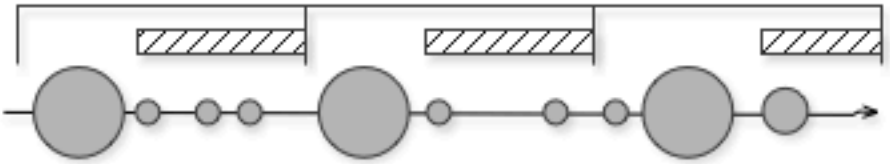
2. Not as Good: Frequent requests in bursts



3. Bad: A few random large requests



4. Bad: Large scan operations



Wie hier dargestellt, kann sich die Auslastungsspitze auf verschiedene Weise auf den bereitgestellten Durchsatz der Tabelle auswirken:

1. Gut: Gleichmäßige Verteilung von Anfragen und Größe
2. Nicht so gut: Häufige Anfragen in Spitzenlasten
3. Schlecht: Ein paar zufällige große Anfragen
4. Schlecht: Große Scan-Operationen

Anstatt eine große `Scan`-Operation zu nutzen, können Sie die folgenden Techniken verwenden, um die Auswirkung auf einen Scan eines bereitgestellten Durchsatzes einer Tabelle zu minimieren.

- Reduzieren der Seitengröße

  Da eine Scan-Operation eine ganze Seite (standardmäßig 1 MB) liest, können Sie die Auswirkung auf die Scan-Operation verringern, indem Sie eine kleinere Seitengröße festlegen. Die `Scan`-Operation bietet den Parameter Limit an, den Sie verwenden können, um die Seitengröße für Ihre Anforderung festzulegen. Jede `Query` oder `Scan`-Anforderung, die über eine kleinere Seitengröße verfügt, verwendet weniger Leseoperationen und erzeugt eine „Pause“ zwischen jeder Anforderung. Angenommen, jedes Element hat eine Größe von 4 KB und Sie legen die Seitengröße auf 40 Elemente fest. Eine `Query`-Anforderung würde dann nur 20 schließlich konsistente Lesevorgänge oder 40 stark konsistente Lesevorgänge verbrauchen. Mit einer größeren Anzahl von kleineren `Query`- oder `Scan`-Operationen können die anderen wichtigen Anforderungen ohne Einschränkung erfolgreich sein.
- Isolieren von Scan-Operationen

  DynamoDB ist für eine einfache Skalierbarkeit ausgelegt. Daher kann eine Anwendung Tabellen für unterschiedliche Zwecke erstellen, möglicherweise sogar Inhalte über mehrere Tabellen duplizieren. Sie möchten Scans in einer Tabelle durchführen, die keinen „unternehmenskritischen“ Datenverkehr annimmt. Einige Anwendungen bearbeiten diese Last, indem sie den Datenverkehr stündlich zwischen zwei Tabellen austauschen – eine für den kritischen Datenverkehr und eine für die Buchhaltung. Andere Anwendungen können dies tun, indem Sie jeden Schreibvorgang in zwei Tabellen durchführen: eine "unternehmenskritische"- und eine "Schatten"-Tabelle.

Konfigurieren Sie Ihre Anwendung so, dass jede Anforderung wiederholt wird, die einen Antwortcode empfängt, der anzeigt, dass Sie Ihren bereitgestellten Durchsatz überschritten haben. Oder erhöhen Sie den bereitgestellten Durchsatz für Ihre Tabelle mithilfe der `UpdateTable`-Operation. Wenn

Ihr Workload temporäre Spitzen aufweist, die dazu führen, dass der Durchsatz gelegentlich die bereitgestellte Stufe überschreitet, wiederholen Sie die Anforderung mit einem exponentiellen Backoff. Weitere Informationen zum Implementieren eines exponentiellen Backoff finden Sie unter Wiederholversuche bei Fehlern und exponentielles Backoff.

Nutzen von parallelen Scans

Viele Anwendungen können von parallelen `Scan`-Operationen eher profitieren als sequenzielle Scans. Zum Beispiel kann eine Anwendung, die eine große Tabelle historischer Daten verarbeitet, einen parallelen Scan viel schneller durchführen als einen sequenziellen Scan. Mehrere Worker-Threads in einem Hintergrund-„Sweeper"-Prozess könnten eine Tabelle mit einer niedrigen Priorität scannen, ohne den Datenverkehr der Produktion zu beeinflussen. In jedem dieser Beispiele wird ein paralleler `Scan` so verwendet, dass die bereitgestellten Durchsatzressourcen für andere Anwendungen nicht behindert werden.

Obwohl parallele Scans nützlich sein können, stellen sie möglicherweise hohe Ansprüche an den bereitgestellten Durchsatz. Bei einem parallelen Scan verfügt Ihre Anwendung über mehrere Worker, die alle gleichzeitig `Scan`-Vorgänge ausführen. Dies kann die bereitgestellte Lesekapazität Ihrer Tabelle schnell verbrauchen. In diesem Fall werden andere Anwendungen, die auf die Tabelle zugreifen müssen, möglicherweise eingeschränkt.

Ein paralleler Scan kann die richtige Wahl sein, wenn die folgenden Bedingungen erfüllt sind:

- Die Tabellengröße ist 20 GB oder größer.
- Der bereitgestellte Lesedurchsatz der Tabelle wird nicht voll ausgeschöpft.
- Sequenzielle `Scan`-Operationen sind zu langsam.

Wählen TotalSegments

Die beste Einstellung für `TotalSegments` hängt ab von Ihren spezifischen Daten, den Einstellungen des bereitgestellten Durchsatzes der Tabelle und Ihren Leistungsanforderungen. Sie müssen wahrscheinlich experimentieren, um alles so hinzubekommen, dass es passt. Wir empfehlen, dass Sie mit einem einfachen Verhältnis beginnen, z. B. ein Segment pro 2 GB Daten. Beispiel: Sie können für eine 30 GB Tabelle `TotalSegments` auf 15 (30 GB/2 GB) festlegen. Ihre Anwendung verwendet dann 15 Workers, wobei jeder Worker ein anderes Segment scannt.

Sie können auch einen Wert für `TotalSegments` auswählen, der auf Client-Ressourcen basiert. Sie können `TotalSegments` auf eine beliebige Zahl von 1 bis 1 000 000 festlegen und

DynamoDB ermöglicht Ihnen diese Anzahl von Segmenten zu scannen. Wenn beispielsweise Ihr Client die Anzahl der Threads begrenzt, die parallel ausgeführt werden können, ist es möglich `TotalSegments` schrittweise zu erhöhen, bis Sie mit Ihrer Anwendung die beste `Scan`-Leistung erzielen.

Überwachen Sie Ihre parallelen Scans, um die Nutzung des bereitgestellten Durchsatzes zu optimieren, während sichergestellt werden muss, dass Ihren anderen Anwendungen die Ressourcen nicht ausgehen. Erhöhen Sie den Wert für `TotalSegments`, wenn Sie nicht den gesamten bereitgestellten Durchsatz verbrauchen, aber immer noch Einschränkungen in Ihren `Scan`-Anforderungen wahrnehmen. Reduzieren Sie den Wert für `TotalSegments`, wenn die `Scan`-Anforderungen mehr bereitgestellten Durchsatz verbrauchen als Sie verwenden möchten.

Bewährte Methoden für das DynamoDB-Tabellendesign

Laut den allgemeinen Designgrundsätzen in Amazon DynamoDB empfiehlt es sich, die Anzahl der verwendeten Tabellen auf ein Minimum zu beschränken. In den meisten Fällen empfehlen wir Ihnen, eine einzelne Tabelle zu verwenden. Wenn es jedoch nicht möglich ist, eine einzelne oder nur wenige Tabellen zu verwenden, können diese Richtlinien hilfreich sein.

- Das Limit pro Konto kann nicht über 10 000 Tabellen pro Konto erhöht werden. Wenn Ihre Anwendung mehr Tabellen erfordert, planen Sie eine Verteilung der Tabellen auf mehrere Konten. Weitere Informationen finden Sie unter Service-, Konto- und Tabellenkontingente in Amazon DynamoDB.
- Berücksichtigen Sie die Limits der Steuerebene für gleichzeitige Operationen auf Steuerebene, die sich auf Ihre Tabellenverwaltung auswirken könnten.
- Arbeiten Sie mit AWS Lösungsarchitekten zusammen, um Ihre Entwurfsmuster für mehrinstanzenfähige Designs zu validieren.

Verwenden von globalen Tabellen in DynamoDB

Globale Tabellen bauen auf der globalen Reichweite von Amazon DynamoDB auf, um Ihnen eine vollständig verwaltete, multiregionale und multiaktive Datenbank zur Verfügung zu stellen, die schnelle lokale Lese- und Schreibleistung für massiv skalierte, globale Anwendungen bietet. Globale Tabellen replizieren Ihre DynamoDB-Tabellen automatisch in die von Ihnen gewünschten Tabellen. AWS-Regionen Es sind keine Änderungen an der Anwendung erforderlich, da globale Tabellen vorhandene DynamoDB APIs verwenden. Für die Verwendung von globalen Tabellen fallen keine

Vorabkosten an und Sie müssen keine Verpflichtungen im Voraus eingehen. Sie zahlen nur für die tatsächlich genutzten Ressourcen.

Dieser Leitfaden erklärt, wie Sie globale DynamoDB-Tabellen effektiv nutzen können. Er enthält die wichtigsten Fakten über globale Tabellen, erklärt die wichtigsten Anwendungsfälle der Funktion, beschreibt die zwei Konsistenzmodi, stellt eine Systematik mit drei verschiedenen Schreibmodellen vor, die Sie in Betracht ziehen sollten, geht die vier wichtigsten Optionen für die Weiterleitung von Anforderungen durch, die Sie implementieren können, erörtert Möglichkeiten zur Evakuierung einer Region, die live ist, oder einer Region, die offline ist, erklärt, wie Sie über die Planung von Durchsatzkapazitäten nachdenken sollten, und enthält eine Checkliste mit Dingen, die Sie bei der Implementierung globaler Tabellen berücksichtigen sollten.

Dieses Handbuch fügt sich in einen größeren Kontext von Bereitstellungen in AWS mehreren Regionen ein, wie im Whitepaper Grundlagen AWS mehrerer Regionen und in den Entwurfsmustern für Datenstabilität mit Video beschrieben. AWS

Themen

- Wichtige Fakten zum Design von globalen DynamoDB-Tabellen
- Wichtige Fakten zu MREC
- Wichtige Fakten zu MRSC
- Anwendungsfälle für globale MREC-Tabellen in DynamoDB
- Schreibmodi bei Verwendung von globalen DynamoDB-Tabellen
- Routing-Strategien in DynamoDB
- Evakuierungsprozesse
- Planung der Durchsatzkapazität für globale DynamoDB-Tabellen
- Checkliste zur Vorbereitung für globale DynamoDB-Tabellen
- Fazit und Ressourcen

Wichtige Fakten zum Design von globalen DynamoDB-Tabellen

- Es gibt zwei Versionen globaler Tabellen: die aktuelle Version Global Tables Version 2019.11.21 (Current) (manchmal als „V2“ bezeichnet) und Version 2017.11.29 (Legacy) der globalen Tabellen (manchmal als „V1“ bezeichnet). In diesem Leitfaden geht es ausschließlich um die aktuelle Version.

- DynamoDB (ohne globale Tabellen) ist ein regionaler Service, was bedeutet, dass er hochgradig verfügbar und intrinsisch widerstandsfähig gegenüber Infrastrukturausfällen, einschließlich des Ausfalls einer gesamten Availability Zone. Eine DynamoDB-Tabelle für eine einzelne Region ist für eine Verfügbarkeit von 99,99 % konzipiert. Weitere Informationen finden Sie unter DynamoDB – Service Level Agreement (SLA).
- Eine globale DynamoDB-Tabelle repliziert ihre Daten zwischen zwei oder mehr Regionen. Eine DynamoDB-Tabelle für mehrere Regionen ist für eine Verfügbarkeit von 99,999 % konzipiert. Bei richtiger Planung kann mithilfe von globalen Tabellen eine Architektur geschaffen werden, die regionalen Ausfällen standhält.
- In DynamoDB gibt es keinen globalen Endpunkt. Alle Anforderungen werden an einen regionalen Endpunkt gestellt, der auf die für diese Region lokale Instance der globalen Tabelle zugreift.
- Aufrufe an DynamoDB sollten nicht regionsübergreifend erfolgen. Als bewährte Methode sollte eine Anwendung, die sich in einer Region befindet, nur auf den lokalen DynamoDB-Endpunkt für diese Region direkt zugreifen. Wenn innerhalb einer Region (auf der DynamoDB-Ebene oder im umgebenden Stack) Probleme festgestellt werden, sollte der Endbenutzer-Datenverkehr an einen anderen Anwendungsendpunkt weitergeleitet werden, der in einer anderen Region gehostet wird. Globale Tabellen stellen sicher, dass Anwendung in jeder Region Zugriff auf dieselben Daten hat.

Konsistenzmodi

Sie können bei der Erstellung einer globalen Tabelle den Konsistenzmodus konfigurieren. Globale Tabellen unterstützen zwei Konsistenzmodi: Multi-Region Eventual Consistency (MREC) und Multi-Region Strong Consistency (MRSC), der im Juni 2025 eingeführt wurde.

Wenn Sie beim Erstellen einer globalen Tabelle keinen Konsistenzmodus angeben, wird für die globale Tabelle standardmäßig MREC verwendet. Replikate, die mit unterschiedlichen Konsistenzmodi konfiguriert wurden, sind in einer globalen Tabelle nicht zulässig. Sie können den Konsistenzmodus einer globalen Tabelle nach der Erstellung nicht mehr ändern.

Wichtige Fakten zu MREC

- Globale Tabellen, die MRSC verwenden, nutzen auch ein Aktiv-Aktiv-Replikationsmodell. Aus Sicht von DynamoDB ist die Tabelle in jeder Region gleichberechtigt, Lese- und Schreibanforderungen anzunehmen. Nach Erhalt einer Schreibanforderung repliziert die lokale Replikattabelle den Schreibvorgang im Hintergrund in andere teilnehmende Remote-Regionen.
- Elemente werden einzeln repliziert. Elemente, die im Rahmen einer einzelnen Transaktion aktualisiert wurden, können möglicherweise nicht zusammen repliziert werden.

- Jede Tabellenpartition in der Quellregion repliziert ihre Schreibvorgänge parallel zu allen anderen Partitionen. Die Reihenfolge der Schreibvorgänge innerhalb einer Remote-Region entspricht möglicherweise nicht der Reihenfolge der Schreibvorgänge in der Quellregion. Weitere Informationen zu Tabellenpartitionen finden Sie im Blogbeitrag Skalierung von DynamoDB: Wie sich Partitionen, Hotkeys und Split for Heat auf die Leistung auswirken.
- Ein neu geschriebenes Element wird in der Regel innerhalb einer Sekunde auf alle Replikattabellen verteilt. Die Verteilung in nahegelegene Regionen erfolgt tendenziell schneller.
- Amazon CloudWatch stellt für jedes Regionspaar eine `ReplicationLatency` Metrik bereit. Zur Berechnung dieser Metrik wird die Ankunftszeit eingehender Elemente mit der Zeit verglichen, zu der sie ursprünglich geschrieben wurden, und ein Durchschnitt ermittelt. Die Zeitangaben werden innerhalb CloudWatch der Quellregion gespeichert. Der Vergleich der durchschnittlichen und maximalen Zeiten kann hilfreich sein, um die durchschnittliche und die stärkste Replikationsverzögerung zu ermitteln. Für diese Latenz gibt es kein SLA.
- Wenn ein einzelnes Element ungefähr zu derselben Zeit (innerhalb dieses `ReplicationLatency`-Fensters) in zwei verschiedenen Regionen aktualisiert wird und der zweite Schreibvorgang vor der Replikation des ersten Schreibvorgangs stattfindet, kann es zu Schreibkonflikten kommen. Globale Tabellen, die MREC verwenden, lösen solche Konflikte nach dem Prinzip „Wer zuletzt schreibt, gewinnt“, basierend auf dem Zeitstempel der Schreibvorgänge. Der erste Vorgang „verliert“ gegen den zweiten Vorgang. Diese Konflikte werden nicht in CloudWatch oder AWS CloudTrail aufgezeichnet.
- Jedes Element verfügt über einen Zeitstempel für den letzten Schreibvorgang, der als private Systemeigenschaft gespeichert wird. Zur Umsetzung des Prinzips „Wer zuletzt schreibt, gewinnt“ wird ein bedingter Schreibvorgang verwendet, der voraussetzt, dass der Zeitstempel des eingehenden Elements größer als der Zeitstempel des vorhandenen Elements ist.
- In einer globalen Tabelle werden alle Elemente in alle teilnehmenden Regionen repliziert. Wenn Sie unterschiedliche Replikationsbereiche haben möchten, können Sie mehrere globale Tabellen erstellen und jeder Tabelle verschiedene teilnehmende Regionen zuweisen.
- Die lokale Region akzeptiert Schreibvorgänge auch dann, wenn die Replikatregion offline ist oder die `ReplicationLatency` wächst. Die lokale Tabelle versucht weiterhin, Elemente in die Remote-Tabelle zu replizieren, bis dies für jedes Element erfolgreich ist.
- Sollte eine Region vollständig offline gehen, was sehr unwahrscheinlich ist, werden alle ausstehenden ausgehenden und eingehenden Replikationen erneut versucht, wenn sie später wieder online ist. Es sind keine besonderen Maßnahmen erforderlich, um die Tabellen wieder zu synchronisieren. Das Prinzip Wer zuletzt schreibt, gewinnt, stellt sicher, dass die Daten letztendlich konsistent werden.

- Sie können einer DynamoDB-MREC-Tabelle jederzeit eine neue Region hinzufügen. DynamoDB kümmert sich um die erste Synchronisierung und die fortlaufende Replikation. Sie können eine Region (selbst die ursprüngliche Region) auch entfernen. Dadurch wird die lokale Tabelle in dieser Region gelöscht.

Wichtige Fakten zu MRSC

- Globale Tabellen, die MRSC verwenden, nutzen auch ein Aktiv-Aktiv-Replikationsmodell. Aus Sicht von DynamoDB ist die Tabelle in jeder Region gleichberechtigt, Lese- und Schreibanforderungen anzunehmen. Elementänderungen im Replikat einer globalen MRSC-Tabelle werden synchron in mindestens eine andere Region repliziert, bevor der Schreibvorgang eine erfolgreiche Antwort zurückgibt.
- Strikt konsistente Lesevorgänge für ein beliebiges MRSC-Replikat geben immer die neueste Version eines Elements zurück. Bei bedingten Schreibvorgängen wird immer der Bedingungsausdruck mit der aktuellen Version des Elements verglichen. Aktualisierungen beziehen sich immer auf die neueste Version eines Elements.
- Letztendlich konsistente Lesevorgänge für ein MRSC-Replikat schließen möglicherweise keine Änderungen ein, die kürzlich in einer anderen Region vorgenommen wurden, und möglicherweise nicht einmal Änderungen, die erst vor ganz Kurzem in derselben Region durchgeführt wurden.
- Ein Schreibvorgang schlägt mir der Ausnahme `ReplicatedWriteConflictException` fehl, wenn versucht wird, ein Element zu ändern, das bereits in einer anderen Region geändert wird. Schreibvorgänge, bei denen die Ausnahme `ReplicatedWriteConflictException` auftritt, können wiederholt werden und sind erfolgreich, wenn das Element in einer anderen Region nicht mehr geändert wird.
- Mit MRSC sind die Latenzen bei Schreibvorgängen und bei strikt konsistenten Lesevorgängen höher. Diese Vorgänge erfordern eine regionsübergreifende Kommunikation. Diese Kommunikation mehr Latenz mit sich bringen, die sich je nach der Round-Trip-Latenz zwischen der Region, auf die zugegriffen wird, und der nächstgelegenen Region, die in der globalen Tabelle aufgeführt ist, erhöht. Weitere Informationen finden Sie in der AWS re:Invent 2024-Präsentation Multi-Region strong consistency with DynamoDB global tables. Bei letztendlich konsistenten Lesevorgängen gibt es keine zusätzliche Latenz. Sie können diese Latenzen in Ihren Regionen mit einem Open-Source-Testtool berechnen.
- Elemente werden einzeln repliziert. Globale Tabellen, die MRSC verwenden, unterstützen die Transaktion nicht. APIs

- Eine globale MRSC-Tabelle muss in genau drei Regionen bereitgestellt werden. Sie können eine globale MRSC-Tabelle mit drei Replikaten oder zwei Replikaten und einem Witness konfigurieren. Ein Witness ist eine Komponente einer globalen MRSC-Tabelle, die aktuelle Daten enthält, die in Replikate der globalen Tabelle geschrieben wurden. Ein Witness bietet eine optionale Alternative zu einem vollständigen Replikat und unterstützt gleichzeitig die Verfügbarkeitsarchitektur von MRSC. Sie können für einen Witness keine Lese- oder Schreibvorgänge ausführen. Für einen Witness fallen keine Speicher- oder Schreibkosten an. Ein Witness befindet sich in einer anderen Region als die beiden Replikate.
- Sie erstellen eine globale MRSC-Tabelle, indem Sie einer vorhandenen DynamoDB-Tabelle, die keine Daten enthält, ein Replikat und einen Witness oder zwei Replikate hinzufügen. Sie können einer vorhandenen globalen MRSC-Tabelle keine zusätzlichen Replikate hinzufügen. Ein einzelnes Replikat oder ein Witness kann nicht aus einer globalen MRSC-Tabelle gelöscht werden. Sie können zwei Replikate oder ein Replikat und einen Witness aus einer globalen MRSC-Tabelle löschen. Im zweiten Szenario wird das verbleibende Replikat in eine DynamoDB-Tabelle mit einer einzelnen Region konvertiert.
- Sie können anhand der API-Ausgabe ermitteln, ob für eine globale MRSC-Tabelle ein Zeuge konfiguriert ist und in welcher Region er konfiguriert ist. DescribeTable Der Zeuge gehört DynamoDB und wird von diesem verwaltet. Er erscheint nicht in Ihrer Region, AWS-Konto in der er konfiguriert ist.
- Die globalen MRSC-Tabellen sind in den folgenden Region-Sets verfügbar:
  - Region-Set USA: USA Ost (Nord-Virginia), USA Ost (Ohio), USA West (Oregon)
  - Region-Set Europa: Europa (Frankfurt), Europa (Irland), Europa (London), Europa (Paris)
  - Region-Set Asien-Pazifik: Asien-Pazifik (Tokio), Asien-Pazifik (Seoul) und Asien-Pazifik (Osaka).
- Globale MRSC-Tabellen können sich nicht über Region-Sets erstrecken. Beispielsweise kann eine globale MRSC-Tabelle keine Replikate aus Region-Sets der USA und der EU enthalten.
- Die Funktion „Time to Live“ (TTL) wird von globalen MRSC-Tabellen nicht unterstützt.
- Lokale sekundäre Indizes (LSIs) werden für globale MRSC-Tabellen nicht unterstützt.
- CloudWatch Contributor Insights-Informationen werden nur für die Region gemeldet, in der ein Vorgang stattgefunden hat.
- Die lokale Region akzeptiert alle Lese- und Schreibvorgänge, solange es eine zweite Region gibt, in der ein Replikat oder ein Witness gehostet wird, um das Quorum einzurichten. Wenn keine zweite Region verfügbar ist, kann die lokale Region nur letztendlich konsistente Lesevorgänge verarbeiten.

- Im unwahrscheinlichen Fall, dass eine Region vollständig offline geht, holt sie automatisch auf, wenn sie später wieder online ist. Solange der Vorgang nicht abgeschlossen ist, geben Schreibvorgänge und stark konsistente Lesevorgänge nur in der Region, in der der Vorgang stattfindet, Fehler zurück, während Anfragen an andere Regionen weiterhin normal ausgeführt werden. Schließlich werden bei konsistenten Lesevorgängen in die aufholende Region die Daten zurückgegeben, die bisher in die Region übertragen wurden, und zwar mit dem üblichen lokalen Konsistenzverhalten zwischen dem Leader-Knoten und den lokalen Replikaten. Es sind keine besonderen Maßnahmen erforderlich, um die Tabellen wieder zu synchronisieren.

Anwendungsfälle für globale MREC-Tabellen in DynamoDB

Globale MREC-Tabellen bieten folgende Vorteile:

- Lesevorgänge mit niedrigerer Latenz. Platzieren Sie eine Kopie der Daten näher am Endbenutzer, um die Netzwerklatenz bei Lesevorgängen zu reduzieren. Die Daten werden so aktuell gehalten wie der `ReplicationLatency`-Wert.
- Schreibvorgänge mit niedrigerer Latenz. Sie können in eine nahegelegene Region schreiben, um die Netzwerklatenz und die für den Schreibvorgang benötigte Zeit zu reduzieren. Der Schreibverkehr muss sorgfältig weitergeleitet werden, um Konflikte zu vermeiden. Die Methoden für die Weiterleitung werden unter Routing-Strategien in DynamoDB ausführlicher erörtert.
- Nahtlose Migration von Regionen. Sie können eine neue Region hinzufügen und die alte Region löschen, um eine Bereitstellung ohne Ausfallzeit auf der Datenebene von einer Region zu einer anderen zu migrieren.

Die globalen MREC- und MRSC-Tabellen bieten beide den folgenden Vorteil:

- Verbesserte Resilienz und Notfallwiederherstellung. Wenn es in einer Region zu Leistungseinbußen oder einem vollständigen Ausfall kommt, können Sie diese evakuieren. Evakuieren bedeutet, einige oder alle Anforderungen, die in dieser Region eingehen, zu verschieben. Durch die Verwendung von globalen Tabellen erhöht sich das DynamoDB-SLA für den monatlichen Prozentsatz der Betriebszeit von 99,99 % auf 99,999 %. Die Verwendung von MREC unterstützt einen Recovery Point Objective (RPO) und einen Recovery Time Objective (RTO), die in Sekunden gemessen werden. Bei Verwendung von MRSC wird ein RPO von Null unterstützt.

  Zum Beispiel präsentierte Fidelity Investments auf der re:Invent 2022, wie das Unternehmen globale DynamoDB-Tabellen für sein Auftragsverwaltungssystem verwendet. Ziel war es, eine

zuverlässige Verarbeitung mit geringer Latenz in einem Umfang zu erzielen, der mit einer On-Premises-Verarbeitung nicht möglich wäre, dabei aber gleichzeitig die Resilienz gegenüber Ausfällen der Availability Zone und Regionen aufrechtzuerhalten.

Wenn Ihr Ziel Resilienz und Notfallwiederherstellung ist, weisen MRSC-Tabellen zwar höhere Latenzen bei Schreibvorgängen und strikt konsistenten Lesevorgängen auf, unterstützen aber einen RPO von Null. Globale MREC-Tabellen können einen RPO unterstützen, der der verzögerten Replikation zwischen Replikaten entspricht, die je nach Region in der Regel einige Sekunden beträgt.

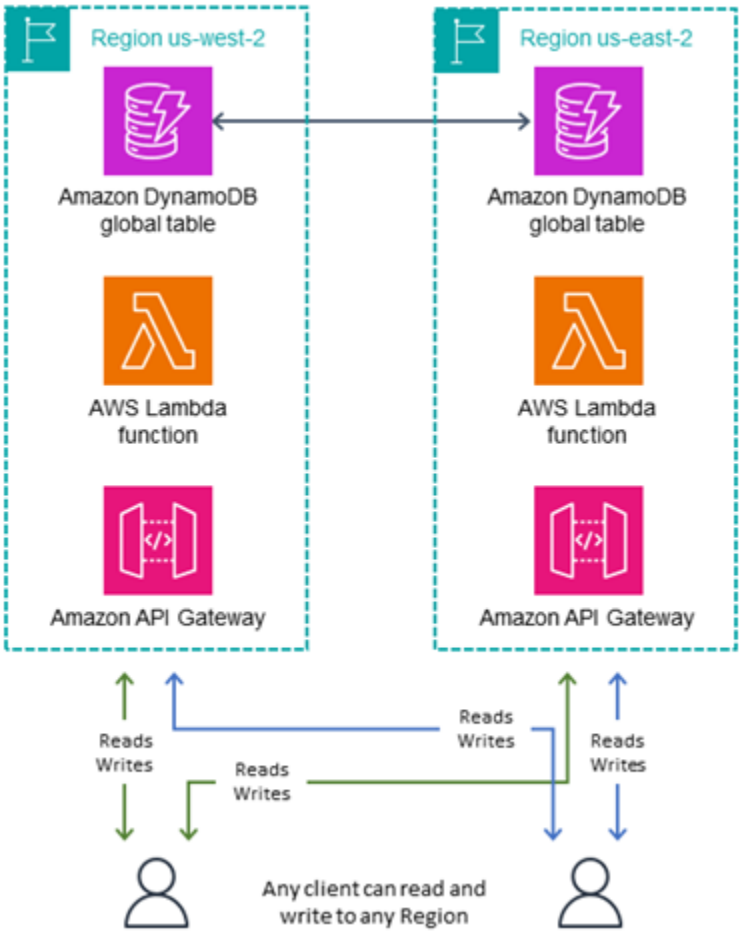
Schreibmodi bei Verwendung von globalen DynamoDB-Tabellen

Globale Tabellen sind auf Tabellenebene immer aktiv-aktiv. Vielleicht möchten Sie sie insbesondere bei MREC-Tabellen jedoch als aktiv-passiv behandeln, indem Sie steuern, wie Schreibanforderungen weitergeleitet werden. Sie könnten sich beispielsweise dafür entscheiden, Schreibanforderungen an eine einzelne Region weiterzuleiten, um mögliche Schreibkonflikte zu vermeiden, die bei MREC-Tabellen vorkommen können.

Es gibt drei Hauptmuster für verwaltete Schreibvorgänge, die in den nächsten drei Abschnitten erläutert werden. Sie sollten überlegen, welches Schreibmuster zu Ihrem Anwendungsfall passt. Diese Wahl wirkt sich auf die Weiterleitung von Anforderungen, die Evakuierung einer Region und die Notfallwiederherstellung aus. Die Anleitung in späteren Abschnitten hängt vom Schreibmodus Ihrer Anwendung ab.

Modus „Schreiben in eine beliebige Region“ (keine Primärregion)

Der Modus Schreiben in eine beliebige Region, der im folgenden Diagramm dargestellt wird, ist vollständig aktiv-aktiv. In diesem Modus bestehen keine Einschränkungen in Bezug darauf, wo ein Schreibvorgang stattfinden kann. Jede Region kann jederzeit Schreibvorgänge akzeptieren. Dies ist der einfachste Modus, der jedoch nur für einige Arten von Anwendungen verwendet werden kann. Dieser Modus ist für alle MRSC-Tabellen geeignet. Er eignet sich auch für MREC-Tabellen, wenn alle Writer idempotent und daher sicher wiederholbar sind, sodass gleichzeitige oder wiederholte Schreibvorgänge in verschiedenen Regionen nicht miteinander in Konflikt geraten. Er kann beispielsweise verwendet werden, wenn ein Benutzer seine Kontaktdaten aktualisiert. Dieser Modus eignet sich auch gut für einen Sonderfall der Idempotenz, einen Append-Only-Datensatz, bei dem alle Schreibvorgänge eindeutige Einfügungen unter einem deterministischen Primärschlüssel sind. Schließlich ist dieser Modus für MREC geeignet, wenn das Risiko von Schreibkonflikten akzeptabel wäre.



Der Modus Schreiben in eine beliebige Region ist die am einfachsten zu implementierende Architektur. Die Weiterleitung ist einfacher, da jede Region jederzeit Schreibziel sein kann. Ein Failover ist einfacher, da bei MRSC-Tabellen die Elemente immer synchronisiert werden. Außerdem können bei MRSC-Tabellen alle kürzlichen Schreibvorgänge beliebig oft in jeder sekundären Region wiederholt werden. Wenn möglich, sollten Sie diesen Schreibmodus vorsehen.

Mehrere Video-Streaming-Services verwenden beispielsweise häufig globale Tabellen, um Lesezeichen, Bewertungen, Ansehstatus-Flags usw. zu verfolgen. Diese Bereitstellungen nutzen MREC-Tabellen, da sie Replikate benötigen, die auf der ganzen Welt verstreut sind, wobei jedes Replikat Lese- und Schreibvorgänge mit niedriger Latenz ermöglicht. Für diese Bereitstellungen kann der Modus Schreiben in eine beliebige Region verwendet werden, solange sichergestellt wird, dass jeder Schreibvorgang idempotent ist. Dies ist der Fall, wenn bei jeder Aktualisierung – beispielsweise beim Festlegen eines neuen aktuellen Zeitcodes, beim Zuweisen einer neuen Überprüfung oder beim Einstellen eines neuen Ansehstatus – der neue Status des Benutzers direkt zugewiesen wird und der nächste korrekte Wert für ein Element nicht von seinem aktuellen Wert abhängt. Wenn die Schreibanforderungen des Benutzers zufälligerweise an verschiedene Regionen weitergeleitet

werden, bleibt der letzte Schreibvorgang bestehen und der globale Status wird gemäß der letzten Zuweisung festgelegt. Lesevorgänge in diesem Modus werden letztendlich konsistent, verzögert um den letzten `ReplicationLatency`-Wert.

In einem anderen Beispiel verwendet ein Finanzdienstleistungsunternehmen globale Tabellen als Teil eines Systems, um eine laufende Aufstellung der Debitkartenkäufe für jeden Kunden zu führen und die Cashback-Prämien des Kunden zu berechnen. Das Unternehmen möchte ein `RunningBalance`-Element pro Kunde beibehalten. Dieser Schreibmodus ist naturgemäß nicht idempotent, da beim Eingehen von Transaktionen der Saldo mit einem `ADD`-Ausdruck geändert wird, bei dem der neue korrekte Wert vom aktuellen Wert abhängt. Durch die Verwendung von MRSC-Tabellen kann das Unternehmen in eine beliebige Region schreiben, da jeder `ADD`-Aufruf immer mit dem allerneuesten Wert des Elements ausgeführt wird.

Ein drittes Beispiel betrifft ein Unternehmen, das Online-Werbeplatzierungsdienste anbietet. Dieses Unternehmen hat entschieden, dass ein geringes Risiko eines Datenverlusts akzeptabel wäre, um die Designvereinfachungen des Modus Schreiben in eine beliebige Region zu erreichen. Wenn Anzeigen geschaltet werden, bleiben nur wenige Millisekunden Zeit, um genügend Metadaten abzurufen, zu bestimmen, welche Anzeige geschaltet werden soll, und dann die Ad Impression aufzuzeichnen, damit dem betreffenden Benutzer nicht mehrmals dieselbe Anzeige präsentiert wird. Mit globalen Tabellen kann das Unternehmen sowohl Lesevorgänge mit niedriger Latenz für Endbenutzer auf der ganzen Welt als auch Schreibvorgänge mit niedriger Latenz erzielen. Alle Ad Impressions für einen Benutzer können in einem einzelnen Element aufgezeichnet werden, das als wachsende Liste dargestellt wird. Das Unternehmen verwendet ein Element, anstatt Elemente an eine Elementauflistung anzuhängen, da auf diese Weise ältere Ad Impressions im Rahmen jedes Schreibvorgangs entfernt werden können, ohne Kosten für einen Löschvorgang zu verursachen. Dieser Schreibvorgang ist nicht idempotent. Wenn derselbe Endbenutzer Anzeigen aus mehreren Regionen ungefähr zur gleichen Zeit sieht, besteht die Möglichkeit, dass ein Schreibvorgang für eine Ad Impression den anderen überschreibt. Das Risiko besteht darin, dass ein Benutzer eine Anzeige gelegentlich wiederholt sieht. Das Unternehmen hat entschieden, dass dies akzeptabel ist.

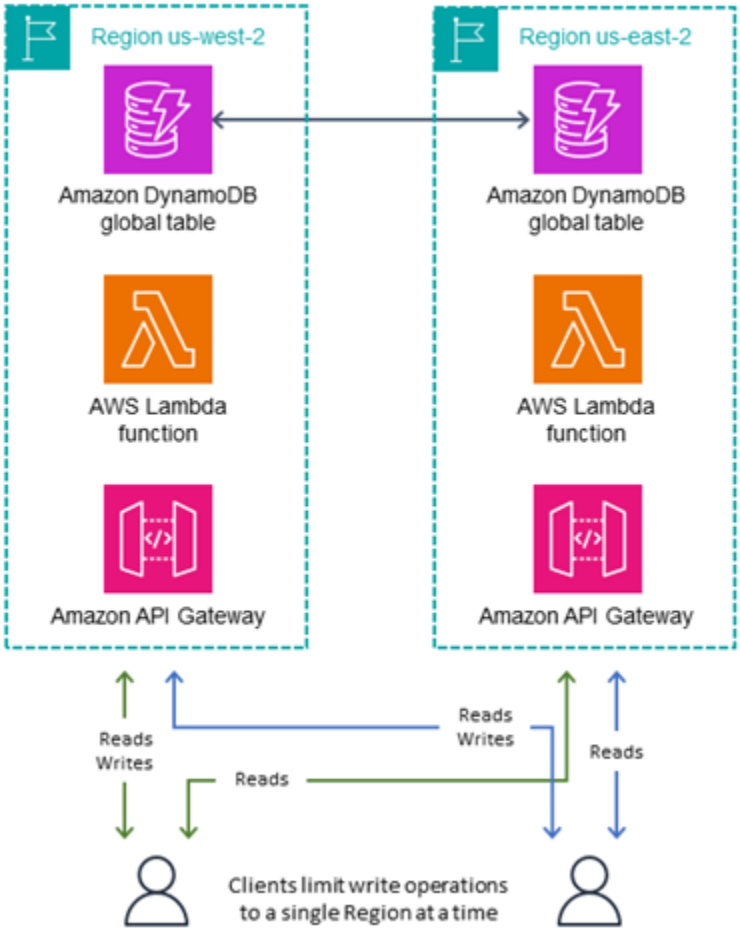
Schreiben in eine Region (einzelne Primärregion)

Der Modus Schreiben in eine Region, der im folgenden Diagramm dargestellt wird, ist aktiv-passiv. Alle Schreibvorgänge in der Tabelle werden an eine einzelne aktive Region weitergeleitet. Beachten Sie, dass DynamoDB das Konzept einer einzelnen aktiven Region nicht kennt. Die Anwendungsweiterleitung außerhalb von DynamoDB verwaltet dies. Der Modus Schreiben in eine Region ist gut für MREC-Tabellen geeignet, bei denen Schreibkonflikte vermieden werden müssen, indem sichergestellt wird, dass Schreibvorgänge jeweils nur in eine Region gelangen.

Dieser Schreibmodus ist hilfreich, wenn Sie bedingte Ausdrücke verwenden möchten und MRSC aus irgendeinem Grund nicht verwenden können, oder wenn Sie Transaktionen ausführen müssen. Diese Ausdrücke sind nur möglich, wenn Sie wissen, dass Sie mit den neuesten Daten arbeiten. Deshalb müssen alle Schreibanforderungen an eine einzelne Region gesendet werden, die über die neuesten Daten verfügt.

Wenn Sie eine MRSC-Tabelle verwenden, sollten Sie der Einfachheit halber vielleicht entscheiden, generell in eine Region schreiben. Das kann beispielsweise dazu beitragen, den Ausbau Ihrer Infrastruktur über DynamoDB hinaus zu minimieren. Der Schreibmodus wäre immer noch das Schreiben in eine beliebige Region, da Sie mit MRSC jederzeit sicher in jede Region schreiben könnten, ohne sich Gedanken über die Konfliktlösung machen zu müssen, die dazu führen würde, dass MREC-Tabellen in eine Region schreiben.

Letztendlich konsistente Lesevorgänge können an alle Replikatregionen geleitet werden, um geringere Latenzen zu erreichen. Strikt konsistente Lesevorgänge müssen an die einzelne Primärregion geleitet werden.



Manchmal ist es infolge eines Regionsausfalls notwendig, die aktive Region zu ändern. Manche Benutzer ändern die derzeit aktive Region nach einem regelmäßigen Zeitplan, z. B. bei der Implementierung einer follow-the-sun Bereitstellung. Dadurch wird die aktive Region in der Nähe der Region mit der höchsten Aktivität platziert (normalerweise dort, wo es Tag ist, daher der Name), was zu Lese- und Schreibvorgängen mit der niedrigsten Latenz führt. Ein Nebeneffekt dabei ist, dass der Code zum Ändern der Region täglich aufgerufen wird und damit vor einer Notfallwiederherstellung gründlich getestet wurde.

Die passive(n) Region(en) kann/können eine verkleinerte Infrastruktur rund um DynamoDB beibehalten, die nur dann aufgebaut wird, wenn sie zur aktiven Region wird. In diesem Handbuch werden die Designs Pilot Light und Warm Standby nicht behandelt. Weitere Informationen finden Sie unter Disaster Recovery (DR) -Architektur unter AWS, Teil III: Pilot Light und Warm Standby.

Der Modus Schreiben in eine Region ist gut geeignet, wenn Sie globale Tabellen für global verteilte Lesevorgänge mit niedriger Latenz verwenden. Ein Beispiel ist ein großes Social-Media-Unternehmen, das in allen Regionen der Welt dieselben Referenzdaten zur Verfügung haben muss. Das Unternehmen aktualisiert die Daten nicht oft, aber wenn eine Aktualisierung durchgeführt wird, werden die Daten nur in eine Region geschrieben, um mögliche Schreibkonflikte zu vermeiden. Lesevorgänge sind immer von jeder Region aus zulässig.

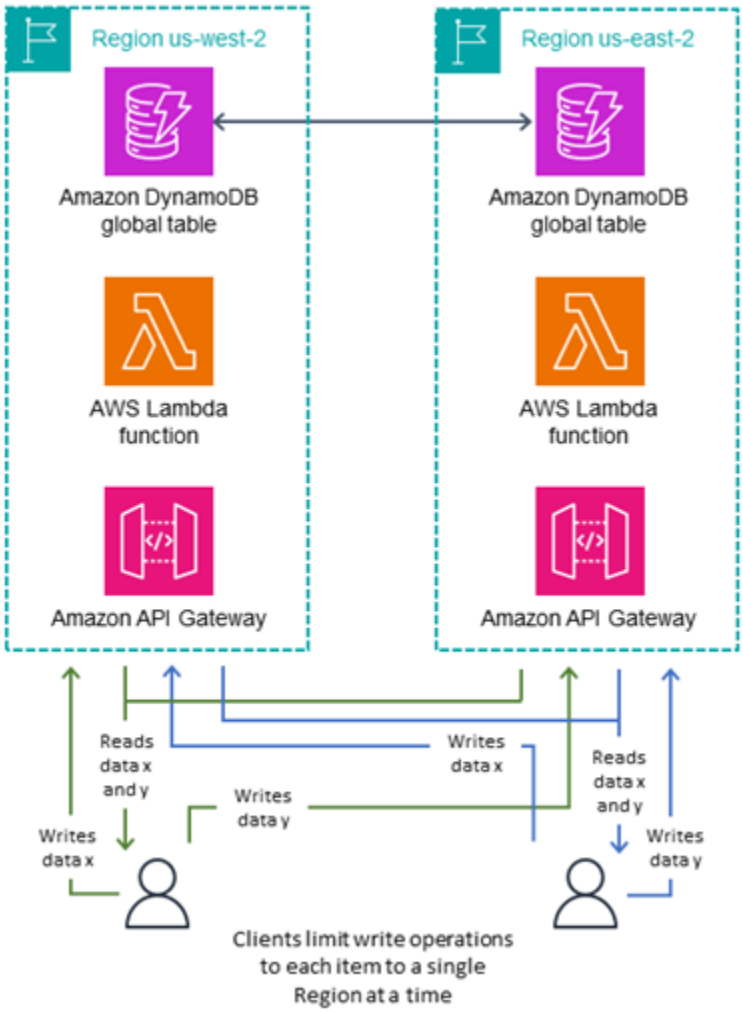
Ein weiteres Beispiel ist das schon erwähnte Finanzdienstleistungsunternehmen, das die tägliche Cashback-Berechnung eingeführt hat. Das Unternehmen nutzte den Modus Schreiben in eine beliebige Region, um den Saldo zu berechnen, aber den Modus Schreiben in eine Region, um Zahlungen zu verfolgen. Diese Arbeit erfordert Transaktionen, die in MRSC-Tabellen nicht unterstützt werden. Daher funktioniert dieser Prozess besser mit einer separaten MREC-Tabelle und dem Modus Schreiben in eine Region.

Schreiben in Ihre Region (gemischte Primärregion)

Der Schreibmodus Schreiben in Ihre Region, der im folgenden Diagramm dargestellt ist, kann mit MREC-Tabellen verwendet werden. Er weist verschiedenen Heimatregionen unterschiedliche Datenteilmengen zu und ermöglicht Schreibvorgänge für ein Element nur über dessen Heimatregion. Dieser Modus ist aktiv-passiv, weist jedoch die aktive Region basierend auf dem Element zu. Jede Region ist Primärregion für ihren eigenen, sich nicht überlappenden Datensatz und Schreibvorgänge müssen überwacht werden, um eine korrekte Lokalisierung sicherzustellen.

Dieser Modus ist ähnlich wie der Modus Schreiben in eine Region, mit der Ausnahme, dass Schreibvorgänge mit geringerer Latenz möglich sind, da die jedem Endbenutzer zugeordneten Daten in größerer Netzwerknähe zu diesem Benutzer platziert werden können. Auch ist die umliegende

Infrastruktur gleichmäßiger auf die Regionen verteilt und der Aufbau der Infrastruktur während eines Failover-Szenarios ist weniger aufwendig, da in allen Regionen ein Teil der Infrastruktur bereits aktiv ist.



Sie können die Heimatregion für Elemente auf verschiedene Arten bestimmen:

- Intrinsisch: Ein bestimmter Aspekt der Daten, z. B. ein spezielles Attribut oder ein Wert, der in den Partitionsschlüssel eingebettet ist, stellt die Heimatregion klar. Diese Technik wird im Blogbeitrag Verwenden von Region Pinning, um eine Heimatregion für Elemente in einer globalen Amazon-DynamoDB-Tabelle festzulegen beschrieben.
- Ausgehandelt: Die Heimatregion jedes Datensatzes wird extern ausgehandelt, z. B. mit einem separaten globalen Dienst, der die Zuweisungen verwaltet. Die Zuweisung kann begrenzt gültig sein. Danach muss sie neu verhandelt werden.
- Tabellenorientiert: Anstelle einer einzelnen replizierenden globalen Tabelle erstellen Sie genauso viele globale Tabellen wie replizierende Regionen. Der Name jeder Tabelle gibt ihre Heimatregion

an. Bei Standardoperationen werden alle Daten in die Heimatregion geschrieben, während andere Regionen eine schreibgeschützte Kopie behalten. Während eines Failovers übernimmt eine andere Region vorübergehend Schreibaufgaben für diese Tabelle.

Stellen Sie sich beispielsweise vor, Sie arbeiten für ein Gaming-Unternehmen. Sie benötigen eine niedrige Latenz bei Lese- und Schreibvorgängen für alle Spieler weltweit. Sie weisen jeden Spieler der ihm am nächsten gelegenen Region zu. In dieser Region werden alle Lese- und Schreibvorgänge ausgeführt, sodass eine hohe read-after-write Konsistenz gewährleistet ist. Wenn ein Spieler jedoch verreist oder es in seiner Heimatregion zu einem Ausfall kommt, steht eine vollständige Kopie seiner Daten in anderen Regionen zur Verfügung und der Spieler kann einer anderen Heimatregion zugewiesen werden.

Stellen Sie sich als weiteres Beispiel vor, Sie arbeiten in einem Videokonferenzunternehmen. Die Metadaten jeder Konferenz werden einer bestimmten Region zugewiesen. Anrufer können die ihnen am nächsten gelegene Region verwenden, um eine möglichst niedrige Latenz zu erzielen. Wenn eine Region ausfällt, ermöglicht die Verwendung globaler Tabellen eine schnelle Wiederherstellung, da das System die Verarbeitung des Anrufs an eine andere Region weitergeben kann, in der sich bereits eine replizierte Kopie der Daten befindet.

Zusammenfassung

- Der Modus „Schreiben in eine beliebige Region“ eignet sich für MRSC-Tabellen und idempotente Aufrufe von MREC-Tabellen.
- Der Modus „Schreiben in eine Region“ eignet sich für idempotente Aufrufe von MREC-Tabellen.
- Der Modus „Schreiben in Ihre Region“ eignet sich für nicht-idempotente Aufrufe von MREC-Tabellen, bei denen es wichtig ist, dass Clients in eine Region schreiben, die sich in ihrer Nähe befindet.

Routing-Strategien in DynamoDB

Der vielleicht komplexeste Teil der Bereitstellung einer globalen Tabelle ist die Verwaltung der Weiterleitung von Anforderungen. Anforderungen müssen zunächst von einem Endbenutzer an eine Region gesendet werden, die auf irgendeine Weise ausgewählt und weitergeleitet wird. Die Anfrage trifft auf einen Stapel von Diensten in dieser Region, einschließlich einer Rechenschicht, die möglicherweise aus einem Load Balancer besteht, der von einer AWS Lambda Funktion, einem Container oder einem Amazon Elastic Compute Cloud (Amazon EC2) -Knoten unterstützt wird, und möglicherweise auf andere Dienste, einschließlich einer anderen Datenbank. Diese

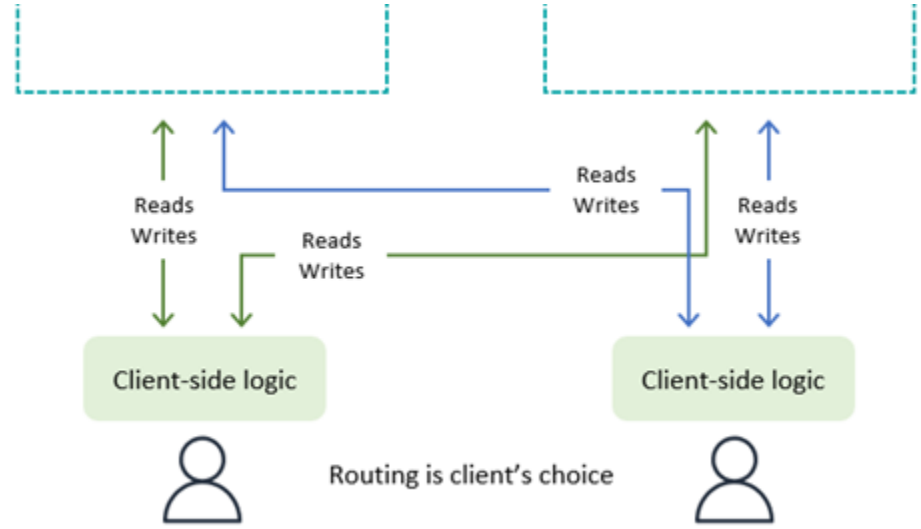
Datenverarbeitungsebene kommuniziert mit DynamoDB. Dazu sollte sie den lokalen Endpunkt für diese Region verwenden. Die Daten in der globalen Tabelle werden in alle anderen teilnehmenden Regionen repliziert und jede Region verfügt über einen ähnlichen Service-Stack rund um ihre DynamoDB-Tabelle.

Die globale Tabelle stellt jedem Stack in den verschiedenen Regionen eine lokale Kopie derselben Daten zur Verfügung. Sie könnten einen einzelnen Stack in einer einzelnen Region vorsehen und im Falle eines Problems mit der lokalen DynamoDB-Tabelle Remote-Aufrufe an den DynamoDB-Endpunkt einer sekundären Region einplanen. Dies stellt keine bewährte Methode dar. Wenn in einer Region ein Problem auftritt, das durch DynamoDB verursacht wird (oder, was wahrscheinlicher ist, durch etwas anderes im Stack oder durch einen anderen Service, der von DynamoDB abhängt), ist es am besten, den Endbenutzer zur Verarbeitung in eine andere Region weiterzuleiten und die Datenverarbeitungsebene dieser anderen Region zu verwenden, die mit ihrem lokalen DynamoDB-Endpunkt kommuniziert. Bei diesem Ansatz wird die problematische Region vollständig umgangen. Um Resilienz zu gewährleisten, benötigen Sie eine Replikation über mehrere Regionen hinweg, wobei sowohl die Datenverarbeitungsebene als auch die Datenebene repliziert werden.

Es gibt zahlreiche alternative Möglichkeiten, die Anforderung eines Endbenutzers zur Verarbeitung an eine Region weiterzuleiten. Die optimale Option ist von Ihrem Schreibmodus und Ihrer Failover-Strategie abhängig. In diesem Abschnitt werden vier Optionen erörtert: Client-gesteuert, Datenverarbeitungsebene, Route 53 und Global Accelerator.

Client-gesteuerte Weiterleitung von Anforderungen

Bei der clientgesteuerten Weiterleitung von Anfragen, wie im folgenden Diagramm dargestellt, verfolgt der Endbenutzer-Client (eine Anwendung, eine Webseite mit JavaScript oder ein anderer Client) die gültigen Anwendungsendpunkte (z. B. ein Amazon API Gateway Gateway-Endpunkt statt eines wörtlichen DynamoDB-Endpunkts) und verwendet seine eigene eingebettete Logik, um die Region auszuwählen, mit der kommuniziert werden soll. Die Entscheidung kann auf der Grundlage einer zufälligen Auswahl, der niedrigsten beobachteten Latenzen, der höchsten beobachteten Bandbreitenmessung oder lokal durchgeführter Zustandsprüfungen getroffen werden.



Ein Vorteil ist, dass sich die Client-gesteuerte Weiterleitung von Anforderungen beispielsweise an die realen Bedingungen des öffentlichen Internetdatenverkehrs anpassen und die Region wechseln kann, falls Leistungseinbußen festgestellt werden. Der Client muss alle potenziellen Endpunkte kennen, doch es kommt nicht oft vor, dass ein neuer regionaler Endpunkt gestartet wird.

Beim Modus Schreiben in eine beliebige Region kann ein Client einseitig seinen bevorzugten Endpunkt auswählen. Wenn der Zugriff auf eine Region beeinträchtigt ist, kann der Client zu einem anderen Endpunkt weiterleiten.

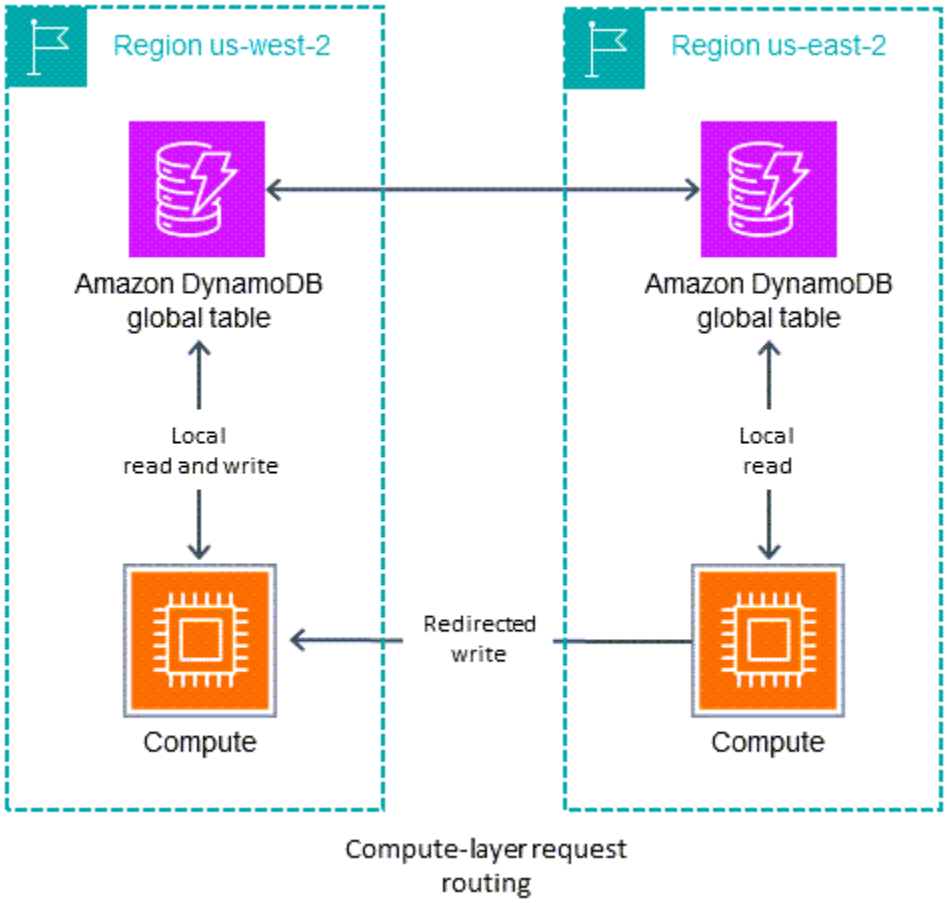
Im Modus Schreiben in eine Region benötigt der Client einen Mechanismus, um seine Schreibvorgänge an die aktuell aktive Region weiterzuleiten. Hierfür könnte es ausreichen, empirisch zu testen, welche Region gerade Schreibvorgänge akzeptiert (wobei jede Ablehnung von Schreibvorgängen vermerkt und auf eine Alternative zurückgegriffen wird), oder es muss ein globaler Koordinator aufgerufen werden, um den aktuellen Anwendungsstatus abzufragen (vielleicht auf der Grundlage der Routing-Steuerung von Amazon Application Recovery Controller (ARC), die ein Quorum-gesteuertes System mit 5 Regionen zur Aufrechterhaltung des globalen Zustands für derartige Anforderungen bereitstellt). Der Client kann entscheiden, ob Lesevorgänge für letztendliche Konsistenz in eine beliebige Region weitergeleitet werden können oder ob sie an die aktive Region weitergeleitet werden müssen, um eine starke Konsistenz zu erzielen. Weitere Informationen finden Sie unter Funktionsweise von Route 53.

Im Modus Schreiben in Ihre Region muss der Client die Heimatregion für den Datensatz ermitteln, mit dem er arbeitet. Wenn der Client beispielsweise einem Benutzerkonto entspricht und jedes Benutzerkonto einer Region zugeordnet ist, kann der Client den entsprechenden Endpunkt von einem globalen Anmeldesystem anfordern.

Ein Finanzdienstleistungsunternehmen beispielsweise, das Benutzer bei der Verwaltung ihrer Geschäftsfinanzen über das Internet unterstützt, könnte globale Tabellen mit dem Modus Schreiben in Ihre Region verwenden. Jeder Benutzer muss sich bei einem zentralen Service anmelden. Dieser Service gibt Anmeldeinformationen sowie den Endpunkt für die Region zurück, für den diese Anmeldeinformationen gelten. Die Anmeldeinformationen sind nur kurze Zeit gültig. Danach handelt die Webseite automatisch eine neue Anmeldung aus, was die Gelegenheit bietet, die Aktivitäten des Benutzers möglicherweise in eine neue Region umzuleiten.

Weiterleitung von Anforderungen auf Datenverarbeitungsebene

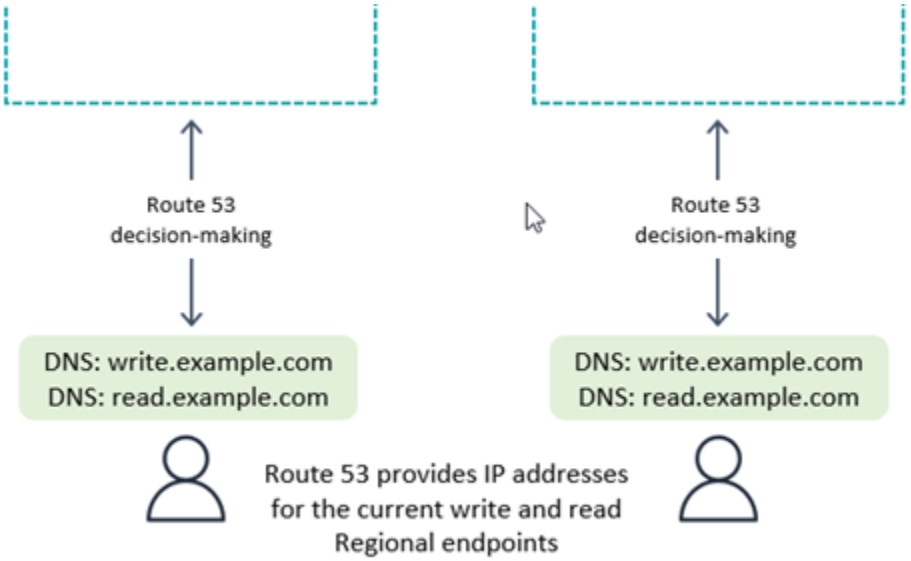
Bei der Weiterleitung von Anforderungen auf Datenverarbeitungsebene, die im folgenden Diagramm dargestellt ist, entscheidet der auf Datenverarbeitungsebene ausgeführte Code, ob er die Anforderung lokal verarbeiten oder an eine Kopie des Codes weitergeben möchte, die in einer anderen Region ausgeführt wird. Wenn Sie den Modus Schreiben in eine Region verwenden, erkennt die Datenverarbeitungsebene möglicherweise, dass es sich nicht um die aktive Region handelt, und erlaubt lokale Lesevorgänge, während alle Schreibvorgänge an eine andere Region weitergeleitet werden. Dieser Code auf Datenverarbeitungsebene muss die Datentopologie und die Regeln für die Weiterleitung kennen und unter Berücksichtigung der aktuellen Einstellungen, die angeben, welche Regionen für welche Daten aktiv sind, zuverlässig durchsetzen. Der äußere Software-Stack innerhalb der Region muss nicht wissen, wie Lese- und Schreibanforderungen von dem Microservice weitergeleitet werden. In einem robusten Design überprüft die empfangende Region, ob sie die aktuelle Primärregion für den Schreibvorgang ist. Ist dies nicht der Fall, wird ein Fehler mit dem Hinweis generiert, dass der globale Zustand korrigiert werden muss. Die empfangende Region könnte den Schreibvorgang auch eine Weile zwischenspeichern, wenn sich die Primärregion gerade ändert. In allen Fällen schreibt der Computing-Stack in einer Region nur auf seinen lokalen DynamoDB-Endpunkt, die Computing-Stacks kommunizieren aber möglicherweise miteinander.



Die Vanguard Group verwendet für diesen Routing-Prozess ein System namens Global Orchestration and Status Tool (GOaST) und eine Bibliothek namens Global Multi-Region Library (GMRlib), wie auf der re:Invent 2022 vorgestellt. Sie verwenden ein einziges primäres Modell. follow-the-sun GOaST behält den globalen Status bei, ähnlich der ARC-Routing-Steuerung, die im vorherigen Abschnitt beschrieben wurde. Unter Verwendung einer globalen Tabelle verfolgt das Unternehmen, welche Region die Primärregion ist und wann der nächste Wechsel der Primärregion geplant ist. Alle Lese- und Schreiboperationen werden ausgeführt GMRlib, was mit GOa ST koordiniert wird. GMRlib ermöglicht die lokale Ausführung von Lesevorgängen mit geringer Latenz. GMRlib Überprüft bei Schreibvorgängen, ob die lokale Region die aktuelle primäre Region ist. Falls ja, wird der Schreibvorgang direkt abgeschlossen. Wenn nicht, GMRlib leitet die Schreibaufgabe an die GMRlib in der primären Region weiter. Diese empfangende Bibliothek bestätigt, dass sie sich ebenfalls als Primärregion betrachtet, und gibt einen Fehler aus, wenn dies nicht der Fall ist, was auf eine Verzögerung bei der Weitergabe des globalen Zustands hindeutet. Dieses Vorgehen bietet einen Validierungsvorteil, da nicht direkt auf einen DynamoDB-Remote-Endpunkt geschrieben wird.

Route-53-Weiterleitung von Anforderungen

Amazon Application Recovery Controller (ARC) ist eine Domain Name Service (DNS)-Technologie. Bei Route 53 fordert der Client seinen Endpunkt an, indem er nach einem bekannten DNS-Domainnamen sucht, und Route 53 gibt die IP-Adresse des regionalen Endpunkts/der regionalen Endpunkte zurück, den/die es für am geeignetsten hält. Dies wird im folgenden Diagramm veranschaulicht. Route 53 verfügt über eine lange Liste von Weiterleitungsrichtlinien, anhand derer die geeignete Region bestimmt wird. Route 53 kann auch ein Failover-Routing durchführen, um Datenverkehr von Regionen wegzuleiten, die die Zustandsprüfung nicht bestehen.



Mit dem Modus Schreiben in eine beliebige Region oder in Kombination mit der Weiterleitung von Anforderungen auf Datenverarbeitungsebene am Back-End kann Route 53 vollständigen Zugriff erhalten, um die Region auf der Grundlage komplexer interner Regeln zurückzugeben, beispielsweise die Region in nächster Netzwerknähe oder in nächster geografischer Nähe oder eine andere Wahl.

Im Modus Schreiben in eine Region können Sie Route 53 so konfigurieren, dass es die derzeit aktive Region zurückgibt (mithilfe von Route 53 ARC). Wenn der Client eine Verbindung mit einer passiven Region herstellen möchte (z. B. für Lesevorgänge), kann er nach einem anderen DNS-Namen suchen.

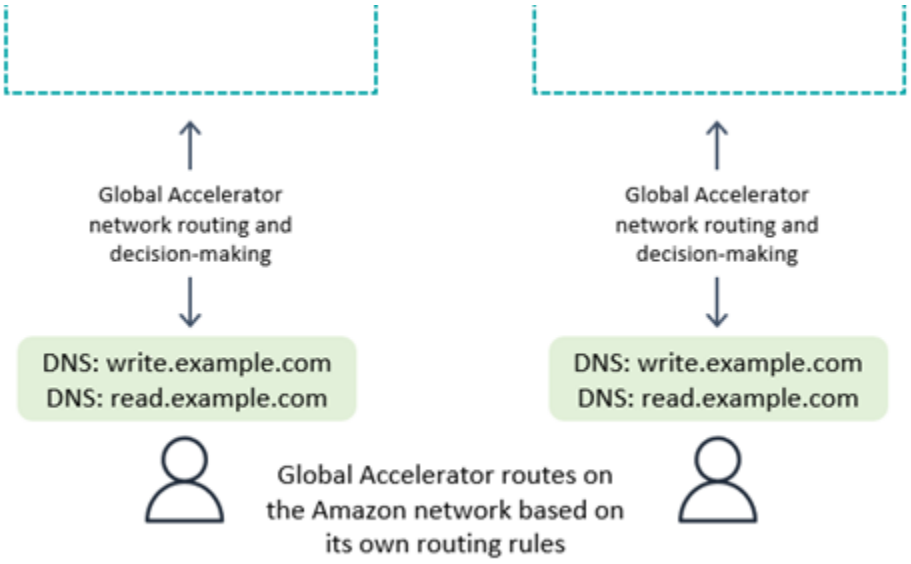
> **Note**
>
> Die IP-Adressen in der Antwort von Route 53 werden von den Clients für eine gewisse Zeit zwischengespeichert, die in der TTL-Einstellung (Time to Live) für den Domainnamen

angegeben ist. Eine längere TTL verlängert das Recovery Time Objective (RTO), damit alle Clients den neuen Endpunkt erkennen. Ein Wert von 60 Sekunden ist typisch für den Failover-Einsatz. Nicht jede Software hält sich perfekt an den Ablauf der DNS-TTL und es kann mehrere Ebenen des DNS-Caching geben, z. B. im Betriebssystem, in der virtuellen Maschine und in der Anwendung.

Im Modus Schreiben in Ihre Region sollte die Verwendung von Route 53 vermieden werden, es sei denn, Sie verwenden auch die Weiterleitung von Anforderungen auf Datenverarbeitungsebene.

Weiterleitung von Anforderungen mit Global Accelerator

Wie im folgenden Diagramm gezeigt, sucht ein Client mit AWS Global Accelerator nach dem bekannten Domainnamen in Route 53. Anstatt jedoch eine IP-Adresse zurückzugewinnen, die einem regionalen Endpunkt entspricht, erhält der Client eine statische Anycast-IP-Adresse, die zum nächstgelegenen AWS Edge-Standort weitergeleitet wird. Ausgehend von diesem Edge-Standort wird der gesamte Datenverkehr über das private AWS -Netzwerk zu einem Endpunkt (z. B. einem Load Balancer oder API Gateway) in einer Region geleitet, die anhand von Weiterleitungsregeln ausgewählt wird, die in Global Accelerator verwaltet werden. Im Vergleich zur Weiterleitung auf der Grundlage von Route-53-Regeln weist die Weiterleitung von Anforderungen mit Global Accelerator geringere Latenzen auf, da der Datenverkehr im öffentlichen Internet reduziert wird. Da Global Accelerator bei der Änderung der Weiterleitungsregeln nicht vom DNS-TTL-Ablauf abhängig ist, kann Global Accelerator die Weiterleitung außerdem schneller anpassen.



Im Modus Schreiben in eine beliebige Region oder in Kombination mit der Weiterleitung von Anforderungen auf Datenverarbeitungsebene am Back-End funktioniert Global Accelerator problemlos. Der Client stellt eine Verbindung zum nächstgelegenen Edge-Standort her und muss sich keine Gedanken darüber machen, welche Region die Anforderung erhält.

Beim Schreiben in eine Region müssen die Weiterleitungsregeln von Global Accelerator Anforderungen an die derzeit aktive Region senden. Sie können Zustandsprüfungen verwenden, die künstlich einen Fehler in einer Region melden, die von Ihrem globalen System nicht als aktive Region angesehen wird. Wie bei DNS ist es möglich, einen alternativen DNS-Domainnamen für die Weiterleitung von Leseanforderungen zu verwenden, wenn die Anforderungen aus einer beliebigen Region stammen können.

Im Modus Schreiben in Ihre Region sollte die Verwendung von Global Accelerator vermieden werden, es sei denn, Sie verwenden auch die Weiterleitung von Anforderungen auf Datenverarbeitungsebene.

Evakuierungsprozesse

Beim Evakuieren einer Region werden Aktivitäten, für gewöhnlich Lese- und Schreibaktivitäten oder Leseaktivitäten, aus dieser Region migriert.

Evakuierung einer Live-Region

Möglicherweise entscheiden Sie sich aus einer Reihe von Gründen dafür, eine aktive Region zu evakuieren: als Teil Ihrer üblichen Geschäftstätigkeit (z. B. wenn Sie den Modus „In eine follow-the-sun Region schreiben“ verwenden), aufgrund einer Geschäftsentscheidung, die derzeit aktive Region zu ändern, als Reaktion auf Fehler im Software-Stack außerhalb von DynamoDB oder weil Sie auf allgemeine Probleme stoßen, wie z. B. höhere Latenzen als üblich innerhalb der Region.

Mit dem Modus Schreiben in eine beliebige Region ist es ganz einfach, eine Live-Region zu evakuieren. Sie können den Datenverkehr über ein beliebiges Routing-System an alternative Regionen weiterleiten und die Schreibvorgänge in der evakuierten Region wie gewohnt replizieren lassen.

Die Modi „In eine Region schreiben“ und „In Ihre Region schreiben“ werden normalerweise mit MREC-Tabellen verwendet. Daher müssen Sie sicherstellen, dass alle Schreibvorgänge in die aktive Region vollständig aufgezeichnet, als Stream verarbeitet und global weitergegeben wurden, bevor Sie mit Schreibvorgängen in der neuen aktiven Region beginnen. So stellen Sie sicher, dass künftige Schreibvorgänge mit der neuesten Version der Daten verarbeitet werden.

Angenommen, Region A ist aktiv und Region B passiv (entweder für die vollständige Tabelle oder für Elemente in Region A). Der typische Evakuierungsmechanismus besteht darin, Schreibvorgänge in A anzuhalten, lange genug zu warten, bis diese Vorgänge vollständig an B weitergegeben wurden, den Architektur-Stack zu aktualisieren, um B als aktiv zu erkennen, und dann die Schreibvorgänge auf B fortzusetzen. Es gibt keine Metrik, die mit absoluter Sicherheit anzeigt, dass Region A ihre Daten vollständig in Region B repliziert hat. Wenn Region A fehlerfrei ist, reicht es in der Regel aus, Schreibvorgänge in Region A anzuhalten und 10-mal den aktuellen Maximalwert der Metrik `ReplicationLatency` abzuwarten, um festzustellen, dass die Replikation abgeschlossen ist. Wenn Region A fehlerhaft ist und andere Bereiche mit erhöhten Latenzen aufweist, würden Sie als Wartezeit ein größeres Vielfaches des Maximalwertes wählen.

Evakuierung einer Offline-Region

Ein Sonderfall ist zu berücksichtigen: Was wäre, wenn Region A ohne vorherige Ankündigung vollständig offline geht? Dies ist äußerst unwahrscheinlich, sollte aber dennoch in Betracht gezogen werden.

Evakuierung einer Offline-MRSC-Tabelle

Wenn dies bei einer MRSC-Tabelle passiert, müssen Sie nichts Besonderes tun. Globale MRSC-Tabellen unterstützen einen Recovery Point Objective (RPO) von Null. Alle erfolgreichen Schreibvorgänge in die MRSC-Tabelle in der Offline-Region sind auch in allen anderen Regionstabellen verfügbar, sodass keine potenzielle Datenlücke besteht, selbst wenn die Region ohne vorherige Ankündigung vollständig offline geht. Unternehmen können weiterhin Replikate verwenden, die sich in den anderen Regionen befinden.

Evakuierung einer Offline-MREC-Tabelle

Wenn dies bei einer MREC-Tabelle passiert, werden alle Schreibvorgänge in Region A, die noch nicht weitergegeben wurden, gespeichert und weitergegeben, nachdem Region A wieder online ist. Die Schreibvorgänge gehen nicht verloren, doch ihre Weitergabe verzögert sich um unbestimmte Zeit.

Wie in diesem Fall vorzugehen ist, entscheidet die Anwendung. Um die Geschäftskontinuität zu wahren, müssen Schreibvorgänge möglicherweise in die neue Primärregion B übertragen werden. Wenn jedoch ein Element in Region B eine Aktualisierung erhält, während die Weitergabe eines Schreibvorgangs für dieses Element aus Region A aussteht, wird die Weitergabe nach dem Prinzip Wer zuletzt schreibt, gewinnt unterdrückt. Jede Aktualisierung in Region B kann eine eingehende Schreibanforderung unterdrücken.

Beim Modus Schreiben in eine beliebige Region können Lese- und Schreibvorgänge in Region B fortgesetzt werden. Dabei wird darauf vertraut, dass die Elemente in Region A irgendwann in Region B übertragen werden, und es wird anerkannt, dass möglicherweise Elemente fehlen, bis Region A wieder online ist. Wenn möglich, beispielsweise bei idempotenten Schreibvorgängen, sollten Sie erwägen, den aktuellen Schreibverkehr wiederzugeben (z. B. durch die Verwendung einer vorgeschalteten Ereignisquelle), um die Lücke möglicherweise fehlender Schreibvorgänge zu schließen und die Konfliktlösung „Wer zuletzt schreibt, gewinnt“ die letztendliche Weiterleitung des eingehenden Schreibvorgangs unterdrücken zu lassen.

Bei den anderen Schreibmodi muss man berücksichtigen, inwieweit die Arbeit mit einem leichten out-of-date Blick auf die Welt weitergehen kann. Eine kurze Zeitspanne von Schreibvorgängen, wie von `ReplicationLatency` verfolgt, wird fehlen, bis Region A wieder online ist. Ist ein Vorankommen der Geschäfte möglich? In einigen Fällen ja, in anderen jedoch möglicherweise nicht ohne zusätzliche Mechanismen zur Risikominderung.

Stellen Sie sich zum Beispiel vor, Sie müssen selbst nach einem vollständigen Ausfall der Region ohne Unterbrechung ein verfügbares Guthaben aufrechterhalten. Sie könnten das Guthaben in zwei Posten aufteilen, einen in Region A und einen in Region B, die zunächst jeweils die Hälfte des Guthabens beinhalten. Hierfür würde der Modus Schreiben in Ihre Region verwendet. Transaktionsaktualisierungen, die in jeder Region verarbeitet werden, würden der lokalen Kopie des Guthabens zugeordnet. Wenn Region A vollständig offline geht, könnte die Arbeit mit der Transaktionsverarbeitung in Region B fortgesetzt werden und die Schreibvorgänge wären auf den Teil des Guthabens in Region B beschränkt. Eine solche Aufteilung des Guthabens führt zu Schwierigkeiten, wenn das Guthaben knapp wird oder wieder ausgeglichen werden muss, doch ist dies ein Beispiel für eine sichere Geschäftserholung auch bei unsicheren ausstehenden Schreibvorgängen.

Stellen Sie sich als weiteres Beispiel vor, Sie erfassen Daten aus Webformularen. Sie können Optimistic Concurrency Control (OCC) verwenden, um Datenelementen Versionen zuzuweisen und die neueste Version als verdecktes Feld in das Webformular einzubetten. Bei jedem Absenden ist der Schreibvorgang nur erfolgreich, wenn die Version in der Datenbank immer noch mit der Version übereinstimmt, für die das Formular erstellt wurde. Wenn die Versionen nicht übereinstimmen, kann das Webformular auf der Grundlage der aktuellen Version in der Datenbank aktualisiert (oder sorgfältig zusammengeführt) werden, und der Benutzer kann erneut fortfahren. Das OCC-Modell schützt in der Regel davor, dass ein anderer Client die Daten überschreibt und eine neue Version der Daten erzeugt. Es kann aber auch bei einem Failover hilfreich sein, wenn ein Client auf ältere Datenversionen stößt. Angenommen, Sie verwenden den Zeitstempel als Version. Das Formular wurde um 12:00 Uhr erstmalig für Region A erstellt,

versucht aber (nach einem Failover), in Region B zu schreiben, und stellt fest, dass die neueste Version in der Datenbank die Uhrzeit 11:59 aufweist. In diesem Szenario kann der Client entweder warten, bis die Version von 12:00 Uhr an Region B weitergegeben wird, und dann auf diese Version schreiben oder auf 11:59 aufbauend eine neue Version 12:01 erstellen (die nach dem Schreiben die nach Wiederherstellung von Region A eingehende Version unterdrücken würde).

Ein drittes Beispiel: Ein Finanzdienstleistungsunternehmen speichert Daten über Kundenkonten und deren Finanztransaktionen in einer DynamoDB-Datenbank. Bei einem vollständigen Ausfall von Region A sollte das Unternehmen sicherstellen, dass alle Schreibaktivitäten im Zusammenhang mit seinen Konten entweder vollständig in Region B verfügbar sind, oder seine Konten als bekannte Teilkonten unter Quarantäne stellen, bis Region A wieder online ist. Anstatt den gesamten Geschäftsverkehr zu unterbrechen, entschied sich das Unternehmen dafür, die Geschäftstätigkeit nur für den winzigen Bruchteil der Konten auszusetzen, bei denen nicht weitergeleitete Transaktionen festgestellt wurden. Um dies zu erreichen, wurde eine dritte Region verwendet, die wir Region C nennen wollen. Vor der Verarbeitung von Schreibvorgänge in Region A stellte das Unternehmen eine kurze Zusammenfassung dieser ausstehenden Vorgänge (z. B. eine neue Transaktionsanzahl für ein Konto) in Region C. Diese Zusammenfassung reichte für Region B aus, um festzustellen, ob ihre Ansicht vollständig aktuell war. Durch diese Maßnahme wurde das Konto ab dem Zeitpunkt des Schreibens in Region C praktisch gesperrt, bis Region A die Schreibvorgänge akzeptierte und Region B sie erhielt. Die Daten in Region C wurden nur im Rahmen eines Failovers verwendet, nach dem Region B ihre Daten mit Region C abgleichen und so feststellen konnte, ob einige ihrer Konten veraltet waren. Diese Konten wurden als unter Quarantäne gestellt, bis im Zuge der Wiederherstellung von Region A die Teildaten an Region B weitergegeben wurden. Falls Region C ausfallen würde, könnte stattdessen eine neue Region D zur Verwendung eingerichtet werden. Die Daten in Region C waren sehr flüchtig, und nach ein paar Minuten verfügte Region D über ausreichend up-to-date Aufzeichnungen der Schreibvorgänge während des Fluges, um sie in vollem Umfang nutzen zu können. Sollte es zu einem Ausfall von Region B kommen, könnte Region A in Kooperation mit Region C, weiterhin Schreibanforderungen annehmen. Das Unternehmen war bereit, Schreibvorgänge mit höherer Latenz zu akzeptieren (in zwei Regionen: C und dann A), und verfügte glücklicherweise über ein Datenmodell, bei dem der Status eines Kontos knapp zusammengefasst werden konnte.

Planung der Durchsatzkapazität für globale DynamoDB-Tabellen

Bei einer Migration des Datenverkehrs von einer Region in eine andere müssen die DynamoDB-Tabelleneinstellungen in Bezug auf die Kapazität sorgfältig geprüft werden.

Hier sind einige Überlegungen zur Verwaltung der Schreibkapazität:

- Eine globale Tabelle muss sich im On-Demand-Modus befinden oder mit aktiviertem Auto Scaling bereitgestellt werden.
- Bei Bereitstellung mit Auto Scaling werden die Schreibeinstellungen (Mindest-, Höchst- und Zielauslastung) regionsübergreifend repliziert. Die Einstellungen für Auto Scaling werden zwar synchronisiert, die tatsächlich bereitgestellte Schreibkapazität kann jedoch unabhängig davon von Region zu Region variieren.
- Ein Grund dafür, dass möglicherweise unterschiedliche Schreibkapazitäten bereitgestellt werden, ist die TTL-Funktion. Wenn Sie TTL in DynamoDB aktivieren, können Sie einen Attributnamen angeben, dessen Wert die Ablaufzeit für das Element angibt, und zwar im Unix-Epochenzeitformat in Sekunden. Nach Ablauf dieser Zeit kann DynamoDB das Element löschen, ohne dass Schreibkosten anfallen. Bei globalen Tabellen können Sie TTL in einer beliebigen Region konfigurieren und diese Einstellung wird automatisch auf andere Regionen repliziert, die der globalen Tabelle zugeordnet sind. Wenn ein Element über eine TTL-Regel gelöscht werden kann, kann dies in jeder Region geschehen. Der Löschvorgang wird ausgeführt, ohne dass Schreibeinheiten in der Quelltabelle verbraucht werden. Die Replikattabellen erhalten jedoch einen replizierten Schreibvorgang für diesen Löschvorgang und es fallen Kosten für replizierte Schreibeinheiten an. TTL wird in MRSC-Tabellen nicht unterstützt.
- Wenn Sie Auto Scaling verwenden, stellen Sie sicher, dass die Einstellung für die maximale bereitgestellte Schreibkapazität hoch genug ist, um alle Schreibvorgänge sowie alle potenziellen TTL-Löschvorgänge zu bewältigen. Auto Scaling passt jede Region ihrem Verbrauch an Schreibkapazität entsprechend an. Für On-Demand-Tabellen gibt es keine Einstellung für die maximale bereitgestellte Schreibkapazität, das Schreibdurchsatz-Limit auf Tabellenebene gibt jedoch an, welche maximale dauerhafte Schreibkapazität die On-Demand-Tabelle zulässt. Das Standardlimit beträgt 40 000, ist jedoch einstellbar. Es wird empfohlen, den Wert hoch genug einzustellen, um alle Schreibvorgänge (einschließlich TTL-Schreibvorgängen) bewältigen zu können, die die On-Demand-Tabelle möglicherweise erfordert. Dieser Wert muss in allen teilnehmenden Regionen gleich sein, wenn Sie globale Tabellen einrichten.

Hier sind einige Überlegungen zur Verwaltung der Lesekapazität:

- Die Einstellungen für die Verwaltung der Lesekapazität dürfen von Region zu Region unterschiedlich sein, da davon ausgegangen wird, dass verschiedene Regionen möglicherweise unterschiedliche Lesemuster haben. Beim erstmaligen Hinzufügen eines globalen Replikats zu einer Tabelle wird die Kapazität der Quellregion übertragen. Nach der Erstellung können Sie die

Einstellungen für die Lesekapazität anpassen und diese Einstellungen werden nicht auf die andere Seite übertragen.

- Wenn Sie DynamoDB Auto Scaling verwenden, stellen Sie sicher, dass die Einstellungen für die maximale bereitgestellte Lesekapazität hoch genug sind, um alle Lesevorgänge in allen Regionen bewältigen zu können. Während des Standardbetriebs wird die Lesekapazität möglicherweise auf die Regionen verteilt, doch bei einem Failover sollte die Tabelle automatisch an den höheren Lese-Workload angepasst werden können. Für On-Demand-Tabellen gibt es keine Einstellung für die maximale bereitgestellte Lesekapazität, das Lesedurchsatz-Limit auf Tabellenebene gibt jedoch an, welche maximale dauerhafte Lesekapazität die On-Demand-Tabelle zulässt. Das Standardlimit beträgt 40 000, ist jedoch einstellbar. Es wird empfohlen, den Wert hoch genug einzustellen, um alle Lesevorgänge bewältigen zu können, die die Tabelle möglicherweise benötigt, falls alle Lesevorgänge an diese einzelne Region weitergeleitet werden sollten.
- Wenn eine Tabelle in einer Region normalerweise keinen Lesedatenverkehr empfängt, nach einem Failover jedoch möglicherweise eine große Menge an Lesedatenverkehr aufnehmen muss, können Sie die Kapazität der Tabelle vorwärmen, damit sie ein höheres Maß an Lesedatenverkehr akzeptieren kann.

ARC verfügt über Bereitschaftsprüfungen, mit deren Hilfe geprüft werden kann, ob DynamoDB-Regionen ähnliche Tabelleneinstellungen und Kontokontingente aufweisen, unabhängig davon, ob Sie Route 53 zum Weiterleiten von Anforderungen verwenden oder nicht. Diese Bereitschaftsprüfungen können auch bei der Anpassung der Kontingente auf Kontoebene hilfreich sein, um eine Übereinstimmung der Kontingente sicherzustellen.

Checkliste zur Vorbereitung für globale DynamoDB-Tabellen

Verwenden Sie die folgende Checkliste für Entscheidungen und Aufgaben bei der Bereitstellung globaler Tabellen.

- Ermitteln Sie, ob Ihr Anwendungsfall mehr von einem MRSC- oder einem MREC-Konsistenzmodus profitiert. Benötigen Sie starke Konsistenz, auch wenn die Latenz ist und andere Kompromisse bestehen?
- Festlegung, wie viele und welche Regionen an der globalen Tabelle teilnehmen sollen. Wenn Sie MRSC verwenden möchten, entscheiden Sie, ob die dritte Region ein Replikat oder ein Witness sein soll.

- Ermittlung des Schreibmodus Ihrer Anwendung. Dies ist nicht dasselbe wie der Konsistenzmodus. Weitere Informationen finden Sie unter Schreibmodi bei Verwendung von globalen DynamoDB-Tabellen.
- Planung Ihrer Strategie für Routing-Strategien in DynamoDB auf der Grundlage Ihres Schreibmodus.
- Definieren Sie Ihre

Beim Evakuieren einer Region werden Aktivitäten, für gewöhnlich Lese- und Schreibaktivitäten oder Leseaktivitäten, aus dieser Region migriert.

Evakuierung einer Live-Region

Möglicherweise entscheiden Sie sich aus einer Reihe von Gründen dafür, eine aktive Region zu evakuieren: als Teil Ihrer üblichen Geschäftstätigkeit (z. B. wenn Sie den Modus „In eine follow-the-sun Region schreiben“ verwenden), aufgrund einer Geschäftsentscheidung, die derzeit aktive Region zu ändern, als Reaktion auf Fehler im Software-Stack außerhalb von DynamoDB oder weil Sie auf allgemeine Probleme stoßen, wie z. B. höhere Latenzen als üblich innerhalb der Region.

Mit dem Modus Schreiben in eine beliebige Region ist es ganz einfach, eine Live-Region zu evakuieren. Sie können den Datenverkehr über ein beliebiges Routing-System an alternative Regionen weiterleiten und die Schreibvorgänge in der evakuierten Region wie gewohnt replizieren lassen.

Die Modi „In eine Region schreiben“ und „In Ihre Region schreiben“ werden normalerweise mit MREC-Tabellen verwendet. Daher müssen Sie sicherstellen, dass alle Schreibvorgänge in die aktive Region vollständig aufgezeichnet, als Stream verarbeitet und global weitergegeben wurden, bevor Sie mit Schreibvorgängen in der neuen aktiven Region beginnen. So stellen Sie sicher, dass künftige Schreibvorgänge mit der neuesten Version der Daten verarbeitet werden.

Angenommen, Region A ist aktiv und Region B passiv (entweder für die vollständige Tabelle oder für Elemente in Region A). Der typische Evakuierungsmechanismus besteht darin, Schreibvorgänge in A anzuhalten, lange genug zu warten, bis diese Vorgänge vollständig an B weitergegeben wurden, den Architektur-Stack zu aktualisieren, um B als aktiv zu erkennen, und dann die Schreibvorgänge auf B fortzusetzen. Es gibt keine Metrik, die mit absoluter Sicherheit anzeigt, dass Region A ihre Daten vollständig in Region B repliziert hat. Wenn Region A fehlerfrei ist, reicht es in der Regel aus, Schreibvorgänge in Region A anzuhalten und 10-mal den aktuellen Maximalwert der Metrik `ReplicationLatency` abzuwarten, um

festzustellen, dass die Replikation abgeschlossen ist. Wenn Region A fehlerhaft ist und andere Bereiche mit erhöhten Latenzen aufweist, würden Sie als Wartezeit ein größeres Vielfaches des Maximalwertes wählen.

Evakuierung einer Offline-Region

Ein Sonderfall ist zu berücksichtigen: Was wäre, wenn Region A ohne vorherige Ankündigung vollständig offline geht? Dies ist äußerst unwahrscheinlich, sollte aber dennoch in Betracht gezogen werden.

Evakuierung einer Offline-MRSC-Tabelle

Wenn dies bei einer MRSC-Tabelle passiert, müssen Sie nichts Besonderes tun. Globale MRSC-Tabellen unterstützen einen Recovery Point Objective (RPO) von Null. Alle erfolgreichen Schreibvorgänge in die MRSC-Tabelle in der Offline-Region sind auch in allen anderen Regionstabellen verfügbar, sodass keine potenzielle Datenlücke besteht, selbst wenn die Region ohne vorherige Ankündigung vollständig offline geht. Unternehmen können weiterhin Replikate verwenden, die sich in den anderen Regionen befinden.

Evakuierung einer Offline-MREC-Tabelle

Wenn dies bei einer MREC-Tabelle passiert, werden alle Schreibvorgänge in Region A, die noch nicht weitergegeben wurden, gespeichert und weitergegeben, nachdem Region A wieder online ist. Die Schreibvorgänge gehen nicht verloren, doch ihre Weitergabe verzögert sich um unbestimmte Zeit.

Wie in diesem Fall vorzugehen ist, entscheidet die Anwendung. Um die Geschäftskontinuität zu wahren, müssen Schreibvorgänge möglicherweise in die neue Primärregion B übertragen werden. Wenn jedoch ein Element in Region B eine Aktualisierung erhält, während die Weitergabe eines Schreibvorgangs für dieses Element aus Region A aussteht, wird die Weitergabe nach dem Prinzip Wer zuletzt schreibt, gewinnt unterdrückt. Jede Aktualisierung in Region B kann eine eingehende Schreibanforderung unterdrücken.

Beim Modus Schreiben in eine beliebige Region können Lese- und Schreibvorgänge in Region B fortgesetzt werden. Dabei wird darauf vertraut, dass die Elemente in Region A irgendwann in Region B übertragen werden, und es wird anerkannt, dass möglicherweise Elemente fehlen, bis Region A wieder online ist. Wenn möglich, beispielsweise bei idempotenten Schreibvorgängen, sollten Sie erwägen, den aktuellen Schreibverkehr wiederzugeben (z. B. durch die Verwendung einer vorgeschalteten Ereignisquelle), um die

Lücke möglicherweise fehlender Schreibvorgänge zu schließen und die Konfliktlösung „Wer zuletzt schreibt, gewinnt“ die letztendliche Weiterleitung des eingehenden Schreibvorgangs unterdrücken zu lassen.

Bei den anderen Schreibmodi muss man berücksichtigen, inwieweit die Arbeit mit einem leichten out-of-date Blick auf die Welt weitergehen kann. Eine kurze Zeitspanne von Schreibvorgängen, wie von `ReplicationLatency` verfolgt, wird fehlen, bis Region A wieder online ist. Ist ein Vorankommen der Geschäfte möglich? In einigen Fällen ja, in anderen jedoch möglicherweise nicht ohne zusätzliche Mechanismen zur Risikominderung.

Stellen Sie sich zum Beispiel vor, Sie müssen selbst nach einem vollständigen Ausfall der Region ohne Unterbrechung ein verfügbares Guthaben aufrechterhalten. Sie könnten das Guthaben in zwei Posten aufteilen, einen in Region A und einen in Region B, die zunächst jeweils die Hälfte des Guthabens beinhalten. Hierfür würde der Modus Schreiben in Ihre Region verwendet. Transaktionsaktualisierungen, die in jeder Region verarbeitet werden, würden der lokalen Kopie des Guthabens zugeordnet. Wenn Region A vollständig offline geht, könnte die Arbeit mit der Transaktionsverarbeitung in Region B fortgesetzt werden und die Schreibvorgänge wären auf den Teil des Guthabens in Region B beschränkt. Eine solche Aufteilung des Guthabens führt zu Schwierigkeiten, wenn das Guthaben knapp wird oder wieder ausgeglichen werden muss, doch ist dies ein Beispiel für eine sichere Geschäftserholung auch bei unsicheren ausstehenden Schreibvorgängen.

Stellen Sie sich als weiteres Beispiel vor, Sie erfassen Daten aus Webformularen. Sie können Optimistic Concurrency Control (OCC) verwenden, um Datenelementen Versionen zuzuweisen und die neueste Version als verdecktes Feld in das Webformular einzubetten. Bei jedem Absenden ist der Schreibvorgang nur erfolgreich, wenn die Version in der Datenbank immer noch mit der Version übereinstimmt, für die das Formular erstellt wurde. Wenn die Versionen nicht übereinstimmen, kann das Webformular auf der Grundlage der aktuellen Version in der Datenbank aktualisiert (oder sorgfältig zusammengeführt) werden, und der Benutzer kann erneut fortfahren. Das OCC-Modell schützt in der Regel davor, dass ein anderer Client die Daten überschreibt und eine neue Version der Daten erzeugt. Es kann aber auch bei einem Failover hilfreich sein, wenn ein Client auf ältere Datenversionen stößt. Angenommen, Sie verwenden den Zeitstempel als Version. Das Formular wurde um 12:00 Uhr erstmalig für Region A erstellt, versucht aber (nach einem Failover), in Region B zu schreiben, und stellt fest, dass die neueste Version in der Datenbank die Uhrzeit 11:59 aufweist. In diesem Szenario kann der Client entweder warten, bis die Version von 12:00 Uhr an Region B weitergegeben wird, und dann auf diese Version schreiben oder

auf 11:59 aufbauend eine neue Version 12:01 erstellen (die nach dem Schreiben die nach

Wiederherstellung von Region A eingehende Version unterdrücken würde).

Ein drittes Beispiel: Ein Finanzdienstleistungsunternehmen speichert Daten über Kundenkonten und deren Finanztransaktionen in einer DynamoDB-Datenbank. Bei einem vollständigen Ausfall von Region A sollte das Unternehmen sicherstellen, dass alle Schreibaktivitäten im Zusammenhang mit seinen Konten entweder vollständig in Region B verfügbar sind, oder seine Konten als bekannte Teilkonten unter Quarantäne stellen, bis Region A wieder online ist. Anstatt den gesamten Geschäftsverkehr zu unterbrechen, entschied sich das Unternehmen dafür, die Geschäftstätigkeit nur für den winzigen Bruchteil der Konten auszusetzen, bei denen nicht weitergeleitete Transaktionen festgestellt wurden. Um dies zu erreichen, wurde eine dritte Region verwendet, die wir Region C nennen wollen. Vor der Verarbeitung von Schreibvorgänge in Region A stellte das Unternehmen eine kurze Zusammenfassung dieser ausstehenden Vorgänge (z. B. eine neue Transaktionsanzahl für ein Konto) in Region C. Diese Zusammenfassung reichte für Region B aus, um festzustellen, ob ihre Ansicht vollständig aktuell war. Durch diese Maßnahme wurde das Konto ab dem Zeitpunkt des Schreibens in Region C praktisch gesperrt, bis Region A die Schreibvorgänge akzeptierte und Region B sie erhielt. Die Daten in Region C wurden nur im Rahmen eines Failovers verwendet, nach dem Region B ihre Daten mit Region C abgleichen und so feststellen konnte, ob einige ihrer Konten veraltet waren. Diese Konten wurden als unter Quarantäne gestellt, bis im Zuge der Wiederherstellung von Region A die Teildaten an Region B weitergegeben wurden. Falls Region C ausfallen würde, könnte stattdessen eine neue Region D zur Verwendung eingerichtet werden. Die Daten in Region C waren sehr flüchtig, und nach ein paar Minuten verfügte Region D über ausreichend up-to-date Aufzeichnungen der Schreibvorgänge während des Fluges, um sie in vollem Umfang nutzen zu können. Sollte es zu einem Ausfall von Region B kommen, könnte Region A in Kooperation mit Region C, weiterhin Schreibanforderungen annehmen. Das Unternehmen war bereit, Schreibvorgänge mit höherer Latenz zu akzeptieren (in zwei Regionen: C und dann A), und verfügte glücklicherweise über ein Datenmodell, bei dem der Status eines Kontos knapp zusammengefasst werden konnte.

the section called “Evakuierungsprozesse” basierend auf Ihrem Konsistenzmodus, Ihrem Schreibmodus und Ihrer Routing-Strategie.

- Erfassung von Metriken zum Zustand, zur Latenz und zu Fehlern in jeder Region. Eine Liste der DynamoDB-Metriken finden Sie im AWS Blogbeitrag Monitoring Amazon DynamoDB for Operational Awareness. Dort finden Sie eine Liste der zu beobachtenden Metriken. Sie sollten

auch synthetische Canarys (künstliche Anforderungen, um Ausfälle zu erkennen, benannt nach dem Kanarienvogel in Kohlebergwerken) verwenden und eine Live-Beobachtung des Kundendatenverkehrs nutzen. Nicht alle Probleme werden in den DynamoDB-Metriken erkennbar sein.

- Wenn Sie MREC verwenden, legen Sie die Alarme für jeden anhaltenden Anstieg bei `ReplicationLatency` ein. Ein Anstieg könnte auf eine versehentliche Fehlkonfiguration hinweisen, bei der die globale Tabelle in verschiedenen Regionen unterschiedliche Schreibeinstellungen aufweist, wodurch es zu fehlgeschlagenen replizierten Anforderungen und höheren Latenzen kommt. Auch auf eine regionale Störung könnte ein solcher Anstieg hindeuten. Ein gutes Beispiel ist die Generierung einer Warnung, wenn der aktuelle Durchschnitt 180 000 Millisekunden überschreitet. Sie können auch darauf achten, ob der Wert für `ReplicationLatency` auf 0 fällt, was auf eine verzögerte Replikation hindeutet.
- Zuweisung ausreichend hoher Einstellungen für die maximale Lese- und Schreibkapazität für jede globale Tabelle.
- Ermittlung der Gründe für die Evakuierung einer Region im Voraus. Wenn die Entscheidung menschliches Urteilsvermögen erfordert, sollten alle Überlegungen dokumentiert werden. Diese Arbeit sollte bereits im Voraus sorgfältig und nicht unter Stress durchgeführt werden.
- Unterhaltung eines Runbooks für jede Aktion, die bei der Evakuierung einer Region stattfinden muss. Normalerweise ist der Aufwand für die globalen Tabellen sehr gering, doch das Verschieben des restlichen Stacks kann komplex sein.

> **Note**
>
> Bei Failover-Verfahren hat es sich bewährt, nur auf Operationen auf der Datenebene, nicht auf Operationen auf der Steuerebene zu setzen, da einige Operationen auf Steuerebene bei Regionsausfällen beeinträchtigt sein können.

Weitere Informationen finden Sie im AWS Blogbeitrag Build resilient applications with Amazon DynamoDB global tables: Part 4.

- Regelmäßiger Test aller Aspekte des Runbooks, einschließlich Evakuierungen von Regionen. Ein nicht getestetes Runbook ist ein unzuverlässiges Runbook.
- Ziehen Sie die Nutzung von AWS Resilience Hub in Betracht, um die Resilienz Ihrer gesamten Anwendung (einschließlich globaler Tabellen) zu bewerten. Das Hub bietet in seinem Dashboard einen umfassenden Überblick über die Widerstandsfähigkeit Ihres gesamten Anwendungsportfolios.

- Ziehen Sie die Nutzung von ARC-Bereitschaftsprüfungen in Betracht, um die aktuelle Konfiguration Ihrer Anwendung zu bewerten und Abweichungen von den bewährten Methoden zu verfolgen.
- Führen Sie beim Schreiben von Zustandsprüfungen für die Verwendung mit Route 53 oder Global Accelerator eine Reihe von Aufrufen aus, die den vollständigen Datenbankfluss abdecken. Wenn Sie Ihre Überprüfung darauf beschränken, nur zu bestätigen, dass der DynamoDB-Endpunkt aktiv ist, können Sie viele Fehlermodi wie AWS Identity and Access Management (IAM-) Konfigurationsfehler, Probleme bei der Codebereitstellung, Fehler im Stack außerhalb von DynamoDB, überdurchschnittliche Lese- oder Schreiblatenzen usw. nicht abdecken.

Häufig gestellte Fragen (FAQ) zur Bereitstellung globaler Tabellen

Wie hoch sind die Preise für globale Tabellen?

- Ein Schreibvorgang in einer herkömmlichen DynamoDB-Tabelle wird in Schreibkapazitätseinheiten (WCUsfür bereitgestellte Tabellen) oder Schreibanforderungseinheiten (WRUs) für On-Demand-Tabellen berechnet. Beim Schreiben eines 5-KB-Elements fällt eine Gebühr von 5 Einheiten an. Ein Schreibvorgang in eine globale Tabelle wird in replizierten Schreibkapazitätseinheiten (rWCUs, für bereitgestellte Tabellen) oder replizierten Schreibanforderungseinheiten (rWRUs, für On-Demand-Tabellen) berechnet. r WCUs und r WRUs werden genauso berechnet wie und. WGUs WRUs
- rWCU- und rWRU-Gebühren fallen in jeder Region an, in der das Element direkt oder durch Replikation geschrieben wird. Es fallen Gebühren für regionsübergreifende Datenübertragungen an.
- Wenn in einen globalen sekundären Index (GSI) geschrieben wird, gilt dies als lokaler Schreibvorgang und es werden reguläre Schreibeinheiten verwendet.
- Derzeit ist keine reservierte Kapazität für r WCUs oder r WRUs verfügbar. Der Kauf von reservierter Kapazität für WCUs kann für Tabellen, die Schreibeinheiten GSIs verbrauchen, von Vorteil sein.
- Wenn Sie einer globalen Tabelle eine neue Region hinzufügen, bootstrapt DynamoDB die neue Region automatisch und stellt sie Ihnen in Rechnung, als ob es sich um eine Tabellenwiederherstellung handeln würde, basierend auf der GB-Größe der Tabelle. Es fallen auch Gebühren für regionsübergreifende Datenübertragungen an.

Welche Regionen werden von globalen Tabellen unterstützt?

Global Tables Version 2019.11.21 (aktuell) unterstützt alle AWS-Regionen für MREC-Tabellen und die folgenden Regionsgruppen für MRSC-Tabellen:

- Region-Set USA: USA Ost (Nord-Virginia), USA Ost (Ohio), USA West (Oregon)
- Region-Set Europa: Europa (Frankfurt), Europa (Irland), Europa (London), Europa (Paris)
- Region-Set Asien-Pazifik: Asien-Pazifik (Tokio), Asien-Pazifik (Seoul) und Asien-Pazifik (Osaka).

Wie wird mit globalen Tabellen umgegangen GSIs ?

Wenn Sie in Version 2019.11.21 (Aktuell) der globalen Tabellen einen GSI in einer Region erstellen, wird dieser automatisch auch in anderen teilnehmenden Regionen erstellt und automatisch ausgefüllt.

Wie stoppe ich die Replikation einer globalen Tabelle?

- Sie können eine Replikattabelle genauso löschen, wie Sie jede andere Tabelle löschen würden. Wenn Sie die globale Tabelle löschen, wird die Replikation in die betreffende Region beendet und die in dieser Region gespeicherte Tabellenkopie wird gelöscht. Sie können die Replikation jedoch nicht stoppen, solange Sie Kopien der Tabelle als unabhängige Entitäten behalten, und Sie können die Replikation auch nicht unterbrechen.
- Eine MRSC-Tabelle muss in genau drei Regionen bereitgestellt werden. Um die Replikate zu löschen, müssen Sie alle Replikate und den Witness löschen, sodass die MRSC-Tabelle zu einer lokalen Tabelle wird.

Wie interagieren DynamoDB-Streams mit globalen Tabellen?

- Jede globale Tabelle erzeugt einen unabhängigen Stream, der auf allen ihren Schreibvorgängen basiert, unabhängig davon, wo diese gestartet wurden. Sie können wählen, ob Sie den DynamoDB-Stream in einer Region oder in allen Regionen (unabhängig voneinander) nutzen möchten. Wenn Sie lokale Schreibvorgänge, aber keine replizierten Schreibvorgänge verarbeiten möchten, können Sie jedem Element Ihr eigenes Regionsattribut hinzufügen, um die schreibende Region zu identifizieren. Sie können dann einen Lambda-Ereignisfilter verwenden, um die Lambda-Funktion nur für Schreibvorgänge in der lokalen Region aufzurufen. Dies ist bei Einfügungen und Aktualisierungen hilfreich, aber nicht bei Löschvorgängen.
- Globale Tabellen, die für Multi-Region Eventual Consistency (MREC-Tabellen) konfiguriert sind, replizieren Änderungen, indem sie diese aus einem DynamoDB-Stream in einer Replikattabelle auslesen und auf alle anderen Replikattabellen anwenden. DynamoDB ist daher standardmäßig für alle Replikate in einer globalen MREC-Tabelle aktiviert und kann in diesen Replikaten nicht deaktiviert werden. Der MREC-Replikationsprozess kann mehrere Änderungen in einem kurzen Zeitraum zu einem einzigen replizierten Schreibvorgang kombinieren. Daher kann der Stream jedes Replikats leicht unterschiedliche Datensätze enthalten. DynamoDB-Streams-Datensätze

in MREC-Replikaten werden immer elementweise sortiert, wobei die Reihenfolge zwischen den Elementen sich von Replikat zu Replikat unterscheiden kann.

- Globale Tabellen, die für Multi-Region Strong Consistency (MRSC-Tabellen) konfiguriert sind, verwenden keine DynamoDB Streams für die Replikation, sodass diese Funktion in MRSC-Replikaten standardmäßig nicht aktiviert ist. DynamoDB Streams können in einem MRSC-Replikat aktiviert werden. DynamoDB-Streams-Datensätze in MRSC-Replikaten sind für jedes Replikat identisch und werden immer elementweise sortiert, wobei die Reihenfolge zwischen den Elementen sich von Replikat zu Replikat unterscheiden kann.

Wie werden Transaktionen in globalen Tabellen gehandhabt?

- Transaktionale Operationen in MRSC-Tabellen führen zu Fehlern.
- Transaktionale Operationen in MREC-Tabellen bieten ACID-Garantien (Atomizität, Konsistenz, Isolierung und Zuverlässigkeit) nur in der Region, in der der Schreibvorgang ursprünglich durchgeführt wurde. Regionsübergreifende Transaktionen werden in globalen Tabellen nicht unterstützt. Wenn Sie beispielsweise eine globale MREC-Tabelle mit Replikaten in den Regionen USA Ost (Ohio) und USA West (Oregon) nutzen und eine `TransactWriteItems`-Operation in der Region USA Ost (Ohio) durchführen, sind unter Umständen partiell durchgeführte Transaktionen in der Region USA West (Oregon) zu beobachten, während die Änderungen repliziert werden. Die Änderungen werden erst dann in die anderen Regionen repliziert, nachdem sie in der Quellregion in die Datenbank eingetragen wurden.

Wie interagieren globale Tabellen mit dem DynamoDB-Accelerator-Cache (DAX)?

Globale Tabellen umgehen DAX, indem sie DynamoDB direkt aktualisieren. DAX ist daher nicht bewusst, dass er veraltete Daten enthält. Der DAX-Cache wird erst aktualisiert, wenn die TTL des Caches abläuft.

Werden Tags auf Tabellen weitergegeben?

Nein, Tags werden nicht automatisch weitergegeben.

Sollte ich Tabellen in allen Regionen sichern oder nur in einer?

Die Antwort hängt davon ab, zu welchem Zweck Sie die Sicherung erstellen.

- Wenn Sie die Beständigkeit Ihrer Daten sicherstellen möchten, bietet DynamoDB diesen Schutz bereits. Der Service gewährleistet Datenbeständigkeit.

- Wenn Sie einen Snapshot für historische Aufzeichnungen aufbewahren möchten (beispielsweise um regulatorische Anforderungen zu erfüllen), sollte eine Sicherung in einer Region ausreichen. Sie können das Backup in weitere Regionen kopieren, indem Sie AWS Backup
- Wenn Sie irrtümlich gelöschte oder geänderte Daten wiederherstellen möchten, verwenden Sie DynamoDB point-in-time Recovery (PITR) in einer Region.

Wie stelle ich globale Tabellen bereit mit? CloudFormation

- CloudFormation stellt eine DynamoDB-Tabelle und eine globale Tabelle als zwei separate Ressourcen dar: `AWS::DynamoDB::Table` und. `AWS::DynamoDB::GlobalTable` Ein Ansatz besteht darin, alle Tabellen, die potenziell global sein können, mithilfe des `GlobalTable`-Konstrukts zu erstellen, um sie zunächst als eigenständige Tabellen beizubehalten und später, falls erforderlich, Regionen hinzuzufügen.
- CloudFormationIn wird jede globale Tabelle unabhängig von der Anzahl der Replikate von einem einzelnen Stack in einer einzigen Region gesteuert. Wenn Sie Ihre Vorlage bereitstellen, werden alle Replikate als Teil eines einzigen Stack-Vorgangs CloudFormation erstellt und aktualisiert. Sie sollten nicht dieselbe AWS::DynamoDB::GlobalTable-Ressource in mehreren Regionen bereitstellen. Dies führt zu Fehlern und wird nicht unterstützt. Wenn Sie Ihre Anwendungsvorlage in mehreren Regionen bereitstellen, können Sie Bedingungen verwenden, um die `AWS::DynamoDB::GlobalTable`-Ressource in einer einzigen Region zu erstellen. Alternativ können Sie Ihre `AWS::DynamoDB::GlobalTable`-Ressourcen in einem von Ihrem Anwendungs-Stack getrennten Stack definieren und sicherstellen, dass er in einer einzigen Region bereitgestellt wird.
- Wenn Sie eine reguläre Tabelle haben und diese in eine globale Tabelle konvertieren und gleichzeitig verwalten möchten, legen Sie die Löschrichtlinie auf fest`Retain`, entfernen Sie die Tabelle aus dem Stapel, konvertieren Sie die Tabelle in eine globale Tabelle in der Konsole und importieren Sie dann die globale Tabelle als neue Ressource in den Stack. CloudFormation Weitere Informationen finden Sie im AWS GitHub Repository.
- Die kontoübergreifende Replikation wird zu diesem Zeitpunkt nicht unterstützt.

Fazit und Ressourcen

Globale DynamoDB-Tabellen haben nur sehr wenige Kontrollen, müssen aber dennoch sorgfältig geprüft werden. Sie müssen Ihren Schreibmodus, Ihr Weiterleitungsmodell und Ihre Evakuierungsprozesse festlegen. Sie müssen Ihre Anwendung in jeder Region instrumentieren und

bereit sein, Ihre Weiterleitung anzupassen oder eine Evakuierung durchzuführen, um die globale Integrität zu erhalten. Der Vorteil ist, dass Sie auf diese Weise einen global verteilten Datensatz mit Lese- und Schreibvorgängen mit niedriger Latenz erhalten, der auf eine Verfügbarkeit von 99,999 % ausgelegt ist.

Weitere Informationen zu globalen DynamoDB-Tabellen finden Sie in den folgenden Ressourcen:

- DynamoDB-Dokumentation
- Amazon Application Recovery Controller
- Bereitschaftsprüfung in ARC (AWS Dokumentation)
- Route-53-Routing-Richtlinien
- AWS Globaler Beschleuniger
- DynamoDB – Service Level Agreement
- AWS Grundlagen für mehrere Regionen (Whitepaper)AWS
- Entwurfsmuster für Datenresilienz mit AWS(Präsentation re:Invent 2022)AWS
- Wie Fidelity Investments und Reltio mit Amazon DynamoDB modernisiert wurden (AWS Präsentation re:Invent 2022)
- Entwurfsmuster und bewährte Verfahren für mehrere Regionen (Präsentation re:Invent 2022)AWS
- Disaster Recovery (DR) -Architektur aktiviert AWS, Teil III: Pilot Light und Warm Standby (Blogbeitrag)AWS
- Verwenden Sie Regions-Pinning, um eine Heimatregion für Elemente in einer globalen Amazon DynamoDB-Tabelle festzulegen (Blogbeitrag)AWS
- Überwachung von Amazon DynamoDB im Hinblick auf betriebliches Bewusstsein (AWS Blogbeitrag)
- Skalierung von DynamoDB: Wie sich Partitionen, Hotkeys und Splits für die Wärmeentwicklung auf die Leistung auswirken (AWS Blogbeitrag)
- Starke regionsübergreifende Konsistenz mit globalen DynamoDB-Tabellen (AWS re:Invent 2024-Präsentation)

Bewährte Methoden für die Verwaltung der Steuerebene in DynamoDB

> **Note**
>
> DynamoDB führt für die Steuerebene ein Drosselungslimit von 2 500 Anfragen pro Sekunde ein, mit der Option für einen erneuten Versuch. Weitere Informationen finden Sie weiter unten.

Mit Operationen auf der DynamoDB-Steuerebene können Sie DynamoDB-Tabellen und -Objekte verwalten, die von Tabellen wie Indizes abhängig sind. Weitere Informationen zu diesen Operationen finden Sie unter Steuerebene.

Unter bestimmten Umständen müssen Sie möglicherweise Maßnahmen ergreifen und Daten, die von Aufrufen auf der Steuerebene zurückgegeben werden, als Teil Ihrer Geschäftslogik verwenden. Beispielsweise müssen Sie möglicherweise den von `ProvisionedThroughput` zurückgegeben Wert von `DescribeTable` kennen. Befolgen Sie unter diesen Umständen die folgenden bewährten Methoden:

- Fragen Sie die DynamoDB-Steuerebene nicht übermäßig ab.
- Kombinieren Sie die Aufrufe der Steuerebene und die Aufrufe der Datenebene nicht innerhalb desselben Codes.
- Behandeln Sie Drosselungen bei Anfragen auf der Steuerebene und versuchen Sie es erneut mit einem Backoff.
- Rufen Sie Änderungen an einer bestimmten Ressource von einem einzigen Client aus auf und verfolgen Sie sie.
- Anstatt Daten für dieselbe Tabelle mehrmals in kurzen Intervallen abzurufen, sollten Sie die Daten zur Verarbeitung zwischenspeichern.

Bewährte Methoden für die Verwendung von Massendatenoperationen in DynamoDB

DynamoDB unterstützt Massenoperationen wie `BatchWriteItem`, mit denen Sie bis zu 25 `PutItem`- und `DeleteItem`-Anforderungen zusammen ausführen können. `BatchWriteItem`

unterstützt jedoch keine `UpdateItem`-Operationen. Bei Massenaktualisierungen liegt der Unterschied in den Anforderungen und der Art der Aktualisierung. Sie können andere DynamoDB verwenden, APIs z. B. `TransactWriteItems` für Batchgrößen von bis zu 100. Wenn mehr Elemente betroffen sind, können Sie Dienste wie AWS Glue Amazon EMR AWS Step Functions oder benutzerdefinierte Skripts und Tools wie DynamoDB-Shell für Massenaktualisierungen verwenden.

Themen

- Bedingte Massenaktualisierung
- Effiziente Massenoperationen

Bedingte Massenaktualisierung

DynamoDB unterstützt Massenoperationen wie `BatchWriteItem`, mit denen Sie bis zu 25 `PutItem`- und `DeleteItem`-Anforderungen in einem einzigen Batch ausführen können. `BatchWriteItem` unterstützt jedoch keine `UpdateItem`-Operationen und Bedingungsausdrücke. Um dieses Problem zu umgehen, können Sie andere DynamoDB verwenden, APIs z. B. `TransactWriteItems` für Batchgrößen von bis zu 100.

Wenn mehr Elemente betroffen sind und ein großer Teil der Daten geändert werden muss, können Sie Dienste wie Amazon EMR AWS Step Functions oder benutzerdefinierte Skripts und Tools wie DynamoDB-Shell für effiziente Massenaktualisierungen verwenden. AWS Glue

Wann dieses Muster verwendet werden sollte

- DynamoDB-Shell ist kein Anwendungsfall, der für die Produktion unterstützt wird.
- `TransactWriteItems` – bis zu 100 einzelne Updates mit oder ohne Bedingungen, die als ACID nach dem „alles oder nichts“-Prinzip gebündelt ausgeführt werden. `TransactWriteItems`-Aufrufe können auch mit einem `ClientRequestToken` versehen werden, falls Ihre Anwendung Idempotenz benötigt, was bedeutet, dass mehrere identische Aufrufe dieselbe Wirkung haben wie ein einziger Aufruf. Dadurch wird sichergestellt, dass Sie dieselbe Transaktion nicht mehrmals ausführen und am Ende einen falschen Datenstatus erhalten.

  Kompromiss — Es wird zusätzlicher Durchsatz verbraucht. 2 WCUs pro 1 KB Schreibvorgang anstelle der standardmäßigen 1 WGU pro 1 KB Schreibvorgang.
- PartiQL `BatchExecuteStatement` – bis zu 25 Updates mit oder ohne Bedingungen. `BatchExecuteStatement` gibt immer eine erfolgreiche Antwort auf die gesamte Anforderung

sowie eine Liste der Antworten auf einzelne Operationen zurück, wobei die Reihenfolge gewahrt bleibt.

Kompromiss – Bei größeren Batches ist zusätzliche Client-seitige Logik erforderlich, um Anforderungen in Batches von 25 zu verteilen. Bei der Festlegung der Wiederholungsstrategie müssen individuelle Fehlerantworten berücksichtigt werden.

Codebeispiele

Diese Codebeispiele verwenden die boto3-Bibliothek, das AWS SDK für Python. In diesen Beispielen wird vorausgesetzt, dass Sie boto3 installiert und mit den entsprechenden AWS -Anmeldeinformationen konfiguriert haben.

Gehen wir von einer Inventardatenbank für einen Elektrogerätehändler aus, der über mehrere Lager in europäischen Städten verfügt. Da es Ende des Sommers ist, möchte der Verkäufer die Tischventilatoren ausräumen, um Platz für andere Lagerbestände zu schaffen. Der Verkäufer möchte einen Preisnachlass für alle Tischventilatoren gewähren, die aus Lagern in Italien geliefert werden, aber nur, wenn ein Vorratsbestand von 20 Tischventilatoren vorhanden ist. Die DynamoDB-Tabelle heißt inventory. Sie hat ein Schlüsselschema mit Partitionsschlüssel sku, einer eindeutigen Kennung für jedes Produkt, und einen Sortierschlüssel warehouse, der eine Kennung für ein Lager ist.

Der folgende Python-Code zeigt, wie diese bedingte Massenaktualisierung mithilfe eines `BatchExecuteStatement`-API-Aufrufs durchgeführt wird.

```
import boto3

client=boto3.client("dynamodb")

before_image=client.query(TableName='inventory', KeyConditionExpression='sku=:pk_val
 AND begins_with(warehouse, :sk_val)', ExpressionAttributeValues={':pk_val':
{'S':'F123'},':sk_val':{'S':'WIT'}},
 ProjectionExpression='sku,warehouse,quantity,price')
print("Before update: ", before_image['Items'])

response=client.batch_execute_statement(
        Statements=[
            {'Statement': 'UPDATE inventory SET price=price-5 WHERE sku=? AND
 warehouse=? AND quantity > 20', 'Parameters': [{'S':'F123'}, {'S':'WITTUR1'}],
 'ReturnValuesOnConditionCheckFailure': 'ALL_OLD'},
```

```
            {'Statement': 'UPDATE inventory SET price=price-5 WHERE sku=? AND
 warehouse=? AND quantity > 20', 'Parameters': [{'S':'F123'}, {'S':'WITROM1'}],
 'ReturnValuesOnConditionCheckFailure': 'ALL_OLD'},
            {'Statement': 'UPDATE inventory SET price=price-5 WHERE sku=? AND
 warehouse=? AND quantity > 20', 'Parameters': [{'S':'F123'}, {'S':'WITROM2'}],
 'ReturnValuesOnConditionCheckFailure': 'ALL_OLD'},
            {'Statement': 'UPDATE inventory SET price=price-5 WHERE sku=? AND
 warehouse=? AND quantity > 20', 'Parameters': [{'S':'F123'}, {'S':'WITROM5'}],
 'ReturnValuesOnConditionCheckFailure': 'ALL_OLD'},
            {'Statement': 'UPDATE inventory SET price=price-5 WHERE sku=? AND
 warehouse=? AND quantity > 20', 'Parameters': [{'S':'F123'}, {'S':'WITVEN1'}],
 'ReturnValuesOnConditionCheckFailure': 'ALL_OLD'},
            {'Statement': 'UPDATE inventory SET price=price-5 WHERE sku=? AND
 warehouse=? AND quantity > 20', 'Parameters': [{'S':'F123'}, {'S':'WITVEN2'}],
 'ReturnValuesOnConditionCheckFailure': 'ALL_OLD'},
            {'Statement': 'UPDATE inventory SET price=price-5 WHERE sku=? AND
 warehouse=? AND quantity > 20', 'Parameters': [{'S':'F123'}, {'S':'WITVEN3'}],
 'ReturnValuesOnConditionCheckFailure': 'ALL_OLD'},
        ],
        ReturnConsumedCapacity='TOTAL'
    )

after_image=client.query(TableName='inventory', KeyConditionExpression='sku=:pk_val
 AND begins_with(warehouse, :sk_val)', ExpressionAttributeValues={':pk_val':
{'S':'F123'},':sk_val':{'S':'WIT'}},
 ProjectionExpression='sku,warehouse,quantity,price')
print("After update: ", after_image['Items'])
```

Die Ausführung erzeugt die folgende Ausgabe für Beispieldaten:

```
Before update:  [{'quantity': {'N': '20'}, 'warehouse': {'S': 'WITROM1'}, 'price':
 {'N': '40'}, 'sku': {'S': 'F123'}}, {'quantity': {'N': '25'}, 'warehouse': {'S':
 'WITROM2'}, 'price': {'N': '40'}, 'sku': {'S': 'F123'}}, {'quantity': {'N':
 '28'}, 'warehouse': {'S': 'WITROM5'}, 'price': {'N': '38'}, 'sku': {'S': 'F123'}},
 {'quantity': {'N': '26'}, 'warehouse': {'S': 'WITTUR1'}, 'price': {'N': '40'}, 'sku':
 {'S': 'F123'}}, {'quantity': {'N': '10'}, 'warehouse': {'S': 'WITVEN1'}, 'price':
 {'N': '38'}, 'sku': {'S': 'F123'}}, {'quantity': {'N': '20'}, 'warehouse': {'S':
 'WITVEN2'}, 'price': {'N': '38'}, 'sku': {'S': 'F123'}}, {'quantity': {'N': '50'},
 'warehouse': {'S': 'WITVEN3'}, 'price': {'N': '35'}, 'sku': {'S': 'F123'}}]
After update:  [{'quantity': {'N': '20'}, 'warehouse': {'S': 'WITROM1'}, 'price': {'N':
 '40'}, 'sku': {'S': 'F123'}}, {'quantity': {'N': '25'}, 'warehouse': {'S': 'WITROM2'},
 'price': {'N': '35'}, 'sku': {'S': 'F123'}}, {'quantity': {'N': '28'}, 'warehouse':
```

```
 {'S': 'WITROM5'}, 'price': {'N': '33'}, 'sku': {'S': 'F123'}}, {'quantity': {'N':
 '26'}, 'warehouse': {'S': 'WITTUR1'}, 'price': {'N': '35'}, 'sku': {'S': 'F123'}},
 {'quantity': {'N': '10'}, 'warehouse': {'S': 'WITVEN1'}, 'price': {'N': '38'}, 'sku':
 {'S': 'F123'}}, {'quantity': {'N': '20'}, 'warehouse': {'S': 'WITVEN2'}, 'price':
 {'N': '38'}, 'sku': {'S': 'F123'}}, {'quantity': {'N': '50'}, 'warehouse': {'S':
 'WITVEN3'}, 'price': {'N': '30'}, 'sku': {'S': 'F123'}}]
```

Da es sich um eine begrenzte Operation für ein internes System handelt, wurden die Anforderungen an die Idempotenz nicht berücksichtigt. Es ist möglich, einen zusätzlichen Integritätsschutz festzulegen, z. B. dass Preisaktualisierungen nur dann durchgeführt werden sollten, wenn der Preis höher als 35 und weniger als 40 ist, damit die Aktualisierungen robuster sind.

Alternativ können wir dieselbe Massenaktualisierung auch mit `TransactWriteItems` durchführen, falls strengere Anforderungen an Idempotenz und ACID gelten. Es ist jedoch wichtig, daran zu denken, dass entweder alle Operationen im Transaktionspaket ausgeführt werden oder das gesamte Paket fehlschlägt.

Nehmen wir an, dann aufgrund einer Hitzewelle in Italien die Nachfrage nach Tischventilatoren stark gestiegen ist. Der Verkäufer möchte seine Kosten für Schreibtischventilatoren beim Versand aus jedem Lager in Italien um 20 Euro erhöhen, aber die Aufsichtsbehörde erlaubt diese Kostenerhöhung nur, wenn die aktuellen Kosten für den gesamten Lagerbestand unter 70 Euro liegen. Es ist wichtig, dass der Preis im gesamten Inventar nur ein einziges Mal aktualisiert wird und nur dann, wenn die Kosten in jedem Lager weniger als 70 Euro betragen.

Der folgende Python-Code zeigt, wie diese Massenaktualisierung mithilfe eines `TransactWriteItems`-API-Aufrufs durchgeführt wird.

```
import boto3

client=boto3.client("dynamodb")

before_image=client.query(TableName='inventory', KeyConditionExpression='sku=:pk_val
 AND begins_with(warehouse, :sk_val)', ExpressionAttributeValues={':pk_val':
{'S':'F123'},':sk_val':{'S':'WIT'}},
 ProjectionExpression='sku,warehouse,quantity,price')
print("Before update: ", before_image['Items'])

response=client.transact_write_items(
        ClientRequestToken='UUIDAWS124',
```

```
        TransactItems=[
            {'Update': { 'Key': {'sku': {'S':'F123'}, 'warehouse': {'S':'WITTUR1'}},
  'UpdateExpression': 'SET price = price + :inc', 'ConditionExpression': 'price < :cap',
  'ExpressionAttributeValues': { ':inc': {'N': '20'}, ':cap': {'N': '70'}}, 'TableName':
  'inventory', 'ReturnValuesOnConditionCheckFailure': 'ALL_OLD'}},
            {'Update': { 'Key': {'sku': {'S':'F123'}, 'warehouse': {'S':'WITROM1'}},
  'UpdateExpression': 'SET price = price + :inc', 'ConditionExpression': 'price < :cap',
  'ExpressionAttributeValues': { ':inc': {'N': '20'}, ':cap': {'N': '70'}}, 'TableName':
  'inventory', 'ReturnValuesOnConditionCheckFailure': 'ALL_OLD'}},
            {'Update': { 'Key': {'sku': {'S':'F123'}, 'warehouse': {'S':'WITROM2'}},
  'UpdateExpression': 'SET price = price + :inc', 'ConditionExpression': 'price < :cap',
  'ExpressionAttributeValues': { ':inc': {'N': '20'}, ':cap': {'N': '70'}}, 'TableName':
  'inventory', 'ReturnValuesOnConditionCheckFailure': 'ALL_OLD'}},
            {'Update': { 'Key': {'sku': {'S':'F123'}, 'warehouse': {'S':'WITROM5'}},
  'UpdateExpression': 'SET price = price + :inc', 'ConditionExpression': 'price < :cap',
  'ExpressionAttributeValues': { ':inc': {'N': '20'}, ':cap': {'N': '70'}}, 'TableName':
  'inventory', 'ReturnValuesOnConditionCheckFailure': 'ALL_OLD'}},
            {'Update': { 'Key': {'sku': {'S':'F123'}, 'warehouse': {'S':'WITVEN1'}},
  'UpdateExpression': 'SET price = price + :inc', 'ConditionExpression': 'price < :cap',
  'ExpressionAttributeValues': { ':inc': {'N': '20'}, ':cap': {'N': '70'}}, 'TableName':
  'inventory', 'ReturnValuesOnConditionCheckFailure': 'ALL_OLD'}},
            {'Update': { 'Key': {'sku': {'S':'F123'}, 'warehouse': {'S':'WITVEN2'}},
  'UpdateExpression': 'SET price = price + :inc', 'ConditionExpression': 'price < :cap',
  'ExpressionAttributeValues': { ':inc': {'N': '20'}, ':cap': {'N': '70'}}, 'TableName':
  'inventory', 'ReturnValuesOnConditionCheckFailure': 'ALL_OLD'}},
            {'Update': { 'Key': {'sku': {'S':'F123'}, 'warehouse': {'S':'WITVEN3'}},
  'UpdateExpression': 'SET price = price + :inc', 'ConditionExpression': 'price < :cap',
  'ExpressionAttributeValues': { ':inc': {'N': '20'}, ':cap': {'N': '70'}}, 'TableName':
  'inventory', 'ReturnValuesOnConditionCheckFailure': 'ALL_OLD'}},
        ],
        ReturnConsumedCapacity='TOTAL'
    )

after_image=client.query(TableName='inventory', KeyConditionExpression='sku=:pk_val
 AND begins_with(warehouse, :sk_val)', ExpressionAttributeValues={':pk_val':
{'S':'F123'},':sk_val':{'S':'WIT'}},
 ProjectionExpression='sku,warehouse,quantity,price')
print("After update: ", after_image['Items'])
```

Die Ausführung erzeugt die folgende Ausgabe für Beispieldaten:

```
Before update:  [{'quantity': {'N': '20'}, 'warehouse': {'S': 'WITROM1'}, 'price':
 {'N': '60'}, 'sku': {'S': 'F123'}}, {'quantity': {'N': '25'}, 'warehouse': {'S':
 'WITROM2'}, 'price': {'N': '55'}, 'sku': {'S': 'F123'}}, {'quantity': {'N':
 '28'}, 'warehouse': {'S': 'WITROM5'}, 'price': {'N': '53'}, 'sku': {'S': 'F123'}},
 {'quantity': {'N': '26'}, 'warehouse': {'S': 'WITTUR1'}, 'price': {'N': '55'}, 'sku':
 {'S': 'F123'}}, {'quantity': {'N': '10'}, 'warehouse': {'S': 'WITVEN1'}, 'price':
 {'N': '58'}, 'sku': {'S': 'F123'}}, {'quantity': {'N': '20'}, 'warehouse': {'S':
 'WITVEN2'}, 'price': {'N': '58'}, 'sku': {'S': 'F123'}}, {'quantity': {'N': '50'},
 'warehouse': {'S': 'WITVEN3'}, 'price': {'N': '50'}, 'sku': {'S': 'F123'}}]
After update:  [{'quantity': {'N': '20'}, 'warehouse': {'S': 'WITROM1'}, 'price': {'N':
 '80'}, 'sku': {'S': 'F123'}}, {'quantity': {'N': '25'}, 'warehouse': {'S': 'WITROM2'},
 'price': {'N': '75'}, 'sku': {'S': 'F123'}}, {'quantity': {'N': '28'}, 'warehouse':
 {'S': 'WITROM5'}, 'price': {'N': '73'}, 'sku': {'S': 'F123'}}, {'quantity': {'N':
 '26'}, 'warehouse': {'S': 'WITTUR1'}, 'price': {'N': '75'}, 'sku': {'S': 'F123'}},
 {'quantity': {'N': '10'}, 'warehouse': {'S': 'WITVEN1'}, 'price': {'N': '78'}, 'sku':
 {'S': 'F123'}}, {'quantity': {'N': '20'}, 'warehouse': {'S': 'WITVEN2'}, 'price':
 {'N': '78'}, 'sku': {'S': 'F123'}}, {'quantity': {'N': '50'}, 'warehouse': {'S':
 'WITVEN3'}, 'price': {'N': '70'}, 'sku': {'S': 'F123'}}]
```

Es gibt mehrere Ansätze für die Durchführung von Massenaktualisierungen in DynamoDB. Der geeignete Ansatz hängt von Faktoren wie den Anforderungen an die and/or ACID-Idempotenz, der Anzahl der zu aktualisierenden Elemente und der Vertrautheit mit ihnen ab. APIs

Effiziente Massenoperationen

Wann dieses Muster verwendet werden sollte

Diese Muster sind nützlich, um Massenaktualisierungen für DynamoDB-Elemente effizient durchzuführen.

- DynamoDB-Shell ist kein Anwendungsfall, der für die Produktion unterstützt wird.
- `TransactWriteItems` – bis zu 100 einzelne Updates mit oder ohne Bedingungen, die als Alles-oder-Nichts-ACID-Paket ausgeführt werden.

  Kompromiss — Es wird zusätzlicher Durchsatz verbraucht, 2 WCUs pro 1 KB Schreibvorgang.
- PartiQL `BatchExecuteStatement` – bis zu 25 Updates mit oder ohne Bedingungen.

  Kompromiss – Es ist eine zusätzliche Logik erforderlich, um Anforderungen in Batches von 25 zu verteilen.
- AWS Step Functions — Massenoperationen mit begrenzter Geschwindigkeit für Entwickler, die damit vertraut sind. AWS Lambda

Kompromiss – Die Ausführungszeit ist umgekehrt proportional zur Ratenbegrenzung. Limitiert durch das maximale Timeout der Lambda-Funktion. Die Funktionalität beinhaltet, dass Datenänderungen, die zwischen dem Lesen und dem Schreiben auftreten, überschrieben werden können. Weitere Informationen finden Sie unter Backfilling an Amazon DynamoDB Time to Live attribute using Amazon EMR: Part 2.

- AWS Glue und Amazon EMR — ratenbegrenzter Massenbetrieb mit verwalteter Parallelität. Bei Anwendungen oder Updates, die nicht zeitkritisch sind, können diese Optionen im Hintergrund ausgeführt werden und verbrauchen dann nur einen kleinen Prozentsatz des Durchsatzes. Beide Dienste verwenden die emr-dynamodb-connector, um DynamoDB-Operationen auszuführen. Diese Services führen einen umfangreichen Lesevorgang aus, gefolgt von einem umfangreichen Schreibvorgang bei aktualisierten Elementen mit einer Option zur Ratenbegrenzung.

  Kompromiss – Die Ausführungszeit ist umgekehrt proportional zur Ratenbegrenzung. Die Funktionalität beinhaltet, dass Datenänderungen, die zwischen dem Lesen und dem Schreiben auftreten, überschrieben werden können. Sie können nicht aus globalen sekundären Indizes () lesen. GSIs Weitere Informationen finden Sie unter Backfilling an Amazon DynamoDB Time to Live attribute using Amazon EMR: Part 2.
- DynamoDB-Shell – Massenoperationen mit Ratenbegrenzung und SQL-ähnlichen Abfragen. Sie können aus Effizienzgründen GSIs daraus lesen.

  Kompromiss – Die Ausführungszeit ist umgekehrt proportional zur Ratenbegrenzung. Weitere Informationen finden Sie unter Rate limited bulk operations in DynamoDB Shell.

Verwenden des Musters

Massenaktualisierungen können erhebliche Auswirkungen auf die Kosten haben, insbesondere wenn Sie den On-Demand-Durchsatzmodus verwenden. Es gibt einen Kompromiss zwischen Geschwindigkeit und Kosten, wenn Sie den Modus mit bereitgestelltem Durchsatz verwenden. Ein sehr strikt festgelegter rate-limit-Parameter kann zu einer sehr langen Verarbeitungszeit führen. Sie können die Geschwindigkeit der Aktualisierung anhand der durchschnittlichen Elementgröße und der Ratenbegrenzung grob bestimmen.

Alternativ können Sie anhand der erwarteten Dauer des Aktualisierungsvorgangs und der durchschnittlichen Elementgröße den für den Vorgang erforderlichen Durchsatz ermitteln. Die zu den einzelnen Mustern gehörenden Blog-Referenzen enthalten Einzelheiten zur Strategie, der Implementierung und den Einschränkungen bei der Verwendung des Musters. Weitere Informationen finden Sie unter Cost-effective bulk processing with Amazon DynamoDB.

Es gibt mehrere Möglichkeiten, Massenaktualisierungen für eine dynamische DynamoDB-Tabelle durchzuführen. Welcher Ansatz am besten geeignet ist, hängt von Faktoren wie den Anforderungen an die and/or ACID-Idempotenz, der Anzahl der zu aktualisierenden Punkte und der Vertrautheit ab. APIs Es ist wichtig, den Kompromiss zwischen Kosten und Zeit zu berücksichtigen. Die meisten der oben erläuterten Ansätze bieten eine Option zur Ratenbegrenzung des Durchsatzes, der von der Massenaktualisierung verwendet wird.

Bewährte Methoden für den Umgang mit gleichzeitigen Updates in DynamoDB

In verteilten Systemen können mehrere Prozesse oder Benutzer versuchen, dieselben Daten gleichzeitig zu ändern. Ohne Kontrolle der Parallelität können diese gleichzeitigen Schreibvorgänge dazu führen, dass Aktualisierungen verloren gehen, Daten inkonsistent sind oder es zu Wettläufen kommt. DynamoDB bietet mehrere Mechanismen, mit denen Sie den gleichzeitigen Zugriff verwalten und die Datenintegrität aufrechterhalten können.

> **Note**
>
> Einzelne Schreibvorgänge `UpdateItem` sind z. B. atomar und werden unabhängig von der Parallelität immer mit der neuesten Version des Elements ausgeführt. Sperrstrategien sind erforderlich, wenn Ihre Anwendung ein Element lesen und es dann basierend auf dem gelesenen Wert zurückschreiben muss (ein read-modify-write Zyklus), da ein anderer Prozess das Element zwischen dem Lesen und dem Schreiben ändern könnte.

Es gibt zwei Hauptstrategien für den Umgang mit gleichzeitigen Aktualisierungen:

- Optimistisches Sperren — Geht davon aus, dass Konflikte selten sind. Es ermöglicht gleichzeitigen Zugriff und erkennt Konflikte beim Schreiben mithilfe von bedingten Schreibvorgängen. Wenn ein Konflikt erkannt wird, schlägt der Schreibvorgang fehl und die Anwendung kann es erneut versuchen.
- Pessimistisches Sperren — Geht davon aus, dass Konflikte wahrscheinlich sind. Es verhindert den gleichzeitigen Zugriff, indem es sich exklusiven Zugriff auf eine Ressource verschafft, bevor sie geändert wird. Andere Prozesse müssen warten, bis die Sperre aufgehoben wird.

In der folgenden Tabelle sind die in DynamoDB verfügbaren Ansätze zusammengefasst:

Ansatz	Mechanismus	Am besten geeignet für
Optimistische Sperre	Versionsattribut + bedingte Schreibvorgänge	Geringer Streitwert, kostengünstige Wiederholungsversuche
Pessimistisches Sperren (Transaktionen)	`TransactWriteItems`	Atomarität mehrerer Gegenstände, mäßiger Streit
Pessimistisches Sperren (Lockclient)	Dedizierte Sperrtabelle mit Lease und Heartbeat	Langfristige Workflows, verteilte Koordination

Optimistische Sperre mit Versionsnummer

Optimistisches Sperren ist eine Strategie, die Konflikte beim Schreiben erkennt, anstatt sie zu verhindern. Jedes Element enthält ein Versionsattribut, das mit jeder Aktualisierung erhöht wird. Wenn Sie ein Element aktualisieren, fügen Sie einen Bedingungsausdruck hinzu, der überprüft, ob die Versionsnummer mit dem Wert übereinstimmt, den Ihre Anwendung zuletzt gelesen hat. Wenn das Element in der Zwischenzeit von einem anderen Prozess geändert wurde, schlägt die Bedingung fehl und DynamoDB gibt a zurück. `ConditionalCheckFailedException`

Wann sollte optimistisches Sperren verwendet werden

Optimistisches Sperren eignet sich gut, wenn:

- Möglicherweise aktualisieren mehrere Benutzer oder Prozesse dasselbe Element, Konflikte treten jedoch selten auf.
- Ein erneuter Versuch, einen fehlgeschlagenen Schreibvorgang zu wiederholen, ist für Ihre Anwendung kostengünstig.
- Sie möchten den Aufwand und die Komplexität der Verwaltung verteilter Sperren vermeiden.

Zu den häufigsten Beispielen gehören Inventaraktualisierungen im E-Commerce, Plattformen für die gemeinsame Bearbeitung und Aufzeichnungen über Finanztransaktionen.

Kompromisse

Mehraufwand bei hohem Datenkonflikt erneut versuchen

In Umgebungen mit hoher Parallelität steigt die Wahrscheinlichkeit von Konflikten, was zu mehr Wiederholungen und höheren Schreibkosten führen kann

Komplexität der Implementierung

Das Hinzufügen von Versionskontrollen zu Elementen und die Handhabung bedingter Prüfungen erhöhen die Komplexität der Anwendungslogik. Das AWS SDK for Java v2 Enhanced Client bietet integrierten Support durch die `@DynamoDbVersionAttribute`Anmerkung, die Versionsnummern automatisch für Sie verwaltet.

Musterdesign

Fügen Sie jedem Element ein Versionsattribut hinzu. Hier ist ein einfacher Schemaentwurf:

- Partitionsschlüssel — Ein eindeutiger Bezeichner für jedes Element (z. B.`ItemId`).
- Attribute:
  - `ItemId` – der eindeutige Bezeichner für das Element
  - `Version` – eine Ganzzahl, die die Versionsnummer des Elements darstellt
  - `QuantityLeft` – der verbleibende Bestand des Elements

Wenn ein Element zum ersten Mal erstellt wird, wird das Attribut `Version` auf 1 gesetzt. Mit jeder Aktualisierung wird die Versionsnummer um 1 erhöht.

ItemID (Partitionsschlüssel)	Version	QuantityLeft
Bananen	1	10
Äpfel	1	5
Orangen	1	7

Implementierung

Gehen Sie wie folgt vor, um optimistisches Sperren zu implementieren:

1. Die aktuelle Version des Elements.

```
def get_item(item_id):
    response = table.get_item(Key={'ItemID': item_id})
    return response['Item']

item = get_item('Bananas')
current_version = item['Version']
```

2. Aktualisieren Sie das Element mithilfe eines Bedingungsausdrucks, der die Versionsnummer überprüft.

```
def update_item(item_id, qty_bought, current_version):
    try:
        response = table.update_item(
            Key={'ItemID': item_id},
            UpdateExpression="SET QuantityLeft = QuantityLeft - :qty, Version
 = :new_v",
            ConditionExpression="Version = :expected_v",
            ExpressionAttributeValues={
                ':qty': qty_bought,
                ':new_v': current_version + 1,
                ':expected_v': current_version
            },
            ReturnValues="UPDATED_NEW"
        )
        return response
    except ClientError as e:
        if e.response['Error']['Code'] == 'ConditionalCheckFailedException':
            print("Version conflict: another process updated this item.")
        raise
```

3. Behandeln Sie Konflikte, indem Sie es erneut mit einem neuen Lesevorgang versuchen.

   Für jeden erneuten Versuch ist ein zusätzlicher Lesevorgang erforderlich. Beschränken Sie daher die Gesamtzahl der Wiederholungen.

```
def update_with_retry(item_id, qty_bought, max_retries=3):
    for attempt in range(max_retries):
        item = get_item(item_id)
        try:
            return update_item(item_id, qty_bought, item['Version'])
        except ClientError as e:
```

```
            if e.response['Error']['Code'] != 'ConditionalCheckFailedException':
                raise
            print(f"Retry {attempt + 1}/{max_retries}")
    raise Exception("Update failed after maximum retries.")
```

Für Java-Anwendungen bietet das AWS SDK for Java v2 Enhanced Client integrierte Unterstützung für optimistisches Sperren über die `@DynamoDbVersionAttribute`Anmerkung, die Versionsnummern automatisch für Sie verwaltet.

Weitere Hinweise zu Bedingungsausdrücken finden Sie unterCLI-Beispiel für DynamoDB-Bedingungsausdrücke.

Pessimistisches Sperren mit DynamoDB-Transaktionen

DynamoDB-Transaktionen bieten einen all-or-nothing Ansatz für gruppierte Operationen. Wenn Sie verwenden`TransactWriteItems`, überwacht DynamoDB alle Elemente in der Transaktion. Wenn ein Element während der Transaktion durch einen anderen Vorgang geändert wird, wird die gesamte Transaktion storniert und DynamoDB gibt a zurück. `TransactionCanceledException` Dieses Verhalten bietet eine Form der pessimistischen Parallelitätskontrolle, da widersprüchliche gleichzeitige Änderungen verhindert und nicht erst im Nachhinein erkannt werden.

Wann sollten Transaktionen zum Sperren verwendet werden

Transaktionen eignen sich gut, wenn:

- Sie müssen mehrere Elemente atomar aktualisieren, entweder innerhalb derselben Tabelle oder tabellenübergreifend.
- Ihre Geschäftslogik erfordert all-or-nothing Semantik — entweder sind alle Änderungen erfolgreich oder es werden keine angewendet.

Zu den häufigsten Beispielen gehören der Transfer von Geld zwischen Konten, das Aufgeben von Bestellungen, mit denen sowohl das Inventar als auch die Bestelltabelle aktualisiert werden, und der Austausch von Gegenständen zwischen Spielern in einem Spiel.

Kompromisse

Höhere Schreibkosten

Bei Elementen mit einer Größe von bis zu 1 KB verbrauchen Transaktionen 2 WCUs pro Element (eine für die Vorbereitung, eine für die Übertragung), verglichen mit 1 WCU für einen Standardschreibvorgang.

Limit für Artikel

Eine einzelne Transaktion kann bis zu 100 Aktionen in einer oder mehreren Tabellen umfassen.

Konfliktsensitivität

Wenn ein Element in der Transaktion durch einen anderen Vorgang geändert wird, schlägt die gesamte Transaktion fehl. In Szenarien mit hohem Streitaufkommen kann dies zu häufigen Stornierungen führen.

Implementierung

Das folgende Beispiel verwendet die atomare Übertragung `TransactWriteItems` von Inventar zwischen zwei Artikeln. Wenn ein anderer Prozess einen der Artikel während der Transaktion ändert, wird der gesamte Vorgang rückgängig gemacht.

```
import boto3

client = boto3.client('dynamodb')

def transfer_inventory(source_id, target_id, quantity):
    try:
        client.transact_write_items(
            TransactItems=[
                {
                    'Update': {
                        'TableName': 'Inventory',
                        'Key': {'ItemID': {'S': source_id}},
                        'UpdateExpression': 'SET QuantityLeft = QuantityLeft - :qty',
                        'ConditionExpression': 'QuantityLeft >= :qty',
                        'ExpressionAttributeValues': {
                            ':qty': {'N': str(quantity)}
                        }
                    }
```

```
                },
                {
                    'Update': {
                        'TableName': 'Inventory',
                        'Key': {'ItemID': {'S': target_id}},
                        'UpdateExpression': 'SET QuantityLeft = QuantityLeft + :qty',
                        'ExpressionAttributeValues': {
                            ':qty': {'N': str(quantity)}
                        }
                    }
                }
            ]
        )
        return True
    except client.exceptions.TransactionCanceledException as e:
        print(f"Transaction canceled: {e}")
        return False
```

In diesem Beispiel überprüft der Bedingungsausdruck, ob ausreichend Inventar vorhanden ist, aber kein Versionsattribut erforderlich ist. DynamoDB bricht die Transaktion automatisch ab, wenn ein Element in der Transaktion durch einen anderen Vorgang zwischen den Vorbereitungs- und Übertragungsphasen geändert wird. Dies ermöglicht die pessimistische Parallelitätskontrolle — widersprüchliche gleichzeitige Änderungen werden durch die Transaktion selbst verhindert.

> **Note**
>
> Sie können Transaktionen mit optimistischem Sperren kombinieren, indem Sie Versionsprüfungen als zusätzliche Bedingungsausdrücke hinzufügen. Dies bietet eine zusätzliche Schutzebene, ist jedoch nicht erforderlich, damit die Transaktion Konflikte erkennt.

Weitere Informationen finden Sie unter Verwalten komplexer Workflows mit DynamoDB-Transaktionen.

Verteiltes Sperren mit dem DynamoDB Lock Client

Für Anwendungen, die traditionelle lock-acquire-release Semantik erfordern, ist der DynamoDB Lock Client eine Open-Source-Bibliothek, die verteiltes Sperren mithilfe einer DynamoDB-Tabelle als Sperrspeicher implementiert. Dieser Ansatz ist nützlich, wenn Sie den Zugriff auf eine externe

Ressource (z. B. ein S3-Objekt oder eine gemeinsam genutzte Konfiguration) über mehrere Anwendungsinstanzen hinweg koordinieren müssen.

Der Lock-Client ist als Open-Source-Java-Bibliothek verfügbar.

Funktionsweise

Der Lock-Client verwendet eine dedizierte DynamoDB-Tabelle, um Sperren zu verfolgen. Jede Sperre wird als ein Element mit den folgenden Schlüsselattributen dargestellt:

- Ein Partitionsschlüssel, der die gesperrte Ressource identifiziert.
- Eine Leasingdauer, die angibt, wie lange die Sperre gültig ist. Wenn der Sperrenhalter abstürzt oder nicht mehr reagiert, läuft die Sperre nach Ablauf der Leasingdauer automatisch ab.
- Ein Taktsignal, das der Schlossinhaber regelmäßig sendet, um den Leasingvertrag zu verlängern. Dadurch wird verhindert, dass die Sperre abläuft, während der Inhaber die Bearbeitung noch aktiv durchführt.

Der Lock-Client verwendet bedingte Schreibvorgänge, um sicherzustellen, dass jeweils nur ein Prozess eine Sperre erhalten kann. Wenn eine Sperre bereits aktiviert ist, kann der Aufrufer wählen, ob er warten und es erneut versuchen möchte oder ob er sofort fehlschlagen möchte.

Wann sollte der Lock-Client verwendet werden

Der Lock-Client eignet sich gut, wenn:

- Sie müssen den Zugriff auf eine gemeinsam genutzte Ressource über mehrere Anwendungsinstanzen oder Microservices hinweg koordinieren.
- Der kritische Abschnitt dauert lange (Sekunden bis Minuten), und im Konfliktfall wäre ein erneuter Versuch, den gesamten Vorgang zu wiederholen, teuer.
- Sie benötigen ein automatisches Ablaufen der Sperren, um Prozessfehler ordnungsgemäß behandeln zu können.

Zu den häufigsten Beispielen gehören die Orchestrierung verteilter Workflows, die Koordination von Cron-Jobs über mehrere Instanzen hinweg und die Verwaltung des Zugriffs auf gemeinsam genutzte externe Ressourcen.

Kompromisse

Zusätzliche Infrastruktur

Erfordert eine dedizierte DynamoDB-Tabelle für die Sperrverwaltung mit zusätzlicher Lese- und Schreibkapazität für Sperroperationen und Heartbeats.

Abhängigkeit von der Uhr

Das Ablaufen von Sperren hängt von Zeitstempeln ab. Ein erheblicher Zeitversatz zwischen den Clients kann zu unerwartetem Verhalten führen, insbesondere bei kurzen Leasingdauern.

Deadlock-Risiko

Wenn Ihre Anwendung Sperren für mehrere Ressourcen erwirbt, müssen Sie diese in einer konsistenten Reihenfolge aktivieren, um Deadlocks zu vermeiden. Die Leasingdauer bietet ein Sicherheitsnetz, da Sperren von Inhabern, die nicht darauf reagieren, automatisch aufgehoben werden.

Implementierung

Das folgende Beispiel zeigt, wie der DynamoDB Lock Client verwendet wird, um eine Sperre zu erwerben und aufzuheben:

```
import java.io.IOException;
import java.util.Optional;
import java.util.concurrent.TimeUnit;
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;

final DynamoDbClient dynamoDB = DynamoDbClient.builder()
    .region(Region.US_WEST_2)
    .build();

final AmazonDynamoDBLockClient lockClient = new AmazonDynamoDBLockClient(
    AmazonDynamoDBLockClientOptions.builder(dynamoDB, "Locks")
        .withTimeUnit(TimeUnit.SECONDS)
        .withLeaseDuration(10L)
        .withHeartbeatPeriod(3L)
        .withCreateHeartbeatBackgroundThread(true)
        .build());

// Try to acquire a lock on a resource
final Optional<LockItem> lock =
```

```
    lockClient.tryAcquireLock(AcquireLockOptions.builder("my-shared-
resource").build());

if (lock.isPresent()) {
    try {
        // Perform operations that require exclusive access
        processSharedResource();
    } finally {
        // Always release the lock when done
        lockClient.releaseLock(lock.get());
    }
} else {
    System.out.println("Failed to acquire lock.");
}

lockClient.close();
```

> **Important**
>
> Geben Sie immer Sperren in einem `finally` Block auf, um sicherzustellen, dass Sperren auch dann aufgehoben werden, wenn Ihre Verarbeitungslogik eine Ausnahme auslöst. Nicht freigegebene Sperren blockieren andere Prozesse, bis das Leasing abläuft.

Sie können auch einen einfachen Sperrmechanismus ohne die Lock-Client-Bibliothek implementieren, indem Sie bedingte Schreibvorgänge direkt verwenden. Im folgenden Beispiel wird `UpdateItem` mit einem Bedingungsausdruck eine Sperre `DeleteItem` abgerufen und aufgehoben:

```
from datetime import datetime, timedelta
from boto3.dynamodb.conditions import Attr

def acquire_lock(table, resource_name, owner_id, ttl_seconds):
    """Attempt to acquire a lock. Returns True if successful."""
    expiry = (datetime.now() + timedelta(seconds=ttl_seconds)).isoformat()
    now = datetime.now().isoformat()
    try:
        table.update_item(
            Key={'LockID': resource_name},
            UpdateExpression='SET #owner = :owner, #expiry = :expiry',
            ConditionExpression=Attr('LockID').not_exists() |
 Attr('ExpiresAt').lt(now),
            ExpressionAttributeNames={'#owner': 'OwnerID', '#expiry': 'ExpiresAt'},
```

```
            ExpressionAttributeValues={':owner': owner_id, ':expiry': expiry}
        )
        return True
    except table.meta.client.exceptions.ConditionalCheckFailedException:
        return False

def release_lock(table, resource_name, owner_id):
    """Release a lock. Only succeeds if the caller is the lock owner."""
    try:
        table.delete_item(
            Key={'LockID': resource_name},
            ConditionExpression=Attr('OwnerID').eq(owner_id)
        )
        return True
    except table.meta.client.exceptions.ConditionalCheckFailedException:
        return False
```

Dieser Ansatz verwendet einen Bedingungsausdruck, um sicherzustellen, dass eine Sperre nur dann erworben werden kann, wenn sie nicht existiert oder abgelaufen ist, und dass sie nur von dem Prozess freigegeben werden kann, der sie erworben hat. Erwägen Sie, Time to Live (TTL) in der Sperrtabelle zu aktivieren, um abgelaufene Sperrobjekte automatisch zu bereinigen.

Wählen Sie eine Strategie zur Kontrolle der Parallelität

Verwenden Sie die folgenden Richtlinien, um den richtigen Ansatz für Ihren Workload zu wählen:

Verwenden Sie optimistisches Sperren in folgenden Fällen:

- Konflikte treten selten auf.
- Der Versuch, einen fehlgeschlagenen Schreibvorgang zu wiederholen, ist kostengünstig.
- Sie aktualisieren jeweils ein einzelnes Element.

Verwenden Sie Transaktionen, wenn:

- Sie müssen mehrere Elemente atomar aktualisieren.
- Sie benötigen all-or-nothing Semantik für alle Elemente oder Tabellen.
- Sie müssen Bedingungsprüfungen mit Schreibvorgängen in einem einzigen Vorgang kombinieren.

Verwenden Sie den Lock-Client, wenn:

- Sie müssen den Zugriff auf externe Ressourcen über verteilte Prozesse hinweg koordinieren.
- Der kritische Abschnitt ist langwierig und es ist kostspielig, bei Konflikten erneut zu versuchen.

- Sie benötigen ein automatisches Ablaufen der Sperren, um Prozessfehler zu beheben.

> **Note**
>
> Wenn Sie globale DynamoDB-Tabellen verwenden, beachten Sie, dass globale Tabellen für gleichzeitige Aktualisierungen eine Abstimmungsstrategie verwenden, bei der der letzte Writer gewinnt. Optimistisches Sperren mit Versionsnummern funktioniert regionsübergreifend nicht wie erwartet, da ein Schreibvorgang in einer Region einen gleichzeitigen Schreibvorgang in einer anderen Region ohne Versionsprüfung überschreiben kann. Entwerfen Sie Ihre Anwendung so, dass sie Konflikte auf Anwendungsebene behandelt, wenn Sie globale Tabellen verwenden.

Bewährte Methoden zum Verständnis Ihrer AWS Abrechnungs- und Nutzungsberichte in DynamoDB

In diesem Dokument werden die `UsageType`-Abrechnungscodes für Gebühren im Zusammenhang mit DynamoDB erläutert.

AWS stellt Kosten- und Nutzungsberichte (CUR) bereit, die Daten für die in Anspruch genommenen Dienste enthalten. Sie können AWS Cost and Usage Report es verwenden, um Abrechnungsberichte in Amazon S3 im CSV-Format zu veröffentlichen. Bei der Einrichtung des CUR können Sie wählen, ob Sie Zeiträume nach Stunden, Tagen oder Monaten einteilen und ob Sie die Nutzung nach Ressourcen-ID aufschlüsseln möchten. Weitere Informationen zum Generieren eines CUR finden Sie unter Erstellen von Kosten- und Nutzungsberichten.

Im CSV-Export finden Sie die relevanten Attribute, die für jede Zeile aufgelistet sind. Im Folgenden finden Sie Beispiele für Attribute, die enthalten sein können:

- lineitem/UsageStartDate: Das Startdatum und die Startzeit für den Einzelposten in UTC, einschließlich.
- lineitem/UsageEndDate: Das Enddatum und die Endzeit für den entsprechenden Zeileneintrag in UTC, ausschließlich.
- lineitem/ProductCode: Für DynamoDB ist das „DB“ AmazonDynamo
- lineitem/UsageType: Ein spezifischer Beschreibungscode für die Art der Nutzung, wie in diesem Dokument aufgeführt

- lineitem/Operation: ein Name, der den Kontext für die Gebühr bereitstellt, z. B. der Name der Operation, durch die die Gebühr entstanden ist (optional)
- lineitem/ResourceId: Der Bezeichner für die Ressource, die die Nutzung verursacht hat. verfügbar, wenn der CUR eine Aufschlüsselung nach Ressourcen-ID enthält
- lineitem/UsageAmount: Die Menge der Nutzung, die während des angegebenen Zeitraums angefallen ist.
- lineitem/UnblendedCost: Die Kosten für diese Nutzung.
- lineitem/LineItemDescription: Textbeschreibung des Einzelpostens.

Weitere Informationen zum CUR-Datenwörterbuch finden Sie unter Kosten- und Nutzungsbericht (CUR) 2.0. Beachten Sie, dass die genauen Namen je nach Kontext variieren.

`UsageType` ist eine Zeichenfolge mit einem Wert wie `ReadCapacityUnit-Hrs`, `USW2-ReadRequestUnits`, `EU-WriteCapacityUnit-Hrs` oder `USE1-TimedPITRStorage-ByteHrs`. Jeder Nutzungstyp beginnt mit einem optionalen Regionspräfix. Falls nicht vorhanden, weist dies auf die Region us-east-1 hin. Falls vorhanden, können Sie die Zuordnung des kurzen Regionscode für die Fakturierung zum herkömmlichen Regionscode und -namen der folgenden Tabelle entnehmen.

Die Verwendung namens `USW2-ReadRequestUnits` weist beispielsweise auf in der Region us-west-2 verbrauchte Leseanforderungseinheiten hin.

Regionscode für die Fakturierung	Regionscode	Name der Region
AFS1	af-south-1	Afrika (Kapstadt)
APE1	ap-east-1	Asien-Pazifik (Hongkong)
APN1	ap-northeast-1	Asien-Pazifik (Tokio)
APN2	ap-northeast-2	Asien-Pazifik (Seoul)
APN3	ap-northeast-3	Asien-Pazifik (Osaka)
APS1	ap-southeast-1	Asien-Pazifik (Singapur)
APS2	ap-southeast-2	Asien-Pazifik (Sydney)

Regionscode für die Fakturierung	Regionscode	Name der Region
APS3	ap-south-1	Asien-Pazifik (Mumbai)
APS4	ap-southeast-3	Asien-Pazifik (Jakarta)
APS5	ap-south-2	Asien-Pazifik (Hyderabad)
APS6	ap-southeast-4	Asien-Pazifik (Melbourne)
CAN1	ca-central-1	Kanada (Zentral)
EU	eu-west-1	Europa (Irland)
EUC1	eu-central-1	Europa (Frankfurt)
EUC2	eu-central-2	Europa (Zürich)
EUN1	eu-north-1	Europa (Stockholm)
EUS1	eu-south-1	Europa (Milan)
EUS2	eu-south-2	Europa (Spain)
EUW1	eu-west-1	Europa (Irland)
EUW2	eu-west-2	Europa (London)
EUW3	eu-west-3	Europa (Paris)
ILC1	Il-central-1	Israel (Tel Aviv)
MEC1	me-central-1	Naher Osten (VAE)
MES1	me-south-1	Middle East (Bahrain)
SAE1	sa-east-1	Südamerika (São Paulo)
USE1 (Standard)	us-east-1	USA Ost (Nord-Virginia)
USE2	us-east-2	USA Ost (Ohio)

Regionscode für die Fakturierung	Regionscode	Name der Region
UGE1	us-gov-east-1	US-Regierung Ost
UGW1	us-gov-west-1	US-Regierung West
USW1	us-west-1	USA West (Nordkalifornien)
USW2	us-west-2	USA West (Oregon)

In den folgenden Abschnitten verwenden wir das `REG-UsageType`-Muster, wenn wir die Gebühren für DynamoDB durchgehen, wobei REG die Region angibt, in der die Nutzung stattgefunden hat, und usageType der Code für die Art der Gebühr ist. Wenn Sie beispielsweise einen Einzelposten für `USW1- ReadCapacityUnit-Hrs` in Ihrer CSV-Datei sehen, bedeutet das, dass die Nutzung der bereitgestellten Lesekapazität in US-West-1 erfolgt ist. In diesem Fall würde die Liste `REG-ReadCapacityUnit-Hrs` enthalten.

Themen

- Durchsatzkapazität
- Streams
- Speicher
- Backup und Backup
- Datenübertragung
- CloudWatch Einblicke der Mitwirkenden
- DynamoDB Accelerator (DAX)

Durchsatzkapazität

Lese- und Schreibvorgänge mit bereitgestellter Kapazität

Wenn Sie eine DynamoDB-Tabelle im Modus bereitgestellter Kapazität erstellen, geben Sie die Lese- und Schreibkapazität an, die Ihre Anwendung voraussichtlich benötigt. Der Nutzungstyp hängt von Ihrer Tabellenklasse ab (Standard oder Standard-Infrequent Access). Sie stellen Lese- und Schreibvorgänge auf der Grundlage der Nutzungsrate pro Sekunde bereit, die Gebühren werden jedoch pro Stunde basierend auf der bereitgestellten Kapazität berechnet.

UsageType	Einheiten	Granularity	Description
REG- ReadCapacityUnit -Stunden	RCU-hours	Stunde	Gebühren für Lesevorgänge im Modus bereitgestellter Kapazität unter Verwendung der Tabellenklasse Standard
REG-IA-ReadCapacityUnit-Hrs	RCU-hours	Stunde	Gebühren für Lesevorgänge im Modus bereitgestellter Kapazität unter Verwendung der Tabellenklasse Standard-IA
REG- -Stunden WriteCapacityUnit	WCU-hours	Stunde	Gebühren für Schreibvorgänge im Modus bereitgestellter Kapazität unter Verwendung der Tabellenklasse Standard
REG-IA-WriteCapacityUnit-Hrs	WCU-hours	Stunde	Gebühren für Schreibvorgänge im Modus bereitgestellter Kapazität unter Verwendung der Tabellenklasse Standard-IA

Lese- und Schreibvorgänge mit reservierter Kapazität

Mit reservierter Kapazität bezahlen Sie im Vorfeld eine einmalige Gebühr und verpflichten sich zu einer voraussichtlichen Mindestnutzung während eines bestimmten Zeitraums. Reservierte Kapazität wird zu einem vergünstigten Stundensatz abgerechnet. Kapazitäten, die Sie über ihre reservierten Kapazitäten hinaus nutzen, werden zu Standardpreisen für bereitgestellte Kapazitäten verrechnet. Reservierte Kapazität ist für bereitgestellte Lese- und Schreibkapazitätseinheiten (RCU und WCU) mit einer einzelnen Region in DynamoDB-Tabellen verfügbar, die die Standard-Tabellenklasse verwenden. Sowohl reservierte Kapazitäten für 1 Jahr als auch für 3 Jahre werden auf dieselbe Weise abgerechnet. SKUs

UsageType	Einheiten	Granularity	Description
REG-:dynamodb.read HeavyUsage	RCU-hours	Im Voraus, dann monatlich	Gebühren für Lesevorgänge mit reservierter Kapazität : eine einmalige Vorauszahlung und eine monatliche Gebühr zu Beginn jedes Monats, die alle vergünstigten zugesagten RCU-Stunden während des Monats abdeckt, Wird passende Nullkosten-Einzelp osten haben. REG-ReadCapacityUnit-Hrs
REG--: dynamodb. write HeavyUsage	WCU-hours	Im Voraus, dann monatlich	Gebühren für Schreibvorgänge mit reservierter Kapazität : eine einmalige Vorauszahlung und eine monatliche Gebühr zu Beginn jedes Monats, die alle vergünstigten

UsageType	Einheiten	Granularity	Description
			zugesagten WCU-Stunden während des Monats abdeckt, Wird passende Nullkosten-Einzelposten haben. REG-WriteCapacityUnit-Hrs

Lese- und Schreibvorgänge mit On-Demand-Kapazität

Wenn Sie eine DynamoDB-Tabelle im On-Demand-Kapazitätsmodus erstellen, zahlen Sie nur für die Lese- und Schreibvorgänge, die Ihre Anwendung ausführt. Die Preise für Lese- und Schreibanforderungen hängen von Ihrer Tabellenklasse ab.

UsageType	Einheiten	Granularity	Description
REG- ReadRequestUnits	RRUs	Einheit	Gebühren für Lesevorgänge im On-Demand-Kapazitätsmodus mit Tabellenklasse Standard
REG-IA- ReadRequestUnits	RRUs	Einheit	Gebühren für Lesevorgänge im On-Demand-Kapazitätsmodus mit Tabellenklasse Standard-IA
REG- WriteRequestUnits	WRUs	Einheit	Gebühren für Schreibvorgänge im On-Demand-Kapazitätsmodus

UsageType	Einheiten	Granularity	Description
			mit Tabellenklasse Standard
REG-IA- WriteRequestUnits	WRUs	Einheit	Gebühren für Schreibvorgänge im On-Demand-Kapazitätsmodus mit Tabellenklasse Standard-IA

Lese- und Schreibvorgänge in globalen Tabellen

DynamoDB berechnet Gebühren für die Nutzung globaler Tabellen auf der Grundlage der in jeder Replikattabelle verwendeten Ressourcen. Bei bereitgestellten globalen Tabellen werden Schreibanforderungen für globale Tabellen in replizierten WCUs (rWCU) statt in Standardtabellen gemessen, WCUs und Schreibvorgänge in globale Sekundärindizes in globalen Tabellen werden in Form von repliziert gemessen. WCUs Bei globalen On-Demand-Tabellen werden Schreibanforderungen in replizierten WRUs (rwRU) statt in Standardwerten gemessen. WRUs Die Anzahl von r WCUs oder r, die für die WRUs Replikation verwendet werden, hängt von der Version der globalen Tabellen ab, die Sie verwenden. Die Preisgestaltung hängt von Ihrer Tabellenklasse ab.

Schreibvorgänge in globale Sekundärindizes (GSIs) werden mit Standard-Schreibeinheiten (WCUs und WRUs) abgerechnet. Leseanforderungen und Datenspeicher werden genauso abgerechnet wie Tabellen mit einer einzelnen Region.

Wenn Sie ein Tabellenreplikat hinzufügen, um eine globale Tabelle in neuen Regionen zu erstellen oder zu erweitern, berechnet DynamoDB eine Tabellenwiederherstellung in den hinzugefügten Regionen pro Gigabyte an wiederhergestellten Daten. Wiederhergestellte Daten werden wie folgt berechnet. REG-RestoreDataSize-Bytes Einzelheiten finden Sie unter Backup und Wiederherstellung für DynamoDB. Für die regionsübergreifende Replikation und das Hinzufügen von Replikaten zu Tabellen, die Daten enthalten, fallen außerdem Gebühren für die ausgehende Datenübertragung an.

Wenn Sie den On-Demand-Kapazitätsmodus für Ihre globalen DynamoDB-Tabellen wählen, zahlen Sie nur für die Ressourcen, die Ihre Anwendung für jede Replikattabelle verwendet.

UsageType	Einheiten	Granularity	Description
REG- ReplWrite CapacityUnit - Stunden	rWCU-hours	Stunde	Globale Tabelle, bereitgestellt, Tabellenklasse Standard
REG-IA- -Stunden ReplWriteCapacityUnit	rWCU-hours	Stunde	Globale Tabelle, bereitgestellt, Tabellenklasse Standard-IA
REG- ReplWrite RequestUnits	rWRU	Einheit	Globale Tabelle, On-Demand, Tabellenklasse Standard
REG-IA- ReplWrite RequestUnits	rWRU	Einheit	Globale Tabelle, On-Demand, Tabellenklasse Standard-IA

Streams

DynamoDB verfügt über zwei Streaming-Technologien, DynamoDB Streams und Kinesis. Für beide gibt es separate Preise.

DynamoDB Streams berechnet Gebühren für das Lesen von Daten in Leseanforderungseinheiten. Jeder `GetRecords`-API-Aufruf wird als Streams-Leseanforderung abgerechnet. Für `GetRecords` API-Aufrufe, die im Rahmen von DynamoDB-Triggern oder von globalen DynamoDB-Tabellen im AWS Lambda Rahmen der Replikation aufgerufen werden, fallen keine Gebühren an.

UsageType	Einheiten	Granularity	Description
REG-Streams-RequestsCount	Anzahl	Einheit	Leseanforderungseinheiten für DynamoDB Streams

Amazon Kinesis Data Streams erhebt Gebühren in Änderungsdatenerfassungseinheiten. DynamoDB berechnet für jeden Schreibvorgang (bis zu 1 KB) eine Änderungsdatenerfassungseinheit. Für Schreibvorgänge, die größer als 1 KB sind, sind zusätzliche Änderungsdatenerfassungseinheiten erforderlich. Sie zahlen nur für die Schreibvorgänge, die Ihre Anwendung ausführt, ohne die Durchsatzkapazität der Tabelle verwalten zu müssen.

UsageType	Einheiten	Granularity	Description
REG- ChangeDataCaptureUnits - Kinesis	CDC-Einheiten	Einheit	Änderungsdatenerfassungseinheiten für Kinesis Data Streams

Speicher

DynamoDB misst die Größe Ihrer fakturierbaren Daten, indem es die Roh-Byte-Größe Ihrer Daten mit einem Speicher-Overhead pro Element addiert, der von den Funktionen abhängt, die Sie aktiviert haben.

> **Note**
>
> Die Werte für die Speichernutzung im CUR sind im Vergleich zu den Speicherwerten bei Verwendung von `DescribeTable` höher, da `DescribeTable` den Speicher-Overhead pro Element nicht berücksichtigt.

Der Speicher wird stündlich berechnet, aber monatlich abgerechnet, wobei der Durchschnitt der Stundengebühren zugrunde gelegt wird.

Obwohl der `ByteHrs`-Speicher `UsageType` als Suffix verwendet, wird die Speichernutzung im CUR in GB gemessen und in GB/Monat berechnet.

UsageType	Einheiten	Granularity	Description
REG- TimedStorage - ByteHrs	GB	Monat	Speichermenge, die von Ihren DynamoDB-Tabellen und -Indizes

UsageType	Einheiten	Granularity	Description
			für Tabellen mit der Tabellenklasse Standard verwendet wird
REG-IA- - TimedStorage ByteHrs	GB	Monat	Speichermenge, die von Ihren DynamoDB-Tabellen und -Indizes für Tabellen mit der Tabellenklasse Standard-IA verwendet wird

Backup und Backup

DynamoDB bietet zwei Arten von Backups: Backups mit zeitpunktbezogener Wiederherstellung (PITR) und On-Demand-Backups. Benutzer können Daten aus diesen Backups auch in DynamoDB-Tabellen wiederherstellen. Die folgenden Gebühren beziehen sich sowohl auf Backups als auch auf Wiederherstellungen.

Gebühren für Backup-Speicher fallen am ersten Tag des Monats an. Anpassungen werden im Laufe des Monats vorgenommen, wenn Backups hinzugefügt oder entfernt werden. Weitere Informationen finden Sie im Blog Understanding Amazon DynamoDB On-Demand-Backups and Billing.

UsageType	Einheiten	Granularity	Description
REG- TimedBackupStorage - ByteHrs	GB	Monat	Der Speicherplatz, der von On-Demand-Backups Ihrer DynamoDB-Tabellen und lokalen sekundären Indizes verbraucht wird

UsageType	Einheiten	Granularity	Description
Zeitgesteuert PITRStorage - ByteHrs	GB	Monat	Der von point-in-time Wiederherstellungs -Backups (PITR) verwendete Speicher. wobei DynamoDB den ganzen Monat über kontinuierlich die Größe Ihrer PITR-fähigen Tabellen überwacht, um Ihre Backup-Ge bühren und Speicherr echnungen zu ermitteln, solange PITR aktiviert ist
REG- -Bytes RestoreDataSize	GB	Größe	Die Gesamtgröße der wiederhergestellte n Daten (einschli eßlich Tabellendaten, lokaler sekundärer Indizes und globaler sekundärer Indizes), gemessen in GB aus DynamoDB-Backups

AWS Backup

AWS Backup ist ein vollständig verwalteter Backup-Service, der es einfach macht, die Sicherung von Daten zwischen AWS Diensten in der Cloud und vor Ort zu zentralisieren und zu automatisieren. AWS Backup wird für die Speicherung (warme oder kalte Lagerung), Wiederherstellungsaktivitäten und die regionsübergreifende Datenübertragung in Rechnung gestellt. Die folgenden `UsageType` Gebühren werden unter „AWS Backup“ und ProductCode nicht unter „AmazonDynamoDB“ aufgeführt.

UsageType	Einheiten	Granularity	Description
REG- WarmStorage -ByteHrs -DynamoDB	GB	Monat	Der Speicher, der von DynamoDB-Backups verwendet wird, die von AWS Backup im Laufe des Monats verwaltet werden, gemessen in GB pro Monat
REG- CrossRegion - WarmBytes - DynamoDB	GB	Größe	Die Daten wurden entweder innerhalb desselben Kontos oder auf ein anderes Konto in eine andere AWS Region übertragen. AWS Gebühren für regionsübergreifende Übertragungen fallen beim Kopieren von Backups von einer Region in eine andere Region an und werden immer dem Konto in Rechnung gestellt, von dem die Daten übertragen werden
REG-Wiederherstellung- -DynamoDB WarmBytes	GB	Größe	Die Gesamtgröße der aus dem Warmspeicher wiederhergestellten Daten, gemessen in GB

UsageType	Einheiten	Granularity	Description
REG- ColdStorage -ByteHrs -DynamoDB	GB	Monat	Der Kaltspeicher, der von DynamoDB-Backups verwendet wird, die von AWS Backup im Laufe des Monats verwaltet werden, gemessen in GB pro Monat
REG-Wiederherstellung- -DynamoDB ColdBytes	GB	Monat	Die Gesamtgröße der aus dem Kaltspeicher wiederhergestellten Daten, gemessen in GB

Export und Import

Sie können Daten aus DynamoDB nach Amazon S3 exportieren oder Daten aus Amazon S3 in eine neue DynamoDB-Tabelle importieren.

Obwohl `UsageType` `Bytes` als Suffix verwendet, wird die Export- und Importnutzung im CUR in GB gemessen und berechnet.

UsageType	Einheiten	Granularity	Description
REG- ExportDataSize -Bytes	GB	Größe	Die Gebühr für das Exportieren von Daten nach S3, wobei DynamoDB diese für Daten auf der Grundlage der Größe der DynamoDB-Basistabelle (Tabellen daten und lokale sekundäre Indizes)

UsageType	Einheiten	Granularity	Description
			zu dem Zeitpunkt berechnet, an dem der Export erstellt wurde
REG- -Bytes ImportDataSize	GB	Größe	Die Gebühr für das Importieren von Daten aus S3, wobei die Größe auf der Grundlage der unkomprimierten Objektgröße der Daten in Amazon S3 berechnet wird Für den Import in Tabellen mit GSIs fallen keine zusätzlic hen Gebühren an.
REG- Increment alExportDataSize - Bytes	GB	Größe	Die Gebühr für die Größe der Daten, die beim kontinuierlichen Backup verarbeitet wurden, um inkrement elle Exporte zu erstellen

Datenübertragung

Datenübertragungsaktivitäten können in Verknüpfung mit dem DynamoDB-Service angezeigt werden. DynamoDB erhebt keine Gebühren für eingehende Datenübertragungen und keine Gebühren für Datenübertragungen zwischen DynamoDB und anderen AWS Diensten innerhalb derselben AWS Region (mit anderen Worten, 0,00 USD pro GB). Daten, die zwischen AWS Regionen übertragen werden (z. B. zwischen DynamoDB in der Region USA Ost [Nord-Virginia] und Amazon EC2 in der Region EU [Irland]), werden auf beiden Seiten der Übertragung berechnet.

UsageType	Einheiten	Granularity	Description
REG-In DataTransfer Byte	GB	Einheiten	Datenübertragung aus dem Internet in DynamoDB
REG- DataTransfer - Out-Bytes	GB	Einheiten	Datenübertragung aus DynamoDB in das Internet

CloudWatch Einblicke der Mitwirkenden

CloudWatch Contributor Insights for DynamoDB ist ein Diagnosetool zur Identifizierung der am häufigsten aufgerufenen und gedrosselten Schlüssel in Ihrer DynamoDB-Tabelle. Die folgenden `UsageType` Gebühren werden unter „“ und nicht unter „DB“ AmazonCloudWatch aufgeführt. ProductCode AmazonDynamo

UsageType	Einheiten	Granularity	Description
REG-CW: Contributor EventsManaged	Ereignisse, die verarbeitet wurden	Einheiten	Die Anzahl der verarbeiteten DynamoDB-Ereignisse. Beispiel: In einer Tabelle, in der CloudWatch Contributor Insights aktiviert ist, wird jedes Mal, wenn ein Element gelesen oder geschrieben wird, es als ein Ereignis gezählt. Wenn die Tabelle über einen Sortierschlüssel verfügt, fallen Gebühren für zwei Ereignisse an.

UsageType	Einheiten	Granularity	Description
REG-CW: ContributorRulesManaged	Anzahl der Regeln	Monat	DynamoDB erstellt Regeln, um die am häufigsten aufgerufenen Elemente und die am häufigsten gedrosselten Schlüssel zu identifizieren, wenn Sie Cloud Watch Contributor Insights aktivieren. Diese Gebühr fällt für die Regeln an, die für jede Entität (Tabellen und GSIs) hinzugefügt wurden, die für die Protokollierung CloudWatch von Erkenntnissen von Mitwirkenden konfiguriert sind.

DynamoDB Accelerator (DAX)

DynamoDB Accelerator (DAX) wird auf der Grundlage des für den Service ausgewählten Instance-Typs stundenweise abgerechnet. Die folgenden Gebühren beziehen sich auf die bereitgestellten DynamoDB-Accelerator-Instances. Die folgenden `UsageType` Gebühren werden unter „AmazonDAX“ und ProductCode nicht unter „AmazonDynamoDB“ aufgeführt.

UsageType	Einheiten	Granularity	Description
REG-:dax-NodeUsage <INSTANCETYPE>	Knotenstunden	Stunde	Die stündliche Nutzung eines bestimmten Instance-Typs. Die Preise

UsageType	Einheiten	Granularity	Description
			verstehen sich pro verbrauchter Knotenstunde, von dem Zeitpunkt, an dem ein Knoten gestartet wird, bis zum Zeitpunkt, an dem er beendet wird. Jede einzelne verbrauchte Knotenstunde wird als volle Stunde abgerechnet. DAX-Gebühren für jeden Knoten in einem DAX-Cluster. Wenn Sie einen Cluster mit mehreren Knoten haben, würden Sie in Ihrem Fakturierungsbericht mehrere Einzelposten sehen.

Der Instance-Typ wird einer der Werte aus der folgenden Liste sein. Weitere Informationen zu Knotentypen finden Sie unter Knoten.

- r3.2xlarge, r4.8xlarge oder r5.8xlarge
- r3.4xlarge, r4.large oder r5.large
- r3.8xlarge, r4.xlarge oder r5.xlarge
- r3.2xlarge, r5.12xlarge oder t2.medium
- r3.4xlarge, r4.large oder r5.large
- r3.xlarge, r5.16xlarge oder t2.small
- r4.16xlarge, r5.24xlarge oder t3.medium

- r4.2xlarge, r5.2xlarge oder t3.small
- r4.4xlarge oder r5.4xlarge

Migrieren einer DynamoDB-Tabelle zwischen zwei Konten

Sie können eine Amazon DynamoDB-Tabelle von einem Konto zu einem anderen migrieren, um eine Strategie mit mehreren Konten oder eine Backup-Strategie zu implementieren. Sie können dies auch aus Test-, Debugging- oder Compliance-Gründen tun. Ein häufiger Anwendungsfall ist das Kopieren von DynamoDB-Tabellen in Produktions-, Staging-, Test- und Entwicklungsumgebungen, in denen jede Umgebung ein anderes Konto verwendet. AWS

DynamoDB bietet zwei Optionen für die Migration von Tabellen von einem AWS Konto zu einem anderen:

- AWS Backup für kontoübergreifendes Backup and Restore: AWS Backup ist ein vollständig verwalteter Backup-Service, mit dem Sie Backups für mehrere AWS Dienste zentral verwalten können. Mit der kontoübergreifenden Backup- und Wiederherstellungsfunktion können Sie eine DynamoDB-Tabelle in einem Konto sichern und das Backup in einem anderen Konto in derselben AWS -Organisation wiederherstellen.
- DynamoDB-Export und Import nach Amazon S3: Mithilfe der DynamoDB-Funktionen Export und Import in Amazon S3 können Sie einen vollständigen Export in einen Amazon S3 S3-Bucket durchführen und diese Daten dann in eine neue Tabelle in einem anderen Konto importieren. AWS Dieser Ansatz eignet sich, wenn Sie zwischen Konten migrieren müssen, die nicht Teil derselben AWS Organisation sind, oder wenn Sie diese nicht verwenden möchten. AWS Backup

> **Note**
>
> Der Import aus Amazon S3 unterstützt keine Tabellen mit lokalen Sekundärindizes (LSIs), aber globale Sekundärindizes (GSIs). Weitere Informationen zu LSIs und GSIs finden Sie unter. Verbessern des Datenzugriffs mit sekundären Indizes in DynamoDB

Themen

- Migrieren Sie eine Tabelle, die AWS Backup für kontoübergreifende Backups und Wiederherstellungen verwendet wird
- Migrieren einer Tabelle mithilfe des Exports nach S3 und des Imports aus S3

Migrieren Sie eine Tabelle, die AWS Backup für kontoübergreifende Backups und Wiederherstellungen verwendet wird

Voraussetzungen

- Quell- und AWS Zielkonten müssen derselben Organisation im AWS Organizations-Service angehören
- Gültige AWS Identity and Access Management (IAM) -Berechtigungen zum Erstellen und Verwenden AWS Backup von Tresoren

Weitere Informationen zum Einrichten kontenübergreifender Backups finden Sie unter Kontenübergreifende Sicherungskopien erstellen. AWS

Preisinformationen

AWS Gebühren für das Backup (basierend auf der Tabellengröße), das Kopieren von Daten zwischen AWS Regionen (basierend auf der Datenmenge), für die Wiederherstellung (basierend auf der Datenmenge) und für alle laufenden Speichergebühren. Um laufende Gebühren zu vermeiden, können Sie das Backup löschen, wenn Sie es nach der Wiederherstellung nicht benötigen.

Weitere Informationen zur Preisgestaltung finden Sie unter AWS Backup – Preise.

Schritt 1: Aktivieren erweiterter Funktionen für DynamoDB und kontoübergreifende Backups

1. Greifen Sie sowohl im Quell- als auch im AWS Zielkonto auf die AWS Management Console zu und öffnen Sie die AWS Backup-Konsole.
2. Wählen Sie die Option Einstellungen aus.
3. Vergewissern Sie sich unter Erweiterte Funktionen für Amazon DynamoDB-Backups, dass Erweiterte Funktionen aktiviert ist. Ist dies nicht der Fall, wählen Sie Aktivieren aus.
4. Wählen Sie unter Kontenübergreifende Verwaltung für Kontenübergreifendes Backup die Option Aktivieren aus.

Schritt 2: Erstellen eines Backup-Tresors im Quell- und Zielkonto

1. Öffnen Sie in den AWS Quellkonten die AWS Backup-Konsole.
2. Wählen Sie Backup vaults (Sicherungstresore) aus.

3. Wählen Sie Create backup vault (Sicherungstresor erstellen) aus.
4. Kopieren und speichern Sie den Amazon-Ressourcennamen (ARN) der erstellten Backup-Tresore und des AWS Zielkontos.
5. Sie benötigen sowohl den ARNs Quell- als auch den Ziel-Backup-Tresor, wenn Sie die DynamoDB-Tabellensicherung zwischen Konten kopieren möchten.

Schritt 3: Erstellen eines DynamoDB-Tabellen-Backups im Quellkonto

1. Wählen Sie auf der Seite AWS -Backup-Dashboard die Option On-Demand-Backup erstellen aus.
2. Wählen Sie im Abschnitt Einstellungen die Option DynamoDB als Ressourcentyp und dann den Tabellennamen aus.
3. Wählen Sie in der Dropdown-Liste Backup-Tresor den Backup-Tresor aus, den Sie im Quellkonto erstellt haben.
4. Wählen Sie den gewünschten Aufbewahrungszeitraum aus.
5. Wählen Sie On-Demand-Backup erstellen.
6. Überwachen Sie den Status des Backup-Auftrags auf der Registerkarte Backup-Aufträge der Seite AWS -Backup-Aufträge.

Schritt 4: Kopieren des DynamoDB-Tabellen-Backups vom Quellkonto zum Zielkonto

1. Öffnen Sie nach Abschluss des Backup-Jobs die AWS Backup Konsole im Quellkonto und wählen Sie Backup-Tresore aus.
2. Wählen Sie unter Backups das DynamoDB-Tabellen-Backup aus. Wählen Sie Aktionen und dann Kopieren aus.
3. Geben Sie die AWS Region des Zielkontos ein.
4. Geben Sie unter ARN des externen Tresors den ARN des Backup-Tresors ein, den Sie im Zielkonto erstellt haben.
5. Aktivieren Sie im Backup-Tresor des Zielkontos den Zugriff von einem Quellkonto, um das Kopieren von Backups zu ermöglichen.

Schritt 5: Wiederherstellen des DynamoDB-Tabellen-Backups im Zielkonto

1. Öffnen Sie im AWS Zielkonto die AWS Backup Konsole und wählen Sie Backup-Tresore

2. Wählen Sie unter Backups das Backup aus, das Sie aus dem Quellkonto kopiert haben. Wählen Sie Aktionen und dann Wiederherstellen aus.
3. Geben Sie den Namen für die neue DynamoDB-Tabelle, die Verschlüsselung, die diese neue Tabelle haben wird, den Schlüssel, mit dem die Wiederherstellung verschlüsselt werden soll, und andere Optionen ein.
4. Wenn die Wiederherstellung abgeschlossen ist, lautet der Tabellenstatus Aktiv.

Migrieren einer Tabelle mithilfe des Exports nach S3 und des Imports aus S3

Voraussetzungen

- Sie müssen Point-in-Time Recovery (PITR) für Ihre Tabelle aktivieren, um den Export nach S3 durchführen zu können. Weitere Informationen finden Sie unter point-in-timeWiederherstellung in DynamoDB aktivieren.
- Gültige IAM-Berechtigungen zur Durchführung des Exports. Weitere Informationen finden Sie unter Anfordern eines Tabellenexports in DynamoDB.
- Gültige IAM-Berechtigungen, die für die Durchführung des Imports ausreichen. Weitere Informationen finden Sie unter Anfordern eines Tabellenexports in DynamoDB.

Preisinformationen

AWS Gebühren für PITR (basierend auf der Größe der Tabelle und der Dauer, für die PITR aktiviert ist). Wenn Sie PITR außer für den Export nicht benötigen, können Sie die Option nach Abschluss des Exports deaktivieren. AWS erhebt außerdem Gebühren für Anforderungen an S3, für das Speichern der exportierten Daten in S3 und für den Import (basierend auf der unkomprimierten Größe der importierten Daten).

Weitere Informationen zu DynamoDB-Preisen finden Sie unter DynamoDB – Preise.

> **Note**
>
> Beim Import von S3 nach DynamoDB gibt es Beschränkungen für die Größe und Anzahl der Objekte. Weitere Informationen finden Sie unter Importkontingente.

Anfordern eines Tabellenexports nach Amazon S3

1. Melden Sie sich bei der AWS Management Console an und öffnen Sie die DynamoDB-Konsole.
2. Klicken Sie im Navigationsbereich links in der Konsole auf Exports to S3 (Exporte nach S3).
3. Wählen Sie eine Quelltabelle und den Ziel-S3-Bucket aus. Geben Sie andernfalls die URL des Zielkonto-Buckets ein und verwenden Sie dabei das `s3://bucketname/prefix`-Format. `/prefix` ist ein optionaler Ordner, um Ihren Ziel-Bucket organisiert zu halten.
4. Wählen Sie Vollständiger Export aus. Bei einem vollständigen Export wird der vollständige Tabellen-Snapshot Ihrer Tabelle zu dem von Ihnen angegebenen Zeitpunkt ausgegeben.
   a. Wählen Sie Aktuelle Zeit aus, um den letzten vollständigen Tabellen-Snapshot zu exportieren.
   b. Wählen Sie unter Exportiertes Dateiformat „DynamoDB JSON“ oder „Amazon Ion“ aus. Die Standardoption ist „DynamoDB JSON“.
5. Klicken Sie auf das Symbol Export (Exportieren), um mit dem Export zu beginnen.
6. Der Export kleiner Tabellen sollte in wenigen Minuten abgeschlossen sein, aber bei Tabellen im Terabyte-Bereich kann der Vorgang mehr als eine Stunde dauern.

Anfordern eines Tabellenimports aus Amazon S3

1. Melden Sie sich bei der AWS Management Console an und öffnen Sie die DynamoDB-Konsole.
2. Klicken Sie im Navigationsbereich links in der Konsole auf Import from S3 (Import aus S3).
3. Wählen Sie auf der angezeigten Seite Import from S3 (Import aus S3).
4. Geben Sie die Quell-URL aus Amazon S3 ein. Sie können diese auch über die Schaltfläche S3 durchsuchen finden. Das erwartete Format des Pfades ist `s3://bucket/prefix/AWSDynamoDB/<XXXXXXXX-XXXXXX>/data/`.
5. Geben Sie an, ob Sie der S3-Bucket-Besitzer sind.
6. Wählen Sie unter Dateikomprimierung importieren die Option GZIP aus, die dem Export entspricht.
7. Wählen Sie unter Dateiformat importieren die Option DynamoDB JSON aus, die dem Export entspricht.
8. Klicken Sie auf Weiter. Wählen Sie unter Tabellendetails angeben die Optionen für die neue Tabelle aus, die zum Speichern Ihrer Daten erstellt wird.

9. Klicken Sie auf Weiter. Passen Sie unter Tabelleneinstellungen konfigurieren gegebenenfalls zusätzliche Tabelleneinstellungen an.
10. Wählen Sie erneut Next (Weiter), um Ihre Importoptionen zu überprüfen, und klicken Sie dann auf Import (Importieren), um die Importaufgabe zu starten. Ihre neue Tabelle wird unter Importe aus S3 mit dem Status Wird importiert aufgeführt. Während dieser Zeit können Sie nicht auf die Tabelle zugreifen. Der Import kleiner Tabellen sollte in wenigen Minuten abgeschlossen sein, aber bei Tabellen im Terabyte-Bereich kann der Vorgang mehr als eine Stunde dauern.
11. Sobald der Import abgeschlossen ist, wird der Status Aktiv angezeigt und Sie können die Tabelle verwenden.

Synchronisieren von Tabellen während der Migration

Wenn Sie Schreibvorgänge in der Quelltabelle für die Dauer der Migration unterbrechen können, sollten die Quelle und Ausgabe nach der Migration exakt übereinstimmen. Wenn Sie Schreibvorgänge nicht unterbrechen können, liegt die Zieltabelle nach der Migration normalerweise etwas hinter der Quelltabelle zurück. Um im Vergleich zur Quelltabelle aufzuholen, können Sie mithilfe von Streaming (DynamoDB Streams oder Kinesis Data Streams für DynamoDB) die Schreibvorgänge wiedergeben, die seit dem Backup oder Export in der Quelltabelle stattgefunden haben.

Sie sollten mit dem Lesen der Stream-Datensätze vor dem Zeitstempel beginnen, an dem Sie die Quelltabelle nach S3 exportiert haben. Wenn der Export nach S3 beispielsweise um 14:00 Uhr erfolgte und der Import in die Zieltabelle um 23:00 Uhr abgeschlossen wurde, sollten Sie das Lesen des DynamoDB-Streams um 13:58 Uhr starten. In der Tabelle mit den Streaming-Optionen für die Erfassung von Änderungsdaten sind die Funktionen der einzelnen Streaming-Modelle zusammengefasst.

Die Verwendung von DynamoDB Streams mit Lambda bietet einen optimierten Ansatz für die Synchronisierung von Daten zwischen den DynamoDB-Quell- und -Zieltabellen. Sie können eine Lambda-Funktion verwenden, um jeden Schreibvorgang in der Zieltabelle erneut wiederzugeben.

> **Note**
>
> Die Elemente werden 24 Stunden lang in DynamoDB Streams aufbewahrt. Planen Sie deshalb, Ihr Backup und die Wiederherstellung oder den Export und Import innerhalb dieses Zeitfensters abzuschließen.

Vorgeschriebene Anleitung zur Integration von DAX in DynamoDB-Anwendungen

DynamoDB Accelerator (DAX) ist ein DynamoDB-kompatibler Caching-Service, der schnelle In-Memory-Leistung für anspruchsvolle Anwendungen wie leseintensive Anwendungen bietet. Mit DAX können Sie Reaktionszeiten in Mikrosekunden für den Zugriff auf häufig angeforderte Daten erreichen. Diese vorgeschriebene Anleitung für DynamoDB Accelerator bietet umfassende Einblicke und bewährte Methoden für die Integration von DAX in Ihre DynamoDB-Anwendungen.

Die Anleitung bietet grundlegendes Wissen für alle, die mit DAX noch nicht vertraut sind oder ihre bestehenden Konfigurationen optimieren möchten. In dieser Anleitung werden verschiedene Themen behandelt, z. B. wann DAX verwendet werden sollte und wie Sie einen DAX-Cluster erstellen. Sie enthält außerdem praktische Beispiele und ausführliche Erläuterungen, die Ihnen helfen, DAX effektiv in Ihren Projekten zu implementieren. Schließlich bietet diese Anleitung erweiterte Strategien, die Sie implementieren müssen, um die DAX-Caching-Funktionen optimal zu nutzen und so schnelle und skalierbare Anwendungen zu gewährleisten.

Themen

- Bewerten der Eignung von DAX für Ihre Anwendungsfälle
- Konfigurieren Ihres DAX-Clients
- Konfigurieren Ihres DAX-Clusters
- Dimensionieren Ihres DAX-Clusters
- Bereitstellen eines Clusters
- Verwalten von Cluster-Operationen
- Überwachen von DAX

Bewerten der Eignung von DAX für Ihre Anwendungsfälle

In diesem Abschnitt wird erklärt, wann und warum DAX verwendet werden sollte. Anhand dieser Anleitung können Sie feststellen, ob die Integration von DAX in DynamoDB für die Workload-Muster, Leistungsanforderungen und Datenkonsistenzanforderungen Ihrer Anwendung von Vorteil ist. Sie behandelt auch Szenarien, in denen DAX möglicherweise nicht geeignet ist, z. B. für schreibintensive Workloads und selten abgerufene Daten.

In diesem Abschnitt

- Wann und warum Sie sich für DAX entscheiden sollten
- Wann Sie DAX nicht verwenden sollten

Wann und warum Sie sich für DAX entscheiden sollten

Sie können in verschiedenen Szenarien erwägen, DAX zu Ihrem Anwendungs-Stack hinzuzufügen. Verwenden Sie DAX beispielsweise, um die Gesamtlatenz von Leseanforderungen an DynamoDB zu reduzieren oder um wiederholte Lesevorgänge derselben Daten aus einer Tabelle zu minimieren. Die folgende Liste enthält Beispiele für Szenarien, in denen Sie die Vorteile der Integration von DAX in DynamoDB nutzen können:

- Hohe Leistungsanforderungen
  - Lesevorgänge mit geringer Latenz – Sie sollten die Verwendung von DAX in Betracht ziehen, wenn Ihre Anwendung Reaktionszeiten in Mikrosekunden für letztendlich konsistente Lesevorgänge benötigt. DAX kann außerdem die Reaktionszeit für den Zugriff auf häufig gelesene Daten drastisch reduzieren.
- Leseintensive Workloads
  - Leseintensive Anwendungen — Bei Anwendungen mit einem hohen read-to-write Verhältnis, z. B. 10:1 oder mehr, führt DAX zu mehr Cache-Treffern und weniger veralteten Daten. Dadurch werden Lesevorgänge in einer Tabelle reduziert. Um zu verhindern, dass veraltete Daten aus dem Cache gelesen werden, wenn Ihre Anwendung schreibintensiv ist, stellen Sie sicher, dass Sie einen niedrigeren Verwenden von Time to Live (TTL) in DynamoDB-Wert für den Cache festlegen.
  - Caching häufiger Abfragen – Wenn Ihre Anwendung häufig dieselben Daten liest, z. B. beliebte Produkte auf einer E-Commerce-Plattform, kann DAX diese Anfragen direkt aus dem Cache heraus bearbeiten.
- Stoßweise Datenverkehrsmuster
  - Reibungslosere Tabellenskalierung – DAX hilft dabei, die Auswirkungen plötzlicher Datenverkehrsspitzen auszugleichen. DAX bietet einen Puffer, mit dem die Kapazität von DynamoDB-Tabellen ordnungsgemäß hochskaliert werden kann, wodurch das Risiko einer Drosselung beim Lesen verringert wird.
  - Höherer Lesedurchsatz für jedes Element – DynamoDB weist jedem Element individuelle Partitionen zu. Eine Partition beginnt jedoch, Lesevorgänge eines Elements zu drosseln, wenn 3 000 Lesekapazitätseinheiten (RCU) erreicht werden. Mit DAX können Sie die Anzahl der Lesevorgänge eines einzelnen Elements auf mehr als 3 000 RCU skalieren.

- Kostenoptimierung
  - Reduzierung der DynamoDB-Kosten – Durch das Lesen aus DAX können die an eine DynamoDB-Tabelle gesendeten Lesevorgänge reduziert werden, was sich dann direkt auf die Kosten auswirken kann. Bei einer hohen Cache-Trefferrate können die reduzierten Lesekosten für Tabellen die Kosten eines DAX-Clusters übersteigen, was zu einer Reduzierung der Nettokosten führt.
- Anforderungen für Datenkonsistenz.
  - Letztendliche Konsistenz – DAX unterstützt letztendlich konsistente Lesevorgänge. Dadurch eignet sich DAX für Anwendungsfälle, in denen eine sofortige Konsistenz nicht entscheidend ist.
  - Write-Through-Caching – Schreibvorgänge, die Sie in DAX vornehmen, werden als Write-Through bezeichnet Sobald DAX bestätigt hat, dass ein Element in DynamoDB geschrieben wurde, speichert es diese Elementversion im Element-Cache. Dieser Write-Through-Mechanismus trägt dazu bei, eine engere Datenkonsistenz zwischen Cache und Datenbank aufrechtzuerhalten, verwendet jedoch zusätzliche DAX-Cluster-Ressourcen.

Wann Sie DAX nicht verwenden sollten

DAX ist zwar leistungsstark, aber nicht für alle Szenarien geeignet. Die folgende Liste enthält Beispiele für Szenarien, in denen die Integration von DAX in DynamoDB nicht geeignet ist:

- Schreibintensive Workloads – Der Hauptvorteil von DAX ist die Beschleunigung von Lesevorgängen, aber Schreibvorgänge verbrauchen mehr DAX-Ressourcen als Lesevorgänge. Wenn Ihre Anwendung überwiegend schreibintensiv ist, sind die Vorteile von DAX möglicherweise begrenzt.
- Selten gelesene Daten – Wenn Ihre Anwendung selten auf Daten oder auf eine Vielzahl selten wiederverwendeter Daten (kalte Daten) zugreift, wird es wahrscheinlich zu einer niedrigen cache hit ratio kommen. In diesem Fall rechtfertigt der Aufwand für die Wartung des Caches die Leistungssteigerung möglicherweise nicht.
- Massenlese- oder -schreibvorgänge – Wenn Ihre Anwendung mehr Massenschreibvorgänge als einzelne Schreibvorgänge durchführt, sollten Sie DAX umgehen. Darüber hinaus sollten Sie für Massenlesevorgänge vollständige Tabellenscans direkt für eine DynamoDB-Tabelle ausführen.
- Starke Konsistenz- oder Transaktionsanforderungen — DAX leitet stark konsistente Lese- und TransactGetItemsAufrufe an eine DynamoDB-Tabelle weiter. Sie sollten diese Lesevorgänge rund um den DAX-Cluster durchführen, um die Nutzung von Cluster-Ressourcen zu vermeiden. Auf

diese Weise gelesene Elemente werden nicht zwischengespeichert. Daher hat das Routing solcher Elemente über DAX keinen Sinn.

- Einfache Anwendungen mit geringen Leistungsanforderungen – Für Anwendungen mit geringen Leistungsanforderungen und Toleranz gegenüber direkter DynamoDB-Latenz sind die Komplexität und die Kosten von DAX möglicherweise nicht notwendig. DynamoDB bewältigt einen hohen Durchsatz allein und bietet eine Leistung im einstelligen Millisekundenbereich.
- Komplexe Abfrageanforderungen, die über den Schlüssel-Wert-Zugriff hinausgehen – DAX ist für Schlüssel-Wert-Zugriffsmuster optimiert. Wenn Ihre Anwendung komplexe Abfragefunktionen mit komplexer Filterung erfordert, z. B. für Abfrage- und Scan-Operationen, sind die Vorteile des DAX-Cachings möglicherweise begrenzt.

  Verwenden Sie in diesen Situationen Amazon ElastiCache (Redis OSS) als Alternative. ElastiCache (Redis OSS) unterstützt erweiterte Datenstrukturen wie Listen, Sätze und Hashes. Es bietet außerdem Funktionen wie Pub/Sub, Geodatenindizes und Skripting.
- Compliance-Anforderungen – DAX bietet derzeit nicht dieselben Compliance-Akkreditierungen wie DynamoDB. Beispielsweise hat DAX die SOC-Akkreditierung noch nicht erhalten.

Konfigurieren Ihres DAX-Clients

Der DAX-Cluster ist ein instanzbasierter Cluster, auf den über verschiedene DAX zugegriffen werden kann. SDKs Jedes SDK bietet Entwicklern konfigurierbare Optionen wie requestTimeout und Verbindungen, um spezifische Anwendungsanforderungen zu erfüllen.

Bei der Konfiguration Ihres DAX-Clients ist der Umfang Ihrer Client-Anwendung von entscheidender Bedeutung, insbesondere das Verhältnis von Client-Instances zu DAX-Server-Instances (maximal 11). Große Client-Instance-Flotten können zahlreiche Verbindungen zu DAX-Server-Instances generieren und diese möglicherweise überfordern. In dieser Anleitung werden bewährte Methoden für die DAX-Client-Konfiguration beschrieben.

Best Practices

1. Client-Instances – Implementieren Sie Singleton-Client-Instances, um sicherzustellen, dass Instances anforderungsübergreifend wiederverwendet werden. Weitere Informationen zur Implementierung finden Sie in the section called “Schritt 4: Ausführen einer Beispielanwendung”.
2. Anforderungs-Timeouts – Anwendungen benötigen zwar oft niedrige Anforderungs-Timeouts, um eine minimale Latenz für Upstream-Systeme zu gewährleisten, aber zu niedrige Timeouts können zu Problemen führen. Niedrige Timeouts können häufige Wiederverbindungen zu Server-

Instances auslösen, wenn auf DAX-Servern vorübergehende Latenzspitzen auftreten. Wenn es zu einem Timeout kommt, beendet der DAX-Client die bestehende Serverknotenverbindung und richtet eine neue ein. Da der Verbindungsaufbau ressourcenintensiv ist, können zahlreiche aufeinanderfolgende Verbindungen die DAX-Server überlasten. Wir empfehlen Folgendes:

- Behalten Sie die Standardeinstellungen für das Anforderungs-Timeout bei.
- Wenn niedrigere Timeouts erforderlich sind, implementieren Sie separate Anwendungs-Threads mit niedrigeren Timeout-Werten und schließen Sie Wiederholungsmechanismen mit exponentiellem Backoff ein.

3. Verbindungs-Timeout – Für die meisten Anwendungen empfehlen wir, die Standardeinstellungen für das Verbindungs-Timeout beizubehalten.
4. Gleichzeitige Verbindungen — Bestimmte Verbindungen SDKs, z. B. JavaV2, ermöglichen die Anpassung gleichzeitiger Verbindungen zum DAX-Server. Wesentliche Überlegungen:
   - DAX-Server-Instances können bis zu 40 000 gleichzeitige Verbindungen verarbeiten.
   - Die Standardeinstellungen eignen sich für die meisten Anwendungsfälle.
   - Große Client-Instances in Kombination mit vielen gleichzeitigen Verbindungen können Server überlasten.
   - Niedrigere Werte für gleichzeitige Verbindungen reduzieren das Risiko einer Serverüberlastung.
   - Beispiel für eine Leistungsberechnung:
     - Unter der Annahme einer Anforderungslatenz von 1 ms kann jede Verbindung theoretisch 1 000 Anforderungen pro Sekunde verarbeiten.
     - Bei einem Cluster mit 3 Knoten kann eine einzelne Client-Instance, die eine Verbindung mit allen Knoten herstellt, 3 000 Anforderungen pro Sekunde verarbeiten.
     - Bei 10 Verbindungen kann der Client etwa 30 000 Anforderungen pro Sekunde verarbeiten.

       Empfehlung: Beginnen Sie mit niedrigeren gleichzeitigen Verbindungseinstellungen und überprüfen Sie diese anhand von Leistungstests mit den zu erwartenden Produktions-Workload-Mustern.

Konfigurieren Ihres DAX-Clusters

Der DAX-Cluster ist ein verwalteter Cluster, dessen Konfiguration Sie jedoch an Ihre Anwendungsanforderungen anpassen können. Aufgrund der engen Integration in DynamoDB-API-Operationen sollten Sie bei der Integration Ihrer Anwendung in DAX die folgenden Aspekte berücksichtigen.

In diesem Abschnitt

- DAX-Preisgestaltung
- Element-Cache und Abfrage-Cache
- Auswahl der TTL-Einstellung für die Caches
- Caching mehrerer Tabellen mit einem DAX-Cluster
- Datenreplikation in DAX und globalen DynamoDB-Tabellen
- Verfügbarkeit von DAX in Regionen
- DAX-Caching-Verhalten

DAX-Preisgestaltung

Die Kosten eines Clusters hängen von der Anzahl und Größe der bereitgestellten Knoten ab. Für jeden Knoten wird jede Stunde, die er im Cluster ausgeführt wird, in Rechnung gestellt. Weitere Informationen finden Sie unter Amazon DynamoDB – Preise.

Cache-Treffer verursachen keine DynamoDB-Kosten, wirken sich jedoch auf die DAX-Cluster-Ressourcen aus. Cache-Fehler verursachen DynamoDB-Lesekosten und benötigen DAX-Ressourcen. Schreibvorgänge verursachen DynamoDB-Schreibkosten und wirken sich auf die DAX-Cluster-Ressourcen aus, die den Schreibvorgang als Proxy ausführen.

Element-Cache und Abfrage-Cache

DAX verwaltet einen Element-Cache und einen Abfrage-Cache. Wenn Sie die Unterschiede zwischen diesen Caches verstehen, können Sie besser bestimmen, welche Leistungs- und Konsistenzmerkmale sie Ihrer Anwendung bieten.

Cache-Merkmale	Element-Cache	Abfrage-Cache
Zweck	Speichert die Ergebnisse von GetItemund API-Operationen. BatchGetItem	Speichert die Ergebnisse der Query- und Scan-API-Operationen. Bei diesen Operationen können mehrere Elemente zurückgegeben werden, die auf Abfragebedingungen und nicht auf bestimmten Elementschlüsseln basieren.

Cache-Merkmale	Element-Cache	Abfrage-Cache
Art des Zugriffs	Verwendet einen schlüssel basierten Zugriff. Wenn eine Anwendung Daten mit `GetItem` oder `BatchGetItem` anfordert , überprüft DAX zunächst den Element-Cache anhand des Primärschlüssels der angeforderten Elemente. Wenn das Element zwischeng espeichert und nicht abgelaufen ist, gibt DAX es sofort zurück, ohne auf die DynamoDB-Tabelle zuzugreif en.	Nutzt einen parameter basierten Zugriff. DAX speichert den Ergebniss atz von `Query`- und `Scan`-API-Operationen im Cache. DAX stellt nachfolgenden Anforderungen dieselben Parameter mit den gleichen Abfragebedingungen, der gleichen Tabelle und dem gleichen Index aus dem Cache bereit. Dies reduziert die Reaktionszeiten und den DynamoDB-Lesedurch satzverbrauch erheblich.

Cache-Merkmale	Element-Cache	Abfrage-Cache
Cache-Invalidierung	DAX repliziert in den folgenden Szenarien automatisch aktualisierte Elemente in den Element-Cache der Knoten im DAX-Cluster: • Sie schreiben eine Elementaktualisierung über den Cache. • Sie lesen eine aktualisierte Elementversion aus der Tabelle.	Der Abfrage-Cache ist schwieriger zu invalidieren als der Element-Cache. Elementaktualisierungen werden möglicherweise nicht direkt zwischengespeicherten Abfragen oder Scans zugeordnet. Sie müssen die TTL des Abfrage-Caches sorgfältig anpassen, um die Datenkonsistenz zu gewährleisten. Schreibvorgänge über DAX oder die Basistabelle werden erst im Abfrage-Cache wiedergegeben, wenn die TTL die zuvor zwischengespeicherte Antwort ablaufen lässt und DAX eine neue Abfrage in DynamoDB durchführt.
Globaler sekundärer Index	Da die `GetItem`-API-Operation für lokale sekundäre Indizes oder globale sekundäre Indizes nicht unterstützt wird, speichert der Element-Cache nur Lesevorgänge aus der Basistabelle im Cache.	Im Abfrage-Cache werden Abfragen sowohl für Tabellen als auch für Indizes zwischengespeichert.

Auswahl der TTL-Einstellung für die Caches

Die TTL bestimmt den Zeitraum, für den Daten im Cache gespeichert werden, bevor sie veraltet sind. Nach diesem Zeitraum werden die Daten bei der nächsten Anfrage automatisch aktualisiert. Die Auswahl der richtigen TTL-Einstellung für Ihre DAX-Caches erfordert ein ausgewogenes

Verhältnis zwischen der Optimierung der Anwendungsleistung und der Datenkonsistenz. Da es keine universelle TTL-Einstellung gibt, die für alle Anwendungen geeignet ist, hängt die optimale TTL-Einstellung von den spezifischen Eigenschaften und Anforderungen Ihrer Anwendung ab. Wir empfehlen, mit einer konservativen TTL-Einstellung zu beginnen und dabei diese vorgeschriebenen Anleitungen zu verwenden. Passen Sie dann Ihre TTL-Einstellung iterativ auf der Grundlage der Leistungsdaten und Erkenntnisse Ihrer Anwendung an.

DAX verwaltet eine LRU-Liste (Least Recently Used) für den Element-Cache. In der LRU-Liste wird nachverfolgt, wann Elemente zum ersten Mal in den Cache geschrieben oder zuletzt aus dem Cache gelesen wurden. Wenn der Arbeitsspeicher des DAX-Knotens voll ist, bereinigt DAX ältere Elemente, auch wenn sie noch nicht abgelaufen sind, um Platz für neue Elemente zu schaffen. Der LRU-Algorithmus ist immer aktiviert und kann nicht vom Benutzer konfiguriert werden.

Berücksichtigen Sie bei der Festlegung einer TTL-Dauer, die für Ihre Anwendungen geeignet ist, die folgenden Punkte:

Verstehen Ihrer Datenzugriffsmuster

- Leseintensive Workloads – Legen Sie für Anwendungen mit leseintensiven Workloads und seltenen Datenaktualisierungen eine längere TTL-Dauer fest, um die Anzahl der Cache-Fehler zu reduzieren. Eine längere TTL-Dauer reduziert außerdem die Notwendigkeit, auf die zugrunde liegende DynamoDB-Tabelle zuzugreifen.
- Schreibintensive Workloads – Legen Sie für Anwendungen mit häufigen Aktualisierungen, die nicht über DAX geschrieben werden, eine kürzere TTL-Dauer fest, um sicherzustellen, dass der Cache mit der Datenbank konsistent bleibt. Eine kürzere TTL-Dauer verringert außerdem das Risiko, dass veraltete Daten bereitgestellt werden.

Bewerten der Leistungsanforderungen Ihrer Anwendung

- Latenzempfindlichkeit – Wenn für Ihre Anwendung eher eine geringe Latenz als Datenaktualität erforderlich ist, sollten Sie eine längere TTL-Dauer verwenden. Eine längere TTL-Dauer maximiert die Cache-Treffer, wodurch die durchschnittliche Leselatenz reduziert wird.
- Durchsatz und Skalierbarkeit – Eine längere TTL-Dauer reduziert die Belastung für DynamoDB-Tabellen und verbessert den Durchsatz und die Skalierbarkeit. Sie sollten dies jedoch mit dem up-to-date Datenbedarf in Einklang bringen.

Analysieren der Cache-Bereinigung und Speichernutzung

- Cache-Speicherlimits – Überwachen Sie die Speichernutzung Ihres DAX-Clusters. Bei einer längeren TTL-Dauer können mehr Daten im Cache gespeichert werden, wodurch möglicherweise die Speicherlimits erreicht und LRU-basierte Bereinigungen durchgeführt werden.

Verwenden von Metriken und Überwachung, um die TTL anzupassen

Überprüfen Sie regelmäßig Metriken, z. B. Cache-Treffer und -Fehler sowie die CPU- und Speicherauslastung. Passen Sie Ihre TTL-Einstellung auf der Grundlage dieser Metriken an, um ein optimales Gleichgewicht zwischen Leistung und Datenaktualität zu finden. Wenn die Anzahl der Cache-Fehler hoch und die Speicherauslastung gering ist, erhöhen Sie die TTL-Dauer, um die Cache-Trefferrate zu erhöhen.

Berücksichtigen der Geschäftsanforderungen und Compliance

Richtlinien für die Datenaufbewahrung schreiben möglicherweise die maximale TTL-Dauer vor, die Sie für sensible oder personenbezogene Daten festlegen können.

Verhalten im Cache bei Festlegung der TTL auf Null

Wenn Sie die TTL auf 0 setzen, zeigen der Element-Cache und der Abfrage-Cache das folgende Verhalten:

- Element-Cache – Elemente im Cache werden nur aktualisiert, wenn eine LRU-Bereinigung oder eine Write-Through-Operation stattfindet.
- Abfrage-Cache – Abfrageantworten werden nicht zwischengespeichert.

Caching mehrerer Tabellen mit einem DAX-Cluster

Für Workloads mit mehreren kleinen DynamoDB-Tabellen, die keine einzelnen Caches benötigen, speichert ein einzelner DAX-Cluster Anfragen für diese Tabellen zwischen. Dies ermöglicht eine flexiblere und effizientere Nutzung von DAX, insbesondere für Anwendungen, die auf mehrere Tabellen zugreifen und leistungsstarke Lesevorgänge erfordern.

Ähnlich wie bei der DynamoDB-Datenebene APIs benötigen DAX-Anfragen einen Tabellennamen. Wenn Sie mehrere Tabellen im selben DAX-Cluster verwenden, benötigen Sie keine spezielle Konfiguration. Sie müssen jedoch sicherstellen, dass die Sicherheitsberechtigungen des Clusters den Zugriff auf alle zwischengespeicherten Tabellen ermöglichen.

Überlegungen zur Verwendung von DAX mit mehreren Tabellen

Wenn Sie DAX mit mehreren DynamoDB-Tabellen verwenden, sollten Sie die folgenden Punkte berücksichtigen:

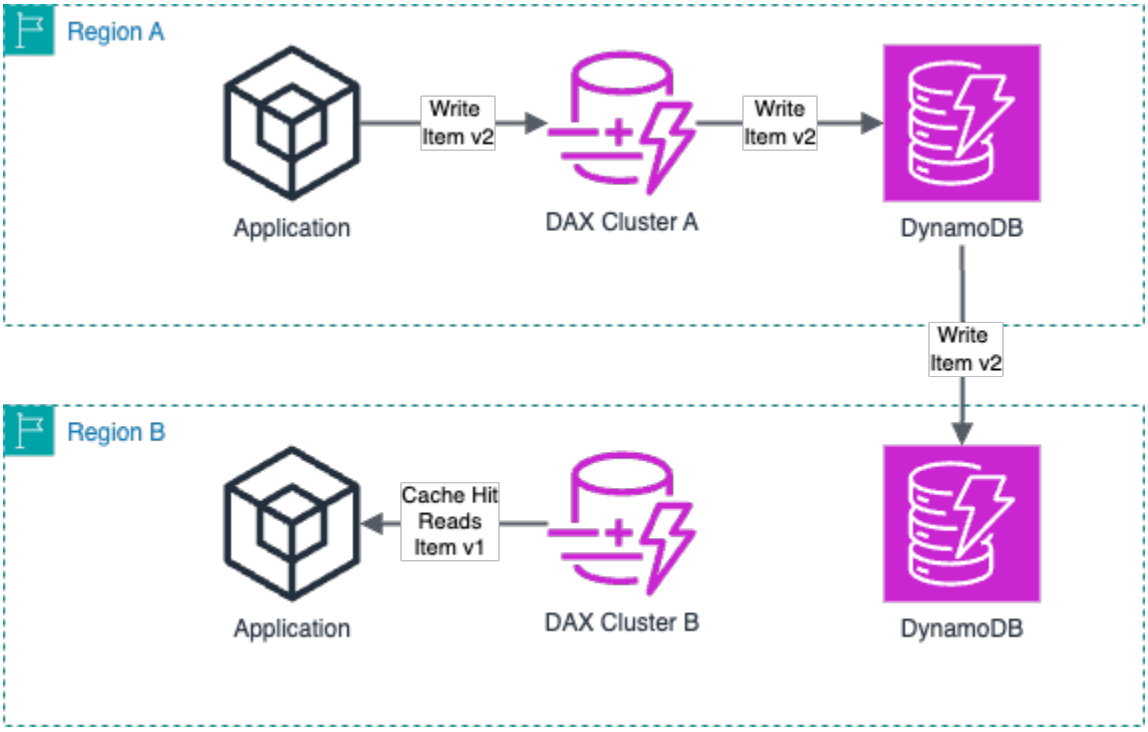
- Speicherverwaltung – Wenn Sie DAX mit mehreren Tabellen verwenden, sollten Sie die Gesamtgröße Ihres Arbeitsdatensatzes berücksichtigen. Alle Tabellen in Ihrem Datensatz teilen sich denselben Speicherplatz des von Ihnen ausgewählten Knotentyps.
- Ressourcenzuweisung – Die Ressourcen des DAX-Clusters werden von allen zwischengespeicherten Tabellen gemeinsam genutzt. Eine Tabelle mit hohem Datenverkehrsaufkommen kann jedoch dazu führen, dass Daten aus den benachbarten kleineren Tabellen bereinigt werden.
- Skaleneffekte – Gruppieren Sie kleinere Ressourcen zu einem größeren DAX-Cluster, um den durchschnittlichen Datenverkehr auf ein stabileres Muster zu reduzieren. Für die Gesamtzahl der Leseressourcen, die der DAX-Cluster benötigt, ist es außerdem wirtschaftlich, drei oder mehr Knoten zu haben. Dadurch wird auch die Verfügbarkeit aller zwischengespeicherten Tabellen im Cluster erhöht.

Datenreplikation in DAX und globalen DynamoDB-Tabellen

DAX ist ein regionsbasierter Service, sodass ein Cluster nur den Datenverkehr innerhalb seiner AWS-Region kennt. Globale Tabellen umgehen den Cache, wenn sie Daten aus einer anderen Region replizieren.

Eine längere TTL-Dauer kann dazu führen, dass veraltete Daten länger in Ihrer sekundären Region verbleiben als in der primären Region. Dies kann zu Cache-Fehlern im lokalen Cache der sekundären Region führen.

Das folgende Diagramm zeigt die Datenreplikation auf globaler Tabellenebene in der Quellregion A. Der DAX-Cluster in Region B erkennt die neu replizierten Daten aus der Quellregion A nicht sofort.



Verfügbarkeit von DAX in Regionen

Nicht alle Regionen, die DynamoDB-Tabellen unterstützen, bieten Unterstützung für die Bereitstellung von DAX-Clustern. Wenn Ihre Anwendung eine geringe Leselatenz über DAX erfordert, überprüfen Sie zunächst die Liste der Regionen, die DAX unterstützen. Wählen Sie dann eine Region für Ihre DynamoDB-Tabelle aus.

DAX-Caching-Verhalten

DAX führt Metadaten- und negatives Caching durch. Wenn Sie sich mit diesen Caching-Verhaltensweisen vertraut machen, können Sie die Attributmetadaten von zwischengespeicherten Elementen und negativen Cache-Einträgen effektiv verwalten.

- Metadaten-Caching – DAX-Cluster behalten auf unbestimmte Zeit Metadaten zu den Attributnamen zwischengespeicherter Elemente bei. Diese Metadaten bleiben auch dann bestehen, wenn das Element abläuft oder aus dem Cache bereinigt wird.

  Anwendungen, die eine unbegrenzte Anzahl von Attributnamen verwenden, können im Laufe der Zeit zu einer Erschöpfung des Speichers im DAX-Cluster führen. Diese Einschränkung gilt nur für Attributnamen auf oberster Ebene, aber nicht für verschachtelte Attributnamen. Beispiele für unbegrenzte Attributnamen sind Zeitstempel und Sitzung. UUIDs IDs Sie können zwar

Zeitstempel und Sitzung IDs als Attributwerte verwenden, wir empfehlen jedoch, kürzere und besser vorhersehbare Attributnamen zu verwenden.

- Negatives Caching – Wenn ein Cache-Fehler auftritt und der Lesevorgang aus einer DynamoDB-Tabelle keine passenden Elemente ergibt, fügt DAX dem entsprechenden Element- oder Abfrage-Cache einen negativen Cache-Eintrag hinzu. Dieser Eintrag bleibt bestehen, bis die Cache-TTL-Dauer abläuft oder ein Write-Through erfolgt. DAX gibt diesen negativen Cache-Eintrag weiterhin für künftige Anforderungen zurück.

  Wenn das negative Caching-Verhalten nicht zu Ihrem Anwendungsmuster passt, lesen Sie die DynamoDB-Tabelle direkt, wenn DAX ein leeres Ergebnis zurückgibt. Wir empfehlen außerdem, eine kürzere TTL-Cachedauer festzulegen, um langanhaltende leere Ergebnisse im Cache zu vermeiden und die Konsistenz mit der Tabelle zu verbessern.

Dimensionieren Ihres DAX-Clusters

Die Gesamtkapazität und Verfügbarkeit eines DAX-Clusters hängt vom Knotentyp und der Anzahl der Knoten ab. Mehr Knoten im Cluster erhöhen die Lesekapazität, nicht jedoch die Schreibkapazität. Größere Knotentypen (bis zu r5.8xlarge) können mehr Schreibvorgänge verarbeiten, aber zu wenige Knoten können die Verfügbarkeit beeinträchtigen, wenn ein Knoten ausfällt. Weitere Informationen zur Dimensionierung Ihres DAX-Clusters finden Sie im DAX-Clustergrößenleitfaden.

In den folgenden Abschnitten werden die verschiedenen Dimensionierungsaspekte erörtert, die Sie berücksichtigen sollten, um ein ausgewogenes Verhältnis zwischen Knotentyp und Knotenanzahl herzustellen und so einen skalierbaren und kostengünstigen Cluster zu erstellen.

In diesem Abschnitt

- Planen der Verfügbarkeit
- Planen des Schreibdurchsatzes
- Planen des Lesedurchsatzes
- Planen der Datensatzgröße
- Berechnen der ungefähren Cluster-Kapazitätsanforderungen
- Ungefähre Cluster-Durchsatzkapazität nach Knotentyp
- Skalieren der Schreibkapazität in DAX-Clustern

Planen der Verfügbarkeit

Bei der Dimensionierung eines DAX-Clusters sollten Sie sich zunächst auf dessen angestrebte Verfügbarkeit konzentrieren. Die Verfügbarkeit eines geclusterten Service wie DAX ist eine Dimension der Gesamtzahl der Knoten im Cluster. Da ein Cluster mit einem Knoten keine Fehlertoleranz hat, entspricht seine Verfügbarkeit der eines Knotens. In einem Cluster mit 10 Knoten hat der Verlust eines einzelnen Knotens nur minimale Auswirkungen auf die Gesamtkapazität des Clusters. Dieser Verlust hat keine direkten Auswirkungen auf die Verfügbarkeit, da die verbleibenden Knoten weiterhin Leseanforderungen erfüllen können. Um Schreibvorgänge wieder aufzunehmen, nominiert DAX schnell einen neuen Primärknoten.

DAX ist VPC-basiert. Es verwendet eine Subnetzgruppe, um zu bestimmen, in welchen Availability Zones Knoten ausgeführt werden können und welche IP-Adressen aus den Subnetzen verwendet werden sollen. Für Produktions-Workloads empfehlen wir dringend, DAX mit mindestens drei Knoten in verschiedenen Availability Zones zu verwenden. Dadurch wird sichergestellt, dass dem Cluster mehr als ein Knoten zur Bearbeitung von Anforderungen zur Verfügung steht, selbst wenn ein einzelner Knoten oder eine Availability Zone ausfällt. Ein Cluster kann bis zu 11 Knoten haben, wobei einer ein Primärknoten und 10 Lesereplikate sind.

Planen des Schreibdurchsatzes

Alle DAX-Cluster verfügen über einen Primärknoten für Write-Through-Anforderungen. Die Größe des Knotentyps für den Cluster bestimmt dessen Schreibkapazität. Durch das Hinzufügen zusätzlicher Lesereplikate wird die Schreibkapazität des Clusters nicht erhöht. Daher sollten Sie die Schreibkapazität bei der Cluster-Erstellung berücksichtigen, da Sie den Knotentyp später nicht ändern können.

Wenn Ihre Anwendung einen Write-Through-Vorgang in DAX durchführen muss, um den Element-Cache zu aktualisieren, sollten Sie eine verstärkte Nutzung von Cluster-Ressourcen in Betracht ziehen, um das Schreiben zu erleichtern. Schreibvorgänge in DAX verbrauchen etwa 25-mal mehr Ressourcen als Cache-Trefferlesevorgänge. Dies erfordert möglicherweise einen größeren Knotentyp als für schreibgeschützte Cluster.

Weitere Hinweise zur Bestimmung, ob Write-Through oder Write-Around für Ihre Anwendung am besten geeignet ist, finden Sie unter Strategien für Schreibvorgänge.

Planen des Lesedurchsatzes

Die Lesekapazität eines DAX-Clusters hängt von der Cache-Trefferrate Ihres Workloads ab. Da DAX Daten aus DynamoDB liest, wenn ein Cache-Fehler auftritt, verbraucht es ungefähr zehnmal

mehr Cluster-Ressourcen als ein Cache-Treffer. Um die Anzahl der Cache-Treffer zu erhöhen, erhöhen Sie die TTL-Einstellung des Caches, um den Zeitraum zu definieren, für den ein Element im Cache gespeichert wird. Eine höhere TTL-Dauer erhöht jedoch die Wahrscheinlichkeit, dass ältere Elementversionen gelesen werden, sofern Aktualisierungen nicht über DAX geschrieben werden.

Um sicherzustellen, dass der Cluster über ausreichende Lesekapazität verfügt, skalieren Sie den Cluster horizontal, wie unter Horizontale Skalierung eines Clusters beschrieben. Durch das Hinzufügen weiterer Knoten werden Lesereplikate zum Ressourcenpool hinzugefügt, während das Entfernen von Knoten die Lesekapazität reduziert. Wenn Sie die Anzahl der Knoten und deren Größe für einen Cluster auswählen, sollten Sie sowohl die minimale als auch die maximale benötigte Lesekapazität berücksichtigen. Wenn Sie einen Cluster mit kleineren Knotentypen nicht horizontal skalieren können, um Ihre Leseanforderungen zu erfüllen, verwenden Sie einen größeren Knotentyp.

Planen der Datensatzgröße

Jeder verfügbare Knotentyp hat eine festgelegte Speichergröße für DAX, um Daten zwischenzuspeichern. Wenn ein Knotentyp zu klein ist, passt der Arbeitsdatensatz, den eine Anwendung anfordert, nicht in den Speicher, was zu Cache-Fehlern führt. Da größere Knoten größere Caches unterstützen, sollten Sie einen Knotentyp verwenden, der größer ist als der geschätzte Datensatz, den Sie zwischenspeichern müssen. Ein größerer Cache verbessert auch die Cache-Trefferrate.

Sie erhalten möglicherweise abnehmende Ergebnisse, wenn Sie Elemente mit wenigen wiederholten Lesevorgängen zwischenspeichern. Berechnen Sie die Speichergröße für Elemente, auf die häufig zugegriffen wird, und stellen Sie sicher, dass der Cache groß genug ist, um diesen Datensatz zu speichern.

Berechnen der ungefähren Cluster-Kapazitätsanforderungen

Sie können den Gesamtkapazitätsbedarf Ihres Workloads abschätzen, um Ihnen bei der Auswahl der geeigneten Größe und Anzahl der Clusterknoten zu helfen. Für diese Schätzung berechnen Sie die Variable für die normalisierte Anforderung pro Sekunde (Normalized RPS). Diese Variable stellt die Gesamtzahl der Arbeitseinheiten dar, die der DAX-Cluster für Ihre Anwendung unterstützen muss, einschließlich Cache-Treffer, Cache-Fehler und Schreibvorgänge. Verwenden Sie die folgenden Eingaben, um die Variable Normalized RPS zu berechnen:

- `ReadRPS_CacheHit` – Gibt die Anzahl der Lesevorgänge pro Sekunde an, die zu einem Cache-Treffer führen.

- `ReadRPS_CacheMiss` – Gibt die Anzahl der Lesevorgänge pro Sekunde an, die zu einem Cache-Fehler führen.
- `WriteRPS` – Gibt die Anzahl der Schreibvorgänge pro Sekunde an, die DAX durchlaufen.
- `DaxNodeCount` – Gibt die Anzahl der Knoten im DAX-Cluster an.
- `Size` – Gibt die Größe des zu schreibenden oder lesenden Elements in KB an, aufgerundet auf das nächste KB.
- `10x_ReadMissFactor` – Stellt einen Wert von 10 dar. Wenn ein Cache-Fehler auftritt, verwendet DAX ungefähr zehnmal mehr Ressourcen als bei Cache-Treffern.
- `25x_WriteFactor` – Stellt einen Wert von 25 dar, da ein DAX-Write-Through etwa 25-mal mehr Ressourcen verbraucht als Cache-Treffer.

Anhand der folgenden Formel können Sie die Variable Normalized RPS berechnen.

```
Normalized RPS = (ReadRPS_CacheHit * Size) + (ReadRPS_CacheMiss * Size *
 10x_ReadMissFactor) + (WriteRequestRate * 25x_WriteFactor * Size * DaxNodeCount)
```

Betrachten wir beispielsweise folgende Eingabewerte:

- `ReadRPS_CacheHit` = 50 000
- `ReadRPS_CacheMiss` = 1 000
- `ReadMissFactor` = 1
- `Size` = 2 KB
- `WriteRPS` = 100
- `WriteFactor` = 1
- `DaxNodeCount` = 3

Wenn Sie diese Werte in die Formel einsetzen, können Sie die Variable Normalized RPS wie folgt berechnen.

```
Normalized RPS = (50,000 Cache Hits/Sec * 2KB) + (1,000 Cache Misses/Sec * 2KB * 10) +
 (100 Writes/Sec * 25 * 2KB * 3)
```

In diesem Beispiel beträgt der berechnete Wert für Normalized RPS 135 000. Dieser Normalized RPS-Wert berücksichtigt jedoch nicht die Tatsache, dass die Cluster-Auslastung unter 100 % bleibt

oder dass Knoten verloren gehen. Wir empfehlen, zusätzliche Kapazität zu berücksichtigen. Ermitteln Sie dazu den größeren von zwei Multiplikationsfaktoren: Zielauslastung oder Knotenverlusttoleranz. Multiplizieren Sie dann den Normalized RPS-Wert mit dem größeren Faktor, um eine Zielanforderung pro Sekunde (Target RPS) zu erhalten.

- Zielauslastung

  Da Leistungseinbußen die Anzahl der Cache-Fehler erhöhen, empfehlen wir, den DAX-Cluster nicht mit einer Auslastung von 100 % auszuführen. Idealerweise sollten Sie die Cluster-Auslastung bei oder unter 70 % halten. Um dies zu erreichen, multiplizieren Sie den Normalized RPS-Wert mit 1,43.

- Toleranz gegenüber Knotenverlusten

  Wenn ein Knoten ausfällt, muss Ihre Anwendung in der Lage sein, ihre Anforderungen auf die verbleibenden Knoten zu verteilen. Um sicherzustellen, dass die Knoten unter einer Auslastung von 100 % bleiben, wählen Sie einen Knotentyp aus, der groß genug ist, um zusätzlichen Datenverkehr aufzunehmen, bis der ausgefallene Knoten wieder online ist. Bei einem Cluster mit weniger Knoten muss jeder Knoten einen größeren Anstieg des Datenverkehrs tolerieren, wenn ein Knoten ausfällt.

  Beim Ausfall eines Primärknotens, führt DAX automatisch einen Failover auf ein Lesereplikat durch und bestimmt dieses zum neuen Primärknoten. Wenn ein Replikatknoten ausfällt, können andere Nodes im DAX-Cluster nach wie vor Anforderungen verarbeiten, bis der ausgefallene Knoten wiederhergestellt ist.

  Beispielsweise erfordert ein DAX-Cluster mit 3 Knoten und einem Knotenausfall zusätzliche Kapazität von 50 % auf den verbleibenden zwei Knoten. Dafür ist ein Multiplikationsfaktor von 1,5 erforderlich. Umgekehrt erfordert ein Cluster mit 11 Knoten und einem ausgefallenen Knoten zusätzliche Kapazität von 10 % auf den verbleibenden Knoten oder einen Multiplikationsfaktor von 1,1.

Anhand der folgenden Formel können Sie die Variable Target RPS berechnen.

```
Target RPS = Normalized RPS * CEILING(TargetUtilization, NodeLossTolerance)
```

Gehen Sie als Beispiel von den folgenden Werten für die Berechnung von Target RPS aus:

- `Normalized RPS`= 135 000
- `TargetUtilization`= 1,43

Da wir eine maximale Cluster-Auslastung von 70 % anstreben, setzen wir `TargetUtilization` auf 1,43.

- `NodeLossTolerance` = 1,5

  Nehmen wir an, wir verwenden einen Cluster mit 3 Knoten und legen daher `NodeLossTolerance` auf eine Kapazität von 50 % fest.

Wenn Sie diese Werte in die Formel einsetzen, können Sie die Variable Target RPS wie folgt berechnen.

```
Target RPS = 135,000 * CEILING(1.43, 1.5)
```

Da in diesem Beispiel der Wert von `NodeLossTolerance` größer als `TargetUtilization` ist, berechnen wir den Wert von Target RPS mit `NodeLossTolerance`. Dies ergibt einen Target RPS-Wert von 202 500, was der Gesamtkapazität entspricht, die der DAX-Cluster unterstützen muss. Um die Anzahl der Knoten zu ermitteln, die Sie in einem Cluster benötigen, ordnen Sie den Target RPS-Wert einer entsprechenden Spalte in der folgenden Tabelle zu. Für dieses Beispiel eines Target RPS-Werts von 202 500 benötigen Sie den Knotentyp dax.r5.large mit drei Knoten.

Ungefähre Cluster-Durchsatzkapazität nach Knotentyp

Mit der Target RPS formula können Sie die Cluster-Kapazität für verschiedene Knotentypen schätzen. Die folgende Tabelle zeigt ungefähre Kapazitäten für Cluster mit 1, 3, 5 und 11 Knotentypen. Diese Kapazitäten ersetzen nicht die Notwendigkeit, Lasttests für DAX mit Ihren eigenen Daten und Anforderungsmustern durchzuführen. Darüber hinaus beinhalten diese Kapazitäten keine Instances vom Typ T, da diese nicht über eine feste CPU-Kapazität verfügen. Die Einheit für alle Werte in der folgenden Tabelle ist Normalized RPS.

Knotentyp (Speicher)	1 Knoten	3 Knoten	5 Knoten	11 Knoten
dax.r5.24xlarge (768 GB)	1 Mio.	3 Mio.	5 Mio.	11 Mio.
dax.r5.16xlarge (512 GB)	1 Mio.	3 Mio.	5 Mio.	11 Mio.

Knotentyp (Speicher)	1 Knoten	3 Knoten	5 Knoten	11 Knoten
dax.r5.12xlarge (384 GB)	1 Mio.	3 Mio.	5 Mio.	11 Mio.
dax.r5.8xlarge (256 GB)	1 Mio.	3 Mio.	5 Mio.	11 Mio.
dax.r5.4xlarge (128 GB)	600 000	1,8 Mio.	3 Mio.	6,6 Mio.
dax.r5.2xlarge (64 GB)	300 000	900 000	1,5 Mio.	3,3 Mio.
dax.r5.xlarge (32 GB)	150 000	450 000	750 000	1,65 Mio.
dax.r5.xlarge (16 GB)	75 000	225 000	375 000	825 000

Aufgrund der Höchstgrenze von 1 Million NPS (Netzwerkoperationen pro Sekunde) für jeden Knoten tragen Knoten des Typs dax.r5.8xlarge oder größer nicht zur zusätzlichen Cluster-Kapazität bei. Knotentypen, die größer als 8xlarge sind, tragen möglicherweise nicht zur Gesamtdurchsatzkapazität des Clusters bei. Solche Knotentypen können jedoch hilfreich sein, um einen größeren Arbeitsdatensatz im Speicher zu speichern.

Skalieren der Schreibkapazität in DAX-Clustern

Jeder Schreibvorgang in DAX verbraucht 25 normalisierte Anforderungen auf jedem Knoten. Da es für jeden Knoten ein Limit von 1 Million RPS gibt, ist ein DAX-Cluster auf 40 000 Schreibvorgänge pro Sekunde begrenzt, wobei die Lesenutzung nicht berücksichtigt wird.

Wenn Ihr Anwendungsfall mehr als 40 000 Schreibvorgänge pro Sekunde im Cache erfordert, müssen Sie separate DAX-Cluster verwenden und die Schreibvorgänge als Shard auf diese verteilen. Ähnlich wie bei DynamoDB können Sie den Partitionsschlüssel für die Daten, die Sie in den Cache schreiben, hashen. Verwenden Sie dann den Modulo, um zu bestimmen, auf welchen Shard die Daten geschrieben werden sollen.

Im folgenden Beispiel wird der Hash einer Eingabezeichenfolge berechnet. Anschließend wird der Modulo des Hashwerts mit 10 berechnet.

```
def hash_modulo(input_string):
    # Compute the hash of the input string
    hash_value = hash(input_string)

    # Compute the modulus of the hash value with 10
    bucket_number = hash_value % 10

    return bucket_number

#Example usage
if _name_ == "_main_":
    input_string = input("Enter a string: ")
    result = hash_modulo(input_string)
    print(f"The hash modulo 10 of '{input_string}' is: {result}.")
```

Bereitstellen eines Clusters

Die Erstellung eines neuen DAX-Clusters erfordert Konfigurationen, die über die für DynamoDB erforderlichen hinausgehen. Diese Konfigurationen eignen sich insbesondere für Netzwerke, da DAX auf Amazon VPC basiert. Dadurch haben Sie die vollständige Kontrolle über Ihre virtuelle Netzwerkumgebung, einschließlich Ressourcenplatzierung, Konnektivität und Sicherheit. In diesem Abschnitt werden die bewährten Methoden für die Einstellungen vorgestellt, die bei der Cluster-Erstellung benötigt werden.

Weitere Informationen zum Auswählen von Cluster-Knoten finden Sie unter Dimensionieren Ihres DAX-Clusters.

In diesem Abschnitt

- Konfigurieren von Netzwerken
- Konfigurieren der Sicherheit
- Parametergruppe
- Wartungsfenster

Konfigurieren von Netzwerken

DAX verwendet eine Subnetzgruppe, um zu bestimmen, in welchen Availability Zones Knoten ausgeführt werden können und welche IP-Adressen aus den Subnetzen verwendet werden sollen. Um die Latenz zwischen Ihrer Anwendung und DAX zu minimieren, sollten die Subnetze und Availability Zones für Ihre Anwendungsserver und den DAX-Cluster identisch sein.

Wir empfehlen Ihnen, die DAX-Knoten über mehrere Availability Zones zu verteilen. Dafür verwenden Sie die Standardoption „Automatische Zuweisung“.

Bewährte Methoden zur Einrichtung Ihrer VPC finden Sie unter Erste Schritte mit Amazon VPC im Benutzerhandbuch zu Amazon VPC.

Konfigurieren der Sicherheit

In diesem Abschnitt werden die Sicherheitsmaßnahmen beschrieben, die Sie für Ihre Anwendungen implementieren sollten, die DAX verwenden. Außerdem wird kurz die Unterstützung beschrieben, die DAX für die Datenverschlüsselung bietet.

IAM

DAX und DynamoDB verfügen über separate Mechanismen für die Zugriffskontrolle. DAX benötigt eine IAM-Rolle für den Zugriff auf Ihre DynamoDB-Tabellen. Diese Rolle sollte dem Prinzip der geringsten Rechte folgen und nur Zugriff auf bestimmte Tabellen und DynamoDB-Operationen wie GetItemund gewähren. PutItem Weitere Informationen zu den Zugriffskontrollmechanismen von DAX finden Sie unter DAX-Zugriffskontrolle.

Verschlüsselung

Sie konfigurieren die Verschlüsselung im Ruhezustand und die Verschlüsselung während der Übertragung bei der Erstellung eines DAX-Clusters. Standardmäßig sind beide aktiviert. Wir empfehlen Ihnen, die standardmäßigen Verschlüsselungseinstellungen beizubehalten, solange geschäftliche Anforderungen dies nicht verhindern. Weitere Informationen erhalten Sie unter DAX-Verschlüsselung im Ruhezustand und DAX-Verschlüsselung während der Übertragung.

Parametergruppe

DAX wendet auf jeden Knoten in einem Cluster eine Reihe von Konfigurationen an, die als Parametergruppe bezeichnet werden. Sie können diese Konfiguration ändern, nachdem Sie den Cluster erstellt haben.

Die DAX-Parametergruppe enthält TTL-Einstellungen für den Element-Cache und den Abfrage-Cache. Die Standardeinstellung für die TTL-Dauer ist 5 Minuten. Sie können die TTL-Dauer auf einen beliebigen Ganzzahlwert überschreiben, der größer oder gleich 1 Millisekunde ist.

Sie können Parametergruppen nicht ändern, wenn eine laufende DAX-Instance diese verwendet. Sie können die Parametergruppenwerte während der Ausfallzeit eines DAX-Clusters ändern.

Wartungsfenster

Um gelegentliche Software-Upgrades und -Patches für Ihre Knoten zu ermöglichen, ist für den DAX-Cluster ein wöchentliches Wartungsfenster konfiguriert. Während dieses Wartungsfensters führt DAX fortlaufende Aktualisierungen für die Knoten durch. Bei Clustern mit mehr als einem Knoten geht die Verfügbarkeit des Clusters während dieser Aktualisierungen nicht verloren, die Cluster-Kapazität wird jedoch reduziert, bis der Knoten zurückkehrt. Wenn in Ihrem Unternehmen eine vorhersehbare Zeit mit geringer Auslastung zu erwarten ist, sollten Sie erwägen, das Wartungsfenster manuell auf diese Zeit einzustellen.

Verwalten von Cluster-Operationen

DAX übernimmt die Wartung und Integrität des Clusters für Sie. Sie müssen jedoch operative Eingaben bereitstellen, um den Cluster horizontal oder vertikal gemäß Ihren Nutzungsmustern zu skalieren. In diesem Abschnitt wird der empfohlene Prozess zum Skalieren Ihrer DAX-Cluster beschrieben.

In diesem Abschnitt

- Horizontale Skalierung eines Clusters
- Vertikale Skalierung eines Clusters

Horizontale Skalierung eines Clusters

Die Skalierung eines DAX-Clusters beinhaltet die Anpassung seiner Kapazität an die Durchsatzanforderungen. Diese Anpassung erfolgt, indem die Anzahl der Knoten (Replikate) im Cluster während der Ausführung erhöht oder verringert wird. Dieser Prozess, der als horizontale Skalierung bezeichnet wird, hilft dabei, den Workload auf mehr Knoten zu verteilen oder bei geringer Nachfrage auf weniger Knoten zu konsolidieren.

Sie können Ihren DAX-Cluster mit dem Befehl `decrease-replication-factor` oder `increase-replication-factor` in der AWS CLI horizontal auf- und abskalieren.

Erhöhen des Replikationsfaktors (aufskalieren)

Durch Erhöhen des Replikationsfaktors eines DAX-Clusters werden dem Cluster mehr Knoten hinzugefügt. Das folgende Beispiel zeigt die Ausgabe des Befehls `increase-replication-factor`:

```
aws dax increase-replication-factor \
    --cluster-name yourClusterName  \
    --new-replication-factor desiredReplicationFactor
```

- In diesem Befehl gibt das Argument `cluster-name` den Namen Ihres Clusters an, Beispiel, *yourClusterName*.
- Das Argument `new-replication-factor` gibt die Gesamtzahl der Knoten an, die dem Cluster nach der Skalierung hinzugefügt werden sollen. Dazu gehören der Primärknoten und die Replikatknoten. Wenn Ihr Cluster beispielsweise derzeit über 3 Knoten verfügt und Sie 2 weitere Knoten hinzufügen möchten, legen Sie den Wert für `new-replication-factor` auf 5 fest.

Verringern des Replikationsfaktors (abskalieren)

Wenn Sie den Replikationsfaktor eines DAX-Clusters verringern, werden Knoten aus dem Cluster entfernt. Das Entfernen von Knoten kann dazu beitragen, die Kosten in Zeiten geringer Nachfrage zu senken. Das folgende Beispiel zeigt die Ausgabe des Befehls `decrease-replication-factor`:

```
aws dax decrease-replication-factor \
    --cluster-name yourClusterName  \
    --new-replication-factor desiredReplicationFactor
```

- In diesem Befehl gibt das Argument `cluster-name` den Namen Ihres Clusters an, Beispiel, *yourClusterName*.
- Das Argument `new-replication-factor` gibt die reduzierte Anzahl von Knoten in Ihrem Cluster nach der Skalierung an. Diese Zahl muss niedriger als der aktuelle Replikationsfaktor sein und den Primärknoten einschließen. Wenn Ihr Cluster beispielsweise derzeit über 5 Knoten verfügt und Sie 2 Knoten entfernen möchten, legen Sie den Wert für `new-replication-factor` auf 3 fest.

Überlegungen zur horizontalen Skalierung

Berücksichtigen Sie beim Planen der horizontalen Skalierung Folgendes:

- Primärknoten – Der DAX-Cluster umfasst einen Primärknoten. Der Replikationsfaktor beinhaltet diesen Primärknoten. Ein Replikationsfaktor von 3 bedeutet beispielsweise einen Primärknoten und zwei Replikatknoten.
- Verfügbarkeit – Das Hinzufügen oder Entfernen von DAX-Knoten ändert die Verfügbarkeit und Fehlertoleranz des Clusters. Mehr Knoten können die Verfügbarkeit verbessern, erhöhen aber auch die Kosten.
- Datenmigration – Wenn Sie den Replikationsfaktor erhöhen, übernimmt DAX automatisch die Datenverteilung über die neue Gruppe von Knoten. Wenn ein neuer Knoten mit der Bereitstellung von Datenverkehr beginnt, ist sein Cache bereits vorgewärmt. Während dieses Vorgangs kann es jedoch zu vorübergehenden Auswirkungen auf die Leistung während der Datenmigration kommen.

Achten Sie darauf, Ihre DAX-Cluster während und nach dem Skalierungsvorgang genau zu beobachten, um sicherzustellen, dass sie erwartungsgemäß funktionieren, und nehmen Sie bei Bedarf weitere Anpassungen vor.

Vertikale Skalierung eines Clusters

Um die Knotengröße eines vorhandenen Clusters vertikal zu skalieren, müssen Sie einen neuen Cluster erstellen und den Anwendungsdatenverkehr zum neuen Cluster migrieren. Die Migration zu einem neuen Cluster mit verschiedenen Knoten umfasst mehrere Schritte, um einen reibungslosen Übergang mit minimalen Auswirkungen auf die Leistung und Verfügbarkeit Ihrer Anwendung sicherzustellen.

Beachten Sie die folgenden Punkte, um einen neuen Cluster für die vertikale Skalierung Ihrer Knotengröße zu erstellen:

- Greifen Sie auf Ihr aktuelles Setup zu – Überprüfen Sie die Metriken Ihres aktuellen DAX-Clusters, um die neue Knotengröße und -menge zu ermitteln, die Sie benötigen. Verwenden Sie diese Informationen als Eingabe, um Ihre Clustergröße zu definieren. Weitere Informationen finden Sie unter Dimensionieren Ihres DAX-Clusters.
- Richten Sie einen neuen DAX-Cluster ein – Erstellen Sie einen neuen DAX-Cluster mit dem Knotentyp und der Knotenmenge, die Sie ermittelt haben. Sie können die vorhandenen Konfigurationseinstellungen aus Ihrer Parametergruppe verwenden, sofern Sie keine Anpassungen vornehmen müssen.
- Synchronisieren Sie Ihre Daten – Da DAX eine Caching-Ebene für DynamoDB ist, müssen Sie Daten nicht direkt migrieren. Der neue DAX-Cluster hat jedoch solange keinen Ihrer Arbeitsdatensätze im Arbeitsspeicher, bis Sie Datenverkehr an ihn senden.

- Aktualisieren Sie die Anwendungskonfiguration – Aktualisieren Sie die Konfiguration Ihrer Anwendung so, dass sie auf den Endpunkt des neuen DAX-Clusters verweist. Je nach Konfiguration Ihrer Anwendung müssen Sie möglicherweise den Code ändern oder Umgebungsvariablen aktualisieren.

  Um die Auswirkungen beim Wechsel zu einem neuen Cluster zu reduzieren, leiten Sie Canary-Datenverkehr von einem kleinen Teil Ihrer Anwendungsflotte an den neuen Cluster weiter. Sie können dies tun, indem Sie langsam Anwendungsaktualisierungen bereitstellen oder einen gewichteten DNS-Routing-Eintrag vor Ihrem DAX-Endpunkt verwenden.
- Überwachen und optimieren Sie Ihre Konfiguration – Nachdem Sie zum neuen DAX-Cluster gewechselt haben, sollten Sie dessen Metriken und Protokolle überwachen, um eventuelle Probleme zu erkennen. Seien Sie bereit, die Anzahl der Knoten basierend auf aktualisierten Workload-Mustern anzupassen.

  Solange der neue Cluster Ihren Arbeitsdatensatz nicht ordnungsgemäß zwischenspeichert, werden Sie höhere Cache-Fehlerraten und Latenzen feststellen.
- Nehmen Sie den alten Cluster außer Betrieb – Wenn Sie sicher sind, dass der neue Cluster wie erwartet funktioniert, können Sie den alten DAX-Cluster sicher außer Betrieb nehmen, um unnötige Kosten zu vermeiden.

Überwachen von DAX

Sie können wichtige Metriken überwachen, z. B. die Cache-Trefferrate, um eine optimale DAX-Cluster-Leistung sicherzustellen, Probleme zu diagnostizieren und zu bestimmen, wann Sie den Cluster skalieren müssen. Durch die regelmäßige Überprüfung der wichtigsten Kennzahlen können Sie die Leistung, Stabilität und Kosteneffizienz aufrechterhalten, indem Sie den Cluster gemäß Ihren Workload-Anforderungen skalieren. Weitere Informationen zum Überwachen von DAX finden Sie unter Produktionsüberwachung.

Die folgende Liste enthält einige der wichtigsten Kennzahlen, die Sie überwachen sollten:

- Cache-Trefferrate – Zeigt, wie effektiv DAX zwischengespeicherte Daten bereitstellt, sodass weniger auf die zugrunde liegenden DynamoDB-Tabellen zugegriffen werden muss. Wenige Cache-Fehler für den Cluster deuten auf eine gute Caching-Effizienz hin. Wenige Cache-Treffer weisen jedoch darauf hin, dass Sie möglicherweise die TTL-Einstellung für das Caching erneut überprüfen müssen oder der Workload nicht für das Caching geeignet ist.

Verwenden Sie Amazon CloudWatch , um die Cache-Trefferquote Ihres DAX-Clusters zu berechnen. Vergleichen Sie die Metriken `ItemCacheHits`, `ItemCacheMisses`, `QueryCacheHits` und `QueryCacheMisses`, um diese Rate zu erhalten. Die folgende Formel zeigt, wie die Cache-Trefferrate berechnet wird. Um die Rate mit dieser Formel zu berechnen, teilen Sie Ihre Cache-Treffer durch die Summe Ihrer Cache-Treffer und -Fehler.

```
Cache hit ratio = Cache hits / (Cache hits + Cache misses)
```

Die Cache-Trefferrate ist eine Zahl zwischen 0 und 1, die als Prozentsatz dargestellt wird. Ein höherer Prozentsatz weist auf eine insgesamt bessere Cache-Auslastung hin.

- ErrorRequestCount— Anzahl der Anfragen, die zu Benutzerfehlern geführt haben, die vom Knoten oder Cluster gemeldet wurden. `ErrorRequestCount`umfasst Anfragen, die vom Knoten oder Cluster gedrosselt wurden. Durch die Überwachung von Benutzerfehlern können Sie Skalierungsfehlkonfigurationen oder häufig auftretende item/partition Muster in Ihrer Anwendung identifizieren.
- Betriebslatenzen – Die Überwachung der Latenz von Lese- und Schreibvorgängen zum und vom DAX-Cluster kann Ihnen dabei helfen, Leistungsengpässe zu identifizieren. Zunehmende Latenzen können auf Probleme mit Ihrer DAX-Cluster-Konfiguration oder Ihrem Netzwerk bzw. auf die Notwendigkeit einer Skalierung hinweisen.
- Netzwerkverbrauch – Behalten Sie die Metriken `NetworkBytesIn` und `NetworkBytesOut` im Auge, um den Netzwerkdatenverkehr Ihres DAX-Clusters zu überwachen. Ein unerwarteter Anstieg des Netzwerkdurchsatzes könnte zu mehr Client-Anforderungen oder ineffizienten Abfragemustern führen, die bedingen, dass mehr Daten übertragen werden.

  Durch die Überwachung des Netzwerkverbrauchs können Sie die Kosten für Ihren DAX-Cluster verwalten. Sie können außerdem sicherstellen, dass das Netzwerk nicht zu einem Engpass für die Cluster-Leistung wird.
- Bereinigungsrate – Zeigt an, wie oft Elemente aus Ihrem Cache entfernt werden, um Platz für neue Elemente zu schaffen. Wenn die Bereinigungsrate im Laufe der Zeit zunimmt, ist Ihr Cache möglicherweise zu klein oder Ihre Caching-Strategie ist nicht effektiv.

  Überwachen Sie die `EvictedSize` Metrik CloudWatch , um festzustellen, ob Ihre Cachegröße für Ihre Arbeitslast ausreichend ist. Wenn die bereinigte Gesamtgröße weiter zunimmt, müssen Sie Ihren DAX-Cluster möglicherweise hochskalieren, um einen größeren Cache aufzunehmen.

- CPU-Auslastung – Dies bezieht sich auf den Prozentsatz der CPU-Auslastung des Knotens oder Clusters. Dies ist eine wichtige Metrik, die für jede Datenbank oder jedes Caching-System überwacht werden muss. Eine hohe CPU-Auslastung kann bedeuten, dass Ihr DAX-Cluster möglicherweise überlastet ist und skaliert werden muss, um der gestiegenen Nachfrage gerecht zu werden.

  Überwachen Sie die Metrik `CPUUtilization` für Ihren DAX-Cluster. Wenn sich Ihre CPU-Auslastung kontinuierlich 70–80 % nähert oder überschreitet, sollten Sie ein Hochskalieren Ihres DAX-Clusters in Betracht ziehen, wie im folgenden Abschnitt beschrieben.

  Wenn die Anzahl der an DAX gesendeten Anforderungen die Kapazität eines Knotens überschreitet, begrenzt DAX die Rate, mit der zusätzliche Anforderungen akzeptiert werden. Dazu gibt sie a zurück ThrottlingException. DAX die CPU-Auslastung Ihres Clusters kontinuierlich aus, um zu bestimmen, welche Anforderungsmenge verarbeitet werden kann, während gleichzeitig ein fehlerfreier Cluster-Status beibehalten wird.

  Sie können die `ThrottledRequestCount` Metrik überwachen, anhand derer DAX veröffentlicht CloudWatch. Wenn diese Ausnahmen regelmäßig angezeigt werden, sollten Sie die Skalierung des Clusters in Erwägung ziehen.

Skalieren Ihres DAX-Clusters mithilfe von Überwachungsdaten

Sie können feststellen, ob Sie Ihren DAX-Cluster hoch- oder herunterskalieren müssen, indem Sie dessen Leistungsmetriken überwachen.

- Hoch- oder Herunterskalieren – Wenn Ihr DAX-Cluster eine hohe CPU-Auslastung, geringe Cache-Treffer (nach der Optimierung der Caching-Strategie) oder hohe Betriebslatenzen aufweist, sollten Sie Ihren Cluster hochskalieren. Das Hinzufügen weiterer Knoten, was auch als Aufskalieren bezeichnet wird, kann dazu beitragen, die Last gleichmäßiger zu verteilen. Für Workloads mit steigenden Schreibvorgängen pro Sekunde müssen Sie möglicherweise leistungsstärkere Knoten auswählen (Hochskalieren).
- Herunterskalieren – Wenn Sie konsistent eine niedrige CPU-Auslastung und Betriebslatenzen unter Ihren Schwellenwerten beobachten, haben Sie möglicherweise zu viele Ressourcen bereitgestellt. In solchen Fällen sollten Sie die Knoten herunterskalieren, um die Kosten zu senken. Sie können die Anzahl der Knoten in Zeiten geringer Auslastung auf 1 reduzieren, können den Cluster aber nicht vollständig herunterfahren.

DynamoDB mit anderen Diensten verwenden AWS

Amazon DynamoDB ist in andere AWS Services integriert, sodass Sie sich wiederholende Aufgaben automatisieren oder Anwendungen erstellen können, die mehrere Services umfassen.

Themen

- Konfiguration von AWS Anmeldeinformationen mit Amazon Cognito für DynamoDB
- Integrieren mit Amazon Redshift
- Verarbeiten von DynamoDB-Daten mit Apache Hive in Amazon EMR
- Integration von DynamoDB in Amazon S3
- DynamoDB Zero-ETL-Integration mit Amazon Lakehouse SageMaker
- DynamoDB Zero-ETL-Integration mit Amazon Service OpenSearch
- Integration von DynamoDB mit Amazon EventBridge
- Integrieren von DynamoDB in Amazon Managed Streaming für Apache Kafka
- Bewährte Methoden für die Integration mit DynamoDB

Konfiguration von AWS Anmeldeinformationen mit Amazon Cognito für DynamoDB

Die empfohlene Methode zum Abrufen von AWS Anmeldeinformationen für Ihre Web- und Mobilanwendungen ist die Verwendung von Amazon Cognito. Amazon Cognito hilft Ihnen, zu vermeiden, dass Ihre AWS Anmeldeinformationen in Ihren Dateien fest codiert werden. Es verwendet AWS Identity and Access Management (IAM-) Rollen, um temporäre Anmeldeinformationen für die authentifizierten und nicht authentifizierten Benutzer Ihrer Anwendung zu generieren.

Gehen Sie beispielsweise wie folgt vor, um Ihre JavaScript Dateien so zu konfigurieren, dass sie eine nicht authentifizierte Amazon Cognito Cognito-Rolle für den Zugriff auf den Amazon DynamoDB-Webservice verwenden.

So konfigurieren Sie Anmeldeinformationen für die Integration in Amazon Cognito

1. Erstellen Sie einen Amazon-Cognito-Identitäten-Pool, der nicht authentifizierte Identitäten zulässt.

```
aws cognito-identity create-identity-pool \
```

```
    --identity-pool-name DynamoPool \
    --allow-unauthenticated-identities \
    --output json
{
    "IdentityPoolId": "us-west-2:12345678-1ab2-123a-1234-a12345ab12",
    "AllowUnauthenticatedIdentities": true,
    "IdentityPoolName": "DynamoPool"
}
```

2. Kopieren Sie die folgende Richtlinie in eine Datei mit dem Namen `myCognitoPolicy.json`. Ersetzen Sie die Identitätspool-ID (*us-west-2:12345678-1ab2-123a-1234-a12345ab12*) durch Ihre eigene, die Sie im vorherigen Schritt `IdentityPoolId` erhalten haben.

   JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Federated": "cognito-identity.amazonaws.com"
      },
      "Action": "sts:AssumeRoleWithWebIdentity",
      "Condition": {
        "StringEquals": {
          "cognito-identity.amazonaws.com:aud": "us-
west-2:12345678-1ab2-123a-1234-a12345ab12"
        },
        "ForAnyValue:StringLike": {
          "cognito-identity.amazonaws.com:amr": "unauthenticated"
        }
      }
    }
  ]
}
```

3. Erstellen Sie eine IAM-Rolle, welche die vorherige Richtlinie übernimmt. Auf diese Weise wird Amazon Cognito zu einer vertrauenswürdigen Entität, welche die `Cognito_DynamoPoolUnauth`-Rolle übernehmen kann.

```
aws iam create-role --role-name Cognito_DynamoPoolUnauth \
```

```
--assume-role-policy-document file://PathToFile/myCognitoPolicy.json --output json
```

4. Gewähren Sie der Rolle `Cognito_DynamoPoolUnauth` vollen Zugriff auf DynamoDB, indem Sie eine verwaltete Richtlinie anfügen (`AmazonDynamoDBFullAccess`).

```
aws iam attach-role-policy --policy-arn arn:aws:iam::aws:policy/
AmazonDynamoDBFullAccess \
--role-name Cognito_DynamoPoolUnauth
```

> **Note**
>
> Alternativ können Sie einen fein abgestimmten Zugriff auf DynamoDB gewähren. Weitere Informationen finden Sie unter Verwenden von IAM-Richtlinienbedingungen für die differenzierte Zugriffskontrolle.

5. Rufen Sie den Amazon-Ressourcennamen der IAM-Rolle ab und kopieren Sie ihn.

```
aws iam get-role --role-name Cognito_DynamoPoolUnauth --output json
```

6. Fügen Sie die `Cognito_DynamoPoolUnauth`-Rolle dem `DynamoPool`-Identitäten-Pool hinzu. Das anzugebende Format ist `KeyName=string`, wobei `KeyName` `unauthenticated` ist und die Zeichenfolge der Rollen-ARN ist, der im vorigen Schritt erhalten wurde.

```
aws cognito-identity set-identity-pool-roles \
--identity-pool-id "us-west-2:12345678-1ab2-123a-1234-a12345ab12" \
--roles unauthenticated=arn:aws:iam::123456789012:role/Cognito_DynamoPoolUnauth --
output json
```

7. Geben Sie die Amazon-Cognito-Anmeldeinformationen in den Dateien an. Ändern Sie `IdentityPoolId` und `RoleArn` entsprechend.

```
AWS.config.credentials = new AWS.CognitoIdentityCredentials({
IdentityPoolId: "us-west-2:12345678-1ab2-123a-1234-a12345ab12",
RoleArn: "arn:aws:iam::123456789012:role/Cognito_DynamoPoolUnauth"
});
```

Sie können Ihre JavaScript Programme jetzt mit Amazon Cognito Cognito-Anmeldeinformationen für den DynamoDB-Webservice ausführen. Weitere Informationen finden Sie unter Festlegen von Anmeldeinformationen in einem Webbrowser im AWS SDK für JavaScript -Erste-Schritte-Leitfaden.

Integrieren mit Amazon Redshift

Amazon Redshift ist ein schneller, vollständig verwalteter Data-Warehouse-Service für Datenmengen im Petabyte-Bereich, mit dem Sie im Zusammenspiel mit Ihren vorhandenen Business-Intelligence-Tools alle Ihre Daten einfach, effizient und wirtschaftlich analysieren können.

DynamoDB und Amazon Redshift können zusammen verwendet werden, um unterschiedliche Datenspeicher- und Verarbeitungsanforderungen innerhalb einer Anwendung oder eines Datenökosystems zu erfüllen.

In den folgenden Themen finden Sie detailliertere Informationen zur Integration von DynamoDB in Amazon Redshift.

Themen

- Überlegungen zu einer kontoübergreifenden Integration in CMK
- Null-ETL-Integration von DynamoDB in Amazon Redshift
- Laden von Daten aus DynamoDB in Amazon Redshift mit dem Befehl COPY

Überlegungen zu einer kontoübergreifenden Integration in CMK

Bei dem Versuch einer Integration von DynamoDB in Amazon Redshift geht die erste Aktion von Amazon Redshift aus. Ohne die entsprechenden Berechtigungen könnte diese Aktion zu einem unbemerkten Fehler führen. In den folgenden Abschnitten werden die Berechtigungen beschrieben, die für diese kontenübergreifende Integration erforderlich sind.

Erforderliche Richtlinien und Berechtigungen AWS KMS

Ersetzen Sie die Platzhalter in den folgenden Beispielen:

- `111122223333`: Die AWS-Konto ID, unter der Amazon Redshift gehostet wird
- `444455556666`: Die AWS-Konto ID, unter der DynamoDB gehostet wird
- `REDSHIFT_ROLE_NAME`: Der von Amazon Redshift verwendete Name für die IAM-Rolle
- `REGION`: Der AWS-Region Ort, an dem sich Ihre Ressourcen befinden
- `TABLE_NAME`: Der Name der DynamoDB-Tabelle.
- `KMS_KEY_ID`: Die ID des KMS-Schlüssels

KMS-Schlüsselrichtlinie im DynamoDB-Konto

Die folgende AWS KMS wichtige Richtlinie ermöglicht den kontoübergreifenden Zugriff zwischen Ihren DynamoDB- und Amazon Redshift Redshift-Services. In diesem Beispiel enthält das Konto 444455556666 die DynamoDB-Tabelle und den AWS KMS Schlüssel, während das Konto 111122223333 den Amazon Redshift Redshift-Cluster enthält, der Zugriff benötigt, um die Daten zu entschlüsseln.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "Enable IAM User Permissions",
            "Effect": "Allow",
            "Principal": {
                "AWS": "arn:aws:iam::444455556666:root"
            },
            "Action": "kms:*",
            "Resource": "*"
        },
        {
            "Sid": "Allow Redshift to use the key",
            "Effect": "Allow",
            "Principal": {
                "AWS": "arn:aws:iam::111122223333:role/REDSHIFT_ROLE_NAME"
            },
            "Action": [
                "kms:Decrypt",
                "kms:DescribeKey",
                "kms:GenerateDataKey",
                "kms:GenerateDataKeyWithoutPlaintext"
            ],
            "Resource": "*"
        }
    ]
}
```

IAM-Richtlinie für die Amazon-Redshift-Rolle (im Amazon-Redshift-Konto)

Die folgende IAM-Richtlinie ermöglicht einem Amazon Redshift Redshift-Service den Zugriff auf DynamoDB-Tabellen und die zugehörigen AWS KMS Verschlüsselungsschlüssel in einem kontoübergreifenden Szenario. In diesem Beispiel enthält das Konto 444455556666 die DynamoDB-Ressourcen und AWS KMS Schlüssel, auf die der Amazon Redshift Redshift-Service zugreifen muss.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "AllowDynamoDBAccess",
            "Effect": "Allow",
            "Action": [
                "dynamodb:DescribeTable",
                "dynamodb:BatchGetItem",
                "dynamodb:Scan",
                "dynamodb:Query",
                "dynamodb:BatchGetItem",
                "dynamodb:GetItem",
                "dynamodb:GetRecords",
                "dynamodb:GetShardIterator",
                "dynamodb:DescribeStream",
                "dynamodb:ListStreams"
            ],
            "Resource": [
                "arn:aws:dynamodb:*:444455556666:table/TABLE_NAME",
                "arn:aws:dynamodb:*:444455556666:table/TABLE_NAME/stream/*"
            ]
        },
        {
            "Sid": "AllowKMSAccess",
            "Effect": "Allow",
            "Action": [
                "kms:Decrypt",
                "kms:DescribeKey",
                "kms:GenerateDataKey",
                "kms:GenerateDataKeyWithoutPlaintext"
            ],
            "Resource": "arn:aws:kms:us-east-1:444455556666:key/KMS_KEY_ID"
```

```
        }
    ]
}
```

Vertrauensstellung für die Amazon-Redshift-Rolle

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "Service": "redshift.amazonaws.com"
            },
            "Action": "sts:AssumeRole"
        }
    ]
}
```

DynamoDB-Tabellenrichtlinie (bei Verwendung von ressourcenbasierten Richtlinien)

Die folgende ressourcenbasierte Richtlinie ermöglicht einem Amazon-Redshift-Service im Konto 111122223333 den Zugriff auf DynamoDB-Tabellen und -Streams im Konto 444455556666. Fügen Sie diese Richtlinie an die DynamoDB-Tabelle an, um den kontenübergreifenden Zugriff zu aktivieren.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "AllowRedshiftAccess",
            "Effect": "Allow",
            "Principal": {
                "AWS": "arn:aws:iam::111122223333:role/REDSHIFT_ROLE_NAME"
            },
            "Action": [
```

```
                "dynamodb:DescribeTable",
                "dynamodb:BatchGetItem",
                "dynamodb:Scan",
                "dynamodb:Query",
                "dynamodb:BatchGetItem",
                "dynamodb:GetItem",
                "dynamodb:GetRecords",
                "dynamodb:GetShardIterator",
                "dynamodb:DescribeStream",
                "dynamodb:ListStreams"
            ],
            "Resource": [
                "arn:aws:dynamodb:*:444455556666:table/TABLE_NAME",
                "arn:aws:dynamodb:*:444455556666:table/TABLE_NAME/stream/*"
            ]
        }
    ]
}
```

Wichtige Überlegungen

1. Stellen Sie sicher, dass sich der KMS-Schlüssel in derselben Region befindet wie die DynamoDB-Tabelle.
2. Der KMS-Schlüssel muss ein vom Kunden verwalteter Schlüssel (CMK) sein, kein. Von AWS verwalteter Schlüssel
3. Wenn Sie globale DynamoDB-Tabellen verwenden, konfigurieren Sie Berechtigungen für alle relevanten Regionen.
4. Erwägen Sie das Hinzufügen von Bedingungsanweisungen, um den Zugriff auf der Grundlage von VPC-Endpunkten oder IP-Bereichen einzuschränken.
5. Für mehr Sicherheit sollten Sie die Verwendung der `aws:PrincipalOrgID`-Bedingungen in Betracht ziehen, um den Zugriff auf die Organisation einzuschränken.
6. Überwachen Sie die Verwendung von KMS-Schlüsseln anhand CloudTrail von CloudWatch Messdaten.

Null-ETL-Integration von DynamoDB in Amazon Redshift

Die Null-ETL-Integration von Amazon DynamoDB in Amazon Redshift ermöglicht eine reibungslose Analyse von DynamoDB-Daten ohne jegliche Codierung. Durch dieses vollständig verwaltete Feature

werden DynamoDB-Tabellen automatisch in eine Amazon-Redshift-Datenbank repliziert, sodass Benutzer SQL-Abfragen und Analysen für ihre Daten ausführen können, ohne komplexe ETL-Prozesse einrichten zu müssen. Die Integration funktioniert durch die Replikation von Daten aus der DynamoDB-Tabelle in die Amazon-Redshift-Datenbank.

Geben Sie zur Einrichtung einfach eine DynamoDB-Tabelle als Quelle und eine Amazon-Redshift-Datenbank als Ziel an. Bei der Aktivierung exportiert die Integration die vollständige DynamoDB-Tabelle und füllt die Amazon Redshift-Datenbank. Die für diesen ersten Prozess benötigte Zeit ist abhängig von der Größe der DynamoDB-Tabelle. Durch die Null-ETL-Integration werden dann mithilfe inkrementeller DynamoDB-Exporte alle 15 bis 30 Minuten Aktualisierungen von DynamoDB-Daten in Amazon Redshift repliziert. Das bedeutet, dass die replizierten DynamoDB-Daten in Amazon Redshift automatisch aufbewahrt werden. up-to-date

Nach der Konfiguration können Benutzer die DynamoDB-Daten in Amazon Redshift mithilfe von SQL-Standard-Clients und -Tools analysieren, ohne dass sich dies auf die Leistung der DynamoDB-Tabellen auswirkt. Durch den Wegfall des umständlichen ETL-Prozesses bietet diese Null-ETL-Integration eine schnelle und einfache Möglichkeit, mithilfe von Amazon-Redshift-Analysen und -Funktionen für Machine Learning Erkenntnisse aus DynamoDB-Daten zu gewinnen.

Themen

- Voraussetzungen zum Erstellen einer Null-ETL-Integration von DynamoDB in Amazon Redshift
- Einschränkungen bei der Verwendung der Null-ETL-Integration von DynamoDB in Amazon Redshift
- Erstellen einer Null-ETL-Integration von DynamoDB in Amazon Redshift
- Anzeigen von Null-ETL-Integrationen von DynamoDB in Amazon Redshift
- Löschen von Null-ETL-Integration von DynamoDB in Amazon Redshift

Voraussetzungen zum Erstellen einer Null-ETL-Integration von DynamoDB in Amazon Redshift

1. Voraussetzung für die Integration ist die Erstellung einer DynamoDB-Quelltabelle und eines Amazon-Redshift-Zielclusters. Weitere Informationen hierzu finden Sie unter Schritt 1: Konfigurieren einer DynamoDB-Quelltabelle und Schritt 2: Erstellen eines Data-Warehouse für Amazon Redshift.
2. Für eine Zero-ETL-Integration zwischen Amazon DynamoDB und Amazon Redshift muss in Ihrer DynamoDB-Quelltabelle Recovery (PITR) aktiviert sein. Point-in-time

3. Wenn ressourcenbasierte Richtlinien vorliegen und Sie eine Integration erstellen, bei der sich die DynamoDB-Tabelle und das Data Warehouse von Amazon Redshift im selben Konto befinden, können Sie im Schritt „Integration erstellen“ mit der Option Reparieren Sie es für mich die erforderlichen Ressourcenrichtlinien automatisch sowohl auf DynamoDB als auch auf Amazon Redshift anwenden.

   Wenn Sie eine Integration erstellen, bei der sich Ihre DynamoDB-Tabelle und Ihr Amazon Redshift Data Warehouse in unterschiedlichen AWS Konten befinden, müssen Sie die folgende Ressourcenrichtlinie auf Ihre DynamoDB-Tabelle anwenden.

   JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid":
 "StatementthatallowsAmazonRedshiftservicetoDescribeTableandExportTable",
            "Effect": "Allow",
            "Principal": {
                "Service": "redshift.amazonaws.com"
            },
            "Action": [
                "dynamodb:ExportTableToPointInTime",
                "dynamodb:DescribeTable"
            ],
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "aws:SourceAccount": "111122223333"
                },
                "ArnEquals": {
                    "aws:SourceArn": "arn:aws:redshift:us-
east-1:111122223333:integration:*"
                }
            }
        },
        {
            "Sid":
 "StatementthatallowsAmazonRedshiftservicetoDescribeTableandExportTable",
            "Effect": "Allow",
            "Principal": {
                "Service": "redshift.amazonaws.com"
```

```
            },
            "Action": "dynamodb:DescribeExport",
            "Resource": "arn:aws:dynamodb:us-east-1:111122223333:table/table-
name/export/*",
            "Condition": {
                "StringEquals": {
                    "aws:SourceAccount": "111122223333"
                },
                "ArnEquals": {
                    "aws:SourceArn": "arn:aws:redshift:us-
east-1:111122223333:integration:*"
                }
            }
        }
    ]
}
```

Möglicherweise müssen Sie auch die Ressourcenrichtlinie im Data Warehouse von Amazon Redshift konfigurieren. Weitere Informationen finden Sie unter Konfigurieren einer Autorisierung unter Verwendung der Amazon-Redshift-API.

4. Bei identitätsbasierten IAM-Richtlinien:

   a. Der Benutzer, der die Integration erstellt, benötigt eine identitätsbasierte Richtlinie, die die folgenden Aktionen autorisiert: `GetResourcePolicy`, `PutResourcePolicy` und `UpdateContinuousBackups`.

   > **Note**
   >
   > In den folgenden Beispielrichtlinien wird die Ressource als `arn:aws:redshift{-serverless}` angezeigt. Dieses Beispiel zeigt, dass der ARN je nachdem, ob es sich bei dem Namespace um einen Amazon-Redshift-Cluster oder einen Amazon-Redshift-Serverless-Namespace handelt, `arn:aws:redshift` oder `arn:aws:redshift-serverless` lauten kann.

   JSON

```
{
    "Version":"2012-10-17",
```

```
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "dynamodb:ListTables"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "dynamodb:GetResourcePolicy",
                "dynamodb:PutResourcePolicy",
                "dynamodb:UpdateContinuousBackups"
            ],
            "Resource": [
                "arn:aws:dynamodb:us-east-1:111122223333:table/table-name"
            ]
        },
        {
            "Sid": "AllowRedshiftDescribeIntegration",
            "Effect": "Allow",
            "Action": [
                "redshift:DescribeIntegrations"
            ],
            "Resource": "*"
        },
        {
            "Sid": "AllowRedshiftCreateIntegration",
            "Effect": "Allow",
            "Action": "redshift:CreateIntegration",
            "Resource": "arn:aws:redshift:us-
east-1:111122223333:integration:*"
        },
        {
            "Sid": "AllowRedshiftModifyDeleteIntegration",
            "Effect": "Allow",
            "Action": [
                "redshift:ModifyIntegration",
                "redshift:DeleteIntegration"
            ],
            "Resource": "arn:aws:redshift:us-
east-1:111122223333:integration:uuid"
        },
```

```
        {
            "Sid": "AllowRedshiftCreateInboundIntegration",
            "Effect": "Allow",
            "Action": "redshift:CreateInboundIntegration",
            "Resource": "arn:aws:redshift:us-
east-1:111122223333:namespace:uuid"
        }
    ]
}
```

b. Der Benutzer, der für die Konfiguration des Amazon-Redshift-Ziel-Namespace verantwortlich ist, benötigt eine identitätsbasierte Richtlinie, die die folgenden Aktionen autorisiert: `PutResourcePolicy`, `DeleteResourcePolicy` und `GetResourcePolicy`.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "redshift:PutResourcePolicy",
                "redshift:DeleteResourcePolicy",
                "redshift:GetResourcePolicy"
            ],
            "Resource": [
                "arn:aws:redshift:us-east-1:111122223333:cluster:cluster-
name"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "redshift:DescribeInboundIntegrations"
            ],
            "Resource": [
                "arn:aws:redshift:us-east-1:111122223333:cluster:cluster-
name"
            ]
        }
    ]
}
```

5. Berechtigungen für Verschlüsselungsschlüssel

   Wenn die DynamoDB-Quelltabelle mit einem vom Kunden verwalteten AWS KMS -Schlüssel verschlüsselt ist, müssen Sie dem KMS-Schlüssel die folgende Richtlinie hinzufügen. Anhand dieser Richtlinie kann Amazon Redshift mit dem KMS-Schlüssel Daten aus der verschlüsselten Tabelle exportieren.

```
{
    "Sid": "Statement to allow Amazon Redshift service to perform Decrypt operation
 on the source DynamoDB Table",
    "Effect": "Allow",
    "Principal": {
        "Service": [
            "redshift.amazonaws.com"
        ]
    },
    "Action": "kms:Decrypt",
    "Resource": "*",
    "Condition": {
        "StringEquals": {
            "aws:SourceAccount": "<account>"
        },
        "ArnEquals": {
            "aws:SourceArn": "arn:aws:redshift:<region>:<account>:integration:*"
        }
    }
}
```

Sie können auch die Schritte unter Erste Schritte mit Null-ETL-Integrationen im Amazon-Redshift-Verwaltungshandbuch ausführen, um die Berechtigungen für den Amazon-Redshift-Namespace zu konfigurieren.

Einschränkungen bei der Verwendung der Null-ETL-Integration von DynamoDB in Amazon Redshift

Die folgenden allgemeinen Einschränkungen gelten für die aktuelle Version dieser Integration. Diese Einschränkungen können sich in nachfolgenden Versionen ändern.

> **Note**
>
> Lesen Sie zusätzlich zu den unten aufgeführten Einschränkungen auch die allgemeinen Überlegungen zur Verwendung von Null-ETL-Integrationen. Weitere Informationen finden Sie unter Überlegungen bei der Verwendung von Null-ETL-Integrationen mit Amazon Redshift im Amazon-Redshift-Verwaltungshandbuch.

- Die DynamoDB-Tabelle und der Amazon-Redshift-Cluster müssen sich in derselben Region befinden.
- Die DynamoDB-Quelltabelle muss entweder mit einem Amazon-eigenen oder einem vom Kunden verwalteten Schlüssel verschlüsselt werden. AWS KMS Die von Amazon verwaltete Verschlüsselung wird für die DynamoDB-Quelltabelle nicht unterstützt.

Erstellen einer Null-ETL-Integration von DynamoDB in Amazon Redshift

Bevor Sie eine Null-ETL-Integration erstellen, müssen Sie zuerst die DynamoDB-Quelltabelle und dann das Ziel-Data-Warehouse von Amazon Redshift einrichten.

Schritt 1: Konfigurieren einer DynamoDB-Quelltabelle

Um eine Zero-ETL-Integration mit Amazon Redshift zu erstellen, müssen Sie point-in-time Recovery (PITR) für Ihre Tabelle aktivieren. Wenn Sie die PITR-Option nicht aktiviert haben, kann dies während der Einrichtung der Integration von der Konsole übernommen werden. Einzelheiten zur Aktivierung von PITR finden Sie unter Wiederherstellung. Point-in-time

Schritt 2: Erstellen eines Data-Warehouse für Amazon Redshift

Wenn Sie noch kein Data-Warehouse für Amazon Redshift eingerichtet haben, können Sie dies jetzt tun. Informationen zum Erstellen einer Serverless-Arbeitsgruppe für Amazon Redshift finden Sie unter Erstellen einer Arbeitsgruppe mit einem Namespace. Informationen zum Erstellen eines Amazon-Redshift-Clusters finden Sie unter Erstellen eines Clusters.

Damit die Integration erfolgreich durchgeführt werden kann, muss für die Zielarbeitsgruppe oder den Ziel-Cluster für Amazon Redshift der Parameter „enable_case_sensitive_identifier parameter“ aktiviert sein. Weitere Informationen zum Aktivieren der Unterscheidung zwischen Groß- und Kleinschreibung finden Sie unter Aktivieren der Unterscheidung zwischen Groß- und Kleinschreibung für Ihr Data Warehouse im Amazon-Redshift-Managementleitfaden.

Nachdem die Einrichtung der Amazon-Redshift-Arbeitsgruppe oder des -Clusters abgeschlossen ist, müssen Sie Ihr Data Warehouse konfigurieren. Weitere Informationen finden Sie im Amazon-Redshift-Managementleitfaden unter Null-ETL-Integrationen.

Schritt 3: Erstellen einer Null-ETL-Integration für DynamoDB

Führen Sie vor dem Erstellen einer Null-ETL-Integration die Aufgaben im Abschnitt Voraussetzungen zum Erstellen einer Null-ETL-Integration von DynamoDB in Amazon Redshift aus. Das Erstellen einer Integration zwischen DynamoDB und Amazon Redshift ist ein zweistufiger Prozess. Erstellen Sie zunächst eine Integration aus der DynamoDB-Datenbank und hängen Sie dann eine Amazon-Redshift-Datenbank an die neu erstellte Integration an.

Erstellen einer Null-ETL-Integration

1. Melden Sie sich bei der AWS Management Console an und öffnen Sie die Amazon DynamoDB DynamoDB-Konsole unter. https://console.aws.amazon.com/dynamodbv2
2. Wählen Sie im Navigationsbereich Integrationen aus.
3. Wählen Sie Null-ETL-Integration erstellen und dann Amazon Redshift aus.
4. Dadurch gelangen Sie zur Amazon-Redshift-Konsole. Lesen Sie den unter Erstellen einer Null-ETL-Integration für DynamoDB den Abschnitt „DynamoDB“, bevor Sie fortfahren.

Anzeigen von Null-ETL-Integrationen von DynamoDB in Amazon Redshift

Sie können die Details einer Null-ETL-Integration einblenden, um ihre Konfigurationsinformationen und den aktuellen Status einzusehen.

So zeigen Sie die Details einer Null-ETL-Integration in der Amazon-DynamoDB-Konsole an:

1. Melden Sie sich bei der AWS Management Console an und öffnen Sie die Amazon DynamoDB DynamoDB-Konsole unter. https://console.aws.amazon.com/dynamodbv2
2. Wählen Sie in der DynamoDB-Konsole die Option Integrationen aus.
3. Wählen Sie im Bereich Null-ETL-Integration die Null-ETL-Integration aus, die Sie anzeigen möchten.

So zeigen Sie die Details einer Null-ETL-Integration in der Amazon-Redshift-Konsole an:

1. Melden Sie sich bei der AWS Management Console an und öffnen Sie die Amazon Redshift Redshift-Konsole unterhttps://console.aws.amazon.com/redshiftv2.

2. Folgen Sie den Schritten unter Anzeigen von Null-ETL-Integrationen.

> **Note**
>
> Die möglichen Statusangaben einer Null-ETL-Integration in Amazon Redshift sind unter Anzeigen von Null-ETL-Integrationen im Amazon-Redshift-Verwaltungshandbuch aufgeführt.

Löschen von Null-ETL-Integration von DynamoDB in Amazon Redshift

Wenn Sie eine Null-ETL-Integration löschen, werden die Daten nicht aus DynamoDB oder Amazon Redshift gelöscht, sondern das Senden von Daten aus der Quelltabelle in die Amazon-Redshift-Zieltabelle wird von DynamoDB unterbunden.

So löschen Sie eine Null-ETL-Integration

1. Melden Sie sich bei der AWS Management Console an und öffnen Sie die Amazon DynamoDB DynamoDB-Konsole unter. https://console.aws.amazon.com/dynamodbv2
2. Wählen Sie in der DynamoDB-Konsole die Option Integrationen aus.
3. Wählen Sie im Bereich Null-ETL-Integration die Null-ETL-Integration aus, die Sie löschen möchten.
4. Wählen Sie Manage (Verwalten). Dadurch gelangen Sie zur Seite mit den Integrationsdetails.
5. Um die Löschung zu bestätigen, klicken Sie auf Delete (Löschen).

Laden von Daten aus DynamoDB in Amazon Redshift mit dem Befehl COPY

Amazon Redshift ergänzt Amazon DynamoDB um hochentwickelte Business-Intelligence-Funktionen und eine leistungsfähige Schnittstelle auf SQL-Basis. Wenn Sie Daten aus einer DynamoDB-Tabelle in Amazon Redshift kopieren, können Sie komplexe Datenanalyseabfragen für diese Daten ausführen, einschließlich Joins mit anderen Tabellen im Amazon-Redshift-Cluster.

Im Hinblick auf den bereitgestellten Durchsatz wird eine Kopieroperation aus einer DynamoDB-Tabelle der Lesekapazität der Tabelle zugerechnet. Nachdem die Daten kopiert wurden, wirken sich die SQL-Abfragen in Amazon Redshift nicht auf DynamoDB aus. Dies liegt daran, dass Ihre Abfragen auf eine Kopie der Daten aus DynamoDB und nicht auf DynamoDB selbst wirken.

Vor dem Laden von Daten aus einer DynamoDB-Tabelle müssen Sie zunächst eine Amazon-Redshift-Tabelle als Ziel für die Daten erstellen. Hinweis: Sie kopieren dabei Daten aus einer NoSQL-Umgebung in eine SQL-Umgebung und die Regeln in einer Umgebung gelten nicht unbedingt in der anderen. Im Folgenden sind einige der Unterschiede aufgeführt:

- DynamoDB-Tabellennamen können bis zu 255 Zeichen enthalten, einschließlich „.“ (Punkt) und „-“ (Bindestrich); außerdem muss die Groß- und Kleinschreibung beachtet werden. Amazon-Redshift-Tabellennamen sind auf 127 Zeichen beschränkt, dürfen keine Punkte oder Bindestriche enthalten und die Groß-/Kleinschreibung braucht nicht beachtet werden. Außerdem dürfen bei Tabellennamen keine Konflikte mit einem für Amazon Redshift reservierten Wort auftreten.
- DynamoDB unterstützt das SQL-Konzept NULL nicht. Sie müssen angeben, wie Amazon Redshift leere oder leere Attributwerte in DynamoDB interpretiert und sie entweder als NULLs oder als leere Felder behandelt.
- DynamoDB-Datentypen entsprechen nicht direkt den Datentypen von Amazon Redshift. Sie müssen dafür sorgen, dass jede Spalte in der Amazon-Redshift-Tabelle den richtigen Datentyp und die entsprechende Größe besitzt, um die Daten aus DynamoDB aufnehmen zu können.

Nachfolgend ist ein COPY-Befehl aus Amazon-Redshift-SQL als Beispiel aufgeführt:

```
copy favoritemovies from 'dynamodb://my-favorite-movies-table'
credentials 'aws_access_key_id=<Your-Access-Key-ID>;aws_secret_access_key=<Your-Secret-
Access-Key>'
readratio 50;
```

In diesem Beispiel lautet die Quelltabelle in DynamoDB `my-favorite-movies-table`. Die Zieltabelle in Amazon Redshift ist `favoritemovies`. Die `readratio 50`-Klausel regelt den Verbrauch des bereitgestellten Durchsatzes (in Prozent). In diesem Fall belegt der COPY-Befehl nicht mehr als 50 % der für `my-favorite-movies-table` bereitgestellten Lesekapazitätseinheiten. Wir empfehlen dringend, dieses Verhältnis auf einen Wert festzulegen, der kleiner als der durchschnittliche, nicht genutzte, bereitgestellte Durchsatz ist.

Ausführliche Anweisungen zum Laden von Daten aus DynamoDB in Amazon Redshift finden Sie in den folgenden Abschnitten imAmazon-Redshift-Datenbankentwicklerleitfaden:

- Laden von Daten aus einer DynamoDB-Tabelle
- Der COPY-Befehl
- COPY-Beispiele

Verarbeiten von DynamoDB-Daten mit Apache Hive in Amazon EMR

Amazon DynamoDB ist in Apache Hive integriert, eine Data-Warehousing-Anwendung, die auf Amazon EMR ausgeführt wird. Hive kann Daten in DynamoDB-Tabellen lesen und schreiben und bietet folgende Möglichkeiten:

- Abfragen von Live-DynamoDB-Daten mit einer SQL-ähnlichen Sprache (HiveQL).
- Kopieren von Daten aus einer DynamoDB-Tabelle in einen Amazon-S3-Bucket und umgekehrt.
- Kopieren von Daten aus einer DynamoDB-Tabelle in Hadoop Distributed File System (HDFS) und umgekehrt.
- Durchführen von Join-Vorgängen für DynamoDB-Tabellen.

Themen

- -Übersicht
- Tutorial: Arbeiten mit Amazon DynamoDB und Apache Hive
- Erstellen einer externen Tabelle in Hive
- Verarbeiten von HiveQL-Anweisungen
- Abfragen von Daten in DynamoDB
- Kopieren von Daten in und aus Amazon DynamoDB
- Leistungsoptimierung

-Übersicht

Amazon EMR ist Service, der die schnelle und kosteneffiziente Verarbeitung riesiger Datenmengen erleichtert. Zum Verwenden von Amazon EMR starten Sie einen verwalteten Cluster von Amazon-EC2-Instances mit dem Open-Source-Framework Hadoop. Hadoop ist eine verteilte Anwendung, die den MapReduce Algorithmus implementiert, bei dem eine Aufgabe mehreren Knoten im Cluster zugeordnet wird. Jeder Knoten verarbeitet die ihm zugewiesene Aufgabe parallel mit den anderen Knoten. Die Ausgaben werden letztendlich auf einen einzelnen Knoten reduziert, was zum Endergebnis führt.

Sie können Ihren Amazon-EMR-Cluster so starten, dass er permanent oder vorübergehend ist:

- Ein permanenter Cluster wird ausgeführt, bis er herunterfahren wird. Permanente Cluster sind ideal für die Datenanalyse, für Data Warehousing und andere interaktive Verwendungen.
- Ein vorübergehender Cluster wird ausgeführt, um einen Auftragsverlauf zu verarbeiten, und fährt dann automatisch herunter. Vorübergehende Cluster sind für regelmäßige Verarbeitungsaufgaben, wie das Ausführen von Skripts, ideal.

Weitere Informationen zur Amazon-EMR-Architektur und -Verwaltung finden Sie im Management Guide für Amazon EMR.

Wenn Sie einen Amazon-EMR-Cluster starten, geben Sie die anfängliche Anzahl und Art der Amazon-EC2-Instances an. Sie geben außerdem andere verteilte Anwendungen (zusätzlich zu Hadoop) an, die auf dem Cluster ausgeführt werden sollen. Diese Anwendungen umfassen u. a. Hue, Mahout, Pig und Spark.

Weitere Informationen über Anwendungen für Amazon EMR finden Sie in den Amazon-EMR-Versionshinweisen.

Je nach Cluster-Konfiguration liegen ein oder mehrere der folgenden Knotentypen vor:

- Leader Node — Verwaltet den Cluster und koordiniert die Verteilung der MapReduce ausführbaren Datei und Teilmengen der Rohdaten an die Kern- und Task-Instanzgruppen. Darüber hinaus verfolgt der Leader-Knoten den Status jedes durchgeführten Tasks und überwacht den Zustand der Instance-Gruppen. In jedem Cluster gibt es nur einen Leader-Knoten.
- Kernknoten — Führt MapReduce Aufgaben aus und speichert Daten mithilfe des Hadoop Distributed File System (HDFS).
- Task-Knoten (optional) — Führt MapReduce Aufgaben aus.

Tutorial: Arbeiten mit Amazon DynamoDB und Apache Hive

In diesem Tutorial starten Sie einen Amazon-EMR-Cluster und verwenden Apache Hive zum Verarbeiten von Daten in einer DynamoDB-Tabelle.

Hive ist eine Data-Warehouse-Anwendung für Hadoop, mit der Sie Daten aus mehreren Quellen verarbeiten und analysieren können. Hive bietet eine SQL-ähnliche Sprache, HiveQL, die es Ihnen ermöglicht, mit lokal im Amazon-EMR-Cluster oder in einer externen Datenquelle (wie Amazon DynamoDB) gespeicherten Daten zu arbeiten.

Weitere Informationen finden Sie im Hive-Tutorial.

Themen

- Bevor Sie beginnen
- Schritt 1: Erstellen eines Amazon EC2-Schlüsselpaares
- Schritt 2: Starten eines Amazon-EMR-Clusters
- Schritt 3: Verbinden mit dem Leader-Knoten
- Schritt 4: Laden von Daten in HDFS
- Schritt 5: Kopieren von Daten nach DynamoDB
- Schritt 6: Abfragen der Daten in der DynamoDB-Tabelle
- Schritt 7: (Optional) Bereinigen

Bevor Sie beginnen

Für dieses Tutorial benötigen Sie Folgendes:

- Ein AWS Konto. Wenn Sie kein Konto haben, finden Sie weitere Informationen unter Melden Sie sich an für AWS.
- Einen SSH-Client (Secure Shell). Sie verwenden den SSH-Client, um eine Verbindung mit dem Leader-Knoten des Amazon-EMR-Clusters herzustellen und interaktive Befehle auszuführen. In den meisten Linux-, Unix- und Mac-OS-X-Installationen sind SSH-Clients standardmäßig vorhanden. Windows-Benutzer können den PuTTY-Client, der SSH unterstützt, herunterladen und installieren.

Nächster Schritt

Schritt 1: Erstellen eines Amazon EC2-Schlüsselpaares

Schritt 1: Erstellen eines Amazon EC2-Schlüsselpaares

In diesem Schritt erstellen Sie das Amazon-EC2-Schlüsselpaar, das Sie zum Verbinden eines Amazon-EMR-Leader-Knotens und zum Ausführen von Hive-Befehlen benötigen.

1. Melden Sie sich bei der an AWS-Managementkonsole und öffnen Sie die Amazon EC2 EC2-Konsole unter https://console.aws.amazon.com/ec2/.
2. Wählen Sie eine Region (z. B. `US West (Oregon)`) aus. Dies sollte dieselbe Region sein, in der sich auch Ihre DynamoDB-Tabelle befindet.

3. Wählen Sie im Navigationsbereich die Option Key Pairs aus.
4. Wählen Sie Create Key Pair aus.
5. Geben Sie in Key pair name einen Namen für Ihr Schlüsselpaar ein (z. B. `mykeypair`) und wählen Sie anschließend Create aus.
6. Laden Sie die private Schlüsseldatei herunter. Der Dateiname endet mit `.pem` (z. B. `mykeypair.pem`). Bewahren Sie die Datei mit dem privaten Schlüssel an einem sicheren Ort auf. Sie benötigen die Datei für den Zugriff auf einen Amazon-EMR-Cluster, den Sie mit diesem Schlüsselpaar starten.

   > **Important**
   >
   > Wenn Sie das Schlüsselpaar verlieren, können Sie keine Verbindung mit dem Leader-Knoten Ihres Amazon-EMR-Clusters herstellen.

   Weitere Informationen über Schlüsselpaare finden Sie unter Amazon-EC2-Schlüsselpaare im Amazon-EC2-Benutzerhandbuch.

Nächster Schritt

Schritt 2: Starten eines Amazon-EMR-Clusters

Schritt 2: Starten eines Amazon-EMR-Clusters

In diesem Schritt konfigurieren und starten Sie einen Amazon-EMR-Cluster. Hive und ein Speicher-Handler für DynamoDB sind bereits auf dem Cluster installiert.

1. Öffnen Sie die Amazon EMR-Konsole unter https://console.aws.amazon.com/emr.
2. Wählen Sie Create Cluster aus.
3. Führen Sie auf der Seite Create Cluster - Quick Options die folgenden Schritte aus:

   a. Geben Sie unter Cluster name einen Namen für Ihren Cluster ein (z. B. `My EMR cluster`).

   b. Wählen Sie unter EC2 key pair das Schlüsselpaar aus, das Sie zuvor erstellt haben.

   Übernehmen Sie für die anderen Einstellungen die Standardwerte.
4. Wählen Sie Cluster erstellen.

Es dauert einige Minuten, bis Ihr Cluster gestartet ist. Sie können den Fortschritt auf der Seite Cluster-Details in der Amazon-EMR-Konsole überwachen.

Sobald der Cluster bereit ist, ändert sich der Status in `Waiting`.

Cluster-Protokolldateien und Amazon S3

Ein Amazon-EMR-Cluster generiert Protokolldateien, die Informationen zum Cluster-Status und Debugging-Daten enthalten. Die Standardeinstellungen für Cluster erstellen - Schnelle Optionen umfassen die Einrichtung der Amazon-EMR-Protokollierung.

Falls noch keiner vorhanden ist, AWS-Managementkonsole erstellt der einen Amazon S3 S3-Bucket. Der Bucket-Name ist`aws-logs-`*`account-id`*`-`*`region`*, wo *`account-id`* sich Ihre AWS Kontonummer *`region`* befindet und die Region ist, in der Sie den Cluster gestartet haben (zum Beispiel`aws-logs-123456789012-us-west-2`).

> **Note**
>
> Sie können die Amazon-S3-Konsole verwenden, um die Protokolldateien anzuzeigen. Weitere Informationen finden Sie unter Ansehen von Protokolldateien im Managementleitfaden für Amazon EMR.

Sie können diesen Bucket noch für andere Zwecke als das Protokollieren verwenden. Beispiel: Sie können den Bucket als Speicherort für die Ablage eines Hive-Skripts oder als Ziel für das Exportieren von Daten aus Amazon DynamoDB nach Amazon S3 verwenden.

Nächster Schritt

Schritt 3: Verbinden mit dem Leader-Knoten

Schritt 3: Verbinden mit dem Leader-Knoten

Wenn sich der Status Ihres Amazon-EMR-Clusters in `Waiting` ändert, können Sie über SSH eine Verbindung mit dem Leader-Knoten herstellen und Befehlszeilenoperationen ausführen.

1. Klicken Sie in der Amazon-EMR-Konsole auf den Clusternamen, um den Status anzusehen.
2. Suchen Sie auf der Seite Clusterdetails das Feld öffentliche Leader-DNS. Hierbei handelt es sich um den öffentlichen DNS-Namen für den Leader-Knoten Ihres Amazon-EMR-Clusters.

3. Klicken Sie rechts neben dem DNS-Namen auf den SSH-Link.
4. Befolgen Sie die Anweisungen unter Verbinden mit dem Leader-Knoten über SSH.

   Wählen Sie je nach Betriebssystem die Registerkarte Windows oder Mac/Linux aus und befolgen die Anweisungen zum Herstellen einer Verbindung mit dem Leader-Knoten.

Nachdem Sie eine Verbindung mit dem Leader-Knoten über SSH oder PuTTY hergestellt haben, wird eine Eingabeaufforderung ähnlich der folgenden angezeigt:

```
[hadoop@ip-192-0-2-0 ~]$
```

Nächster Schritt

Schritt 4: Laden von Daten in HDFS

Schritt 4: Laden von Daten in HDFS

In diesem Schritt kopieren Sie eine Datendatei in Hadoop Distributed File System (HDFS) und erstellen dann eine externe Hive-Tabelle, die der Datendatei zugeordnet ist.

Herunterladen der Beispieldaten

1. Laden Sie das Beispieldatenarchiv (`features.zip`) herunter:

   ```
   wget https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/samples/
   features.zip
   ```

2. Extrahieren Sie die Datei `features.txt` aus dem Archiv:

   ```
   unzip features.zip
   ```

3. Zeigen Sie die ersten Zeilen der Datei `features.txt` an:

   ```
   head features.txt
   ```

   Das Ergebnis sollte wie folgt aussehen:

   ```
   1535908|Big Run|Stream|WV|38.6370428|-80.8595469|794
   875609|Constable Hook|Cape|NJ|40.657881|-74.0990309|7
   ```

```
1217998|Gooseberry Island|Island|RI|41.4534361|-71.3253284|10
26603|Boone Moore Spring|Spring|AZ|34.0895692|-111.410065|3681
1506738|Missouri Flat|Flat|WA|46.7634987|-117.0346113|2605
1181348|Minnow Run|Stream|PA|40.0820178|-79.3800349|1558
1288759|Hunting Creek|Stream|TN|36.343969|-83.8029682|1024
533060|Big Charles Bayou|Bay|LA|29.6046517|-91.9828654|0
829689|Greenwood Creek|Stream|NE|41.596086|-103.0499296|3671
541692|Button Willow Island|Island|LA|31.9579389|-93.0648847|98
```

Die `features.txt` Datei enthält eine Teilmenge von Daten des Vereinigte Staaten Board on Geographic Names (http://geonames.usgs). gov/domestic/download_data.htm). Die Felder in jeder Zeile repräsentieren Folgendes:

- Merkmals-ID (eindeutige Kennung)
- Name
- Klasse (See, Wald, Strom usw.)
- Status
- Breitengrad (Grad)
- Längengrad (Grad)
- Höhe (in Fuß)

4. Geben Sie an der Eingabeaufforderung den folgenden Befehl ein:

```
hive
```

Die Eingabeaufforderung ändert sich wie folgt: `hive>`

5. Geben Sie die folgende HiveQL-Anweisung zum Erstellen einer nativen Hive-Tabelle ein:

```
CREATE TABLE hive_features
    (feature_id             BIGINT,
    feature_name            STRING ,
    feature_class           STRING ,
    state_alpha             STRING,
    prim_lat_dec            DOUBLE ,
    prim_long_dec           DOUBLE ,
    elev_in_ft              BIGINT)
    ROW FORMAT DELIMITED
    FIELDS TERMINATED BY '|'
    LINES TERMINATED BY '\n';
```

6. Geben Sie die folgende HiveQL-Anweisung zum Laden der Tabelle mit Daten ein:

```
LOAD DATA
LOCAL
INPATH './features.txt'
OVERWRITE
INTO TABLE hive_features;
```

7. Damit haben Sie eine native Hive-Tabelle, die mit Daten aus der Datei `features.txt` gefüllt wurde. Zum Überprüfen geben Sie die folgende HiveQL-Anweisung ein:

```
SELECT state_alpha, COUNT(*)
FROM hive_features
GROUP BY state_alpha;
```

   Die Ausgabe sollte eine Liste der Bundesstaaten und die Anzahl der geografischen Merkmale in jedem Bundesstaat enthalten.

Nächster Schritt

Schritt 5: Kopieren von Daten nach DynamoDB

Schritt 5: Kopieren von Daten nach DynamoDB

In diesem Schritt kopieren Sie Daten aus der Hive-Tabelle (`hive_features`) in eine neue Tabelle in DynamoDB.

1. Öffnen Sie die DynamoDB-Konsole unter. https://console.aws.amazon.com/dynamodb/
2. Wählen Sie Tabelle erstellen aus.
3. Führen Sie auf der Seite Create DynamoDB table die folgenden Schritte aus:
   a. In der Tabelle, tippen Sie **`Features`** ein.
   b. Geben Sie für Primärer Schlüssel im Feld Partitionsschlüssel, **`Id`** ein. Legen Sie den Datentyp auf Number fest.

      Deaktivieren Sie Standardeinstellungen verwenden. Geben Sie für Provisioned Capacity Folgendes ein:

      - Lesekapazitätseinheiten—`10`

- Schreibkapazitätseinheiten—10

Wählen Sie Erstellen aus.

4. Geben Sie an der Hive-Eingabeaufforderung die folgende HiveQL-Anweisung ein:

```
CREATE EXTERNAL TABLE ddb_features
    (feature_id   BIGINT,
    feature_name  STRING,
    feature_class STRING,
    state_alpha   STRING,
    prim_lat_dec  DOUBLE,
    prim_long_dec DOUBLE,
    elev_in_ft    BIGINT)
STORED BY 'org.apache.hadoop.hive.dynamodb.DynamoDBStorageHandler'
TBLPROPERTIES(
    "dynamodb.table.name" = "Features",

 "dynamodb.column.mapping"="feature_id:Id,feature_name:Name,feature_class:Class,state_alpha
);
```

Sie haben jetzt ein Mapping zwischen Hive und der Tabelle „Funktionen“ in DynamoDB hergestellt.

5. Geben Sie die folgende HiveQL-Anweisung ein, um Daten in DynamoDB zu importieren:

```
INSERT OVERWRITE TABLE ddb_features
SELECT
    feature_id,
    feature_name,
    feature_class,
    state_alpha,
    prim_lat_dec,
    prim_long_dec,
    elev_in_ft
FROM hive_features;
```

Hive reicht einen MapReduce Job ein, der von Ihrem Amazon EMR-Cluster bearbeitet wird. Die Verarbeitung des Auftrags kann einige Minuten dauern.

6. Überprüfen Sie, ob die Daten in DynamoDB geladen wurden:

a. Wählen Sie im Navigationsbereich der DynamoDB-Konsole Tabellen aus.

b. Wählen Sie die Tabelle „Features” und anschließend die Registerkarte Items aus, um die Daten anzuzeigen.

Nächster Schritt

Schritt 6: Abfragen der Daten in der DynamoDB-Tabelle

Schritt 6: Abfragen der Daten in der DynamoDB-Tabelle

In diesem Schritt verwenden Sie HiveQL, um die Tabelle „Funktionen“ in DynamoDB abzufragen. Probieren Sie die folgenden Hive-Abfragen aus:

1. Alle Merkmalstypen (`feature_class`) in alphabetischer Reihenfolge:

```
SELECT DISTINCT feature_class
FROM ddb_features
ORDER BY feature_class;
```

2. Alle Seen, die mit dem Buchstaben "M" beginnen:

```
SELECT feature_name, state_alpha
FROM ddb_features
WHERE feature_class = 'Lake'
AND feature_name LIKE 'M%'
ORDER BY feature_name;
```

3. Staaten mit mindestens drei Merkmalen, die höher als eine Meile (1,6 km) sind:

```
SELECT state_alpha, feature_class, COUNT(*)
FROM ddb_features
WHERE elev_in_ft > 5280
GROUP by state_alpha, feature_class
HAVING COUNT(*) >= 3
ORDER BY state_alpha, feature_class;
```

Nächster Schritt

Schritt 7: (Optional) Bereinigen

Schritt 7: (Optional) Bereinigen

Nachdem Sie das Tutorial abgeschlossen haben, können Sie mit diesem Abschnitt fortfahren, um mehr über das Arbeiten mit DynamoDB-Daten in Amazon EMR zu erfahren. Sie können Ihren Amazon-EMR-Cluster dabei in Betrieb lassen.

Wenn Sie den Cluster nicht mehr benötigen, sollten Sie ihn beenden und alle zugehörigen Ressourcen entfernen. So vermeiden Sie Gebühren für Ressourcen, die Sie nicht benötigen.

1. Beenden Sie den Amazon-EMR-Cluster:
   a. Öffnen Sie die Amazon EMR-Konsole unter https://console.aws.amazon.com/emr.
   b. Wählen Sie den Amazon-EMR-Cluster und anschließend Beenden aus und bestätigen Sie den Vorgang.
2. Löschen Sie die Tabelle „Funktionen“ in DynamoDB:
   a. Öffnen Sie die DynamoDB-Konsole unter. https://console.aws.amazon.com/dynamodb/
   b. Wählen Sie im Navigationsbereich Tables (Tabellen) aus.
   c. Wählen Sie die Tabelle „Funktionen“ aus. Wählen Sie im Menü Actions die Option Delete Table aus.
3. Löschen Sie den Amazon-S3-Bucket mit den Amazon-EMR-Protokolldateien:
   a. Öffnen Sie die Amazon S3 S3-Konsole unter https://console.aws.amazon.com/s3/.
   b. Wählen Sie aus der Liste der Buckets aus`aws-logs-` *`accountID`*`-`*`region`*, wo sich *`accountID`* Ihre AWS Kontonummer *`region`* befindet und in welcher Region Sie den Cluster gestartet haben.
   c. Wählen Sie im Menü Action die Option Delete aus.

Erstellen einer externen Tabelle in Hive

In Tutorial: Arbeiten mit Amazon DynamoDB und Apache Hive haben Sie eine externe Hive-Tabelle erstellt, die einer DynamoDB-Tabelle zugeordnet ist. Als Sie HiveQL-Anweisungen für die externe Tabelle ausstellten, wurden die Lese- und Schreibvorgänge an die DynamoDB-Tabelle weitergeleitet.

Sie können sich eine externe Tabelle als Zeiger auf eine Datenquelle vorstellen, die an einem anderen Ort verwaltet und gespeichert wird. In diesem Fall ist die zugrunde liegende Datenquelle eine DynamoDB-Tabelle. (Die Tabelle muss bereits vorhanden sein. Sie können keine DynamoDB-

Tabelle innerhalb von Hive erstellen, aktualisieren oder löschen.) Zum Erstellen der externen Tabelle verwenden Sie die `CREATE EXTERNAL TABLE`-Anweisung. Danach können Sie HiveQL verwenden, um so mit Daten in DynamoDB zu arbeiten, als wären diese Daten lokal in Hive gespeichert.

> **Note**
>
> Sie können `INSERT`-Anweisungen verwenden, um Daten in eine externe Tabelle einzufügen, und `SELECT`-Anweisungen zum Auswählen von Daten aus der Tabelle. Mit der `UPDATE`- oder `DELETE`-Anweisung können Sie die Daten in der Tabelle jedoch nicht bearbeiten.

Wenn Sie die externe Tabelle nicht mehr benötigen, können Sie sie mit der `DROP TABLE`-Anweisung entfernen. In diesem Fall wird mit `DROP TABLE` nur die externe Tabelle in Hive entfernt. Die Anweisung hat keine Auswirkungen auf die zugrunde liegende DynamoDB-Tabelle oder darin enthaltene Daten.

Themen

- CREATE EXTERNAL TABLE-Syntax
- Datentyp-Mappings

CREATE EXTERNAL TABLE-Syntax

Im Folgenden wird die HiveQL-Syntax zum Erstellen einer externen Hive-Tabelle dargestellt, die einer DynamoDB-Tabelle zugeordnet ist:

```
CREATE EXTERNAL TABLE hive_table

 (hive_column1_name hive_column1_datatype, hive_column2_name hive_column2_datatype...)
STORED BY 'org.apache.hadoop.hive.dynamodb.DynamoDBStorageHandler'
TBLPROPERTIES (
    "dynamodb.table.name" = "dynamodb_table",
    "dynamodb.column.mapping" =
 "hive_column1_name:dynamodb_attribute1_name,hive_column2_name:dynamodb_attribute2_name..."
);
```

Zeile 1 ist der Beginn der `CREATE EXTERNAL TABLE`-Anweisung. Hier geben Sie den Namen der Hive-Tabelle (hive_table) ein, die Sie erstellen möchten.

Zeile 2 gibt die Spalten und Datentypen für hive_table an. Sie müssen Spalten und Datentypen definieren, die den Attributen in der DynamoDB-Tabelle entsprechen.

Zeile 3 ist die `STORED BY`-Klausel. Hier geben Sie eine Klasse an, die die Datenverwaltung zwischen der Hive- und der DynamoDB-Tabelle übernimmt. Für DynamoDB sollte `STORED BY` der Wert `'org.apache.hadoop.hive.dynamodb.DynamoDBStorageHandler'` sein.

Zeile 4 ist der Beginn der `TBLPROPERTIES`-Klausel. Hier definieren Sie die folgenden Parameter für `DynamoDBStorageHandler`:

- `dynamodb.table.name` – Der Name der DynamoDB-Tabelle.
- `dynamodb.column.mapping`– Spaltennamenpaare der Hive-Tabelle und ihre entsprechenden Attribute in der DynamoDB-Tabelle. Jedes Paar hat das Format hive_column_name:dynamodb_attribute_name. Die Paare sind jeweils durch Kommas getrennt.

Beachten Sie Folgendes:

- Der Name der Hive-Tabelle muss nicht mit dem DynamoDB-Tabellennamen identisch sein.
- Die Spaltennamen der Hive-Tabelle müssen nicht mit den Namen in der DynamoDB-Tabelle identisch sein.
- Die von `dynamodb.table.name` angegebene Tabelle muss in DynamoDB vorhanden sein.
- Für `dynamodb.column.mapping`:
  - Sie müssen die Schlüsselschemaattribute für die DynamoDB-Tabelle zuordnen. Dies umfasst die Partitions- und Sortierschlüssel (sofern vorhanden).
  - Die Nicht-Schlüsselattribute der DynamoDB-Tabelle müssen nicht zugeordnet werden. Sie sehen jedoch keine Daten dieser Attribute, wenn Sie die Hive-Tabelle abfragen.
  - Wenn die Datentypen einer Hive-Tabellenspalte und ein DynamoDB-Attribute nicht kompatibel sind, wird beim Abfragen der Hive-Tabelle in diesen Spalten `NULL` angezeigt.

> **Note**
>
> Die `CREATE EXTERNAL TABLE`-Anweisung führt keine Validierung der `TBLPROPERTIES`-Klausel durch. Die Werte, die Sie für `dynamodb.table.name` und `dynamodb.column.mapping` bereitstellen, werden von der `DynamoDBStorageHandler`-Klasse nur ausgewertet, wenn Sie versuchen, auf die Tabelle zuzugreifen.

Datentyp-Mappings

In der folgenden Tabelle sind DynamoDB-Datentypen und kompatible Hive-Datentypen aufgeführt:

DynamoDB-Datentyp	Hive-Datentyp
Zeichenfolge	`STRING`
Zahl	`BIGINT` oder `DOUBLE`
Binär	`BINARY`
Zeichenfolgensatz	`ARRAY<STRING>`
Zahlensatz	`ARRAY<BIGINT>` oder `ARRAY<DOUBLE>`
Binärwertesatz	`ARRAY<BINARY>`

> **Note**
>
> Die folgenden DynamoDB-Datentypen werden von der `DynamoDBStorageHandler`-Klasse nicht unterstützt und können daher nicht mit `dynamodb.column.mapping` verwendet werden:
>
> - Zuordnung
> - Auflisten
> - Boolesch
> - Null
>
> Wenn Sie jedoch mit diesen Datentypen arbeiten müssen, können Sie eine einzelne Entität mit dem Namen `item` erstellen, die das gesamte DynamoDB-Element als Zuordnung von Zeichenfolgen für Schlüssel und Werte in der Zuordnung darstellt. Weitere Informationen finden Sie unter Kopieren von Daten ohne Spaltenmapping.

Wenn Sie ein DynamoDB-Attribut des Typs Zahl zuordnen möchten, müssen Sie einen entsprechenden Hive-Typ auswählen:

- Der Hive-Typ `BIGINT` ist für 8-Byte-Ganzzahlen mit Vorzeichen bestimmt. Er ist mit dem Datentyp `long` in Java identisch.
- Der Hive-Typ `DOUBLE` ist für 8-Bit-Gleitkommazahlen mit doppelter Genauigkeit vorgesehen. Er ist mit dem Typ `double` in Java identisch.

Wenn Sie numerische Daten in DynamoDB mit größerer Genauigkeit gespeichert haben als der ausgewählte Hive-Datentyp aufweist, kann der Zugriff auf die DynamoDB-Daten zu Datenverlust führen.

Wenn Sie Daten des Typs Binärwert von DynamoDB in (Amazon S3) oder HDFS exportieren, werden die Daten als Base64-kodierte Zeichenfolge gespeichert. Wenn Sie Daten aus Amazon S3 oder HDFS in den DynamoDB-Typ Binärwert importieren, müssen Sie sicherstellen, dass die Daten als Base64-Zeichenfolge kodiert sind.

Verarbeiten von HiveQL-Anweisungen

Hive ist eine Anwendung, die auf Hadoop läuft, einem stapelorientierten Framework für die Ausführung von Jobs. MapReduce Wenn Sie eine HiveQL-Anweisung ausgeben, bestimmt Hive, ob es die Ergebnisse sofort zurückgeben kann oder ob es einen Job einreichen muss. MapReduce

Betrachten Sie z. B. die Tabelle ddb_features (aus Tutorial: Arbeiten mit Amazon DynamoDB und Apache Hive). Mit der folgenden Hive-Abfrage werden Abkürzungen für die Bundesstaaten und die Anzahl von Gipfeln in jedem Bundesstaat gedruckt:

```
SELECT state_alpha, count(*)
FROM ddb_features
WHERE feature_class = 'Summit'
GROUP BY state_alpha;
```

Hive gibt die Ergebnisse nicht sofort zurück. Stattdessen sendet es einen MapReduce Job, der vom Hadoop-Framework verarbeitet wird. Hive wartet, bis der Auftrag abgeschlossen ist und zeigt erst dann die Ergebnisse der Abfrage an:

```
AK  2
AL  2
AR  2
AZ  3
CA  7
CO  2
```

```
CT  2
ID  1
KS  1
ME  2
MI  1
MT  3
NC  1
NE  1
NM  1
NY  2
OR  5
PA  1
TN  1
TX  1
UT  4
VA  1
VT  2
WA  2
WY  3
Time taken: 8.753 seconds, Fetched: 25 row(s)
```

Überwachen und Abbrechen von Aufträgen

Wenn Hive einen Hadoop-Auftrag startet, wird die Ausgabe dieses Auftrags gedruckt. Der Auftragsabschlussstatus wird aktualisiert, während der Auftrag bearbeitet wird. In einigen Fällen wird der Status möglicherweise für einen längeren Zeitraum nicht aktualisiert. (Dies kann der Fall sein, wenn Sie eine große DynamoDB-Tabelle mit einer niedrigen, bereitgestellten Lesekapazitätseinstellung abfragen.)

Wenn Sie den Auftrag abbrechen müssen, bevor er abgeschlossen wird, können Sie jederzeit **Ctrl +C** eingeben.

Abfragen von Daten in DynamoDB

Die folgenden Beispiele zeigen einige Möglichkeiten zur Verwendung von HiveQL zum Abfragen von Daten in DynamoDB.

Diese Beispiele beziehen sich auf die Tabelle ddb_features im Tutorial (Schritt 5: Kopieren von Daten nach DynamoDB).

Themen

- Verwenden von Aggregatfunktionen

- Verwenden der GROUP BY- und HAVING-Klauseln
- Verknüpfen von zwei DynamoDB-Tabellen
- Verknüpfen von Tabellen aus anderen Quellen

Verwenden von Aggregatfunktionen

HiveQL bietet integrierte Funktionen für die Zusammenfassung von Datenwerten. Sie können z. B. die Funktion `MAX` verwenden, um den größten Wert für eine ausgewählte Spalte zu finden. Das folgende Beispiel gibt die Erhöhung des höchsten Merkmals im Bundesstaat Colorado zurück.

```
SELECT MAX(elev_in_ft)
FROM ddb_features
WHERE state_alpha = 'CO';
```

Verwenden der GROUP BY- und HAVING-Klauseln

Sie können die `GROUP BY`-Klausel zum Sammeln von Daten über mehrere Datensätze hinweg verwenden. Diese Klausel wird häufig in Verbindung mit einer aggregierten Funktion wie `SUM`, `COUNT`, `MIN` oder `MAX` eingesetzt. Sie können mithilfe der `HAVING`-Klausel auch Ergebnisse verwerfen, die bestimmte Kriterien nicht erfüllen.

Das folgende Beispiel gibt eine Liste der höchsten Erhebungen aus Bundesstaaten mit mehr als fünf Merkmalen in der Tabelle ddb_features zurück.

```
SELECT state_alpha, max(elev_in_ft)
FROM ddb_features
GROUP BY state_alpha
HAVING count(*) >= 5;
```

Verknüpfen von zwei DynamoDB-Tabellen

Das folgende Beispiel zeigt, wie eine Hive-Tabelle (east_coast_states) einer Tabelle in DynamoDB zugeordnet wird. Die `SELECT`-Anweisung ist ein Join dieser beiden Tabellen. Der Join wird auf dem Cluster verarbeitet und zurückgegeben. Der Join wird nicht in DynamoDB ausgeführt.

Stellen Sie sich eine DynamoDB-Tabelle mit dem Namen vor EastCoastStates , die die folgenden Daten enthält:

```
StateName       StateAbbrev
```

```
Maine           ME
New Hampshire   NH
Massachusetts   MA
Rhode Island    RI
Connecticut     CT
New York        NY
New Jersey      NJ
Delaware        DE
Maryland        MD
Virginia        VA
North Carolina  NC
South Carolina  SC
Georgia         GA
Florida         FL
```

Nehmen wir an, die Tabelle ist als externe Hive-Tabelle mit dem Namen "east_coast_states" verfügbar:

```
CREATE EXTERNAL TABLE ddb_east_coast_states (state_name STRING, state_alpha STRING)
STORED BY 'org.apache.hadoop.hive.dynamodb.DynamoDBStorageHandler'
TBLPROPERTIES ("dynamodb.table.name" = "EastCoastStates",
"dynamodb.column.mapping" = "state_name:StateName,state_alpha:StateAbbrev");
```

Der folgende Join gibt die Bundesstaaten an der Ostküste der Vereinigten Staaten zurück, die mindestens drei Merkmale aufweisen:

```
SELECT ecs.state_name, f.feature_class, COUNT(*)
FROM ddb_east_coast_states ecs
JOIN ddb_features f on ecs.state_alpha = f.state_alpha
GROUP BY ecs.state_name, f.feature_class
HAVING COUNT(*) >= 3;
```

Verknüpfen von Tabellen aus anderen Quellen

Im folgenden Beispiel ist "s3_east_coast_states" eine Hive-Tabelle, die einer in Amazon S3 gespeicherten CSV-Datei zugeordnet ist. Die Tabelle ddb_features ist mit Daten in DynamoDB verknüpft. Das folgende Beispiel verknüpft die beiden Tabellen und liefert die geografischen Merkmale der Bundesstaaten, deren Namen mit "New" beginnen.

```
create external table s3_east_coast_states (state_name STRING, state_alpha STRING)
```

```
ROW FORMAT DELIMITED FIELDS TERMINATED BY ','
LOCATION 's3://bucketname/path/subpath/';
```

```
SELECT ecs.state_name, f.feature_name, f.feature_class
FROM s3_east_coast_states ecs
JOIN ddb_features f
ON ecs.state_alpha = f.state_alpha
WHERE ecs.state_name LIKE 'New%';
```

Kopieren von Daten in und aus Amazon DynamoDB

Im Tutorial: Arbeiten mit Amazon DynamoDB und Apache Hive haben Sie Daten aus einer nativen Hive-Tabelle in eine externe DynamoDB-Tabelle kopiert und die externe DynamoDB-Tabelle abgefragt. Die Tabelle ist extern, da sie sich außerhalb von Hive befindet. Auch wenn Sie die zugeordnete Hive-Tabelle verwerfen, ist die Tabelle in DynamoDB davon nicht betroffen.

Bei Hive handelt es sich um eine hervorragende Lösung für das Kopieren von Daten zwischen DynamoDB-Tabellen, Amazon-S3-Buckets, nativen Hive-Tabellen und Hadoop Distributed File System (HDFS). Dieser Abschnitt enthält Beispiele für diese Operationen.

Themen

- Kopieren von Daten zwischen DynamoDB und einer nativen Hive-Tabelle
- Kopieren von Daten zwischen DynamoDB und Amazon S3
- Kopieren von Daten zwischen DynamoDB und HDFS
- Verwenden der Datenkomprimierung
- Lesen von nicht druckbaren UTF-8-Zeichendaten

Kopieren von Daten zwischen DynamoDB und einer nativen Hive-Tabelle

Wenn Sie Daten in einer DynamoDB-Tabelle vorliegen haben, können Sie sie in eine native Hive-Tabelle kopieren. So erhalten Sie einen Snapshot der Daten zum Zeitpunkt des Kopierens.

Dies ist sinnvoll, wenn Sie viele HiveQL-Abfragen ausführen müssen, aber keine bereitgestellte Durchsatzkapazität aus DynamoDB verbrauchen möchten. Da es sich bei den Daten in der systemeigenen Hive-Tabelle um eine Kopie der Daten aus DynamoDB handelt und nicht um „Live-Daten“, sollten Ihre Abfragen nicht erwarten, dass es sich bei den Daten um Daten handelt. up-to-date

> **Note**
>
> Die Beispiele in diesem Abschnitt setzen voraus, dass Sie die Schritte in Tutorial: Arbeiten mit Amazon DynamoDB und Apache Hive befolgt haben und über eine externe Tabelle, die in DynamoDB verwaltet wird (ddb_features), verfügen.

Example Kopieren von DynamoDB in native Hive-Tabelle

Sie können eine native Hive-Tabelle erstellen und sie mit Daten aus ddb_features wie folgt füllen:

```
CREATE TABLE features_snapshot AS
SELECT * FROM ddb_features;
```

Anschließend können Sie die Daten jederzeit aktualisieren:

```
INSERT OVERWRITE TABLE features_snapshot
SELECT * FROM ddb_features;
```

In diesen Beispielen werden mit der Unterabfrage `SELECT * FROM ddb_features` alle Daten aus der Tabelle ddb_features abgerufen. Wenn Sie nur eine Teilmenge der Daten kopieren möchten, verwenden Sie eine `WHERE`-Klausel in der Unterabfrage.

Im folgenden Beispiel wird eine native Hive-Tabelle erstellt, die nur einige der Attribute für Seen und Gipfel enthält:

```
CREATE TABLE lakes_and_summits AS
SELECT feature_name, feature_class, state_alpha
FROM ddb_features
WHERE feature_class IN ('Lake','Summit');
```

Example Kopieren von nativer Hive-Tabelle in DynamoDB

Verwenden Sie die folgende HiveQL-Anweisung, um die Daten aus der nativen Hive-Tabelle in die Tabelle ddb_features zu kopieren:

```
INSERT OVERWRITE TABLE ddb_features
SELECT * FROM features_snapshot;
```

Kopieren von Daten zwischen DynamoDB und Amazon S3

Wenn Sie Daten in einer DynamoDB-Tabelle vorliegen haben, können Sie sie mit Hive in einen Amazon-S3-Bucket kopieren.

Dies ist sinnvoll, wenn Sie ein Datenarchiv in Ihrer DynamoDB-Tabelle erstellen möchten. Angenommen, Sie haben eine Testumgebung, in der Sie mit einer grundlegenden Reihe von Testdaten in DynamoDB arbeiten müssen. Sie können die grundlegenden Daten in einen Amazon-S3-Bucket kopieren und Ihre Tests dann ausführen. Danach können Sie die Testumgebung durch Wiederherstellen der grundlegenden Daten aus dem Amazon-S3-Bucket in DynamoDB zurücksetzen.

Wenn Sie das Tutorial: Arbeiten mit Amazon DynamoDB und Apache Hive durchgearbeitet haben, verfügen Sie bereits über einen Amazon-S3-Bucket, der Ihre Amazon-EMR-Protokolle enthält. Sie können diesen Bucket für die Beispiele in diesem Abschnitt verwenden, wenn Sie den Stammpfad des Buckets kennen:

1. Öffnen Sie die Amazon EMR-Konsole unter https://console.aws.amazon.com/emr.
2. Wählen Sie für Name Ihren Cluster aus.
3. Der URI ist in Log URI unter Configuration Details aufgelistet.
4. Notieren Sie den Stammpfad des Buckets. Die Namenskonvention lautet wie folgt:

   `s3://aws-logs-`*`accountID`*`-`*`region`*

   wo *`accountID`* ist Ihre AWS Konto-ID und Region ist die AWS Region für den Bucket.

> **Note**
>
> Bei diesen Beispielen verwenden wir ein Unterpfad innerhalb des Buckets, wie in diesem Fall: `s3://aws-logs-123456789012-us-west-2/hive-test`

Die folgenden Verfahren setzen voraus, dass Sie die Schritte im Tutorial befolgt haben und über eine externe Tabelle, die in DynamoDB verwaltet wird (ddb_features), verfügen.

Themen

- Kopieren von Daten mit dem Hive-Standardformat
- Kopieren von Daten in einem benutzerdefinierten Format

- Kopieren von Daten ohne Spaltenmapping
- Anzeigen der Daten in Amazon S3

Kopieren von Daten mit dem Hive-Standardformat

Example Von DynamoDB zu Amazon S3

Verwenden Sie eine `INSERT OVERWRITE`-Anweisung, um direkt in Amazon S3 zu schreiben.

```
INSERT OVERWRITE DIRECTORY 's3://aws-logs-123456789012-us-west-2/hive-test'
SELECT * FROM ddb_features;
```

Die Datendatei in Amazon S3 sieht folgendermaßen aus:

```
920709^ASoldiers Farewell Hill^ASummit^ANM^A32.3564729^A-108.33004616135
1178153^AJones Run^AStream^APA^A41.2120086^A-79.25920781260
253838^ASentinel Dome^ASummit^ACA^A37.7229821^A-119.584338133
264054^ANeversweet Gulch^AValley^ACA^A41.6565269^A-122.83614322900
115905^AChacaloochee Bay^ABay^AAL^A30.6979676^A-87.97388530
```

Jedes Feld wird durch ein SoH-Zeichen getrennt (Anfang der Überschrift, 0x01). In der Datei wird SoH als **^A** angezeigt.

Example Von Amazon S3 zu DynamoDB

1. Erstellen Sie eine externe Tabelle, die auf die unformatierten Daten in Amazon S3 verweist.

   ```
   CREATE EXTERNAL TABLE s3_features_unformatted
       (feature_id       BIGINT,
       feature_name      STRING ,
       feature_class     STRING ,
       state_alpha       STRING,
       prim_lat_dec      DOUBLE ,
       prim_long_dec     DOUBLE ,
       elev_in_ft        BIGINT)
   LOCATION 's3://aws-logs-123456789012-us-west-2/hive-test';
   ```

2. Kopieren Sie die Daten nach DynamoDB.

   ```
   INSERT OVERWRITE TABLE ddb_features
   SELECT * FROM s3_features_unformatted;
   ```

Kopieren von Daten in einem benutzerdefinierten Format

Wenn Sie ein eigenes Feldtrennzeichen angeben möchten, können Sie eine externe Tabelle erstellen, die dem Amazon S3-Bucket zugeordnet ist. Sie können diese Vorgehensweise zum Erstellen von Datendateien mit CSV-Werten verwenden.

Example Von DynamoDB zu Amazon S3

1. Erstellen Sie eine externe Hive-Tabelle, die Amazon S3 zugeordnet ist. Stellen Sie dabei sicher, dass die Datentypen mit den Typen der externen DynamoDB-Tabelle konsistent sind.

   ```
   CREATE EXTERNAL TABLE s3_features_csv
       (feature_id       BIGINT,
       feature_name      STRING,
       feature_class     STRING,
       state_alpha       STRING,
       prim_lat_dec      DOUBLE,
       prim_long_dec     DOUBLE,
       elev_in_ft        BIGINT)
   ROW FORMAT DELIMITED
   FIELDS TERMINATED BY ','
   LOCATION 's3://aws-logs-123456789012-us-west-2/hive-test';
   ```

2. Kopieren Sie die Daten aus DynamoDB.

   ```
   INSERT OVERWRITE TABLE s3_features_csv
   SELECT * FROM ddb_features;
   ```

Die Datendatei in Amazon S3 sieht folgendermaßen aus:

```
920709,Soldiers Farewell Hill,Summit,NM,32.3564729,-108.3300461,6135
1178153,Jones Run,Stream,PA,41.2120086,-79.2592078,1260
253838,Sentinel Dome,Summit,CA,37.7229821,-119.58433,8133
264054,Neversweet Gulch,Valley,CA,41.6565269,-122.8361432,2900
115905,Chacaloochee Bay,Bay,AL,30.6979676,-87.9738853,0
```

Example Von Amazon S3 zu DynamoDB

Mit einer einzigen HiveQL-Anweisung können Sie die DynamoDB-Tabelle mit den Daten aus Amazon S3: auffüllen:

```
INSERT OVERWRITE TABLE ddb_features
SELECT * FROM s3_features_csv;
```

Kopieren von Daten ohne Spaltenmapping

Sie können Daten aus DynamoDB in einem unformatierten Format kopieren und sie ohne Angabe von Datentypen oder Spaltenmapping in Amazon S3 schreiben. Sie können diese Methode zum Erstellen eines Archivs von DynamoDB-Daten und zum Speichern in Amazon S3 verwenden.

Example Von DynamoDB zu Amazon S3

1. Erstellen Sie eine externe Tabelle im Zusammenhang mit Ihrer DynamoDB-Tabelle. (In dieser HiveQL-Anweisung ist keine `dynamodb.column.mapping` enthalten.)

   ```
   CREATE EXTERNAL TABLE ddb_features_no_mapping
       (item MAP<STRING, STRING>)
   STORED BY 'org.apache.hadoop.hive.dynamodb.DynamoDBStorageHandler'
   TBLPROPERTIES ("dynamodb.table.name" = "Features");
   ```

2. Erstellen Sie eine weitere externe Tabelle im Zusammenhang mit Ihrem Amazon-S3-Bucket.

   ```
   CREATE EXTERNAL TABLE s3_features_no_mapping
       (item MAP<STRING, STRING>)
   ROW FORMAT DELIMITED
   FIELDS TERMINATED BY '\t'
   LINES TERMINATED BY '\n'
   LOCATION 's3://aws-logs-123456789012-us-west-2/hive-test';
   ```

3. Kopieren Sie die Daten aus DynamoDB in Amazon S3.

   ```
   INSERT OVERWRITE TABLE s3_features_no_mapping
   SELECT * FROM ddb_features_no_mapping;
   ```

Die Datendatei in Amazon S3 sieht folgendermaßen aus:

```
Name^C{"s":"Soldiers Farewell
 Hill"}^BState^C{"s":"NM"}^BClass^C{"s":"Summit"}^BElevation^C{"n":"6135"}^BLatitude^C{"n":"32.
Name^C{"s":"Jones
 Run"}^BState^C{"s":"PA"}^BClass^C{"s":"Stream"}^BElevation^C{"n":"1260"}^BLatitude^C{"n":"41.2
```

```
Name^C{"s":"Sentinel
 Dome"}^BState^C{"s":"CA"}^BClass^C{"s":"Summit"}^BElevation^C{"n":"8133"}^BLatitude^C{"n":"37.
Name^C{"s":"Neversweet
 Gulch"}^BState^C{"s":"CA"}^BClass^C{"s":"Valley"}^BElevation^C{"n":"2900"}^BLatitude^C{"n":"41
Name^C{"s":"Chacaloochee
 Bay"}^BState^C{"s":"AL"}^BClass^C{"s":"Bay"}^BElevation^C{"n":"0"}^BLatitude^C{"n":"30.6979676
```

Jedes Feld beginnt mit einem STX-Zeichen (Textanfang, 0x02) und endet mit einem ETX-Zeichen (Textende, 0x03). In der Datei wird für STX **^B** und für ETX **^C** angegeben.

Example Von Amazon S3 zu DynamoDB

Mit einer einzigen HiveQL-Anweisung können Sie die DynamoDB-Tabelle mit den Daten aus Amazon S3: auffüllen:

```
INSERT OVERWRITE TABLE ddb_features_no_mapping
SELECT * FROM s3_features_no_mapping;
```

Anzeigen der Daten in Amazon S3

Wenn Sie eine Verbindung zum Leader-Knoten über SSH herstellen, können Sie die AWS Command Line Interface (AWS CLI) für den Zugriff auf die Daten verwenden, die von Hive in Amazon S3 geschrieben wurden.

Die folgenden Schritte setzen voraus, dass Sie die Daten aus DynamoDB in Amazon S3 mit einem der in diesem Abschnitt beschriebenen Verfahren kopiert haben.

1. Wenn Sie sich an der Hive-Eingabeaufforderung befinden, beenden Sie die Linux-Eingabeaufforderung.

   ```
   hive> exit;
   ```

2. Führen Sie den Inhalt des Hive-Testverzeichnisses in Ihrem Amazon-S3-Bucket auf. (Dies ist der Speicherort, an den Hive die Daten aus DynamoDB kopiert hat.)

   ```
   aws s3 ls s3://aws-logs-123456789012-us-west-2/hive-test/
   ```

   Die Antwort sollte wie folgt aussehen:

   ```
   2016-11-01 23:19:54 81983 000000_0
   ```

Der Dateiname (000000_0) wird vom System generiert.

3. (Optional) Sie können die Datendatei aus Amazon S3 in das lokale Dateisystem auf dem Leader-Knoten kopieren. Anschließend können Sie mithilfe von Standardbefehlszeilen-Dienstprogrammen von Linux mit den Daten in der Datei arbeiten.

```
aws s3 cp s3://aws-logs-123456789012-us-west-2/hive-test/000000_0 .
```

Die Antwort sollte wie folgt aussehen:

```
download: s3://aws-logs-123456789012-us-west-2/hive-test/000000_0
to ./000000_0
```

> **Note**
>
> Das lokale Dateisystem auf dem Leader-Knoten verfügt über begrenzte Kapazität. Verwenden Sie diesen Befehl nicht mit Dateien, die größer als der im lokalen Dateisystem verfügbare Platz sind.

Kopieren von Daten zwischen DynamoDB und HDFS

Wenn Sie Daten in einer DynamoDB-Tabelle vorliegen haben, können Sie sie mit Hive in das Hadoop Distributed File System (HDFS) kopieren.

Sie können dies tun, wenn Sie einen MapReduce Job ausführen, für den Daten von DynamoDB erforderlich sind. Wenn Sie die Daten aus DynamoDB in HDFS kopieren, kann Hadoop sie verarbeiten, indem alle verfügbaren Knoten im Amazon EMR-Cluster parallel verwendet werden. Wenn der MapReduce Job abgeschlossen ist, können Sie die Ergebnisse von HDFS nach DDB schreiben.

In den folgenden Beispielen verwendet Hive das nachstehende HDFS-Verzeichnis für Lese- und Schreibvorgänge: `/user/hadoop/hive-test`

> **Note**
>
> Die Beispiele in diesem Abschnitt setzen voraus, dass Sie die Schritte in Tutorial: Arbeiten mit Amazon DynamoDB und Apache Hive befolgt haben und über eine externe Tabelle, die in DynamoDB verwaltet wird (ddb_features), verfügen.

Themen

- Kopieren von Daten mit dem Hive-Standardformat
- Kopieren von Daten in einem benutzerdefinierten Format
- Kopieren von Daten ohne Spaltenmapping
- Zugriff auf die Daten in HDFS

Kopieren von Daten mit dem Hive-Standardformat

Example Von DynamoDB in HDFS

Verwenden Sie eine `INSERT OVERWRITE`-Anweisung, um direkt in HDFS zu schreiben.

```
INSERT OVERWRITE DIRECTORY 'hdfs:///user/hadoop/hive-test'
SELECT * FROM ddb_features;
```

Die Datendatei in HDFS sieht folgendermaßen aus:

```
920709^ASoldiers Farewell Hill^ASummit^ANM^A32.3564729^A-108.33004616135
1178153^AJones Run^AStream^APA^A41.2120086^A-79.25920781260
253838^ASentinel Dome^ASummit^ACA^A37.7229821^A-119.584338133
264054^ANeversweet Gulch^AValley^ACA^A41.6565269^A-122.83614322900
115905^AChacaloochee Bay^ABay^AAL^A30.6979676^A-87.97388530
```

Jedes Feld wird durch ein SoH-Zeichen getrennt (Anfang der Überschrift, 0x01). In der Datei wird SoH als **^A** angezeigt.

Example Kopieren von HDFS in DynamoDB

1. Erstellen Sie eine externe Tabelle, die den unformatierten Daten in HDFS zugeordnet ist.

   ```
   CREATE EXTERNAL TABLE hdfs_features_unformatted
   ```

```
    (feature_id       BIGINT,
    feature_name      STRING ,
    feature_class     STRING ,
    state_alpha       STRING,
    prim_lat_dec      DOUBLE ,
    prim_long_dec     DOUBLE ,
    elev_in_ft        BIGINT)
LOCATION 'hdfs:///user/hadoop/hive-test';
```

2. Kopieren Sie die Daten in DynamoDB

```
INSERT OVERWRITE TABLE ddb_features
SELECT * FROM hdfs_features_unformatted;
```

Kopieren von Daten in einem benutzerdefinierten Format

Wenn Sie ein anderes Feldtrennzeichen verwenden möchten, können Sie eine externe Tabelle erstellen, die dem HDFS-Verzeichnis zugeordnet ist. Sie können diese Vorgehensweise zum Erstellen von Datendateien mit CSV-Werten verwenden.

Example Von DynamoDB in HDFS

1. Erstellen Sie eine externe Hive-Tabelle, die HDFS zugeordnet ist. Stellen Sie dabei sicher, dass die Datentypen mit den Typen der externen DynamoDB-Tabelle konsistent sind.

```
CREATE EXTERNAL TABLE hdfs_features_csv
    (feature_id       BIGINT,
    feature_name      STRING ,
    feature_class     STRING ,
    state_alpha       STRING,
    prim_lat_dec      DOUBLE ,
    prim_long_dec     DOUBLE ,
    elev_in_ft        BIGINT)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ','
LOCATION 'hdfs:///user/hadoop/hive-test';
```

2. Kopieren Sie die Daten aus DynamoDB.

```
INSERT OVERWRITE TABLE hdfs_features_csv
SELECT * FROM ddb_features;
```

Die Datendatei in HDFS sieht folgendermaßen aus:

```
920709,Soldiers Farewell Hill,Summit,NM,32.3564729,-108.3300461,6135
1178153,Jones Run,Stream,PA,41.2120086,-79.2592078,1260
253838,Sentinel Dome,Summit,CA,37.7229821,-119.58433,8133
264054,Neversweet Gulch,Valley,CA,41.6565269,-122.8361432,2900
115905,Chacaloochee Bay,Bay,AL,30.6979676,-87.9738853,0
```

Example Kopieren von HDFS in DynamoDB

Mit einer einzigen HiveQL-Anweisung können Sie die DynamoDB-Tabelle mit den Daten aus HDFS auffüllen:

```
INSERT OVERWRITE TABLE ddb_features
SELECT * FROM hdfs_features_csv;
```

Kopieren von Daten ohne Spaltenmapping

Sie können Daten aus DynamoDB in einem unformatierten Format kopieren und sie ohne Angabe von Datentypen oder Spaltenmapping in HDFS schreiben. Sie können diese Methode zum Erstellen eines Archivs von DynamoDB-Daten und zum Speichern in HDFS verwenden.

> **Note**
>
> Wenn Ihre DynamoDB-Tabelle Attribute vom Typ Zuordnung, Liste, Boolescher Wert oder Null enthält, ist dies die einzige Möglichkeit, um Hive zum Kopieren von Daten aus DynamoDB in HDFS zu verwenden.

Example Von DynamoDB in HDFS

1. Erstellen Sie eine externe Tabelle im Zusammenhang mit Ihrer DynamoDB-Tabelle. (In dieser HiveQL-Anweisung ist keine `dynamodb.column.mapping` enthalten.)

   ```
   CREATE EXTERNAL TABLE ddb_features_no_mapping
       (item MAP<STRING, STRING>)
   STORED BY 'org.apache.hadoop.hive.dynamodb.DynamoDBStorageHandler'
   TBLPROPERTIES ("dynamodb.table.name" = "Features");
   ```

2. Erstellen Sie eine weitere externe Tabelle im Zusammenhang mit Ihrem HDFS-Verzeichnis.

```
CREATE EXTERNAL TABLE hdfs_features_no_mapping
    (item MAP<STRING, STRING>)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY '\t'
LINES TERMINATED BY '\n'
LOCATION 'hdfs:///user/hadoop/hive-test';
```

3. Kopieren Sie die Daten aus DynamoDB in HDFS.

```
INSERT OVERWRITE TABLE hdfs_features_no_mapping
SELECT * FROM ddb_features_no_mapping;
```

Die Datendatei in HDFS sieht folgendermaßen aus:

```
Name^C{"s":"Soldiers Farewell
 Hill"}^BState^C{"s":"NM"}^BClass^C{"s":"Summit"}^BElevation^C{"n":"6135"}^BLatitude^C{"n":"32.
Name^C{"s":"Jones
 Run"}^BState^C{"s":"PA"}^BClass^C{"s":"Stream"}^BElevation^C{"n":"1260"}^BLatitude^C{"n":"41.2
Name^C{"s":"Sentinel
 Dome"}^BState^C{"s":"CA"}^BClass^C{"s":"Summit"}^BElevation^C{"n":"8133"}^BLatitude^C{"n":"37.
Name^C{"s":"Neversweet
 Gulch"}^BState^C{"s":"CA"}^BClass^C{"s":"Valley"}^BElevation^C{"n":"2900"}^BLatitude^C{"n":"41
Name^C{"s":"Chacaloochee
 Bay"}^BState^C{"s":"AL"}^BClass^C{"s":"Bay"}^BElevation^C{"n":"0"}^BLatitude^C{"n":"30.6979676
```

Jedes Feld beginnt mit einem STX-Zeichen (Textanfang, 0x02) und endet mit einem ETX-Zeichen (Textende, 0x03). In der Datei wird für STX **^B** und für ETX **^C** angegeben.

Example Kopieren von HDFS in DynamoDB

Mit einer einzigen HiveQL-Anweisung können Sie die DynamoDB-Tabelle mit den Daten aus HDFS auffüllen:

```
INSERT OVERWRITE TABLE ddb_features_no_mapping
SELECT * FROM hdfs_features_no_mapping;
```

Zugriff auf die Daten in HDFS

HDFS ist ein verteiltes Dateisystem, auf das alle Knoten im Amazon EMR-Cluster zugreifen können. Wenn Sie eine Verbindung zum Leader-Knoten über SSH herstellen, können Sie die Befehlszeilen-Tools für den Zugriff auf die Daten verwenden, die von Hive in HDFS geschrieben wurden.

HDFS ist nicht mit dem lokalen Dateisystem auf dem Leader-Knoten identisch. Sie können nicht mit Dateien und Verzeichnissen in HDFS arbeiten und dabei die Linux-Standardbefehle verwenden (wie `cat`, `cp`, `mv` oder `rm`). Verwenden Sie stattdessen den Befehl `hadoop fs` für diese Aufgaben.

Die folgenden Schritte setzen voraus, dass Sie die Daten aus DynamoDB in HDFS mit einem der in diesem Abschnitt beschriebenen Verfahren kopiert haben.

1. Wenn Sie sich an der Hive-Eingabeaufforderung befinden, beenden Sie die Linux-Eingabeaufforderung.

   ```
   hive> exit;
   ```

2. Listet den Inhalt des Verzeichnisses/user/hadoop/hive-test in HDFS auf. (Dies ist der Speicherort, an den Hive die Daten aus DynamoDB kopiert hat.)

   ```
   hadoop fs -ls /user/hadoop/hive-test
   ```

   Die Antwort sollte wie folgt aussehen:

   ```
   Found 1 items
   -rw-r--r-- 1 hadoop hadoop 29504 2016-06-08 23:40 /user/hadoop/hive-test/000000_0
   ```

   Der Dateiname (000000_0) wird vom System generiert.

3. Zeigen Sie den Inhalt der -Datei an:

   ```
   hadoop fs -cat /user/hadoop/hive-test/000000_0
   ```

   > **Note**
   >
   > In diesem Beispiel ist die Datei relativ klein (etwa 29 KB). Seien Sie vorsichtig, wenn Sie diesen Befehl mit Dateien verwenden, die sehr groß sind oder nicht druckbare Zeichen enthalten.

4. (Optional) Sie können die Datendatei aus HDFS in das lokale Dateisystem auf dem Leader-Knoten kopieren. Anschließend können Sie mithilfe von Standardbefehlszeilen-Dienstprogrammen von Linux mit den Daten in der Datei arbeiten.

```
hadoop fs -get /user/hadoop/hive-test/000000_0
```

Mit diesem Befehl wird die Datei nicht überschrieben.

> **Note**
>
> Das lokale Dateisystem auf dem Leader-Knoten verfügt über begrenzte Kapazität. Verwenden Sie diesen Befehl nicht mit Dateien, die größer als der im lokalen Dateisystem verfügbare Platz sind.

Verwenden der Datenkomprimierung

Wenn Sie Hive verwenden, um Daten zwischen verschiedenen Datenquellen zu kopieren, können Sie eine Datenkomprimierung anfordern on-the-fly. Hive bietet mehrere Kompressions-Codecs. Sie können einen Codec während der Hive-Sitzung auswählen. Die Daten werden dann in dem angegebenen Format komprimiert.

Im folgenden Beispiel werden Daten mithilfe des Lempel-Ziv-Oberhumer (LZO) -Algorithmus komprimiert.

```
SET hive.exec.compress.output=true;
SET io.seqfile.compression.type=BLOCK;
SET mapred.output.compression.codec = com.hadoop.compression.lzo.LzopCodec;

CREATE EXTERNAL TABLE lzo_compression_table (line STRING)
ROW FORMAT DELIMITED FIELDS TERMINATED BY '\t' LINES TERMINATED BY '\n'
LOCATION 's3://bucketname/path/subpath/';

INSERT OVERWRITE TABLE lzo_compression_table SELECT *
FROM hiveTableName;
```

Die entsprechende Datei in Amazon S3 erhält einen vom System generierten Namen mit der Erweiterung `.lzo` am Ende (z.B. `8d436957-57ba-4af7-840c-96c2fc7bb6f5-000000.lzo`).

Die verfügbaren Kompressions-Codecs sind:

- `org.apache.hadoop.io.compress.GzipCodec`
- `org.apache.hadoop.io.compress.DefaultCodec`
- `com.hadoop.compression.lzo.LzoCodec`
- `com.hadoop.compression.lzo.LzopCodec`
- `org.apache.hadoop.io.compress.BZip2Codec`
- `org.apache.hadoop.io.compress.SnappyCodec`

Lesen von nicht druckbaren UTF-8-Zeichendaten

Zum Lesen und Schreiben von nicht druckbaren UTF-8-Zeichendaten können Sie beim Erstellen einer Hive-Tabelle die `STORED AS SEQUENCEFILE`-Klausel verwenden. A SequenceFile ist ein Hadoop-Binärdateiformat. Sie müssen Hadoop zum Lesen dieser Datei verwenden. Das folgende Beispiel zeigt, wie Daten aus DynamoDB nach Amazon S3 exportiert werden. Sie können diese Funktionalität für die Verarbeitung von nicht druckbaren UTF-8-kodierten Zeichen verwenden.

```
CREATE EXTERNAL TABLE s3_export(a_col string, b_col bigint, c_col array<string>)
STORED AS SEQUENCEFILE
LOCATION 's3://bucketname/path/subpath/';

INSERT OVERWRITE TABLE s3_export SELECT *
FROM hiveTableName;
```

Leistungsoptimierung

Wenn Sie eine externe Hive-Tabelle erstellen, die einer DynamoDB-Tabelle zugeordnet ist, wird keine Lese- oder Schreibkapazität von DynamoDB belegt. Lese- und Schreibaktivitäten in der Hive-Tabelle (wie `INSERT` oder `SELECT`) wirken sich jedoch direkt als Lese- und Schreibvorgänge in der zugrunde liegenden DynamoDB-Tabelle aus.

Apache Hive auf Amazon EMR implementiert eine eigene Logik für den I/O Lastausgleich in der DynamoDB-Tabelle und versucht, die Möglichkeit zu minimieren, dass der bereitgestellte Durchsatz der Tabelle überschritten wird. Am Ende jeder Hive-Abfrage gibt Amazon EMR Laufzeitmetriken zurück, einschließlich der Anzahl von Überschreitungen des bereitgestellten Durchsatzes. Sie können diese Informationen zusammen mit CloudWatch Metriken in Ihrer DynamoDB-Tabelle verwenden, um die Leistung bei nachfolgenden Anfragen zu verbessern.

Die Amazon-EMR-Konsole bietet grundlegende Überwachungstools für Ihren Cluster. Weitere Informationen finden Sie unter Anzeigen und Überwachen eines Clusters im Managementleitfaden für Amazon EMR.

Außerdem können Sie Ihren Cluster und Hadoop-Aufträge mithilfe von webbasierten Tools wie Hue und Ganglia sowie der Hadoop-Webschnittstelle überwachen. Weitere Informationen finden Sie unter Anzeigen von auf Amazon-EMR-Clustern gehosteten Webschnittstellen im Management Guide für Amazon EMR.

In diesem Abschnitt werden Schritte beschrieben, wie Sie die Leistung von Hive-Operationen für externe DynamoDB-Tabellen optimieren können.

Themen

- DynamoDB – Bereitgestellter Durchsatz
- Anpassen der Mapper
- Weitere Themen

DynamoDB – Bereitgestellter Durchsatz

Wenn Sie HiveQL-Anweisungen für die externe DynamoDB-Tabelle erstellen, nimmt die Klasse `DynamoDBStorageHandler` die entsprechenden DynamoDB-Low-Level-API-Anforderungen vor, die den bereitgestellten Durchsatz verbrauchen. Wenn nicht genügend Lese- und Schreibkapazität für die DynamoDB-Tabelle zur Verfügung steht, wird die Anforderung gedrosselt und die HiveQL-Leistung so beeinträchtigt. Aus diesem Grund sollten Sie sicherstellen, dass die Tabelle über ausreichende Durchsatzkapazität verfügt.

Nehmen Sie zum Beispiel an, dass Sie 100 Lesekapazitätseinheiten für Ihre DynamoDB-Tabelle bereitgestellt haben. Mit dieser Kapazität können Sie 409 600 Byte pro Sekunde (100×4KB Lesekapazitätseinheitsgröße) lesen. Angenommen, die Tabelle enthält 20 GB Daten (21 474 836 480 Byte) und Sie möchten mit der `SELECT`-Anweisung alle Daten unter Verwendung von HiveQL auswählen. Sie können wie folgt schätzen, wie lange die Abfrage dauert:

21 474 836 480/409 600 = 52 429 Sekunden = 14,56 Stunden

In diesem Szenario stellt die DynamoDB-Tabelle einen Engpass dar. Das Hinzufügen weiterer Amazon-EMR-Knoten würde keine Abhilfe schaffen, da der Hive-Durchsatz auf nur 409,600 Byte pro Sekunde beschränkt ist. Die einzige Möglichkeit, um die Bearbeitungsdauer der `SELECT`-Anweisung zu verkürzen, besteht darin, die bereitgestellte Lesekapazität der DynamoDB-Tabelle zu erhöhen.

Sie können eine ähnliche Berechnung anstellen, um zu schätzen, wie lange es dauert, um Daten in eine externe Hive-Tabelle, die einer DynamoDB-Tabelle zugeordnet ist, massenzuladen. Ermitteln Sie die Gesamtzahl der pro Element benötigten Schreibkapazitätseinheiten (weniger als 1 KB = 1, 1 bis 2 KB = 2 usw.) und multiplizieren Sie diese mit der Anzahl der zu ladenden Elemente. Dadurch erhalten Sie die Anzahl der erforderlichen Schreibkapazitätseinheiten. Teilen Sie diese Anzahl von Schreibkapazitätseinheiten, die pro Sekunde zugewiesen werden. Daraus ergibt sich die Anzahl von Sekunden, die für das Laden der Tabelle benötigt wird.

Sie sollten die CloudWatch Metriken für Ihre Tabelle regelmäßig überwachen. Um einen kurzen Übersicht in der DynamoDB-Konsole zu erhalten, wählen Sie Ihre Tabelle und dann die Registerkarte Metrics aus. Hier können Sie verbrauchte Lese- und Schreibkapazitätseinheiten sowie gedrosselte Lese- und Schreibanforderungen einsehen.

Lesekapazität

Amazon EMR verwaltet die Anforderungslast für Ihre DynamoDB-Tabelle entsprechend der bereitgestellten Durchsatzeinstellungen der Tabelle. Wenn in der Auftragsausgabe eine große Zahl von `ProvisionedThroughputExceeded`-Nachrichten enthalten ist, können Sie die Standardleserate anpassen. Ändern Sie dazu die Konfigurationsvariable `dynamodb.throughput.read.percent`. Mit dem Befehl `SET` können Sie diese Variable an der Hive-Eingabeaufforderung festlegen:

```
SET dynamodb.throughput.read.percent=1.0;
```

Diese Variable gilt nur für die aktuelle Hive-Sitzung. Wenn Sie Hive beenden und später erneut aufrufen, erscheint für `dynamodb.throughput.read.percent` wieder der Standardwert.

Der Wert von `dynamodb.throughput.read.percent` kann einschließlich zwischen `0.1` und `1.5` sein. `0.5` stellt die Standardleserate dar. Dies bedeutet, dass Hive versucht, die Hälfte der Lesekapazität der Tabelle zu verbrauchen. Wenn Sie den Wert über `0.5` anheben, erhöht Hive die Anforderungsrate. Durch Senken des Werts unter `0.5` wird die Leseanforderungsrate verringert. (Die tatsächliche Leserate variiert abhängig von Faktoren wie der Tatsache, ob eine einheitliche Verteilung des Schlüssels in der DynamoDB-Tabelle vorliegt.)

Wenn Sie feststellen, dass Hive die bereitgestellte Lesekapazität der Tabelle häufig aufbraucht oder Ihre Leseanforderungen zu stark gedrosselt werden, versuchen Sie `dynamodb.throughput.read.percent` unter `0.5` zu reduzieren. Wenn die Lesekapazität der Tabelle ausreicht und Sie reaktionsfähigere HiveQL-Operationen wünschen, können Sie den Wert über `0.5` festlegen.

Schreibkapazität

Amazon EMR verwaltet die Anforderungslast für Ihre DynamoDB-Tabelle entsprechend der bereitgestellten Durchsatzeinstellungen der Tabelle. Wenn in der Auftragsausgabe eine große Zahl von `ProvisionedThroughputExceeded`-Nachrichten enthalten ist, können Sie die Standardschreibrate anpassen. Ändern Sie dazu die Konfigurationsvariable `dynamodb.throughput.write.percent`. Mit dem Befehl `SET` können Sie diese Variable an der Hive-Eingabeaufforderung festlegen:

```
SET dynamodb.throughput.write.percent=1.0;
```

Diese Variable gilt nur für die aktuelle Hive-Sitzung. Wenn Sie Hive beenden und später erneut aufrufen, erscheint für `dynamodb.throughput.write.percent` wieder der Standardwert.

Der Wert von `dynamodb.throughput.write.percent` kann einschließlich zwischen `0.1` und `1.5` sein. `0.5` stellt die Standardschreibrate dar. Dies bedeutet, dass Hive versucht, die Hälfte der Schreibkapazität der Tabelle zu verbrauchen. Wenn Sie den Wert über `0.5` anheben, erhöht Hive die Anforderungsrate. Durch Senken des Werts unter `0.5` wird die Schreibanforderungsrate verringert. (Die tatsächliche Schreibrate variiert abhängig von Faktoren wie der Tatsache, ob eine einheitliche Verteilung des Schlüssels in der DynamoDB-Tabelle vorliegt.)

Wenn Sie feststellen, dass Hive die bereitgestellte Schreibkapazität der Tabelle häufig aufbraucht oder Ihre Schreibanforderungen zu stark gedrosselt werden, versuchen Sie `dynamodb.throughput.write.percent` unter `0.5` zu reduzieren. Wenn die Kapazität der Tabelle ausreicht und Sie reaktionsfähigere HiveQL-Operationen wünschen, können Sie den Wert über `0.5` festlegen.

Wenn Sie mit Hive Daten in DynamoDB schreiben möchten, stellen Sie sicher, dass die Anzahl der Schreibkapazitätseinheiten größer als die Anzahl der Mapper im Cluster ist. Betrachten Sie z. B. einen Amazon-EMR-Cluster mit 10m1.xlarge-Knoten. Der Knotentyp m1.xlarge stellt 8 Mapper-Aufgaben bereit, sodass der Cluster insgesamt 80 Mapper (10 × 8) enthält. Wenn Ihre DynamoDB-Tabelle weniger als 80 Schreibkapazitätseinheiten umfasst, kann ein Hive-Schreibvorgang möglicherweise den gesamten Schreibdurchsatz für diese Tabelle aufbrauchen.

Um die Anzahl der Mapper für Amazon-EMR-Knotentypen zu ermitteln, lesen Sie den Abschnitt Aufgabenkonfiguration im Amazon-EMR-Entwicklerhandbuch.

Weitere Informationen zu Mappern finden Sie unter Anpassen der Mapper.

Anpassen der Mapper

Wenn Hive einen Hadoop-Auftrag startet, wird dieser von einer oder mehreren Mapper-Aufgaben verarbeitet. Unter der Voraussetzung, dass Ihre DynamoDB-Tabelle über genügend Durchsatzkapazität verfügt, können Sie die Anzahl von Mappern im Cluster ändern und die Leistung so möglicherweise verbessern.

> **Note**
>
> Die Anzahl von Mapper-Aufgaben, die in einem Hadoop-Auftrag verwendet werden, wird von Input Splits beeinflusst, d. h. Hadoop teilt die Daten in logische Blöcke auf. Wenn Hadoop nicht genügend Input Splits vornimmt, können Ihre Schreibvorgänge möglicherweise nicht die gesamte Schreibdurchsatzkapazität der DynamoDB-Tabelle nutzen.

Erhöhen der Anzahl der Mapper

Jeder Mapper in einem Amazon EMR verfügt über eine maximale Leserate von 1 MiB pro Sekunde. Die Anzahl der Mapper in einem Cluster hängt von der Größe der Knoten in Ihrem Cluster ab. (Informationen über Knotengrößen und die Anzahl der Mapper pro Knoten finden Sie unter Aufgabenkonfiguration im Amazon EMR-Entwicklerhandbuch.)

Wenn Ihre DynamoDB-Tabelle über ausreichend Durchsatzkapazität für Lesevorgänge verfügt, können Sie versuchen, die Anzahl der Mapper erhöhen, indem Sie einen der folgenden Schritte ausführen:

- Erhöhen Sie die Größe der Knoten in Ihrem Cluster. Wenn Ihr Cluster z. B. Knoten vom Typ m1.large (drei Mapper pro Knoten) verwendet, können Sie auf m1.xlarge-Knoten (acht Mapper pro Knoten) erhöhen.
- Erhöhen Sie die Anzahl der Knoten in Ihrem Cluster. Wenn Sie z. B. über ein Drei-Knoten-Cluster mit m1.xlarge-Knoten verfügen, stehen insgesamt 24 Mapper zur Verfügung. Wenn Sie die Größe des Clusters mit demselben Knotentyp verdoppeln, erhalten Sie 48 Mapper.

Sie können den verwenden AWS-Managementkonsole , um die Größe oder Anzahl der Knoten in Ihrem Cluster zu verwalten. (Sie müssen den Cluster möglicherweise neu starten, damit diese Änderungen wirksam werden.)

Eine andere Möglichkeit, die Anzahl der Mapper zu erhöhen, besteht darin, den `mapred.tasktracker.map.tasks.maximum` Hadoop-Konfigurationsparameter zu ändern. (Dies

ist ein Hadoop- und kein Hive-Parameter. Sie können ihn nicht interaktiv an der Eingabeaufforderung ändern.) Wenn Sie den Wert `mapred.tasktracker.map.tasks.maximum` erhöhen, steigern Sie die Anzahl der Mapper ohne die Größe oder Anzahl der Knoten zu ändern. Allerdings kann es passieren, dass der Speicherplatz der Cluster-Knoten nicht mehr ausreicht, wenn Sie den Wert zu hoch festlegen.

Sie legen den Wert für `mapred.tasktracker.map.tasks.maximum` als Bootstrap-Aktion fest, wenn Sie Ihren Amazon-EMR-Cluster das erste Mal starten. Weitere Informationen finden Sie unter (Optional) Erstellen von Bootstrap-Aktionen zum Installieren zusätzlicher Software im Management Guide für Amazon EMR.

Reduzieren der Anzahl der Mapper

Wenn Sie die `SELECT`-Anweisung verwenden, um Daten aus einer externen Hive-Tabelle auszuwählen, die DynamoDB zugeordnet ist, kann der Hadoop-Auftrag so viele Aufgaben nutzen wie erforderlich, und zwar bis zur maximalen Anzahl der Mapper im Cluster. In diesem Szenario ist es möglich, dass eine zeitaufwändige Hive-Abfrage die gesamte bereitgestellte Lesekapazität der DynamoDB-Tabelle nutzt, was negative Auswirkungen auf andere Benutzer hat.

Sie können den Parameter `dynamodb.max.map.tasks` nutzen, ob eine Obergrenze für Zuordnungsaufgaben festzulegen:

```
SET dynamodb.max.map.tasks=1
```

Dieser Wert muss gleich oder größer 1 sein. Wenn Hive Ihre Abfrage verarbeitet, verbraucht der entsprechende Hadoop-Auftrag beim Lesen aus der DynamoDB-Tabelle nicht mehr als `dynamodb.max.map.tasks`.

Weitere Themen

Im Folgenden werden weitere Möglichkeiten zum Optimieren von Anwendungen beschrieben, die Hive für den Zugriff auf DynamoDB verwenden.

Retry duration

Standardmäßig führt Hive einen Hadoop-Auftrag erneut aus, wenn innerhalb von zwei Minuten keine Ergebnisse von DynamoDB zurückgegeben werden. Sie können diesen Zeitraum durch Ändern des Parameters `dynamodb.retry.duration` anpassen:

```
SET dynamodb.retry.duration=2;
```

Der Wert muss eine Ganzzahl ungleich Null sein, die die Anzahl der Minuten im Wiederholungsintervall darstellt. Der Standardwert für `dynamodb.retry.duration` ist 2 (Minuten).

Parallele Datenanforderungen

Mehrere Datenanforderungen, entweder von mehr als einem Benutzer oder mehr als einer Anwendung, an eine einzelne Tabelle kann den bereitgestellten Lesedurchsatz erschöpfen und die Leistung beeinträchtigen.

Prozessdauer

Die Datenkonsistenz in DynamoDB hängt von der Reihenfolge der Lese- und Schreibvorgänge auf den einzelnen Knoten ab. Während eine Hive-Abfrage verarbeitet wird, kann eine andere Anwendung neue Daten in die DynamoDB-Tabelle laden oder vorhandene Daten ändern oder löschen. In diesem Fall enthalten die Ergebnisse der Hive-Abfrage möglicherweise nicht die Datenänderungen, die vorgenommen wurden, während die Abfrage ausgeführt wurde.

Abfragezeit

Wenn Hive-Abfragen, die auf eine DynamoDB-Tabelle zugreifen, für Zeiten geplant werden, in denen wenig Anforderungen an die DynamoDB-Tabelle gerichtet werden, verbessert das die Leistung. Beispiel: Wenn die Mehrzahl der Benutzer Ihrer Anwendung in Hamburg leben, können Sie die täglichen Daten um 04:00 Uhr MEZ exportieren, wenn die meisten Benutzer schlafen und keine Datensätze in Ihrer DynamoDB-Datenbank aktualisieren.

Integration von DynamoDB in Amazon S3

Die Import- und Exportfunktionen von Amazon DynamoDB bieten eine einfache und effiziente Möglichkeit, Daten zwischen Amazon-S3- und DynamoDB-Tabellen zu verschieben, ohne Code schreiben zu müssen.

DynamoDB-Import- und Exportfunktionen helfen Ihnen beim Verschieben, Transformieren und Kopieren von DynamoDB-Tabellenkonten. Sie können aus Ihren S3-Quellen importieren und Ihre DynamoDB-Tabellendaten zu Amazon S3 exportieren und AWS-Services wie Athena, Amazon SageMaker AI und AWS Lake Formation verwenden, um Ihre Daten zu analysieren und umsetzbare Erkenntnisse zu gewinnen. Sie können Daten auch direkt in neue DynamoDB-Tabellen importieren, um neue Anwendungen mit skalierbarer Leistung im einstelligen Millisekundenbereich zu erstellen, den Datenaustausch zwischen Tabellen und Konten zu vereinfachen und Ihre Notfallwiederherstellungs- und Business Continuity-Pläne zu vereinfachen.

Themen

- DynamoDB-Datenimport zu Amazon S3: Funktionsweise
- DynamoDB Datenexport zu Amazon S3: Funktionsweise

DynamoDB-Datenimport zu Amazon S3: Funktionsweise

Um Daten zu DynamoDB zu importieren, müssen sich Ihre Daten in einem Amazon S3-Bucket im CSV-, DynamoDB JSON- oder Amazon Ion-Format befinden. Daten können im ZSTD- oder GZIP-Format komprimiert oder direkt in unkomprimierter Form importiert werden. Quelldaten können entweder ein einzelnes Amazon-S3-Objekt oder mehrere Amazon-S3-Objekte sein, die dasselbe Präfix verwenden.

Ihre Daten werden in eine neue DynamoDB-Tabelle importiert, die erstellt wird, wenn Sie die Importanforderung initiieren. Sie können diese Tabelle mit sekundären Indizes erstellen und dann Ihre Daten über alle primären und sekundären Indizes hinweg abfragen und aktualisieren, sobald der Import abgeschlossen ist. Sie können auch ein globales Tabellenreplikat hinzufügen, nachdem der Import abgeschlossen ist.

> **Note**
>
> Während des Amazon-S3-Importvorgangs erstellt DynamoDB eine neue Zieltabelle, in die importiert wird. Der Import in vorhandene Tabellen wird von dieser Funktion derzeit nicht unterstützt.

Der Import aus Amazon S3 verbraucht keine Schreibkapazität für die neue Tabelle, so dass Sie keine zusätzliche Kapazität für den Import von Daten in DynamoDB bereitstellen müssen. Die Preise für Datenimporte basieren auf der unkomprimierten Größe der Quelldaten in Amazon S3, die als Ergebnis des Imports verarbeitet werden. Elemente, die verarbeitet werden, aber aufgrund von Formatierungen oder anderen Inkonsistenzen in den Quelldaten nicht in die Tabelle geladen werden können, werden im Rahmen des Importvorgangs ebenfalls in Rechnung gestellt. Siehe Amazon-DynamoDB-Preise für weitere Informationen.

Sie können Daten in einen S3-Bucket exportieren, der zu einem anderen Konto gehört, wenn Sie über die korrekten Schreibberechtigungen für den jeweiligen Bucket verfügen. Die neue Tabelle kann sich auch in einer anderen Region als die des Quell-Buckets aus Amazon S3 befinden. Weitere Informationen finden Sie unter Amazon Simple Storage Service – Einrichtung und Berechtigungen .

Die Importzeiten stehen in direktem Zusammenhang mit den Merkmalen Ihrer Daten in Amazon S3. Dazu gehören Datengröße, Datenformat, Komprimierungsschema, Einheitlichkeit der Datenverteilung, Anzahl der Amazon-S3-Objekte und andere zugehörige Variablen. Insbesondere Datensätze mit gleichmäßig verteilten Schlüsseln lassen sich schneller importieren als solche ohne diese. Zum Beispiel: Wenn der Schlüssel Ihres sekundären Index den Monat des Jahres für die Partitionierung verwendet und alle Ihre Daten aus dem Monat Dezember stammen, kann der Import dieser Daten deutlich länger dauern.

Es wird erwartet, dass die mit Schlüsseln verknüpften Attribute in der Basistabelle eindeutig sind. Wenn Schlüssel nicht eindeutig sind, werden beim Import die zugehörigen Elemente überschrieben, bis nur die letzte Überschreibung übrig bleibt. Wenn der Primärschlüssel beispielsweise der Monat ist und mehrere Elemente auf den Monat September festgelegt sind, überschreibt jedes neue Element die zuvor geschriebenen Elemente und es bleibt nur ein Element übrig, dessen Primärschlüssel (Monat) auf September festgelegt ist. In solchen Fällen stimmt die Anzahl der verarbeiteten Elemente in der Beschreibung der Importtabelle nicht mit der Anzahl der Elemente in der Zieltabelle überein.

AWS CloudTrail protokolliert alle Konsolen- und API-Aktionen für den Tabellenimport. Weitere Informationen finden Sie unter Protokollieren von DynamoDB-Vorgängen mithilfe von AWS CloudTrail.

Das folgende Video gibt eine Einführung in den direkten Import aus Amazon S3 in DynamoDB.

Importieren aus Amazon S3

Themen

- Anfordern eines Tabellenexports in DynamoDB
- Amazon-S3-Importformate für DynamoDB
- Importformatkontingente und Validierung
- Bewährte Methoden für den Import aus Amazon S3 in DynamoDB

Anfordern eines Tabellenexports in DynamoDB

Der DynamoDB-Import ermöglicht Ihnen, Daten aus einem Amazon-S3-Bucket in eine neue DynamoDB-Tabelle zu importieren. Sie können einen Tabellenimport mit der DynamoDB-Konsole, der CLI CloudFormationoder der DynamoDB-API anfordern.

Wenn Sie den verwenden möchten AWS CLI, müssen Sie ihn zuerst konfigurieren. Weitere Informationen finden Sie unter Zugreifen auf DynamoDB.

Note

- Die Funktion „Tabelle importieren“ interagiert mit mehreren verschiedenen AWS Diensten wie Amazon S3 und CloudWatch. Bevor Sie mit einem Import beginnen, stellen Sie sicher, dass der Benutzer oder die Rolle, die den Import aufruft, APIs über Berechtigungen für alle Dienste und Ressourcen verfügt, von denen die Funktion abhängt.
- Während des Imports dürfen die Amazon-S3-Objekte nicht geändert werden, da dies zum Fehlschlagen oder Abbruch des Vorgangs führen kann.

Weitere Informationen zu Fehlern und zur Fehlerbehebung finden Sie unter Importformatkontingente und Validierung

Themen

- Einrichten von IAM-Berechtigungen
- Beantragen eines Imports mit dem AWS-Managementkonsole
- Details zu früheren Importen finden Sie in AWS-Managementkonsole
- Beantragen eines Imports mit dem AWS CLI
- Einzelheiten zu vergangenen Importen finden Sie in AWS CLI

Einrichten von IAM-Berechtigungen

Sie können Daten aus jedem Amazon S3-Bucket importieren, für den Sie über Leseberechtigung verfügen. Der Quell-Bucket muss sich nicht in derselben Region befinden oder denselben Besitzer wie die Quelltabelle haben. Ihr AWS Identity and Access Management (IAM) muss die relevanten Aktionen für den Amazon S3 S3-Quell-Bucket und die erforderlichen CloudWatch Berechtigungen für die Bereitstellung von Debugging-Informationen enthalten. Nachfolgend eine Beispielrichtlinie.

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Sid": "AllowDynamoDBImportAction",
      "Effect": "Allow",
```

```
      "Action": [
        "dynamodb:ImportTable",
        "dynamodb:DescribeImport"
      ],
      "Resource": "arn:aws:dynamodb:us-east-1:111122223333:table/my-table*"
    },
    {
      "Sid": "AllowS3Access",
      "Effect": "Allow",
      "Action": [
        "s3:GetObject",
        "s3:ListBucket"
      ],
      "Resource": [
        "arn:aws:s3:::your-bucket/*",
        "arn:aws:s3:::your-bucket"
      ]
    },
    {
      "Sid": "AllowCloudwatchAccess",
      "Effect": "Allow",
      "Action": [
        "logs:CreateLogGroup",
        "logs:CreateLogStream",
        "logs:DescribeLogGroups",
        "logs:DescribeLogStreams",
        "logs:PutLogEvents",
        "logs:PutRetentionPolicy"
      ],
      "Resource": "arn:aws:logs:us-east-1:111122223333:log-group/aws-dynamodb/*"
    },
    {
      "Sid": "AllowDynamoDBListImports",
      "Effect": "Allow",
      "Action": "dynamodb:ListImports",
      "Resource": "*"
    }
  ]
}
```

Amazon-S3-Berechtigungen

Wenn Sie einen Import auf einer Bucket-Quelle aus Amazon S3 starten, die einem anderen Konto gehört, stellen Sie sicher, dass die Rolle oder der Benutzer Zugriff auf die Amazon-S3-Objekte hat. Sie können dies überprüfen, indem Sie einen Amazon-S3–Befehl `GetObject` ausführen und die Anmeldeinformationen verwenden. Bei Verwendung der API wird für den Bucket-Besitzer-Parameter von Amazon S3 standardmäßig die Konto-ID des aktuellen Benutzers verwendet. Stellen Sie bei kontoübergreifenden Importen sicher, dass dieser Parameter korrekt mit der Konto-ID des Bucket-Besitzers ausgefüllt ist. Der folgende Code ist ein Beispiel für eine S3-Bucket-Richtlinie im Quellkonto.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {"Sid": "ExampleStatement",
            "Effect": "Allow",
            "Principal": {"AWS": "arn:aws:iam::123456789012:user/Dave"
            },
            "Action": [
                "s3:GetObject",
                "s3:ListBucket"
            ],
            "Resource": "arn:aws:s3:::amzn-s3-demo-bucket/*"
        }
    ]
}
```

AWS Key Management Service

Wenn Sie beim Erstellen der neuen Tabelle für den Import einen Schlüssel zur Verschlüsselung im Ruhezustand auswählen, der nicht DynamoDB gehört, müssen Sie die erforderlichen AWS KMS Berechtigungen für den Betrieb einer DynamoDB-Tabelle bereitstellen, die mit vom Kunden verwalteten Schlüsseln verschlüsselt ist. Weitere Informationen finden Sie unter Autorisieren der Verwendung Ihres Schlüssels. AWS KMS Wenn die Amazon S3 S3-Objekte mit serverseitiger Verschlüsselung KMS (SSE-KMS) verschlüsselt sind, stellen Sie sicher, dass die Rolle oder der Benutzer, der den Import initiiert, Zugriff auf die Entschlüsselung mit dem Schlüssel hat. AWS

KMS Diese Funktion unterstützt keine Amazon-S3-Objekte, die mit vom Kunden bereitgestellten Verschlüsselungsschlüsseln (SSE-C) verschlüsselt sind.

CloudWatch Berechtigungen

Die Rolle oder der Benutzer, die/der den Import initiiert, benötigt Erstellungs- und Verwaltungsberechtigungen für die Protokollgruppe und die Protokollstreams, die mit dem Import verknüpft sind.

Beantragen eines Imports mit dem AWS-Managementkonsole

Das folgende Beispiel zeigt, wie Sie mit der DynamoDB-Konsole vorhandene Daten in eine neue Tabelle mit der Bezeichnung `MusicCollection` importieren.

So fordern Sie einen Tabellenimport an:

1. Melden Sie sich bei der an AWS-Managementkonsole und öffnen Sie die DynamoDB-Konsole unter. https://console.aws.amazon.com/dynamodb/
2. Klicken Sie im Navigationsbereich links in der Konsole auf Import from S3 (Import aus S3).
3. Wählen Sie auf der angezeigten Seite Import from S3 (Import aus S3).
4. Wählen Sie Import aus S3 aus.
5. Geben Sie im Quell-S3-URL die Quell-URL aus Amazon S3 ein.

   Wenn Sie Eigentümer des Quell-Buckets sind, wählen Sie Browse S3 aus, um danach zu suchen. Geben Sie alternativ die URL des Buckets im folgenden Format ein – `s3://bucket/prefix`. Das `prefix` ist ein Amazon-S3-Schlüsselpräfix. Es ist entweder der Amazon-S3-Objektname, den Sie importieren möchten, oder das gemeinsame Schlüsselpräfix aller Amazon-S3-Objekte, die Sie importieren möchten.

   > **Note**
   >
   > Sie können nicht dasselbe Präfix wie Ihre DynamoDB-Exportanforderung verwenden. Die Exportfunktion erstellt eine Ordnerstruktur und Manifestdateien für alle Exporte. Wenn Sie denselben Amazon-S3-Pfad verwenden, führt dies zu einem Fehler.
   > Richten Sie den Import stattdessen auf den Ordner, der bereits Daten aus diesem Export enthält. Das Format des richtigen Pfads lautet in diesem Fall `s3://bucket/prefix/AWSDynamoDB/<XXXXXXXX-XXXXXX>/data/`, wobei `XXXXXXXX-XXXXXX` die Export-ID ist. Sie finden die Export-ID im Export-ARN, das das folgende Format hat – `arn:aws:dynamodb:<Region>:<AccountID>:table/`

<TableName>/export/<XXXXXXXX-XXXXXX>. Beispiel, arn:aws:dynamodb:*us-east-1*:*123456789012*:table/*ProductCatalog*/export/*01234567890123-a1b2c3d4*.

6. Geben Sie an, ob Sie der S3-Bucket-Besitzer sind. Wenn der Quell-Bucket einem anderen Konto gehört, wählen Sie Ein anderes AWS Konto aus. Geben Sie dann die Konto-ID-des Bucket-Besitzers ein.
7. Wählen Sie unter Import file compression (Importdatei-Komprimierung) entweder No compression (Keine Komprimierung), GZIP oder ZSTD.
8. Wählen Sie das entsprechende Importdateiformat aus. Die Optionen sind DynamoDB JSON, Amazon Ion oder CSV. Wenn Sie die Option CSV wählen, haben Sie zwei zusätzliche Optionen: CSV header (CSV-Kopfzeile) und CSV delimiter character (CSV-Trennzeichen).

   Wählen Sie für CSV header (CSV-Kopfzeile), ob die Kopfzeile entweder aus der ersten Zeile der Datei stammt oder angepasst werden soll. Wenn Sie die Option Customize your headers (Kopfzeilen anpassen) wählen, können Sie die Kopfzeilenwerte angeben, mit denen Sie importieren möchten. Durch diese Methode angegebene CSV-Kopfzeilen unterscheiden zwischen Groß- und Kleinschreibung und es wird erwartet, dass sie die Schlüssel der Zieltabelle enthalten.

   Bei CSV delimiter character (CSV-Trennzeichen) legen Sie das Zeichen fest, das die Elemente trennen soll. „Komma“ ist standardmäßig aktiviert. Wenn Sie Custom delimiter character (Benutzerdefiniertes Trennzeichen) auswählen, muss das Trennzeichen mit dem Regex-Muster übereinstimmen: `[,;:|\t ]`.
9. Wählen Sie die Schaltfläche Next (Weiter) und dann die Optionen für die neue Tabelle, die zum Speichern Ihrer Daten erstellt wird.

   > **Note**
   >
   > Primärschlüssel und Sortierschlüssel müssen mit den Attributen in der Datei übereinstimmen, sonst schlägt der Import fehl. Bei Attributen wird zwischen Groß- und Kleinschreibung unterschieden.

10. Wählen Sie erneut Next (Weiter), um Ihre Importoptionen zu überprüfen, und klicken Sie dann auf Import (Importieren), um die Importaufgabe zu starten. Sie sehen zunächst Ihre neue Tabelle unter „Tables (Tabellen)“ mit dem Status „Creating (Erstellen)“. Zu diesem Zeitpunkt ist die Tabelle nicht zugänglich.

11. Sobald der Import abgeschlossen ist, wird der Status als „Active (Aktiv)“ angezeigt und Sie können die Tabelle verwenden.

Details zu früheren Importen finden Sie in AWS-Managementkonsole

Informationen zu Importaufgaben, die Sie in der Vergangenheit ausgeführt haben, finden Sie, indem Sie auf Import from S3 (Import aus S3) in der Navigationsseitenleiste klicken und dann die Registerkarte Exports (Exporte) wählen. Das Importfenster enthält eine Liste aller Importe, die Sie in den letzten 90 Tagen erstellt haben. Wenn Sie den ARN einer Aufgabe auswählen, die auf der Registerkarte „Exports (Exporte)“ aufgeführt ist, werden Informationen über diesen Export abgerufen, einschließlich aller von Ihnen gewählten erweiterten Konfigurationseinstellungen.

Beantragen eines Imports mit dem AWS CLI

Im folgenden Beispiel werden CSV-formatierte Daten aus einem S3-Bucket mit der Bezeichnung „Bucket“ mit einem Präfix in eine neue Tabelle mit der Bezeichnung „target-Table“ importiert.

```
aws dynamodb import-table --s3-bucket-source S3Bucket=bucket,S3KeyPrefix=prefix \
            --input-format CSV --table-creation-parameters '{"TableName":"target-
table","KeySchema":  \
            [{"AttributeName":"hk","KeyType":"HASH"}],"AttributeDefinitions":
[{"AttributeName":"hk","AttributeType":"S"}],"BillingMode":"PAY_PER_REQUEST"}' \
            --input-format-options '{"Csv": {"HeaderList": ["hk", "title", "artist",
 "year_of_release"], "Delimiter": ";"}}'
```

> **Note**
>
> Wenn Sie Ihren Import mit einem durch AWS Key Management Service (AWS KMS) geschützten Schlüssel verschlüsseln möchten, muss sich der Schlüssel in derselben Region wie der Amazon S3 S3-Ziel-Bucket befinden.

Einzelheiten zu vergangenen Importen finden Sie in AWS CLI

Informationen zu Importaufgaben, die Sie in der Vergangenheit ausgeführt haben, finden Sie mithilfe des `list-imports`-Befehls. Dieser Befehl gibt eine Liste aller Importe zurück, die Sie in den letzten 90 Tagen erstellt haben. Beachten Sie, dass die Metadaten für die Importaufgabe nach 90 Tagen ablaufen und Aufträge, die älter sind als diese, in dieser Liste nicht mehr gefunden werden

können; DynamoDB löscht keine Objekte in Ihrem Amazon S3-Bucket oder in der Tabelle, die beim Importieren erstellt wurden.

```
aws dynamodb list-imports
```

Um detaillierte Informationen zu einer bestimmten Importaufgabe, einschließlich erweiterter Konfigurationseinstellungen, abzurufen, verwenden Sie den `describe-import`-Befehl.

```
aws dynamodb describe-import \
    --import-arn arn:aws:dynamodb:us-east-1:123456789012:table/ProductCatalog/exp
```

Amazon-S3-Importformate für DynamoDB

DynamoDB kann Daten in drei Formaten importieren: CSV, DynamoDB JSON und Amazon Ion.

Themen

- CSV
- DynamoDB JSON
- Amazon Ion

CSV

Eine Datei im CSV-Format besteht aus mehreren Elementen, die durch Zeilenumbrüche getrennt sind. Standardmäßig interpretiert DynamoDB die erste Zeile einer Importdatei als Kopfzeile und erwartet, dass Spalten durch Kommata getrennt werden. Sie können auch Kopfzeilen definieren, die angewendet werden, sofern sie mit der Anzahl der Spalten in der Datei übereinstimmen. Wenn Sie Kopfzeilen explizit definieren, wird die erste Zeile der Datei als Werte importiert.

> **Note**
>
> Beim Import aus CSV-Dateien werden alle Spalten außer dem Hash-Bereich und den Schlüsseln Ihrer Basistabelle und sekundären Indizes als DynamoDB-Zeichenfolgen importiert.

Maskieren von doppelten Anführungszeichen mit Escape-Zeichen

Alle doppelten Anführungszeichen in der CSV-Datei müssen mit Escape-Zeichen maskiert werden. Wenn sie nicht maskiert werden, wie im folgenden Beispiel, schlägt der Importvorgang fehl:

```
id,value
"123",Women's Full "Length" Dress
```

Derselbe Importvorgang ist erfolgreich, wenn die Anführungszeichen mit zwei Sätzen doppelter Anführungszeichen maskiert werden:

```
id,value
"""123""","Women's Full ""Length"" Dress"
```

Sobald der Text ordnungsgemäß maskiert und importiert wurde, wird er wie in der ursprünglichen CSV-Datei angezeigt:

```
id,value
"123",Women's Full "Length" Dress
```

Heterogene Artikeltypen importieren

Sie können eine einzelne CSV-Datei verwenden, um verschiedene Elementtypen in eine Tabelle zu importieren. Definieren Sie eine Kopfzeile, die alle Attribute Ihrer Artikeltypen enthält, und lassen Sie die Spalten leer für Attribute, die nicht für einen bestimmten Artikel gelten. Leere Spalten werden im importierten Element weggelassen und nicht als leere Zeichenketten gespeichert.

```
PK,SK,EntityType,Name,Email,OrderDate,Amount,ProductName,Quantity
USER#1,PROFILE,User,Alice,alice@example.com,,,,
USER#1,ORDER#2024-01-15,Order,,,2024-01-15,99.99,,
USER#1,ORDER#2024-02-10,Order,,,2024-02-10,149.50,,
PRODUCT#101,METADATA,Product,,,,,Laptop,50
PRODUCT#102,METADATA,Product,,,,,Mouse,200
USER#2,PROFILE,User,Bob,bob@example.com,,,,
USER#2,ORDER#2024-01-20,Order,,,2024-01-20,75.00,,
PRODUCT#103,METADATA,Product,,,,,Keyboard,150
USER#3,PROFILE,User,Charlie,charlie@example.com,,,,
PRODUCT#104,METADATA,Product,,,,,Monitor,30
```

In diesem Beispiel verwenden Benutzerprofile, Bestellungen und Produkte dieselbe Tabelle. Jeder Elementtyp verwendet nur die für ihn relevanten Spalten.

DynamoDB JSON

Ein Tabellenexport im DynamoDB-JSON-Format kann aus mehreren Elementobjekten bestehen. Jedes einzelne Objekt befindet sich im Standard-JSON-Format von DynamoDB, und Zeilenumbrüche werden als Elementtrennzeichen verwendet. Als zusätzliche Funktion werden zeitpunktbezogene Exporte standardmäßig als Importquelle unterstützt.

> **Note**
>
> Neue Zeilen werden als Elementtrennzeichen für eine Datei im DynamoDB-JSON-Format verwendet und sollten nicht innerhalb eines Elementobjekts verwendet werden.

```
{"Item": {"Authors": {"SS": ["Author1", "Author2"]}, "Dimensions": {"S": "8.5 x 11.0 x
 1.5"}, "ISBN": {"S": "333-3333333333"}, "Id": {"N": "103"}, "InPublication": {"BOOL":
 false}, "PageCount": {"N": "600"}, "Price": {"N": "2000"}, "ProductCategory": {"S":
 "Book"}, "Title": {"S": "Book 103 Title"}}}
{"Item": {"Authors": {"SS": ["Author1", "Author2"]}, "Dimensions": {"S": "8.5 x 11.0 x
 1.5"}, "ISBN": {"S": "444-444444444"}, "Id": {"N": "104"}, "InPublication": {"BOOL":
 false}, "PageCount": {"N": "600"}, "Price": {"N": "2000"}, "ProductCategory": {"S":
 "Book"}, "Title": {"S": "Book 104 Title"}}}
{"Item": {"Authors": {"SS": ["Author1", "Author2"]}, "Dimensions": {"S": "8.5 x 11.0 x
 1.5"}, "ISBN": {"S": "555-5555555555"}, "Id": {"N": "105"}, "InPublication": {"BOOL":
 false}, "PageCount": {"N": "600"}, "Price": {"N": "2000"}, "ProductCategory": {"S":
 "Book"}, "Title": {"S": "Book 105 Title"}}}
```

Amazon Ion

Amazon Ion ist ein reich typisiertes, selbstbeschreibendes, hierarchisches Datenserialisierungsformat, das entwickelt wurde, um schnelle Entwicklungs-, Entkopplungs- und Effizienzprobleme zu bewältigen, denen sich täglich gegenübersehen, während umfangreiche, serviceorientierte Architekturen entwickelt werden.

Wenn Sie eine Tabelle im Ion-Format exportieren, werden die in der Tabelle verwendeten DynamoDB-Datentypen Ion-Datentypen zugeordnet.

S. Nein.	Konvertierung des Datentyps von Ion zu DynamoDB	B
1	`Ion Data Type`	`DynamoDB Representation`
2	`string`	`String (s)`
3	`bool`	`Boolean (BOOL)`
4	`decimal`	`Number (N)`
5	`blob`	`Binary (B)`
6	`list (with type annotation $dynamodb_SS, $dynamodb_NS, or $dynamodb_BS)`	`Set (SS, NS, BS)`
7	`list`	`List`
8	`struct`	`Map`

Elemente in einem Ion-Export werden durch Zeilenumbrüche getrennt. Jede Zeile beginnt mit einer Ion-Versionsmarkierung, gefolgt von einem Element im Ion-Format.

> **Note**
>
> Im folgenden Beispiel wurde ein Element aus einer Datei im Ion-Format aus Gründen der Lesbarkeit in mehreren Zeilen formatiert.

```
$ion_1_0
[
  {
    Item:{
      Authors:$dynamodb_SS::["Author1","Author2"],
      Dimensions:"8.5 x 11.0 x 1.5",
      ISBN:"333-3333333333",
```

```
      Id:103.,
      InPublication:false,
      PageCount:6d2,
      Price:2d3,
      ProductCategory:"Book",
      Title:"Book 103 Title"
    }
  },
  {
    Item:{
      Authors:$dynamodb_SS::["Author1","Author2"],
      Dimensions:"8.5 x 11.0 x 1.5",
      ISBN:"444-4444444444",
      Id:104.,
      InPublication:false,
      PageCount:6d2,
      Price:2d3,
      ProductCategory:"Book",
      Title:"Book 104 Title"
    }
  },
  {
    Item:{
      Authors:$dynamodb_SS::["Author1","Author2"],
      Dimensions:"8.5 x 11.0 x 1.5",
      ISBN:"555-5555555555",
      Id:105.,
      InPublication:false,
      PageCount:6d2,
      Price:2d3,
      ProductCategory:"Book",
      Title:"Book 105 Title"
    }
  }
]
```

Importformatkontingente und Validierung

Importkontingente

Der DynamoDB-Import aus Amazon S3 kann bis zu 50 gleichzeitige Importaufgaben mit einer Gesamtgröße von 15 TB gleichzeitig in den Regionen us-east-1, us-west-2 und eu-west-1 unterstützen. In allen anderen Regionen werden bis zu 50 gleichzeitige Importaufgaben mit einer

Gesamtgröße von 1 TB unterstützt. Jede Importaufgabe kann bis zu 50 000 Amazon-S3-Objekte in allen Regionen aufnehmen. Diese Standardkontingente werden auf jedes Konto angewendet. Wenn Sie der Meinung sind, dass Sie diese Kontingente ändern müssen, wenden Sie sich bitte an Ihr Kontoteam, das diese Anträge fallweise prüft. Weitere Informationen zu den DynamoDB-Grenzwerten finden Sie unter Servicekontingente.

Validierungsfehler

Während des Importvorgangs kann DynamoDB beim Parsen Ihrer Daten auf Fehler stoßen. Für jeden Fehler gibt DynamoDB ein CloudWatch-Protokoll aus und zählt die Gesamtzahl der aufgetretenen Fehler. Wenn das Amazon-S3-Objekt selbst fehlerhaft formatiert ist oder sein Inhalt kein DynamoDB-Element bilden kann, können wir die Verarbeitung des verbleibenden Teils des Objekts überspringen.

> **Note**
>
> Wenn die Amazon-S3-Datenquelle über mehrere Elemente mit demselben Schlüssel verfügt, werden die Elemente überschrieben, bis eines übrig bleibt. Dies kann den Anschein erwecken, als sei 1 Element importiert worden und die anderen seien ignoriert worden. Die doppelten Elemente werden in zufälliger Reihenfolge überschrieben, nicht als Fehler gezählt und nicht an CloudWatch-Protokolle ausgegeben.
> Sobald der Import abgeschlossen ist, können Sie die Gesamtzahl der importierten Elemente, die Gesamtzahl der Fehler und die Gesamtzahl der verarbeiteten Elemente anzeigen. Zur weiteren Fehlerbehebung können Sie auch die Gesamtgröße der importierten Elemente und die Gesamtgröße der verarbeiteten Daten überprüfen.

Es gibt drei Kategorien von Importfehlern: API-Validierungsfehler, Datenvalidierungsfehler und Konfigurationsfehler.

API-Validierungsfehler

API-Validierungsfehler sind Fehler auf Elementebene aus der Sync-API. Häufige Ursachen sind Berechtigungsprobleme, fehlende erforderliche Parameter und Fehler bei der Parametervalidierung. Details dazu, warum der API-Aufruf fehlgeschlagen ist, sind in den von der `ImportTable`-Anforderung ausgelösten Ausnahmen enthalten.

Datenvalidierungsfehler

Datenvalidierungsfehler können entweder auf Element- oder Dateiebene auftreten. Während des Imports werden Elemente basierend auf DynamoDB-Regeln validiert, bevor sie in die Zieltabelle importiert werden. Wenn die Validierung eines Elements fehlschlägt und das Element nicht importiert wird, überspringt die Importaufgabe dieses Element und fährt mit dem nächsten Element fort. Am Ende der Aufgabe wird der Importstatus auf FAILED gesetzt, zusammen mit FailureCode, ItemValidationError und der FailureMessage „Einige der Elemente haben die Validierungsprüfungen nicht bestanden und wurden nicht importiert. Weitere Details finden Sie in den CloudWatch-Fehlerprotokollen.“

Häufige Ursachen für Datenvalidierungsfehler sind Objekte, die nicht analysierbar sind, Objekte im falschen Format (Eingabe gibt DYNAMODB_JSON an, das Objekt befindet sich jedoch nicht in DYNAMODB_JSON) und die Nichtübereinstimmung des Schemas mit den angegebenen Quelltabellenschlüsseln.

Konfigurationsfehler

Konfigurationsfehler sind in der Regel Workflowfehler aufgrund der Berechtigungsvalidierung. Der Import-Workflow überprüft einige Berechtigungen, nachdem die Anforderung angenommen wurde. Wenn es Probleme beim Aufrufen einer der erforderlichen Abhängigkeiten wie Amazon S3 oder CloudWatch gibt, markiert der Prozess den Importstatus als FAILED. `failureCode` und `failureMessage` weisen auf den Grund für den Fehler hin. Gegebenenfalls enthält die Fehlermeldung auch die Anforderungs-ID, mit der Sie den Grund für den Fehler in CloudTrail untersuchen können.

Häufige Konfigurationsfehler sind die falsche URL für den Amazon S3-Bucket und eine fehlende Berechtigung für den Zugriff auf den Amazon-S3-Bucket, CloudWatch-Protokolle und AWS KMS-Schlüssel, die zum Entschlüsseln des Amazon-S3-Objekts verwendet werden. Weitere Informationen finden Sie unter Verwenden von Datenschlüsseln.

Validieren von Amazon-S3-Quellobjekten

Um S3-Quell-Objekte zu validieren, führen Sie die folgenden Schritte aus.

1. Überprüfen Sie das Datenformat und den Komprimierungstyp

    - Stellen Sie sicher, dass alle übereinstimmenden Amazon-S3-Objekte unter dem angegebenen Präfix dasselbe Format haben (DYNAMODB_JSON, DYNAMODB_ION, CSV)

- Stellen Sie sicher, dass alle übereinstimmenden Amazon-S3-Objekte unter dem angegebenen Präfix auf dieselbe Weise komprimiert werden (GZIP, ZSTD, NONE)

> **Note**
>
> Die Amazon-S3-Objekte müssen nicht die entsprechende Erweiterung (.csv/.json/.ion/.gz/.zstd usw.) haben, da das im ImportTable-Aufruf angegebene Eingabeformat Vorrang hat.

2. Überprüfen Sie, ob die Importdaten dem gewünschten Tabellenschema entsprechen

- Stellen Sie sicher, dass jedes Element in den Quelldaten über den Primärschlüssel verfügt. Ein Sortierschlüssel ist für Importe optional.
- Stellen Sie sicher, dass der mit dem Primärschlüssel und einem beliebigen Sortierschlüssel verknüpfte Attributtyp mit dem Attributtyp in der Tabelle und im GSI-Schema übereinstimmt, wie in den Parametern zur Tabellenerstellung angegeben.

Fehlerbehebung

CloudWatch-Protokolle

Bei fehlgeschlagenen Importaufgaben werden detaillierte Fehlermeldungen in CloudWatch-Protokollen veröffentlicht. Um auf diese Protokolle zuzugreifen, rufen Sie zuerst ImportArn aus der Ausgabe ab, sowie describe-import mit diesem Befehl:

```
aws dynamodb describe-import --import-arn arn:aws:dynamodb:us-east-1:ACCOUNT:table/
target-table/import/01658528578619-c4d4e311
}
```

Beispielausgabe:

```
aws dynamodb describe-import --import-arn "arn:aws:dynamodb:us-
east-1:531234567890:table/target-table/import/01658528578619-c4d4e311"
{
    "ImportTableDescription": {
        "ImportArn": "arn:aws:dynamodb:us-east-1:ACCOUNT:table/target-table/
import/01658528578619-c4d4e311",
        "ImportStatus": "FAILED",
        "TableArn": "arn:aws:dynamodb:us-east-1:ACCOUNT:table/target-table",
        "TableId": "7b7ecc22-302f-4039-8ea9-8e7c3eb2bcb8",
```

```
        "ClientToken": "30f8891c-e478-47f4-af4a-67a5c3b595e3",
        "S3BucketSource": {
            "S3BucketOwner": "ACCOUNT",
            "S3Bucket": "my-import-source",
            "S3KeyPrefix": "import-test"
        },
        "ErrorCount": 1,
        "CloudWatchLogGroupArn": "arn:aws:logs:us-east-1:ACCOUNT:log-group:/aws-
dynamodb/imports:*",
        "InputFormat": "CSV",
        "InputCompressionType": "NONE",
        "TableCreationParameters": {
            "TableName": "target-table",
            "AttributeDefinitions": [
                {
                    "AttributeName": "pk",
                    "AttributeType": "S"
                }
            ],
            "KeySchema": [
                {
                    "AttributeName": "pk",
                    "KeyType": "HASH"
                }
            ],
            "BillingMode": "PAY_PER_REQUEST"
        },
        "StartTime": 1658528578.619,
        "EndTime": 1658528750.628,
        "ProcessedSizeBytes": 70,
        "ProcessedItemCount": 1,
        "ImportedItemCount": 0,
        "FailureCode": "ItemValidationError",
        "FailureMessage": "Some of the items failed validation checks and were not
 imported. Please check CloudWatch error logs for more details."
    }
}
```

Rufen Sie die Protokollgruppe und die Import-ID aus der obigen Antwort ab und verwenden Sie sie, um die Fehlerprotokolle abzurufen. Die Import-ID ist das letzte Pfadelement des Felds `ImportArn`. Der Name der Protokollgruppe lautet `/aws-dynamodb/imports`. Der Name des Fehler-Protokollstreams ist `import-id/error`. In diesem Beispiel wäre es `01658528578619-c4d4e311/error`.

Fehlender Schlüssel pk im Element

Wenn das S3-Quellobjekt nicht den Primärschlüssel enthält, der als Parameter angegeben wurde, schlägt der Import fehl. Zum Beispiel, wenn Sie den Primärschlüssel für den Import als Spaltenname „pk“ definieren.

```
aws dynamodb import-table –s3-bucket-source S3Bucket=my-import-
source,S3KeyPrefix=import-test.csv \
            –input-format CSV --table-creation-parameters '{"TableName":"target-
table","KeySchema":  \
            [{"AttributeName":"pk","KeyType":"HASH"}],"AttributeDefinitions":
[{"AttributeName":"pk","AttributeType":"S"}],"BillingMode":"PAY_PER_REQUEST"}'
```

Die Spalte „pk“ fehlt im Quellobjekt `import-test.csv`, das die folgenden Inhalte hat:

```
title,artist,year_of_release
The Dark Side of the Moon,Pink Floyd,1973
```

Dieser Import schlägt aufgrund eines Fehlers bei der Elementvalidierung wegen des fehlenden Primärschlüssels in der Datenquelle fehl.

Beispiel für CloudWatch-Fehlerprotokoll:

```
aws logs get-log-events –log-group-name /aws-dynamodb/imports –log-stream-name
 01658528578619-c4d4e311/error
{
"events": [
{
"timestamp": 1658528745319,
"message": "{\"itemS3Pointer\":{\"bucket\":\"my-import-source\",\"key\":
\"import-test.csv\",\"itemIndex\":0},\"importArn\":\"arn:aws:dynamodb:us-
east-1:531234567890:table/target-table/import/01658528578619-c4d4e311\",\"errorMessages
\":[\"One or more parameter values were invalid: Missing the key pk in the item\"]}",
"ingestionTime": 1658528745414
}
],
"nextForwardToken": "f/36986426953797707963335499204463414460239026137054642176/s",
"nextBackwardToken": "b/36986426953797707963335499204463414460239026137054642176/s"
}
```

Dieses Fehlerprotokoll zeigt an, dass „ein oder mehrere Parameterwerte ungültig waren: Der Schlüssel pk im Element fehlt“. Da dieser Importauftrag fehlgeschlagen ist, existiert jetzt die Tabelle

„Zieltabelle“ und ist leer, da keine Elemente importiert wurden. Das erste Element wurde verarbeitet, und das Objekt hat die Elementvalidierung nicht bestanden.

Um das Problem zu beheben, löschen Sie zuerst die „Zieltabelle“, wenn sie nicht mehr benötigt wird. Verwenden Sie dann entweder einen Primärschlüsselspaltennamen, der im Quellobjekt vorhanden ist, oder aktualisieren Sie die Quelldaten wie folgt:

```
pk,title,artist,year_of_release
Albums::Rock::Classic::1973::AlbumId::ALB25,The Dark Side of the Moon,Pink Floyd,1973
```

Zieltabelle existiert

Wenn Sie eine Importaufgabe starten und eine Antwort wie folgt erhalten:

```
An error occurred (ResourceInUseException) when calling the ImportTable operation:
 Table already exists: target-table
```

Um diesen Fehler zu beheben, müssen Sie einen Tabellennamen wählen, der noch nicht existiert, und den Import wiederholen.

Der angegebene Bucket existiert nicht

Wenn der Quell-Bucket nicht existiert, schlägt der Import fehl und die Details der Fehlermeldung werden in CloudWatch protokolliert.

Beispiel-Importbeschreibung:

```
aws dynamodb –endpoint-url $ENDPOINT describe-import –import-arn "arn:aws:dynamodb:us-
east-1:531234567890:table/target-table/import/01658530687105-e6035287"
{
"ImportTableDescription": {
"ImportArn": "arn:aws:dynamodb:us-east-1:ACCOUNT:table/target-table/
import/01658530687105-e6035287",
"ImportStatus": "FAILED",
"TableArn": "arn:aws:dynamodb:us-east-1:ACCOUNT:table/target-table",
"TableId": "e1215a82-b8d1-45a8-b2e2-14b9dd8eb99c",
"ClientToken": "3048e16a-069b-47a6-9dfb-9c259fd2fb6f",
"S3BucketSource": {
"S3BucketOwner": "531234567890",
"S3Bucket": "BUCKET_DOES_NOT_EXIST",
"S3KeyPrefix": "import-test"
```

```
},
"ErrorCount": 0,
"CloudWatchLogGroupArn": "arn:aws:logs:us-east-1:ACCOUNT:log-group:/aws-dynamodb/
imports:*",
"InputFormat": "CSV",
"InputCompressionType": "NONE",
"TableCreationParameters": {
"TableName": "target-table",
"AttributeDefinitions": [
{
"AttributeName": "pk",
"AttributeType": "S"
}
],
"KeySchema": [
{
"AttributeName": "pk",
"KeyType": "HASH"
}
],
"BillingMode": "PAY_PER_REQUEST"
},
"StartTime": 1658530687.105,
"EndTime": 1658530701.873,
"ProcessedSizeBytes": 0,
"ProcessedItemCount": 0,
"ImportedItemCount": 0,
"FailureCode": "S3NoSuchBucket",
"FailureMessage": "The specified bucket does not exist (Service: Amazon S3; Status
 Code: 404; Error Code: NoSuchBucket; Request ID: Q4W6QYYFDWY6WAKH; S3 Extended Request
 ID: ObqSlLeIMJpQqHLRX2C5Sy7n+8g6iGPwy7ixg7eEeTuEkg/+chU/JF+RbliWytMlkUlUcuCLTrI=;
 Proxy: null)"
}
}
```


Der `FailureCode` lautet `S3NoSuchBucket`. Die `FailureMessage` enthält Details wie die Anforderungs-ID und den Service, der den Fehler ausgelöst hat. Da der Fehler erkannt wurde, bevor die Daten in die Tabelle importiert wurden, wird keine neue DynamoDB-Tabelle erstellt. In einigen Fällen, wenn diese Fehler nach dem Start des Datenimports auftreten, wird die Tabelle mit teilweise importierten Daten beibehalten.

Um diesen Fehler zu beheben, stellen Sie sicher, dass der Quell-Bucket aus Amazon S3 vorhanden ist, und starten Sie den Importvorgang neu.

Bewährte Methoden für den Import aus Amazon S3 in DynamoDB

Im Folgenden werden die bewährten Methoden für den Import aus Amazon S3 in DynamoDB vorgestellt.

Das Limit von 50 000 S3-Objekten nicht überschreiten

Bei jedem Importauftrag werden maximal 50 000 S3-Objekte unterstützt. Wenn Ihr Datensatz mehr als 50 000 Objekte enthält, sollten Sie diese zu größeren Objekten zusammenfassen.

Übermäßig große S3-Objekte vermeiden

S3-Objekte werden parallel importiert. Wenn Sie über viele mittelgroße S3-Objekte verfügen, ist eine parallele Ausführung ohne übermäßigen Overhead möglich. Bei Elementen unter 1 KB sollten Sie 4 000 000 Elemente in jedem S3-Objekt platzieren. Wenn Sie eine größere durchschnittliche Elementgröße haben, platzieren Sie proportional weniger Elemente in jedem S3-Objekt.

Sortierte Daten randomisieren

Wenn ein S3-Objekt Daten in sortierter Reihenfolge enthält, kann es eine fortlaufende Hot-Partition erstellen. In diesem Fall empfängt eine Partition die gesamte Aktivität, danach die nächste Partition usw. Daten in sortierter Reihenfolge sind aufeinanderfolgende Elemente im S3-Objekt, die während des Imports auf dieselbe Zielpartition geschrieben werden. Ein typischer Fall, in dem Daten in sortierter Reihenfolge vorliegen, ist eine CSV-Datei, in der Elemente nach Partitionsschlüssel sortiert werden, sodass wiederholte Elemente denselben Partitionsschlüssel aufweisen.

Zum Vermeiden einer fortlaufenden Hot-Partition empfehlen wir, die Reihenfolge in diesen Fällen zu randomisieren. Dies kann die Leistung verbessern, indem die Schreibvorgänge verteilt werden. Weitere Informationen finden Sie unter Effiziente Verteilung der Schreibaktivität beim Hochladen von Daten in DynamoDB.

Daten komprimieren, damit die Gesamtgröße der S3-Objekte das regionale Limit nicht überschreitet

Beim Import aus S3 gibt es ein Limit für die Gesamtgröße der S3-Objektdaten, die importiert werden sollen. Das Limit liegt in den Regionen us-east-1, us-west-2 und eu-west-1 bei 15 TB und in allen anderen Regionen bei 1 TB. Das Limit basiert auf den Rohgrößen von S3-Objekten.

Durch eine Komprimierung kann das Limit mehr Rohdaten umfassen. Wenn die Komprimierung allein nicht ausreicht, damit der Import innerhalb des Limits liegt, können Sie beim AWS Premium Support eine Kontingenterhöhung beantragen.

Die Auswirkungen der Elementgröße auf die Leistung kennen

Wenn die durchschnittliche Elementgröße sehr klein ist (unter 200 Byte), kann der Import etwas länger dauern als bei größeren Elementgrößen.

Ändern Sie S3-Objekte bei aktiven Importen nicht

Stellen Sie sicher, dass Ihre S3-Quellobjekte während eines Importvorgangs unverändert bleiben. Wenn ein S3-Objekt während eines Imports geändert wird, schlägt der Vorgang mit einem Fehlercode `ObjectModifiedInS3DuringImport` und der Meldung „Das S3-Objekt konnte nicht importiert werden, weil es überschrieben wurde“ fehl.

Wenn dieser Fehler auftritt, starten Sie den Importvorgang mit einer stabilen Version Ihres S3-Objekts neu. Um dieses Problem zu vermeiden, warten Sie, bis der aktuelle Import abgeschlossen ist, bevor Sie Änderungen an den Quelldateien vornehmen.

Den Import ohne globale sekundäre Indizes in Betracht ziehen

Die Dauer einer Importaufgabe kann vom Vorhandensein eines oder mehrerer globaler sekundärer Indizes (GSIs) abhängen. Wenn Sie Indizes mit Partitionsschlüsseln erstellen möchten, die eine geringe Kardinalität aufweisen, können Sie den Import möglicherweise beschleunigen, indem Sie die Indexerstellung aufschieben, bis der Importauftrag abgeschlossen ist (anstatt sie in den Importauftrag einzubeziehen).

> **Note**
>
> Für die Erstellung eines GSI während des Imports fallen keine Schreibgebühren an (für die Erstellung eines GSI nach dem Import hingegen schon).

DynamoDB Datenexport zu Amazon S3: Funktionsweise

Der DynamoDB-Export nach S3 ist eine vollständig verwaltete Lösung für den Export Ihrer DynamoDB-Daten in einen Amazon-S3-Bucket in großem Maßstab. Mithilfe des DynamoDB-Exports nach S3 können Sie Daten aus einer Amazon DynamoDB-Tabelle jederzeit innerhalb Ihres point-in-time Wiederherstellungsfensters (PITR) in einen Amazon S3 S3-Bucket exportieren. Sie müssen PITR in Ihrer Tabelle aktivieren, um die Exportfunktion nutzen zu können. Mit dieser Funktion können Sie Analysen und komplexe Abfragen Ihrer Daten mithilfe anderer AWS Dienste wie Athena, Amazon SageMaker AI AWS Glue, Amazon EMR und durchführen. AWS Lake Formation

Mit dem DynamoDB-Export nach S3 können Sie sowohl vollständige als auch inkrementelle Daten aus Ihrer DynamoDB-Tabelle exportieren. Exporte sind asynchron, verbrauchen keine Lesekapazitätseinheiten (RCUs) und haben keine Auswirkungen auf die Tabellenleistung und Verfügbarkeit. Die unterstützten Exportdateiformate sind DynamoDB JSON und Amazon Ion. Sie können Daten auch in einen S3-Bucket exportieren, der einem anderen AWS Konto gehört, und in eine andere AWS Region. Ihre Daten sind immer verschlüsselt end-to-end.

Vollständige DynamoDB-Exporte werden auf der Grundlage der Größe der DynamoDB-Tabelle (Tabellendaten und lokale Sekundärindizes) zu dem Zeitpunkt berechnet, für den der Export ausgeführt wird. Inkrementelle DynamoDB-Exporte werden auf der Grundlage der Größe der Daten berechnet, die aus Ihren kontinuierlichen Backups für den exportierten Zeitraum verarbeitet wurden. Für den inkrementellen Export wird eine Mindestgebühr von 10 MB berechnet. Zusätzliche Gebühren fallen für das Speichern exportierter Daten in Amazon S3 und für PUT-Anfragen an Ihren Amazon-S3-Bucket an. Weitere Informationen zu diesen Gebühren finden Sie unter Amazon-DynamoDB-Preise und Amazon-S3-Preise.

Einzelheiten zu Servicekontingenten finden Sie unter Exportieren von Tabellen zu Amazon S3.

Themen

- Anfordern eines Tabellenexports in DynamoDB
- Export-Ausgabeformat für DynamoDB-Tabellen

Anfordern eines Tabellenexports in DynamoDB

DynamoDB-Tabellenexporte ermöglichen es Ihnen, Tabellendaten in einen Amazon S3 S3-Bucket zu exportieren, sodass Sie Analysen und komplexe Abfragen Ihrer Daten mithilfe anderer AWS Services wie Athena, Amazon SageMaker AI AWS Glue, Amazon EMR und durchführen können. AWS Lake Formation Sie können einen Tabellenexport mit der AWS-Managementkonsole, der oder der AWS CLI DynamoDB-API anfordern.

> **Note**
>
> Zahlung durch den Anforderer ist für Amazon-S3-Buckets nicht unterstützt.

DynamoDB unterstützt sowohl den vollständigen Export als auch den inkrementellen Export:

- Mit vollständigen Exporten können Sie einen vollständigen Snapshot Ihrer Tabelle von einem beliebigen Zeitpunkt innerhalb des point-in-time Wiederherstellungsfensters (PITR) in Ihren Amazon S3 S3-Bucket exportieren.
- Mit inkrementellen Exporten können Sie Daten aus Ihrer DynamoDB-Tabelle, die zwischen einem bestimmten Zeitraum geändert, aktualisiert oder gelöscht wurden, innerhalb Ihres PITR-Fensters in Ihren Amazon-S3-Bucket exportieren.

Themen

- Voraussetzungen
- Beantragen eines Exports mit dem AWS-Managementkonsole
- Details zu vergangenen Exporten finden Sie in AWS-Managementkonsole
- Einen Export mit dem AWS CLI und anfordern AWS SDKs
- Details zu vergangenen Exporten mit dem AWS CLI und abrufen AWS SDKs

Voraussetzungen

Aktivieren von PITR

PITR muss in Ihrer Tabelle aktiviert sein, um die Funktion zum Exportieren nach S3 verwenden zu können. Einzelheiten zur Aktivierung von PITR finden Sie unter oint-in-timeP-Wiederherstellung. Wenn Sie einen Export für eine Tabelle anfordern, für die PITR nicht aktiviert ist, schlägt Ihre Anfrage fehl und es wird eine Ausnahmemeldung angezeigt: „Beim Aufrufen des `ExportTableToPointInTime` Vorgangs ist ein Fehler aufgetreten (PointInTimeRecoveryUnavailableException): Die Point-in-Time-Wiederherstellung ist für die Tabelle 'my-dynamodb-tablenicht aktiviert“. Sie können einen Export nur für einen Zeitpunkt innerhalb Ihrer konfigurierten PITR-`RecoveryPeriodInDays` anfordern.

Einrichten von S3-Berechtigungen

Sie können Ihre Tabellendaten in jeden Amazon S3-Bucket exportieren, in den Sie schreiben dürfen. Der Ziel-Bucket muss sich nicht in derselben AWS Region befinden oder denselben Besitzer haben wie der Besitzer der Quelltabelle. Ihre AWS Identity and Access Management (IAM-) Richtlinie muss es Ihnen ermöglichen, S3-Aktionen (`s3:AbortMultipartUploads`, `s3:PutObject`, und`s3:PutObjectAcl`) und die DynamoDB-Exportaktion () auszuführen. `dynamodb:ExportTableToPointInTime` Hier ein Beispiel für eine

Beispielrichtlinie, die Ihren Benutzern Berechtigungen zum Ausführen von Exporten in einen S3-Bucket gewährt.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "AllowDynamoDBExportAction",
            "Effect": "Allow",
            "Action": "dynamodb:ExportTableToPointInTime",
            "Resource": "arn:aws:dynamodb:us-east-1:111122223333:table/my-table"
        },
        {
            "Sid": "AllowS3BucketWrites",
            "Effect": "Allow",
            "Action": [
                "s3:AbortMultipartUpload",
                "s3:PutObject",
                "s3:PutObjectAcl"
            ],
            "Resource": "arn:aws:s3:::amzn-s3-demo-bucket/*"
        }
    ]
}
```

Wenn Sie in einen Amazon S3-Bucket schreiben müssen, der sich in einem anderen Konto befindet, oder wenn Sie keine Schreibberechtigungen haben, muss der Eigentümer des Amazon S3-Buckets eine Bucket-Richtlinie hinzufügen, damit Sie von DynamoDB in diesen Bucket exportieren können. Hier sehen Sie eine Beispielrichtlinie für den Amazon S3-Zielbucket.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "ExampleStatement",
            "Effect": "Allow",
```

```
            "Principal": {
                "AWS": "arn:aws:iam::123456789012:user/Dave"
            },
            "Action": [
                "s3:AbortMultipartUpload",
                "s3:PutObject",
                "s3:PutObjectAcl"
            ],
            "Resource": "arn:aws:s3:::amzn-s3-demo-bucket/*"
        }
    ]
}
```

Das Widerrufen dieser Berechtigungen während eines Exports führt zu Teildateien.

> **Note**
>
> Wenn die Tabelle oder der Bucket, in die/den Sie exportieren, mit vom Kunden verwalteten Schlüsseln verschlüsselt ist, müssen die Richtlinien dieses KMS-Schlüssels DynamoDB die Berechtigung zur Verwendung erteilen. Diese Berechtigung wird über das IAM erteilt, das den Exportauftrag User/Role auslöst. Weitere Informationen zur Verschlüsselung, einschließlich Best Practices, finden Sie unter So verwendet DynamoDB AWS KMS und Using a custom KMS key (Verwendung eines benutzerdefinierten KMS-Schlüssels).

Beantragen eines Exports mit dem AWS-Managementkonsole

Das folgende Beispiel zeigt, wie Sie mit der DynamoDB-Konsole eine vorhandene Tabelle namens `MusicCollection` im Zustand eines bestimmten Zeitpunkts wiederherstellen.

> **Note**
>
> Bei diesem Verfahren wird davon ausgegangen, dass Sie die point-in-time Wiederherstellung aktiviert haben. Um es für die `MusicCollection` Tabelle zu aktivieren, wählen Sie auf der Registerkarte Übersicht der Tabelle im Abschnitt Tabellendetails die Option Für oint-in-timeP-Wiederherstellung aktivieren aus.

So fordern Sie einen Tabellenexport an

1. Melden Sie sich bei der an AWS-Managementkonsole und öffnen Sie die DynamoDB-Konsole unter. https://console.aws.amazon.com/dynamodb/
2. Klicken Sie im Navigationsbereich links in der Konsole auf Exports to S3 (Exporte nach S3).
3. Wählen Sie die Schaltfläche Nach S3 exportieren.
4. Wählen Sie eine Quelltabelle und den Ziel-S3-Bucket aus. Wenn der Ziel-Bucket Ihrem Konto gehört, können Sie die Schaltfläche Browse S3 (S3 durchsuchen) verwenden, um ihn zu finden. Geben Sie andernfalls die URL des Buckets ein und verwenden Sie dabei das `s3://bucketname/prefix format.` **`prefix`** ist ein optionaler Ordner, um Ihren Ziel-Bucket organisiert zu halten.
5. Wählen Sie Vollständiger Export oder Inkrementeller Export. Bei einem vollständigen Export wird der vollständige Tabellen-Snapshot Ihrer Tabelle so ausgegeben, wie er zu dem von Ihnen angegebenen Zeitpunkt war. Bei einem inkrementellen Export werden die Änderungen ausgegeben, die während des angegebenen Exportzeitraums an Ihrer Tabelle vorgenommen wurden. Ihre Ausgabe ist komprimiert, sodass sie nur den endgültigen Status des Elements aus dem Exportzeitraum enthält. Das Element wird nur einmal im Export angezeigt, auch wenn es innerhalb desselben Exportzeitraums mehrfach aktualisiert wurde.

   Full export

   1. Wählen Sie den Zeitpunkt aus, ab dem Sie den vollständigen Tabellen-Snapshot exportieren möchten. Dies kann ein beliebiger Zeitpunkt innerhalb des PITR-Zeitfensters sein. Alternativ können Sie Aktuelle Zeit auswählen, um den neuesten Snapshot zu exportieren.
   2. Wählen Sie für Exportiertes Dateiformat zwischen DynamoDB JSON und Amazon Ion. Standardmäßig wird Ihre Tabelle ab dem letzten wiederherstellbaren Zeitpunkt im Point-in-Time-Wiederherstellungsfenster im DynamoDB JSON-Format exportiert und mit einem Amazon S3-Schlüssel (SSE-S3) verschlüsselt. Sie können diese Exporteinstellungen bei Bedarf ändern.

      > **Note**
      >
      > Wenn Sie Ihren Export mit einem durch AWS Key Management Service (AWS KMS) geschützten Schlüssel verschlüsseln möchten, muss sich der Schlüssel in derselben Region wie der Ziel-S3-Bucket befinden.

Incremental export

1. Wählen Sie den Exportzeitraum aus, für den Sie die inkrementellen Daten exportieren möchten. Wählen Sie eine Startzeit innerhalb des PITR-Zeitfensters aus. Die Dauer des Exportzeitraums muss mindestens 15 Minuten betragen und darf nicht länger als 24 Stunden sein. Die Startzeit ist im Exportzeitraums enthalten, die Endzeit nicht.
2. Wählen Sie zwischen dem Absoluten Modus oder dem Relativen Modus.

   a. Im Absoluten Modus werden inkrementelle Daten für den von Ihnen angegebenen Zeitraum exportiert.

   b. Im Relativen Modus werden inkrementelle Daten für einen Exportzeitraum exportiert, der sich auf die Übermittlungszeit Ihres Exportauftrags bezieht.
3. Wählen Sie für Exportiertes Dateiformat zwischen DynamoDB JSON und Amazon Ion. Standardmäßig wird Ihre Tabelle ab dem letzten wiederherstellbaren Zeitpunkt im Point-in-Time-Wiederherstellungsfenster im DynamoDB JSON-Format exportiert und mit einem Amazon S3-Schlüssel (SSE-S3) verschlüsselt. Sie können diese Exporteinstellungen bei Bedarf ändern.

   > **Note**
   >
   > Wenn Sie Ihren Export mit einem durch AWS Key Management Service (AWS KMS) geschützten Schlüssel verschlüsseln möchten, muss sich der Schlüssel in derselben Region wie der S3-Ziel-Bucket befinden.
4. Wählen Sie für den Exportansichtstyp entweder Neue und alte Bilder oder Nur neue Bilder aus. Neues Bild zeigt den aktuellen Status des Elements Altes Bild zeigt den Status des Elements unmittelbar vor dem angegebenen Startdatum und der angegebenen Startzeit Die Standardeinstellung ist Neue und alte Bilder. Weitere Informationen zu neuen und alten Bildern finden Sie unter Inkrementelle Exportausgabe.

6. Klicken Sie auf Exportieren, um den Vorgang zu beginnen.

Die exportierten Daten sind nicht transaktional konsistent. Ihre Transaktionsoperationen können zwischen zwei Exportausgaben aufgeteilt werden. Ein Teil der Elemente kann durch eine Transaktionsoperation geändert und im Export widergespiegelt werden, während ein anderer Teil der Änderungen derselben Transaktion im selben Exportauftrag nicht enthalten ist. Die Exporte sind

jedoch letztendlich konsistent. Wenn eine Transaktion während eines Exports aufgeteilt wird, wird der verbleibende Teil der Transaktion im nächsten zusammenhängenden Export enthalten sein – ohne Duplikate. Die für Exporte verwendeten Zeiträume basieren auf einer internen Systemuhr und können um eine Minute von der lokalen Uhr Ihrer Anwendung abweichen.

Details zu vergangenen Exporten finden Sie in AWS-Managementkonsole

Informationen zu früheren Exportaufträgen finden Sie im Navigationsbereich unter dem Abschnitt Exporte nach S3. Dieser Abschnitt enthält eine Liste aller Exporte, die Sie in den letzten 90 Tagen erstellt haben. Wählen Sie die ARN eines in der Registerkarte Exporte aufgeführten Auftrags aus, um Informationen zu diesem Export abzurufen, einschließlich aller erweiterten Konfigurationseinstellungen, die Sie gewählt haben. Beachten Sie, dass die Metadaten für die Exportaufgabe nach 90 Tagen ablaufen und Aufträge, die älter sind als diese, in dieser Liste nicht mehr gefunden werden, die Objekte in Ihrem S3 Bucket so lange bleiben, wie es ihre Bucket-Richtlinien zulassen. DynamoDB löscht nie eines der Objekte, die es während eines Exports in Ihrem S3 Bucket erstellt hat.

Einen Export mit dem AWS CLI und anfordern AWS SDKs

Die folgenden Beispiele zeigen, wie eine vorhandene Tabelle in einen S3-Bucket exportiert wird.

> **Note**
>
> Bei diesem Verfahren wird davon ausgegangen, dass Sie die point-in-time Wiederherstellung aktiviert haben. Führen Sie den folgenden Befehl aus, um diese Funktion für die Tabelle `MusicCollection` zu aktivieren.
>
> ```
> aws dynamodb update-continuous-backups \
>     --table-name MusicCollection \
>     --point-in-time-recovery-specification PointInTimeRecoveryEnabled=True
> ```

Vollständiger Export

AWS CLI

> **Note**
>
> Wenn Sie einen kontenübergreifenden Tabellenexport anfordern, stellen Sie sicher, dass Sie die Option `--s3-bucket-owner` einfügen.

```
aws dynamodb export-table-to-point-in-time \
  --table-arn arn:aws:dynamodb:us-west-2:111122223333:table/MusicCollection \
  --s3-bucket ddb-export-musiccollection-9012345678 \
  --s3-prefix 2020-Nov \
  --export-format DYNAMODB_JSON \
  --export-time 1604632434 \
  --s3-bucket-owner 9012345678 \
  --s3-sse-algorithm AES256
```

Python

```
import boto3
from datetime import datetime

client = boto3.client('dynamodb')

client.export_table_to_point_in_time(
    TableArn='arn:aws:dynamodb:us-east-1:111122223333:table/TABLE',
    ExportTime=datetime(2023, 9, 20, 12, 0, 0),
    S3Bucket='bucket',
    S3Prefix='prefix',
    S3SseAlgorithm='AES256',
    ExportFormat='DYNAMODB_JSON'
)
```

Java

```
DynamoDbClient client = DynamoDbClient.create();

client.exportTableToPointInTime(b -> b
    .tableArn("arn:aws:dynamodb:us-east-1:111122223333:table/TABLE")
    .exportTime(Instant.parse("2023-09-20T12:00:00Z"))
    .s3Bucket("bucket")
```

```
    .s3Prefix("prefix")
    .s3SseAlgorithm(S3SseAlgorithm.AES256)
    .exportFormat(ExportFormat.DYNAMODB_JSON));
```

.NET

```
var client = new AmazonDynamoDBClient();

await client.ExportTableToPointInTimeAsync(new ExportTableToPointInTimeRequest
{
    TableArn = "arn:aws:dynamodb:us-east-1:111122223333:table/TABLE",
    ExportTime = new DateTime(2023, 9, 20, 12, 0, 0, DateTimeKind.Utc),
    S3Bucket = "bucket",
    S3Prefix = "prefix",
    S3SseAlgorithm = S3SseAlgorithm.AES256,
    ExportFormat = ExportFormat.DYNAMODB_JSON
});
```

JavaScript

```
import { DynamoDBClient, ExportTableToPointInTimeCommand } from "@aws-sdk/client-
dynamodb";

const client = new DynamoDBClient();

await client.send(new ExportTableToPointInTimeCommand({
    TableArn: "arn:aws:dynamodb:us-east-1:111122223333:table/TABLE",
    ExportTime: new Date("2023-09-20T12:00:00Z"),
    S3Bucket: "bucket",
    S3Prefix: "prefix",
    S3SseAlgorithm: "AES256",
    ExportFormat: "DYNAMODB_JSON"
}));
```

Go

```
cfg, _ := config.LoadDefaultConfig(context.TODO())
client := dynamodb.NewFromConfig(cfg)

exportTime := time.Date(2023, 9, 20, 12, 0, 0, 0, time.UTC)
client.ExportTableToPointInTime(context.TODO(),
 &dynamodb.ExportTableToPointInTimeInput{
```

```
    TableArn:       aws.String("arn:aws:dynamodb:us-east-1:111122223333:table/
TABLE"),
    ExportTime:     &exportTime,
    S3Bucket:       aws.String("bucket"),
    S3Prefix:       aws.String("prefix"),
    S3SseAlgorithm: types.S3SseAlgorithmAes256,
    ExportFormat:   types.ExportFormatDynamodbJson,
})
```

Inkrementeller Export

AWS CLI

```
aws dynamodb export-table-to-point-in-time \
  --table-arn arn:aws:dynamodb:REGION:ACCOUNT:table/TABLENAME \
  --s3-bucket BUCKET --s3-prefix PREFIX \
  --incremental-export-specification
 ExportFromTime=1693569600,ExportToTime=1693656000,ExportViewType=NEW_AND_OLD_IMAGES
 \
  --export-type INCREMENTAL_EXPORT
```

Python

```
import boto3
from datetime import datetime

client = boto3.client('dynamodb')

client.export_table_to_point_in_time(
    TableArn='arn:aws:dynamodb:us-east-1:111122223333:table/TABLE',
    IncrementalExportSpecification={
      'ExportFromTime': datetime(2023, 9, 20, 12, 0, 0),
      'ExportToTime': datetime(2023, 9, 20, 13, 0, 0),
      'ExportViewType': 'NEW_AND_OLD_IMAGES'
    },
    ExportType='INCREMENTAL_EXPORT',
    S3Bucket='bucket',
    S3Prefix='prefix',
    S3SseAlgorithm='AES256',
    ExportFormat='DYNAMODB_JSON'
)
```

Java

```
DynamoDbClient client = DynamoDbClient.create();

client.exportTableToPointInTime(b -> b
    .tableArn("arn:aws:dynamodb:us-east-1:111122223333:table/TABLE")
    .exportType(ExportType.INCREMENTAL_EXPORT)
    .incrementalExportSpecification(i -> i
        .exportFromTime(Instant.parse("2023-09-20T12:00:00Z"))
        .exportToTime(Instant.parse("2023-09-20T13:00:00Z"))
        .exportViewType(ExportViewType.NEW_AND_OLD_IMAGES))
    .s3Bucket("bucket")
    .s3Prefix("prefix")
    .s3SseAlgorithm(S3SseAlgorithm.AES256)
    .exportFormat(ExportFormat.DYNAMODB_JSON));
```

.NET

```
var client = new AmazonDynamoDBClient();

await client.ExportTableToPointInTimeAsync(new ExportTableToPointInTimeRequest
{
    TableArn = "arn:aws:dynamodb:us-east-1:111122223333:table/TABLE",
    ExportType = ExportType.INCREMENTAL_EXPORT,
    IncrementalExportSpecification = new IncrementalExportSpecification
    {
        ExportFromTime = new DateTime(2023, 9, 20, 12, 0, 0, DateTimeKind.Utc),
        ExportToTime = new DateTime(2023, 9, 20, 13, 0, 0, DateTimeKind.Utc),
        ExportViewType = ExportViewType.NEW_AND_OLD_IMAGES
    },
    S3Bucket = "bucket",
    S3Prefix = "prefix",
    S3SseAlgorithm = S3SseAlgorithm.AES256,
    ExportFormat = ExportFormat.DYNAMODB_JSON
});
```

JavaScript

```
import { DynamoDBClient, ExportTableToPointInTimeCommand } from "@aws-sdk/client-
dynamodb";

const client = new DynamoDBClient();
```

```
await client.send(new ExportTableToPointInTimeCommand({
    TableArn: "arn:aws:dynamodb:us-east-1:111122223333:table/TABLE",
    ExportType: "INCREMENTAL_EXPORT",
    IncrementalExportSpecification: {
        ExportFromTime: new Date("2023-09-20T12:00:00Z"),
        ExportToTime: new Date("2023-09-20T13:00:00Z"),
        ExportViewType: "NEW_AND_OLD_IMAGES"
    },
    S3Bucket: "bucket",
    S3Prefix: "prefix",
    S3SseAlgorithm: "AES256",
    ExportFormat: "DYNAMODB_JSON"
}));
```

Go

```
cfg, _ := config.LoadDefaultConfig(context.TODO())
client := dynamodb.NewFromConfig(cfg)

fromTime := time.Date(2023, 9, 20, 12, 0, 0, 0, time.UTC)
toTime := time.Date(2023, 9, 20, 13, 0, 0, 0, time.UTC)
client.ExportTableToPointInTime(context.TODO(),
 &dynamodb.ExportTableToPointInTimeInput{
    TableArn:   aws.String("arn:aws:dynamodb:us-east-1:111122223333:table/TABLE"),
    ExportType: types.ExportTypeIncrementalExport,
    IncrementalExportSpecification: &types.IncrementalExportSpecification{
        ExportFromTime: &fromTime,
        ExportToTime:   &toTime,
        ExportViewType: types.ExportViewTypeNewAndOldImages,
    },
    S3Bucket:       aws.String("bucket"),
    S3Prefix:       aws.String("prefix"),
    S3SseAlgorithm: types.S3SseAlgorithmAes256,
    ExportFormat:   types.ExportFormatDynamodbJson,
})
```

> **Note**
>
> Wenn Sie Ihren Export mit einem durch AWS Key Management Service (AWS KMS) geschützten Schlüssel verschlüsseln möchten, muss sich der Schlüssel in derselben Region wie der S3-Ziel-Bucket befinden.

Details zu vergangenen Exporten mit dem AWS CLI und abrufen AWS SDKs

Informationen zu Exportaufgaben, die Sie in der Vergangenheit ausgeführt haben, finden Sie mithilfe des `list-exports`-Befehls. Dieser Befehl gibt eine Liste aller Exporte zurück, die Sie in den letzten 90 Tagen erstellt haben. Beachten Sie, dass, obwohl die Metadaten der Exportaufgabe nach 90 Tagen ablaufen und Aufträge, die älter sind, nicht mehr vom `list-exports` Befehl zurückgegeben werden, die Objekte in Ihrem S3 Bucket so lange erhalten bleiben, wie es ihre Bucket-Richtlinien zulassen. DynamoDB löscht nie eines der Objekte, die es während eines Exports in Ihrem S3 Bucket erstellt hat.

Exporte haben den Status `PENDING`, bis sie entweder erfolgreich sind oder fehlschlagen. Wenn sie erfolgreich sind, ändert sich der Status zu `COMPLETED`. Wenn sie fehlschlagen, ändert sich der Status zu `FAILED` mit `failure_message` und `failure_reason`.

Exporte auflisten

AWS CLI

```
aws dynamodb list-exports \
    --table-arn arn:aws:dynamodb:us-east-1:111122223333:table/ProductCatalog
```

Python

```
import boto3

client = boto3.client('dynamodb')

print(
  client.list_exports(
     TableArn='arn:aws:dynamodb:us-east-1:111122223333:table/TABLE',
  )
)
```

Java

```
DynamoDbClient client = DynamoDbClient.create();

ListExportsResponse response = client.listExports(b -> b
    .tableArn("arn:aws:dynamodb:us-east-1:111122223333:table/TABLE"));

response.exportSummaries().forEach(System.out::println);
```

.NET

```
var client = new AmazonDynamoDBClient();

var response = await client.ListExportsAsync(new ListExportsRequest
{
    TableArn = "arn:aws:dynamodb:us-east-1:111122223333:table/TABLE"
});

response.ExportSummaries.ForEach(Console.WriteLine);
```

JavaScript

```
import { DynamoDBClient, ListExportsCommand } from "@aws-sdk/client-dynamodb";

const client = new DynamoDBClient();

const response = await client.send(new ListExportsCommand({
    TableArn: "arn:aws:dynamodb:us-east-1:111122223333:table/TABLE"
}));

console.log(response.ExportSummaries);
```

Go

```
cfg, _ := config.LoadDefaultConfig(context.TODO())
client := dynamodb.NewFromConfig(cfg)

response, _ := client.ListExports(context.TODO(), &dynamodb.ListExportsInput{
    TableArn: aws.String("arn:aws:dynamodb:us-east-1:111122223333:table/TABLE"),
})

fmt.Println(response.ExportSummaries)
```

Beschreiben Sie den Export

AWS CLI

```
aws dynamodb describe-export \
    --export-arn arn:aws:dynamodb:us-east-1:111122223333:table/ProductCatalog/
export/01695353076000-a1b2c3d4
```

Python

```
import boto3

client = boto3.client('dynamodb')

print(
  client.describe_export(
     ExportArn='arn:aws:dynamodb:us-east-1:111122223333:table/TABLE/
export/01695353076000-06e2188f',
  )['ExportDescription']
)
```

Java

```
DynamoDbClient client = DynamoDbClient.create();

DescribeExportResponse response = client.describeExport(b -> b
    .exportArn("arn:aws:dynamodb:us-east-1:111122223333:table/TABLE/
export/01695353076000-06e2188f"));

System.out.println(response.exportDescription());
```

.NET

```
var client = new AmazonDynamoDBClient();

var response = await client.DescribeExportAsync(new DescribeExportRequest
{
    ExportArn = "arn:aws:dynamodb:us-east-1:111122223333:table/TABLE/
export/01695353076000-06e2188f"
});

Console.WriteLine(response.ExportDescription);
```

JavaScript

```
import { DynamoDBClient, DescribeExportCommand } from "@aws-sdk/client-dynamodb";

const client = new DynamoDBClient();

const response = await client.send(new DescribeExportCommand({
```

```
    ExportArn: "arn:aws:dynamodb:us-east-1:111122223333:table/TABLE/
export/01695353076000-06e2188f"
}));

console.log(response.ExportDescription);
```

Go

```
cfg, _ := config.LoadDefaultConfig(context.TODO())
client := dynamodb.NewFromConfig(cfg)

response, _ := client.DescribeExport(context.TODO(), &dynamodb.DescribeExportInput{
    ExportArn: aws.String("arn:aws:dynamodb:us-east-1:111122223333:table/TABLE/
export/01695353076000-06e2188f"),
})

fmt.Println(response.ExportDescription)
```

Export-Ausgabeformat für DynamoDB-Tabellen

Ein DynamoDB-Tabellenexport umfasst Manifestdateien, zusätzlich zu den Dateien, die Ihre Tabellendaten enthalten. Diese Dateien werden alle in dem Amazon S3-Bucket gespeichert, den Sie in Ihrer Exportanforderung spezifizieren. Die folgenden Abschnitte beschreiben das Format und den Inhalt der einzelnen Ausgabeobjekte.

Vollständige Exportausgabe

Manifestdateien

DynamoDB erstellt Manifestdateien zusammen mit ihren Prüfsummendateien im angegebenen S3-Bucket für jede Exportanforderung.

```
export-prefix/AWSDynamoDB/ExportId/manifest-summary.json
export-prefix/AWSDynamoDB/ExportId/manifest-summary.checksum
export-prefix/AWSDynamoDB/ExportId/manifest-files.json
export-prefix/AWSDynamoDB/ExportId/manifest-files.checksum
```

Sie wählen ein **`export-prefix`**, wenn Sie einen Tabellenexport anfordern. Auf diese Weise können Sie Dateien im Ziel-S3-Bucket organisieren. Die **`ExportId`** ist ein eindeutiges vom Service generiertes Token, um sicherzustellen, dass mehrere Exporte in denselben S3-Bucket und `export-prefix` sich nicht gegenseitig überschreiben.

Bei dem Export wird mindestens 1 Datei pro Partition erstellt. Für leere Partitionen erstellt Ihre Exportanforderung eine leere Datei. Alle Elemente in jeder Datei stammen aus dem Hash-Keyspace der betreffenden Partition.

> **Note**
>
> DynamoDB erstellt auch eine leere Datei namens `_started`, die im selben Verzeichnis wie die Manifestdateien benannt ist. Diese Datei überprüft, ob der Ziel-Bucket beschreibbar ist und der Export begonnen hat. Es kann sicher gelöscht werden.

Das Übersichtsmanifest

Die `manifest-summary.json`-Datei enthält zusammenfassende Informationen zum Exportauftrag. Auf diese Weise können Sie feststellen, welche Datendateien im gemeinsam genutzten Datenordner mit diesem Export verknüpft sind. Das Format lautet folgendermaßen:

```
{
   "version": "2020-06-30",
   "exportArn": "arn:aws:dynamodb:us-east-1:123456789012:table/ProductCatalog/
export/01234567890123-a1b2c3d4",
   "startTime": "2020-11-04T07:28:34.028Z",
   "endTime": "2020-11-04T07:33:43.897Z",
   "tableArn": "arn:aws:dynamodb:us-east-1:123456789012:table/ProductCatalog",
   "tableId": "12345a12-abcd-123a-ab12-1234abc12345",
   "exportTime": "2020-11-04T07:28:34.028Z",
   "s3Bucket": "ddb-productcatalog-export",
   "s3Prefix": "2020-Nov",
   "s3SseAlgorithm": "AES256",
   "s3SseKmsKeyId": null,
   "manifestFilesS3Key": "AWSDynamoDB/01693685827463-2d8752fd/manifest-files.json",
   "billedSizeBytes": 0,
   "itemCount": 8,
   "outputFormat": "DYNAMODB_JSON",
   "exportType": "FULL_EXPORT"
}
```

Die Manifestdateien

Die `manifest-files.json`-Datei enthält Informationen zu den Dateien, die Ihre exportierten Tabellendaten enthalten. Die Datei hat das JSON-Lines-Format, was bedeutet, dass Zeilenumbrüche

als Elementtrennzeichen verwendet werden. Im folgenden Beispiel werden die Details einer Datendatei aus einem Dateimanifest aus Gründen der Lesbarkeit in mehreren Zeilen formatiert.

```
{
"itemCount": 8,
   "md5Checksum": "sQMSpEILNgoQmarvDFonGQ==",
   "etag": "af83d6f217c19b8b0fff8023d8ca4716-1",
   "dataFileS3Key": "AWSDynamoDB/01693685827463-2d8752fd/data/asdl123dasas.json.gz"
}
```

Datendateien

DynamoDB kann Ihre Tabellendaten in zwei Formaten exportieren: DynamoDB JSON und Amazon Ion. Unabhängig vom gewählten Format werden Ihre Daten in mehrere komprimierte Dateien geschrieben, die nach den Schlüsseln benannt sind. Diese Dateien sind auch in der `manifest-files.json`-Datei aufgeführt.

Die Verzeichnisstruktur Ihres Amazon S3-Buckets enthält nach einem vollständigen Export alle Ihre Manifestdateien und Datendateien im Ordner Export-ID.

```
amzn-s3-demo-bucket/DestinationPrefix
.
### AWSDynamoDB
    ### 01693685827463-2d8752fd     // the single full export
    #   ### manifest-files.json     // manifest points to files under 'data' subfolder
    #   ### manifest-files.checksum
    #   ### manifest-summary.json   // stores metadata about request
    #   ### manifest-summary.md5
    #   ### data                    // The data exported by full export
    #   #   ### asdl123dasas.json.gz
    #   #   ...
    #   ### _started                // empty file for permission check
```

DynamoDB-JSON

Ein Tabellenexport im DynamoDB-JSON-Format besteht aus mehreren `Item`-Objekten. Jedes einzelne Objekt befindet sich im Standard-JSON-Format von DynamoDB.

Beachten Sie beim Erstellen benutzerdefinierter Parser für DynamoDB-JSON-Exportdaten, dass das Format JSON Lines ist. Dies bedeutet, dass Zeilenumbrüche als Elementtrennzeichen verwendet werden. Viele AWS Dienste, wie Athena und AWS Glue, analysieren dieses Format automatisch.

Im folgenden Beispiel wurde ein einzelnes Element aus einem DynamoDB-JSON-Export aus Gründen der Lesbarkeit in mehreren Zeilen formatiert.

```
{
    "Item":{
        "Authors":{
            "SS":[
                "Author1",
                "Author2"
            ]
        },
        "Dimensions":{
            "S":"8.5 x 11.0 x 1.5"
        },
        "ISBN":{
            "S":"333-3333333333"
        },
        "Id":{
            "N":"103"
        },
        "InPublication":{
            "BOOL":false
        },
        "PageCount":{
            "N":"600"
        },
        "Price":{
            "N":"2000"
        },
        "ProductCategory":{
            "S":"Book"
        },
        "Title":{
            "S":"Book 103 Title"
        }
    }
}
```

Amazon Ion

Amazon Ion ist ein reich typisiertes, selbstbeschreibendes, hierarchisches Datenserialisierungsformat, das entwickelt wurde, um schnelle Entwicklungs-, Entkopplungs- und Effizienzprobleme zu bewältigen, denen sich täglich gegenübersehen, während umfangreiche,

serviceorientierte Architekturen entwickelt werden. DynamoDB unterstützt den Export von Tabellendaten in Textformat, die eine Obermenge von JSON ist.

Wenn Sie eine Tabelle in das Ion-Format exportieren, werden die in der Tabelle verwendeten DynamoDB-Datentypen Ion-Datentypen abgebildet. DynamoDB-Sets verwenden Ion-Typ-Anmerkungen, um den in der Quelltabelle verwendeten Datentyp zu verdeutlichen.

In der folgenden Tabelle ist die Zuordnung von DynamoDB-Datentypen zu Ion-Datentypen aufgeführt:

DynamoDB-Datentyp	Ion-Darstellung
String (S)	Zeichenfolge
BOOLEAN (BOOL)	bool
Nummer (N)	decimal
Binary (B)	blob
Satz (SS, NS, BS)	Liste (mit Typannotation $DynamoDB_SS, $DynamoDB_NS oder $DynamoDB_Bs)
Liste	auflisten
Map	struct

Elemente in einem Ionen-Export werden durch Zeilenumbrüche getrennt. Jede Zeile beginnt mit einer Ion-Versionsmarkierung, gefolgt von einem Element im Ionen-Format. Im folgenden Beispiel wurde ein Element aus einem Ion-Export aus Gründen der Lesbarkeit in mehreren Zeilen formatiert.

```
$ion_1_0 {
    Item:{
        Authors:$dynamodb_SS::["Author1","Author2"],
        Dimensions:"8.5 x 11.0 x 1.5",
        ISBN:"333-3333333333",
        Id:103.,
        InPublication:false,
        PageCount:6d2,
        Price:2d3,
        ProductCategory:"Book",
        Title:"Book 103 Title"
```

```
    }
}
```

Inkrementelle Exportausgabe

Manifestdateien

DynamoDB erstellt Manifestdateien zusammen mit ihren Prüfsummendateien im angegebenen S3-Bucket für jede Exportanforderung.

```
export-prefix/AWSDynamoDB/ExportId/manifest-summary.json
export-prefix/AWSDynamoDB/ExportId/manifest-summary.checksum
export-prefix/AWSDynamoDB/ExportId/manifest-files.json
export-prefix/AWSDynamoDB/ExportId/manifest-files.checksum
```

Sie wählen ein **`export-prefix`**, wenn Sie einen Tabellenexport anfordern. Auf diese Weise können Sie Dateien im Ziel-S3-Bucket organisieren. Die **`ExportId`** ist ein eindeutiges vom Service generiertes Token, um sicherzustellen, dass mehrere Exporte in denselben S3-Bucket und `export-prefix` sich nicht gegenseitig überschreiben.

Bei dem Export wird mindestens 1 Datei pro Partition erstellt. Für leere Partitionen erstellt Ihre Exportanforderung eine leere Datei. Alle Elemente in jeder Datei stammen aus dem Hash-Keyspace der betreffenden Partition.

> **Note**
>
> DynamoDB erstellt auch eine leere Datei namens `_started`, die im selben Verzeichnis wie die Manifestdateien benannt ist. Diese Datei überprüft, ob der Ziel-Bucket beschreibbar ist und der Export begonnen hat. Es kann sicher gelöscht werden.

Das Übersichtsmanifest

Die `manifest-summary.json`-Datei enthält zusammenfassende Informationen zum Exportauftrag. Auf diese Weise können Sie feststellen, welche Datendateien im gemeinsam genutzten Datenordner mit diesem Export verknüpft sind. Das Format lautet folgendermaßen:

```
{
 "version": "2023-08-01",
 "exportArn": "arn:aws:dynamodb:us-east-1:599882009758:table/export-test/
export/01695097218000-d6299cbd",
```

```
 "startTime": "2023-09-19T04:20:18.000Z",
 "endTime": "2023-09-19T04:40:24.780Z",
 "tableArn": "arn:aws:dynamodb:us-east-1:599882009758:table/export-test",
 "tableId": "b116b490-6460-4d4a-9a6b-5d360abf4fb3",
 "exportFromTime": "2023-09-18T17:00:00.000Z",
 "exportToTime": "2023-09-19T04:00:00.000Z",
 "s3Bucket": "jason-exports",
 "s3Prefix": "20230919-prefix",
 "s3SseAlgorithm": "AES256",
 "s3SseKmsKeyId": null,
 "manifestFilesS3Key": "20230919-prefix/AWSDynamoDB/01693685934212-ac809da5/manifest-
files.json",
 "billedSizeBytes": 20901239349,
 "itemCount": 169928274,
 "outputFormat": "DYNAMODB_JSON",
 "outputView": "NEW_AND_OLD_IMAGES",
 "exportType": "INCREMENTAL_EXPORT"
}
```

Die Manifestdateien

Die `manifest-files.json`-Datei enthält Informationen zu den Dateien, die Ihre exportierten Tabellendaten enthalten. Die Datei hat das JSON-Lines-Format, was bedeutet, dass Zeilenumbrüche als Elementtrennzeichen verwendet werden. Im folgenden Beispiel werden die Details einer Datendatei aus einem Dateimanifest aus Gründen der Lesbarkeit in mehreren Zeilen formatiert.

```
{
"itemCount": 8,
   "md5Checksum": "sQMSpEILNgoQmarvDFonGQ==",
   "etag": "af83d6f217c19b8b0fff8023d8ca4716-1",
   "dataFileS3Key": "AWSDynamoDB/data/sgad6417s6vss4p7owp0471bcq.json.gz"
}
```

Datendateien

DynamoDB kann Ihre Tabellendaten in zwei Formaten exportieren: DynamoDB JSON und Amazon Ion. Unabhängig vom gewählten Format werden Ihre Daten in mehrere komprimierte Dateien geschrieben, die nach den Schlüsseln benannt sind. Diese Dateien sind auch in der `manifest-files.json`-Datei aufgeführt.

Die Datendateien für inkrementelle Exporte sind alle in einem gemeinsamen Datenordner in Ihrem S3-Bucket enthalten. Ihre Manifestdateien befinden sich in Ihrem Export-ID-Ordner.

```
amzn-s3-demo-bucket/DestinationPrefix
.
### AWSDynamoDB
    ### 01693685934212-ac809da5     // an incremental export ID
    #   ### manifest-files.json     // manifest points to files under 'data' folder
    #   ### manifest-files.checksum
    #   ### manifest-summary.json   // stores metadata about request
    #   ### manifest-summary.md5
    #   ### _started                // empty file for permission check
    ### 01693686034521-ac809da5
    #   ### manifest-files.json
    #   ### manifest-files.checksum
    #   ### manifest-summary.json
    #   ### manifest-summary.md5
    #   ### _started
    ### data                        // stores all the data files for incremental
 exports
    #   ### sgad6417s6vss4p7owp0471bcq.json.gz
    #   ...
```

In Ihren Exportdateien enthält die Ausgabe jedes Elements einen Zeitstempel, der angibt, wann das Element in Ihrer Tabelle aktualisiert wurde, und eine Datenstruktur, die angibt, ob es sich um einen `insert`-, `update`- oder `delete`-Vorgang handelt. Der Zeitstempel basiert auf einer internen Systemuhr und kann von der Uhr Ihrer Anwendung abweichen. Für inkrementelle Exporte können Sie zwischen zwei Exportansichtstypen für Ihre Ausgabestruktur wählen: neue und alte Bilder oder nur neue Bilder.

- Neues Bild zeigt den aktuellen Status des Elements
- Altes Bild zeigt den Status des Elements unmittelbar vor dem angegebenen Startdatum und der angegebenen Startzeit

Ansichtstypen können hilfreich sein, wenn Sie sehen möchten, wie das Element innerhalb des Exportzeitraums geändert wurde. Sie können auch nützlich sein, um Ihre Downstream-Systeme effizient zu aktualisieren, insbesondere wenn diese Downstream-Systeme einen Partitionsschlüssel haben, der nicht mit Ihrem DynamoDB-Partitionsschlüssel identisch ist.

Anhand der Struktur der Ausgabe können Sie ableiten, ob es sich bei einem Element in Ihrer inkrementellen Exportausgabe um ein `insert`, `update` oder `delete` handelte. Die Struktur des

inkrementellen Exports und die entsprechenden Operationen sind in der folgenden Tabelle für beide Exportansichtstypen zusammengefasst.

Operation	Nur neue Bilder	Neue und alte Bilder
Einfügen	Schlüssel + neues Bild	Schlüssel + neues Bild
Aktualisierung	Schlüssel + neues Bild	Schlüssel + neues Bild + altes Bild
Delete	Schlüssel	Schlüssel + altes Bild
Einfügen + Löschen	Keine Ausgabe	Keine Ausgabe

DynamoDB-JSON

Ein Tabellenexport im DynamoDB-JSON-Format besteht aus einem Metadaten-Zeitstempel, der die Schreibzeit des Elements angibt, gefolgt von den Schlüsseln des Elements und den Werten. Nachfolgend sehen Sie ein Beispiel für eine DynamoDB-JSON-Ausgabe mit dem Exportansichtstyp Neue und alte Bilder.

```
// Ex 1: Insert
//   An insert means the item did not exist before the incremental export window
//   and was added during the incremental export window

{
   "Metadata": {
     "WriteTimestampMicros": "1680109764000000"
   },
   "Keys": {
     "PK": {
       "S": "CUST#100"
     }
   },
   "NewImage": {
     "PK": {
       "S": "CUST#100"
     },
     "FirstName": {
       "S": "John"
     },
```

```
      "LastName": {
        "S": "Don"
      }
    }
}

// Ex 2: Update
//   An update means the item existed before the incremental export window
//   and was updated during the incremental export window.
//   The OldImage would not be present if choosing "New images only".

{
   "Metadata": {
     "WriteTimestampMicros": "1680109764000000"
   },
   "Keys": {
     "PK": {
       "S": "CUST#200"
     }
   },
   "OldImage": {
     "PK": {
       "S": "CUST#200"
     },
     "FirstName": {
       "S": "Mary"
     },
     "LastName": {
       "S": "Grace"
     }
   },
   "NewImage": {
     "PK": {
       "S": "CUST#200"
     },
     "FirstName": {
       "S": "Mary"
     },
     "LastName": {
       "S": "Smith"
     }
   }
}
```

```
// Ex 3: Delete
//   A delete means the item existed before the incremental export window
//   and was deleted during the incremental export window
//   The OldImage would not be present if choosing "New images only".

{
   "Metadata": {
     "WriteTimestampMicros": "1680109764000000"
   },
   "Keys": {
     "PK": {
       "S": "CUST#300"
     }
   },
   "OldImage": {
     "PK": {
       "S": "CUST#300"
     },
     "FirstName": {
       "S": "Jose"
     },
     "LastName": {
       "S": "Hernandez"
     }
   }
}

// Ex 4: Insert + Delete
//   Nothing is exported if an item is inserted and deleted within the
//   incremental export window.
```

Amazon Ion

Amazon Ion ist ein reich typisiertes, selbstbeschreibendes, hierarchisches Datenserialisierungsformat, das entwickelt wurde, um schnelle Entwicklungs-, Entkopplungs- und Effizienzprobleme zu bewältigen, denen sich täglich gegenübersehen, während umfangreiche, serviceorientierte Architekturen entwickelt werden. DynamoDB unterstützt den Export von Tabellendaten in Textformat, die eine Obermenge von JSON ist.

Wenn Sie eine Tabelle in das Ion-Format exportieren, werden die in der Tabelle verwendeten DynamoDB-Datentypen Ion-Datentypen abgebildet. DynamoDB-Sets verwenden Ion-Typ-Anmerkungen, um den in der Quelltabelle verwendeten Datentyp zu verdeutlichen.

In der folgenden Tabelle ist die Zuordnung von DynamoDB-Datentypen zu Ion-Datentypen aufgeführt:

DynamoDB-Datentyp	Ion-Darstellung
String (S)	Zeichenfolge
BOOLEAN (BOOL)	bool
Nummer (N)	decimal
Binary (B)	blob
Satz (SS, NS, BS)	Liste (mit Typannotation $DynamoDB_SS, $DynamoDB_NS oder $DynamoDB_Bs)
Liste	auflisten
Map	struct

Elemente in einem Ionen-Export werden durch Zeilenumbrüche getrennt. Jede Zeile beginnt mit einer Ion-Versionsmarkierung, gefolgt von einem Element im Ionen-Format. Im folgenden Beispiel wurde ein Element aus einem Ion-Export aus Gründen der Lesbarkeit in mehreren Zeilen formatiert.

```
$ion_1_0 {
    Record:{
        Keys:{
             ISBN:"333-3333333333"
         },
        Metadata:{
            WriteTimestampMicros:1684374845117899.
        },
        OldImage:{
            Authors:$dynamodb_SS::["Author1","Author2"],
            ISBN:"333-3333333333",
            Id:103.,
            InPublication:false,
            ProductCategory:"Book",
            Title:"Book 103 Title"
        },
        NewImage:{
            Authors:$dynamodb_SS::["Author1","Author2"],
```

```
            Dimensions:"8.5 x 11.0 x 1.5",
            ISBN:"333-3333333333",
            Id:103.,
            InPublication:true,
            PageCount:6d2,
            Price:2d3,
            ProductCategory:"Book",
            Title:"Book 103 Title"
        }
    }
}
```

DynamoDB Zero-ETL-Integration mit Amazon Lakehouse SageMaker

Die DynamoDB-Zero-ETL-Integration mit Amazon SageMaker Lakehouse macht die Erstellung benutzerdefinierter Datenverschiebungspipelines überflüssig, da DynamoDB-Daten automatisch nach Amazon Lakehouse repliziert werden. SageMaker Diese codefreie Integration hilft Kunden dabei, Analyse-Workloads für ihre DynamoDB-Daten mithilfe von Amazon SageMaker Lakehouse auszuführen, ohne DynamoDB-Tabellenkapazität zu verbrauchen. Die Integration exportiert automatisch Daten aus Ihrer Tabelle und hält das Ziel auf dem neuesten Stand, normalerweise innerhalb von 15 bis 30 Minuten.

Themen

- DynamoDB Zero-ETL-Integration mit Amazon Lakehouse SageMaker

DynamoDB Zero-ETL-Integration mit Amazon Lakehouse SageMaker

Die Einrichtung einer Integration zwischen der DynamoDB-Tabelle und Amazon SageMaker Lakehouse erfordert Voraussetzungen wie die Konfiguration von IAM-Rollen, die für den Zugriff auf Daten von der Quelle und zum Schreiben in das Ziel AWS Glue verwendet werden, und die Verwendung von KMS-Schlüsseln zur Verschlüsselung der Daten im Zwischen- oder Zielspeicherort.

Themen

- Voraussetzungen für die Erstellung einer DynamoDB-Zero-ETL-Integration mit Amazon Lakehouse SageMaker
- Erstellen einer DynamoDB-Zero-ETL-Integration mit Amazon Lakehouse SageMaker

- CloudWatch Metriken für die Integration anzeigen

Voraussetzungen für die Erstellung einer DynamoDB-Zero-ETL-Integration mit Amazon Lakehouse SageMaker

Um eine Zero-ETL-Integration mit einer DynamoDB-Quelle zu konfigurieren, müssen Sie eine RBAC-Richtlinie (Resource-Based Access) einrichten, die den Zugriff auf und den Export von Daten aus der AWS Glue DynamoDB-Tabelle ermöglicht. Die Richtlinie sollte bestimmte Berechtigungen wie `ExportTableToPointInTime`, `DescribeTable` und `DescribeExport` mit Bedingungen beinhalten, die den Zugriff auf ein bestimmtes AWS-Konto und eine Region einschränken. Weitere Informationen finden Sie unter Konfigurieren einer Amazon-DynamoDB-Quelle.

Point-in-time Recovery (PITR) muss für die Tabelle aktiviert sein, und Sie können die Richtlinie mithilfe von Befehlen anwenden. AWS CLI Die Richtlinie kann weiter verfeinert werden, indem der vollständige ARN für die Integration für eine restriktivere Zugriffskontrolle angegeben wird. Weitere Informationen finden Sie unter Voraussetzungen für die Einrichtung einer Null-ETL-Integration.

Erstellen einer DynamoDB-Zero-ETL-Integration mit Amazon Lakehouse SageMaker

Wenn Sie die Integrationsvoraussetzungen erfüllt haben, können Sie die Null-ETL-Integration gemäß den folgenden Anleitungen erstellen, ändern oder löschen:

Erstellen einer Integration

So wird eine Integration erstellt

1. Melden Sie sich bei der AWS Management Console an und öffnen Sie die Amazon DynamoDB DynamoDB-Konsole unter. https://console.aws.amazon.com/dynamodbv2
2. Wählen Sie im Navigationsbereich Integrationen aus.
3. Wählen Sie Create Zero-ETL integration with Amazon SageMaker Lakehouse und dann Next aus.
4. Informationen zum Erstellen einer Integration finden Sie unter Erstellen einer Integration.
5. Informationen zum Ändern einer Integration finden Sie unter Modifying an integration.
6. Informationen zum Löschen einer Integration finden Sie unter Deleting an integration.
7. Informationen zum Einrichten einer kontenübergreifenden Integration finden Sie unter Einrichten einer kontenübergreifenden Integration.

Aktivieren der Komprimierung für Amazon S3 Tables

Sie können die Komprimierung aktivieren, um die Abfrageleistung in Amazon Athena zu verbessern.

Schließen Sie zunächst die erforderlichen Einstellungen für die Komprimierungsressourcen ab, einschließlich der Konfiguration der erforderlichen IAM-Rolle. Detaillierte Schritte zur Konfiguration der IAM-Rolle finden Sie in der Lake-Formation-Dokumentation. Weitere Informationen finden Sie unter Optimieren von Tabellen für die Komprimierung.

Um die Komprimierung für die während der Integration erstellte AWS Glue Tabelle zu aktivieren, folgen Sie dem Prozess zur Aktivierung der Lake Formation Formation-Komprimierung. Dies trägt dazu bei, die Leistung und Abfrageeffizienz Ihrer Tabelle zu optimieren.

CloudWatch Metriken für die Integration anzeigen

Sobald eine Integration abgeschlossen ist, können Sie diese CloudWatch Metriken und EventBridge Benachrichtigungen sehen, die in Ihrem Konto für jeden AWS Glue Job generiert wurden. Weitere Informationen finden Sie unter Überwachen einer Integration.

DynamoDB Zero-ETL-Integration mit Amazon Service OpenSearch

Amazon DynamoDB bietet über das DynamoDB-Plugin für Ingestion eine Zero-ETL-Integration mit Amazon OpenSearch Service. OpenSearch Amazon OpenSearch Ingestion bietet ein vollständig verwaltetes Erlebnis ohne Code für die Aufnahme von Daten in Amazon Service. OpenSearch

Mit dem DynamoDB-Plug-In für OpenSearch Ingestion können Sie eine oder mehrere DynamoDB-Tabellen als Quelle für die Aufnahme in einen oder mehrere Service-Indizes verwenden. OpenSearch Sie können Ihre OpenSearch Ingestion-Pipelines mit DynamoDB als Quelle entweder über OpenSearch Ingestion oder DynamoDB-Integrationen in der durchsuchen und konfigurieren. AWS-Managementkonsole

- Um mit Ingestion zu beginnen, folgen Sie den Anweisungen im OpenSearch Ingestion-Leitfaden „Erste Schritte“. OpenSearch
- Weitere Informationen zu den Voraussetzungen und allen Konfigurationsoptionen für das DynamoDB-Plugin finden Sie in der Dokumentation zum DynamoDB-Plugin für Ingestion. OpenSearch

Funktionsweise

Das Plugin verwendet den DynamoDB-Export nach Amazon S3, um einen ersten Snapshot zu erstellen, in den geladen werden kann. OpenSearch Nach dem Laden des Snapshots repliziert das Plugin mithilfe von DynamoDB-Streams alle weiteren Änderungen nahezu in Echtzeit. Jedes Element wird in OpenSearch Ingestion als Ereignis verarbeitet und kann mit Prozessor-Plugins geändert werden. Es ist möglich, Attribute zu löschen oder zusammengesetzte Attribute zu erstellen und sie über Routen an verschiedene Indizes zu senden.

Sie müssen point-in-time Recovery (PITR) aktiviert haben, um den Export nach Amazon S3 verwenden zu können. Für die Verwendung müssen außerdem DynamoDB-Streams (mit aktivierter Option Neues und altes Abbild) aktiviert werden. Durch Ausschließen von Exportoptionen ist es möglich, eine Pipeline ohne Snapshot zu erstellen.

Sie können auch eine Pipeline mit nur einem Snapshot und ohne Updates erstellen, indem Sie die Streams-Einstellungen ausschließen. Das Plugin verwendet keinen Lese- oder Schreibdurchsatz für die Tabelle, sodass es sicher verwendet werden kann, ohne den Produktionsdatenverkehr zu beeinträchtigen. Berücksichtigen Sie vor der Erstellung dieser oder anderer Integrationen, dass die Anzahl der parallelen Verbraucher in einem Stream begrenzt ist. Weitere Überlegungen finden Sie unter the section called “Bewährte Methoden für die Integration”.

Bei einfachen Pipelines kann eine einzelne OpenSearch Recheneinheit (OCU) etwa 1 MB Schreibvorgänge pro Sekunde verarbeiten. Dies entspricht etwa 1000 Write-Request-Units (WCUs). Abhängig von der Komplexität Ihrer Pipeline und anderen Faktoren kann diese Anzahl höher oder niedriger ausfallen.

OpenSearch Ingestion unterstützt eine Dead-Letter-Warteschlange (DLQ) für Ereignisse, die nicht behebbare Fehler verursachen. Darüber hinaus kann die Pipeline ohne Benutzereingriff an der Stelle fortgesetzt werden, an der sie aufgehört hat, selbst wenn es zu einer Unterbrechung des Dienstes mit DynamoDB, der Pipeline oder Amazon OpenSearch Service kommt.

Wenn die Unterbrechung länger als 24 Stunden andauert, kann dies einen Verlust von Aktualisierungen zur Folge haben. Die Pipeline verarbeitet allerdings weiterhin die Aktualisierungen, die bei Wiederherstellung der Verfügbarkeit noch vorhanden waren. Sie müssten einen neuen Index erstellen, um alle Unregelmäßigkeiten zu beheben, die auf die ausgelassenen Ereignisse zurückzuführen sind, es sei denn, sie befinden sich in der Warteschlange für unzustellbare Nachrichten.

Alle Einstellungen und Details für das Plugin finden Sie in der Dokumentation zum OpenSearchIngestion DynamoDB-Plug-In.

Integrierte Erstellungsumgebung über die Konsole

DynamoDB und OpenSearch Service verfügen über ein integriertes Erlebnis in der AWS-Managementkonsole, was den Einführungsprozess rationalisiert. Wenn Sie diese Schritte ausführen, wählt der Service automatisch den DynamoDB-Blueprint aus und fügt die entsprechenden DynamoDB-Informationen für Sie hinzu.

Um eine Integration zu erstellen, folgen Sie den Anweisungen im Leitfaden Erste Schritte mit OpenSearch Ingestion. Wenn Sie zu Schritt 3: Pipeline erstellen gelangen, ersetzen Sie die Schritte 1 und 2 durch die folgenden Schritte:

1. Navigieren Sie zur DynamoDB-Konsole.
2. Wählen Sie im Navigationsbereich links die Option Integration aus.
3. Wählen Sie die DynamoDB-Tabelle aus, in die Sie replizieren möchten. OpenSearch
4. Wählen Sie Erstellen aus.

Ab hier können Sie mit dem Rest des Tutorials fortfahren.

Nächste Schritte

Ein besseres Verständnis der Integration von DynamoDB in OpenSearch Service finden Sie im Folgenden:

- Erste Schritte mit Amazon OpenSearch Ingestion
- DynamoDB-Plugin-Konfiguration und ihre Anforderungen

Umgang mit grundlegenden Änderungen am Index

OpenSearch kann Ihrem Index dynamisch neue Attribute hinzufügen. Nachdem die Zuweisungsvorlage allerdings für einen bestimmten Schlüssel festgelegt wurde, müssen Sie zusätzliche Maßnahmen ergreifen, um sie zu ändern. Wenn die Änderung eine erneute Verarbeitung aller Daten in der DynamoDB-Tabelle erfordert, müssen Sie außerdem Maßnahmen ergreifen, um einen neuen Export zu initiieren.

> **Note**
>
> Bei allen genannten Optionen können weiterhin Probleme auftreten, wenn in der DynamoDB-Tabelle Typkonflikte mit der von Ihnen angegebenen Zuweisungsvorlage auftreten. Stellen Sie sicher, dass Sie eine Warteschlange für unzustellbare Nachrichten (DLQ) aktiviert haben. Auf diese Weise können Sie leichter nachvollziehen, was an dem Datensatz falsch sein könnte, der zu einem Konflikt führt, wenn er in Ihren Index aufgenommen wird. OpenSearch

Themen

- Funktionsweise
- Löschen Sie den Index und setzen Sie die Pipeline zurück (Pipeline-zentrierte Option)
- Löschen Sie den Index und setzen Sie die Pipeline zurück (Pipeline-zentrierte Option)
- Erstellen und Speichern eines neuen Index (Online-Option)
- Bewährte Methoden zum Vermeiden und Debuggen von Typkonflikten

Funktionsweise

Im Folgenden finden Sie einen kurzen Überblick über die Maßnahmen für den Umgang mit grundlegenden Änderungen am Index. Sehen Sie sich die step-by-step Verfahren in den folgenden Abschnitten an.

- Pipeline anhalten und starten: Mit dieser Option wird der Status der Pipeline zurückgesetzt, und die Pipeline wird mit einem neuen vollständigen Export neu gestartet. Sie ist nicht destruktiv, d. h. der Index oder Daten in DynamoDB werden nicht gelöscht. Wenn Sie vorher keinen neuen Index erstellen, werden möglicherweise viele Fehler aufgrund von Versionskonflikten ausgegeben, da beim Export versucht wird, ältere Dokumente als die aktuelle `_version` in den Index einzufügen. Diese Fehler können ignoriert werden. Ihnen werden keine Kosten für eine angehaltene Pipeline in Rechnung gestellt.
- Pipeline aktualisieren: Durch diese Option wird die Konfiguration in der Pipeline mit einem Blau/Grün-Ansatz aktualisiert, ohne dass der Status verloren geht. Wenn Sie wesentliche Änderungen an Ihrer Pipeline vornehmen (z. B. neue Routen, Indizes oder Schlüssel zu vorhandenen Indizes hinzufügen), müssen Sie möglicherweise die Pipeline vollständig zurücksetzen und den Index neu erstellen. Bei dieser Option wird kein vollständiger Export durchgeführt.

- Index löschen und neu erstellen: Mit dieser Option werden Ihre Daten und Zuweisungseinstellungen aus Ihrem Index entfernt. Sie sollten dies tun, bevor Sie grundlegende Änderungen an Ihren Zuweisungen vornehmen. Dadurch werden alle Anwendungen, die auf den Index angewiesen sind, unterbrochen, bis der Index neu erstellt und synchronisiert wurde. Durch das Löschen des Index wird kein neuer Export initiiert. Sie sollten Ihren Index erst nach der Aktualisierung der Pipeline löschen. Andernfalls wird er möglicherweise neu erstellt, bevor Sie Ihre Einstellungen aktualisieren.

Löschen Sie den Index und setzen Sie die Pipeline zurück (Pipeline-zentrierte Option)

Dies ist oft die schnellste Option, wenn Sie sich noch in der Entwicklungsphase befinden. Sie löschen Ihren Index in OpenSearch Service und beenden und starten dann Ihre Pipeline, um einen neuen Export all Ihrer Daten zu starten. Dadurch wird sichergestellt, dass keine Konflikte zwischen Zuweisungsvorlagen und vorhandenen Indizes auftreten und dass Daten aus einer unvollständigen verarbeiteten Tabelle nicht verloren gehen.

1. Stoppen Sie die AWS-Managementkonsole Pipeline entweder über das oder mithilfe der StopPipelineAPI-Operation mit dem AWS CLI oder einem SDK.
2. Aktualisieren Sie die Pipeline-Konfiguration mit Ihren neuen Änderungen.
3. Löschen Sie Ihren Index im OpenSearch Service, entweder über einen `REST` API-Aufruf oder über Ihr OpenSearch Dashboard.
4. Starten Sie die Pipeline entweder über die Konsole oder mithilfe der `StartPipeline` API-Operation mit dem AWS CLI oder einem SDK.

   > **Note**
   >
   > Dadurch wird ein neuer vollständiger Export initiiert, für den zusätzliche Kosten anfallen.

5. Achten Sie auf unerwartete Probleme, da für die Erstellung des Index ein neuer Export generiert wird.
6. Vergewissern Sie sich im OpenSearch Service, dass der Index Ihren Erwartungen entspricht.

Nachdem der Export abgeschlossen ist und das Lesen aus dem Stream fortgesetzt wird, sind Ihre DynamoDB-Tabellendaten jetzt im Index verfügbar.

Löschen Sie den Index und setzen Sie die Pipeline zurück (Pipeline-zentrierte Option)

Diese Methode eignet sich gut, wenn Sie viele Iterationen am Indexdesign in OpenSearch Service durchführen müssen, bevor Sie die Pipeline von DynamoDB aus fortsetzen. Dies kann für die Entwicklung nützlich sein, wenn Sie anhand Ihrer Suchmuster eine sehr schnelle Iteration durchführen und vermeiden möchten, dass zwischen den einzelnen Iterationen neue Exporte durchgeführt werden.

1. Stoppen Sie die Pipeline entweder über das oder AWS-Managementkonsole, indem Sie den StopPipelineAPI-Vorgang mit dem AWS CLI oder einem SDK aufrufen.
2. Löschen Sie Ihren Index und erstellen Sie ihn OpenSearch mit der Mapping-Vorlage, die Sie verwenden möchten, neu. Sie können durch manuelles Hinzufügen einiger Beispieldaten überprüfen, ob Ihre Suchanfragen wie beabsichtigt funktionieren. Bei potenziellen Konflikten zwischen den Beispieldaten und den Daten aus DynamoDB löschen Sie die Beispieldaten unbedingt, bevor Sie mit dem nächsten Schritt fortfahren.
3. Wenn Sie eine Indexierungsvorlage in Ihrer Pipeline haben, entfernen Sie diese oder ersetzen Sie sie durch die Vorlage, die Sie bereits in OpenSearch Service erstellt haben. Stellen Sie sicher, dass der Name des Index mit dem Namen in der Pipeline übereinstimmt.
4. Starten Sie die Pipeline entweder über die Konsole oder indem Sie den `StartPipeline` API-Vorgang mit dem AWS CLI oder einem SDK aufrufen.

   > **Note**
   >
   > Dadurch wird ein neuer vollständiger Export initiiert, für den zusätzliche Kosten anfallen.

5. Achten Sie auf unerwartete Probleme, da für die Erstellung des Index ein neuer Export generiert wird.

Nachdem der Export abgeschlossen ist und das Lesen aus dem Stream fortgesetzt wird, sind Ihre DynamoDB-Tabellendaten jetzt im Index verfügbar.

Erstellen und Speichern eines neuen Index (Online-Option)

Diese Methode eignet sich gut, wenn Sie Ihre Zuweisungsvorlage aktualisieren müssen, Ihren Index aber derzeit in der Produktion verwenden. Dadurch wird ein ganz neuer Index erstellt, in den Sie die Anwendung verschieben müssen, nachdem sie synchronisiert und validiert wurde.

> **Note**
>
> Dadurch wird ein weiterer Verbraucher im Stream erstellt. Dies kann ein Problem sein, wenn Sie auch andere Verbraucher wie AWS Lambda oder globale Tabellen haben. Möglicherweise müssen Sie die Aktualisierungen Ihrer vorhandenen Pipeline unterbrechen, um Kapazität zum Laden des neuen Index zu schaffen.

1. Erstellen Sie eine neue Pipeline mit neuen Einstellungen und einem anderen Indexnamen.
2. Überwachen Sie den neuen Index auf unerwartete Probleme.
3. Verschieben Sie die Anwendung in den neuen Index.
4. Halten Sie alte Pipeline an und löschen Sie sie, nachdem Sie überprüft haben, ob alles ordnungsgemäß funktioniert.

Bewährte Methoden zum Vermeiden und Debuggen von Typkonflikten

- Verwenden Sie immer eine Warteschlange für unzustellbare Nachrichten (Dead-Letter Queue, DLQ), um das Debuggen zu erleichtern.
- Verwenden Sie immer eine Indexvorlage mit Zuweisung und legen Sie `include_keys` fest. OpenSearch Service ordnet neue Schlüssel zwar dynamisch zu, dies kann jedoch zu Problemen mit unerwartetem Verhalten (z. B. wenn erwartet wird`GeoPoint`, dass etwas ein ist, das aber als `string` Oder erstellt wird`object`) oder zu Fehlern (z. B. wenn eine `number` Kombination aus `float` AND-Werten `long` vorliegt) auftreten.
- Wenn der vorhandene Index in der Produktion weiterhin funktionieren soll, können Sie auch einen beliebigen der vorherigen Schritte zum Löschen von Indizes ersetzen, indem Sie den Index in der Pipeline-Konfigurationsdatei einfach umbenennen. Dadurch wird ein ganz neuer Index erstellt. Sie müssen Ihre Anwendung dann aktualisieren, sodass sie nach der Fertigstellung des neuen Index darauf verweist.
- Wenn Sie ein Problem mit der Typkonvertierung haben, das Sie mit einem Prozessor beheben, können Sie es mit `UpdatePipeline` testen. Dazu müssen Sie die Warteschlangen für unzustellbare Nachrichten beenden und starten oder bearbeiten, damit alle zuvor übersprungenen fehlerhaften Dokumente korrigiert werden.

Bewährte Methoden für die Arbeit mit DynamoDB Zero-ETL Integration und Service OpenSearch

DynamoDB verfügt über eine DynamoDB-Zero-ETL-Integration mit Amazon Service. OpenSearch Weitere Informationen finden Sie im DynamoDB-Plug-In für OpenSearch Ingestion und in spezifischen Best Practices für Amazon Service. OpenSearch

Konfiguration

- Indizieren Sie nur Daten, die Sie für die Durchführung von Suchvorgängen benötigen. Verwenden Sie immer eine Zuweisungsvorlage (`template_type: index_template` und `template_content`) und `include_keys`, um diese zu implementieren.
- Überwachen Sie Ihre Protokolle auf Fehler, die mit Typkonflikten zusammenhängen. OpenSearch Der Dienst erwartet, dass alle Werte für einen bestimmten Schlüssel denselben Typ haben. Bei Nichtübereinstimmungen werden Ausnahmen generiert. Wenn ein solcher Fehler auftritt, können Sie einen Prozessor hinzufügen, um zu erkennen, ob ein bestimmter Schlüssel immer denselben Wert hat.
- Verwenden Sie im Allgemeinen den Metadatenwert `primary_key` für den Wert `document_id`. In OpenSearch Service entspricht die Dokument-ID dem Primärschlüssel in DynamoDB. Die Verwendung des Primärschlüssels erleichtert das Auffinden des Dokuments und stellt sicher, dass Aktualisierungen konsistent und ohne Konflikte in das Dokument repliziert werden.

  Sie können zum Abrufen des Primärschlüssels die Helper-Funktion `getMetadata` verwenden (z. B. `document_id: "${getMetadata('primary_key')}"`). Wenn Sie einen zusammengesetzten Primärschlüssel verwenden, wird dieser von der Hilfsfunktion verkettet.
- Verwenden Sie im Allgemeinen den Metadatenwert `opensearch_action` für den Wert `action`. Dadurch wird sichergestellt, dass Updates so repliziert werden, dass die Daten in OpenSearch Service dem neuesten Status in DynamoDB entsprechen.

  Sie können zum Abrufen des Primärschlüssels die Helper-Funktion `getMetadata` verwenden (z. B. `action: "${getMetadata('opensearch_action')}"`). In Anwendungsfällen wie Filteroperationen können Sie den Typ des Stream-Ereignisses auch mit `dynamodb_event_name` abrufen. In der Regel sollten Sie ihn jedoch nicht für die Einstellung `action` verwenden.

Beobachtbarkeit

- Verwenden Sie auf Ihren Speichersystemen immer eine Warteschlange (Dead-Letter Queue, DLQ), um gelöschte Ereignisse zu verarbeiten OpenSearch . DynamoDB ist im Allgemeinen weniger strukturiert als OpenSearch Service, und es ist immer möglich, dass etwas Unerwartetes passiert. Mit einer Warteschlange für unzustellbare Nachrichten können Sie einzelne Ereignisse wiederherstellen und sogar den Wiederherstellungsprozess automatisieren. Auf diese Weise müssen Sie nicht den gesamten Index neu erstellen.
- Richten Sie immer Warnmeldungen ein, sodass der erwartete Betrag durch eine verzögerte Replikation nicht überschritten wird. In der Regel kann von einer Minute ausgegangen werden, ohne dass die Warnung zu laut wird. Dies kann variieren, je nachdem, wie stark Ihr Schreibverkehr ist und wie hoch Ihre OpenSearch Compute Unit (OCU) -Einstellungen in der Pipeline sind.

  Wenn die Replikationsverzögerung bei mehr als 24 Stunden liegt, werden im Stream Ereignisse gelöscht, und es treten Probleme mit der Genauigkeit auf, sofern Sie den Index nicht von Grund auf neu erstellen.

Skalierung

- Verwenden Sie Auto Scaling für Pipelines, um die Skalierung nach oben oder unten zu unterstützen OCUs , um sie optimal an die Arbeitslast anzupassen.
- Für bereitgestellte Durchsatztabellen ohne auto Skalierung empfehlen wir die Einstellung auf der OCUs Grundlage Ihrer Schreibkapazitätseinheiten (WCUs) geteilt durch 1000. Legen Sie als Minimum 1 OCU unter diesem Wert (mindestens 1) und als Maximum mindestens 1 OCU über diesem Wert fest.
  - Formel:

    ```
    OCU_minimum = GREATEST((table_WCU / 1000) - 1, 1)
    OCU_maximum = (table_WCU / 1000) + 1
    ```

  - Beispiel: Für Ihre Tabelle wurden 25000 WCUs bereitgestellt. Ihre Pipelines OCUs sollten auf mindestens 24 (25000/1000 — 1) und einen Höchstwert von mindestens 26 (25000/1000 + 1) eingestellt sein.
- Für bereitgestellte Durchsatztabellen mit Auto Scaling empfehlen wir eine Einstellung, die auf Ihrem Mindest- und Höchstwert WCUs, geteilt durch 1000, OCUs basiert. Legen Sie als Minimum 1 OCU unter dem Minimum von DynamoDB und als Maximum mindestens 1 OCU über dem Maximum von DynamoDB fest.

- Formel:

```
OCU_minimum = GREATEST((table_minimum_WCU / 1000) - 1, 1)
OCU_maximum = (table_maximum_WCU / 1000) + 1
```

- Beispiel: Die Tabelle verfügt über eine Auto-Scaling-Richtlinie mit einem Minimum von 8.000 und einem Maximum von 14.000. Ihre Pipelines OCUs sollten auf mindestens 7 (8000/1000 - 1) und maximal 15 (14000/1000 + 1) eingestellt sein.

- Für On-Demand-Durchsatztabellen empfehlen wir eine Einstellung, die auf Ihrem typischen Peak and Valley für Schreibanforderungseinheiten pro Sekunde OCUs basiert. Je nachdem, welche Aggregation Ihnen zur Verfügung steht, müssen Sie möglicherweise den Durchschnitt über einen längeren Zeitraum ermitteln. Legen Sie als Minimum 1 OCU unter dem Minimum von DynamoDB und als Maximum mindestens 1 OCU über dem Maximum von DynamoDB fest.
  - Formel:

```
# Assuming we have writes aggregated at the minute level
OCU_minimum = GREATEST((min(table_writes_1min) / (60 * 1000)) - 1, 1)
OCU_maximum = (max(table_writes_1min) / (60 * 1000)) + 1
```

  - Beispiel: Die Tabelle weist einen durchschnittlichen Tiefstwert von 300 und einen durchschnittlichen Höchstwert von 4.300 Schreibanforderungseinheiten pro Sekunde auf. Ihre Pipelines OCUs sollten auf mindestens 1 (300/1000 — 1, aber mindestens 1) und maximal 5 (4300/1000 + 1) eingestellt sein.
- Folgen Sie den bewährten Methoden zur Skalierung Ihrer Ziel-Serviceindizes. OpenSearch Eine Unterskalierung der Indizes verlangsamt die Aufnahme aus DynamoDB und kann zu Verzögerungen führen.

> **Note**
>
> `GREATEST` ist eine SQL-Funktion, die bei einer gegebenen Anzahl von Argumenten das Argument mit dem größten Wert zurückgibt.

Integration von DynamoDB mit Amazon EventBridge

Amazon DynamoDB verfügt über DynamoDB Streams für die Erfassung von Änderungsdaten und somit die Erfassung von Änderungen in DynamoDB-Tabellen auf Elementebene. DynamoDB

Streams kann Lambda-Funktionen zur Verarbeitung dieser Änderungen aufrufen, wodurch eine ereignisgesteuerte Integration in andere Services und Anwendungen ermöglicht wird. DynamoDB Streams unterstützt auch das Filtern als effiziente und gezielte Ereignisverarbeitungsmethode.

DynamoDB Streams unterstützt bis zu zwei gleichzeitige Verbraucher pro Shard sowie das Filtern mittels Lambda-Ereignisfilterung, sodass nur Elemente verarbeitet werden, die bestimmten Kriterien entsprechen. Manche Kunden benötigen Unterstützung für mehr als zwei Verbraucher. Andere wiederum müssen möglicherweise Änderungsereignisse vor der Verarbeitung anreichern oder erweiterte Filter- und Routingfunktionen verwenden.

Die Integration von DynamoDB mit EventBridge kann diese Anforderungen unterstützen.

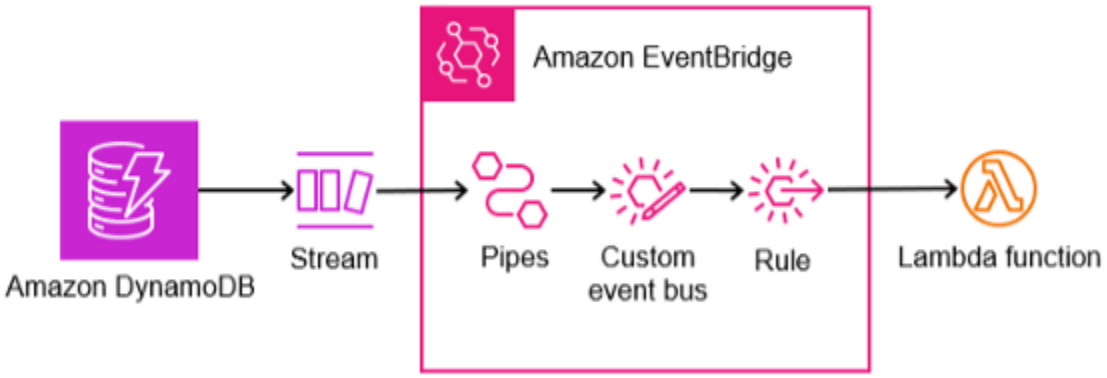
Amazon EventBridge ist ein serverloser Service, der Ereignisse verwendet, um Anwendungskomponenten miteinander zu verbinden, sodass Sie leichter skalierbare, ereignisgesteuerte Anwendungen erstellen können. EventBridge bietet native Integration mit Amazon DynamoDB über EventBridge Pipes und ermöglicht so einen nahtlosen Datenfluss von DynamoDB zu einem Bus. EventBridge Dieser kann dann über eine Reihe von Regeln und Zielen auf mehrere Anwendungen und Services ausgeweitet werden.

Themen

- Funktionsweise
- Erstellen einer Integration über die Konsole
- Nächste Schritte

Funktionsweise

Die Integration zwischen DynamoDB und EventBridge Pipes verwendet DynamoDB Streams, um eine zeitlich geordnete Abfolge von Änderungen auf Elementebene in einer DynamoDB-Tabelle zu erfassen. Jeder auf diese Weise erfasste Datensatz enthält die in der Tabelle geänderten Daten.



Eine EventBridge Pipe verarbeitet Ereignisse aus DynamoDB Streams und leitet sie an ein Ziel wie einen Bus weiter (ein EventBridge Eventbus ist ein Router, der Ereignisse empfängt und sie an Ziele, auch Ziele genannt, weiterleitet). Die Übermittlung ist abhängig von den Regeln, die dem Inhalt des Ereignisses entsprechen. Die Pipe bietet ebenfalls die Möglichkeit, nach bestimmten Ereignissen zu filtern und die Ereignisdaten anzureichern, bevor sie an das Ziel gesendet werden.

EventBridge Unterstützt zwar mehrere Zieltypen, aber bei der Implementierung eines Fan-Out-Designs wird häufig eine Lambda-Funktion als Ziel verwendet. Im folgenden Beispiel wird eine Integration erstellt, die eine Lambda-Funktion zum Ziel hat.

Erstellen einer Integration über die Konsole

Gehen Sie wie folgt vor, um eine Integration über die AWS-Managementkonsole zu erstellen.

1. Aktivieren Sie DynamoDB Streams in der Quelltabelle, indem Sie die Schritte im Abschnitt Aktivieren eines Streams des DynamoDB-Entwicklerhandbuchs befolgen. Wenn DynamoDB Streams in der Quelltabelle bereits aktiviert ist, stellen Sie sicher, dass derzeit weniger als zwei Verbraucher vorhanden sind. Verbraucher können Lambda-Funktionen, DynamoDB Global Tables, Amazon DynamoDB Zero-ETL-Integrationen mit Amazon OpenSearch Service oder Anwendungen sein, die direkt aus Streams lesen, z. B. über den DynamoDB Streams Kinesis-Adapter.
2. Erstellen Sie einen EventBridge Event-Bus, indem Sie die Schritte im Abschnitt EventBridge Amazon-Event-Bus erstellen des EventBridge Benutzerhandbuchs befolgen.
   a. Aktivieren Sie bei der Erstellung des Event Bus die Schemaerkennung.
3. Erstellen Sie eine EventBridge Pipe, indem Sie den Schritten im Abschnitt Erstellen einer EventBridge Amazon-Leitung des EventBridge Benutzerhandbuchs folgen.

a. Wählen Sie bei der Konfiguration der Quelle im Feld Quelle die Option DynamoDB und im Feld DynamoDB Streams den Namen des Streams der Quelltabelle aus.

b. Wählen Sie bei der Konfiguration des Ziels im Feld Zieldienst die Option EventBridge Event-Bus und im Feld Event-Bus als Ziel den in Schritt 2 erstellten Event-Bus aus.

4. Schreiben Sie ein Beispielelement in die DynamoDB-Quelltabelle, um ein Ereignis auszulösen. Dies ermöglicht es EventBridge , das Schema aus dem Beispielelement abzuleiten. Anhand dieses Schemas können Regeln für das Routing von Ereignissen erstellt werden. Wenn Sie beispielsweise ein Entwurfsmuster implementieren, das das Überladen von Attributen beinhaltet, sollen je nach Wert des Sortierschlüssels möglicherweise unterschiedliche Regeln ausgelöst werden. Einzelheiten zum Schreiben eines Elements in DynamoDB finden Sie im Abschnitt Arbeiten mit Elementen und Attributen im DynamoDB-Entwicklerhandbuch.

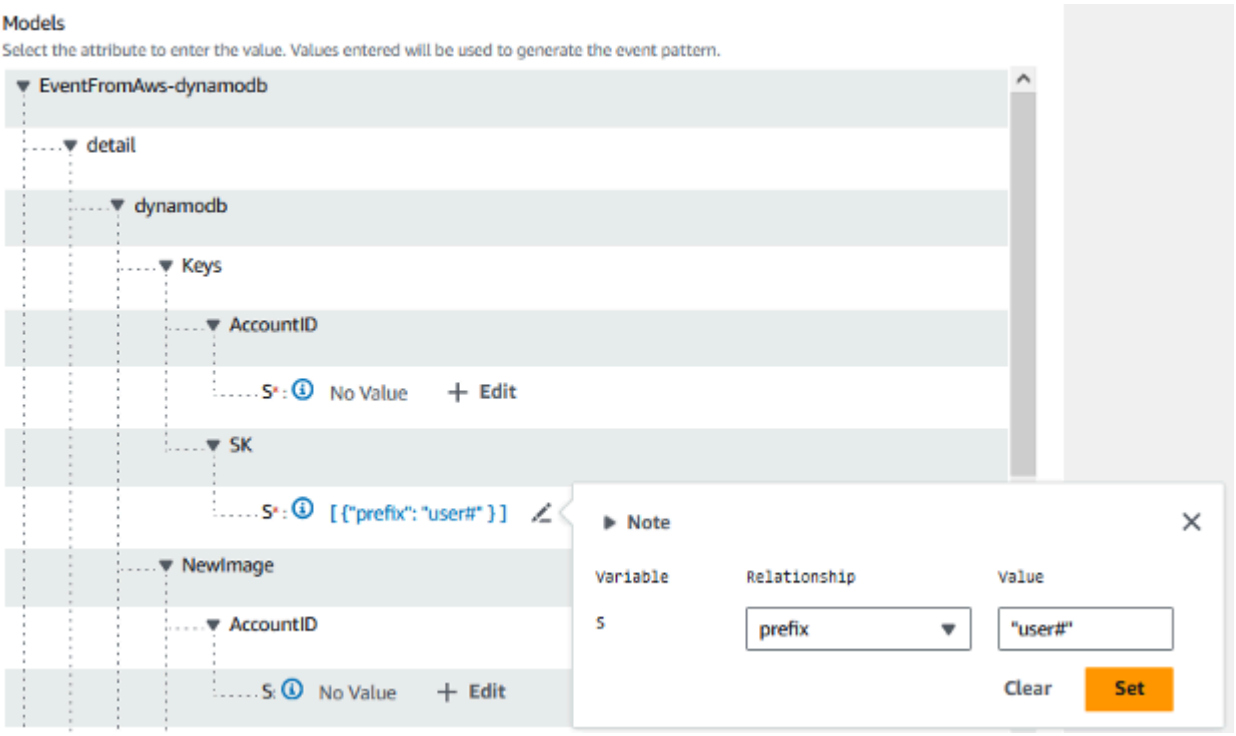
5. Erstellen Sie eine Python-Lambda-Beispielfunktion, die als Ziel verwendet werden soll, und befolgen Sie dabei die Schritte im Abschnitt Erstellen von Lambda-Funktionen mit Python im Lambda-Entwicklerhandbuch. Bei der Erstellung Ihrer Funktion können Sie den folgenden Beispielcode zur Demonstration der Integration verwenden. Wenn es aufgerufen wird, druckt es das `NewImage` und das `OldImage` empfangene Ereignis aus, das in CloudWatch Logs eingesehen werden kann.

```
import json

def lambda_handler(event, context):
    dynamodb = event.get('detail', {}).get('dynamodb', {})
    new_image = dynamodb.get('NewImage')
    old_image = dynamodb.get('OldImage')

    if new_image:
        print("NewImage:", json.dumps(new_image, indent=2))
    if old_image:
        print("OldImage:", json.dumps(old_image, indent=2))

    return {'statusCode': 200, 'body': json.dumps(event)}
```

6. Erstellen Sie eine EventBridge Regel, die Ereignisse an Ihre neue Lambda-Funktion weiterleitet, indem Sie die Schritte im Abschnitt „Regel erstellen, die auf Ereignisse reagiert“ EventBridge befolgen.

a. Wählen Sie bei der Definition der Regeldetails unter Event Bus den Namen des in Schritt 2 erstellten Event Bus aus.

b. Folgen Sie beim Erstellen des Ereignismusters der Anleitung unter Vorhandenes Schema. Hier können Sie die Registrierung erkannter Schemas und das erkannte Schema für Ihr Ereignis auswählen. Auf diese Weise können Sie ein für Ihren Anwendungsfall spezifisches Ereignismuster konfigurieren, nach dem nur Nachrichten weitergeleitet werden, die bestimmten Attributen entsprechen. Eine solche Konfiguration empfiehlt sich beispielsweise, wenn Sie nur Übereinstimmungen in DynamoDB-Elementen finden möchten, deren Sortierschlüssel mit `“user#”` beginnt.



c. Klicken Sie auf Ereignismuster in JSON generieren, nachdem Sie mit dem Entwurf eines Musters anhand des Schemas fertig sind. Wenn Sie stattdessen alle Ereignisse abgleichen möchten, die in DynamoDB-Streams erscheinen, verwenden Sie für das Ereignismuster den folgenden JSON-Code.

```
{
  "source": ["aws.dynamodb"]
}
```

d. Folgen Sie bei der Auswahl von Zielen der Serviceanleitung. AWS Wählen Sie unter „Ein Ziel auswählen“ Option „Lambda-Funktion“ aus. Wählen im Feld „Funktion“ die Lambda-Funktion aus, die Sie in Schritt 5 erstellt haben.

7. Sie können jetzt die Schemaerkennung auf Ihrem Event-Bus beenden, indem Sie die Schritte im Abschnitt Schemaerkennung in Event-Bussen starten oder beenden des EventBridge Benutzerhandbuchs befolgen.
8. Schreiben Sie ein zweites Beispielelement in die DynamoDB-Quelltabelle, um ein Ereignis auszulösen. Überprüfen Sie bei jedem Schritt, ob das Ereignis erfolgreich verarbeitet wurde.
   a. Sehen Sie sich die CloudWatch Metrik PutEventsApproximateSuccessCountfür Ihren Event-Bus an, indem Sie dem EventBridge Abschnitt Monitoring Amazon des EventBridge Benutzerhandbuchs folgen.
   b. Sehen Sie sich Funktionsprotokolle für die Lambda-Funktion gemäß dem Abschnitt Überwachung und Fehlerbehebung bei Lambda-Funktionen im Lambda-Entwicklerhandbuch an. Wenn Ihre Lambda-Funktion den bereitgestellten Beispielcode verwendet, sollten Sie die Daten `NewImage` und `OldImage` aus DynamoDB Streams in der CloudWatch Protokollgruppe Logs sehen.
   c. Sehen Sie sich die Metrik Fehlerzahl und Erfolgsquote (%) für die Lambda-Funktion gemäß dem Abschnitt Überwachung und Fehlerbehebung bei Lambda-Funktionen im Lambda-Entwicklerhandbuch an.

Nächste Schritte

Dieses Beispiel bietet eine grundlegende Integration mit einer einzelnen Lambda-Funktion als Ziel. Ein besseres Verständnis komplexerer Konfigurationen, wie z. B. das Erstellen mehrerer Regeln, das Erstellen mehrerer Ziele, die Integration mit anderen Diensten und das Anreichern von Ereignissen, finden Sie im vollständigen EventBridge Benutzerhandbuch: Erste Schritte mit. EventBridge

> **Note**
>
> Achten Sie auf alle EventBridge Kontingente, die für Ihre Anwendung relevant sein könnten. Die Kapazität von DynamoDB Streams skaliert zwar mit Ihrer Tabelle, die EventBridge Kontingente sind jedoch separat. Zu den üblichen Quoten, die bei einer großen Anwendung beachtet werden müssen, gehören die Begrenzung der Anzahl der Transaktionen pro Sekunde für Aufrufe und die PutEventsDrosselung der Transaktionen pro Sekunde. Diese Kontingente geben die Anzahl der Aufrufe an, die an Ziele gesendet werden können, und die Anzahl der Ereignisse, die pro Sekunde an den Bus übertragen werden können.

Integrieren von DynamoDB in Amazon Managed Streaming für Apache Kafka

Amazon Managed Streaming für Apache Kafka (Amazon MSK) erleichtert das Erfassen und Verarbeiten von Streaming-Daten in Echtzeit mithilfe eines vollständig verwalteten, hochverfügbaren Apache Kafka-Service.

Apache Kafka ist ein verteilter Datenspeicher, der für die Aufnahme und Verarbeitung von Streaming-Daten in Echtzeit optimiert ist. Kafka kann Streams von Datensätzen verarbeiten, sie effektiv in der Reihenfolge speichern, in sie generiert wurden, und sie veröffentlichen und abonnieren.

Aufgrund dieser Funktionen wird Apache Kafka häufig zur Erstellung von Echtzeit-Streaming-Datenpipelines verwendet. Eine Datenpipeline verarbeitet und verschiebt Daten zuverlässig von einem System in ein anderes. Sie kann ein wichtiger Bestandteil einer speziell entwickelten Datenbankstrategie sein, da sie die Verwendung mehrerer Datenbanken erleichtert, die jeweils unterschiedliche Anwendungsfälle unterstützen.

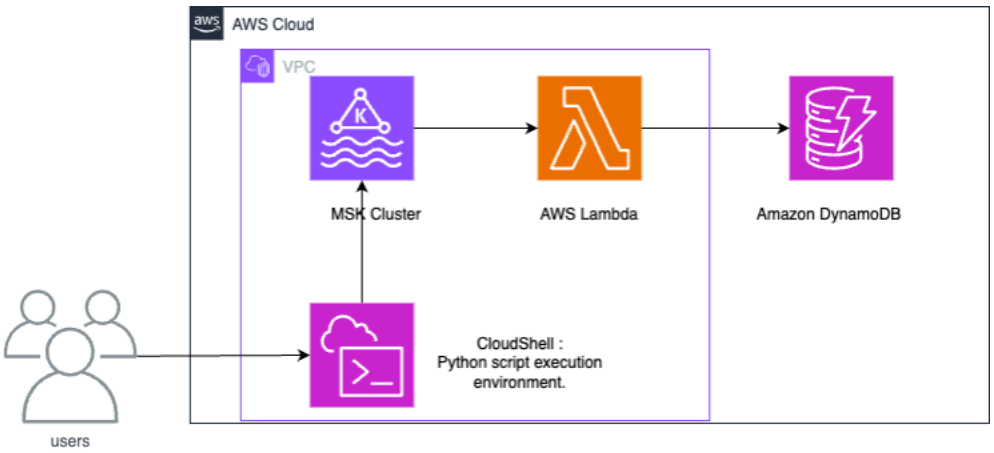
Amazon DynamoDB wird in diesen Datenpipelines häufig zur Unterstützung von Anwendungen eingesetzt, die Schlüssel-Wert- oder Dokumentdatenmodelle verwenden und für die eine grenzenlose Skalierbarkeit mit konsistenter Leistung im einstelligen Millisekundenbereich gewünscht ist.

Themen

- Funktionsweise
- Einrichten einer Integration von Amazon MSK in DynamoDB
- Nächste Schritte

Funktionsweise

Bei der Integration von Amazon MSK in DynamoDB wird eine Lambda-Funktion verwendet, die Datensätze von Amazon MSK verarbeitet und in DynamoDB schreibt.



Lambda fragt intern neue Nachrichten von Amazon MSK ab und ruft dann synchron die Ziel-Lambda-Funktion auf. Die Ereignisnutzlast der Lambda-Funktion enthält Stapel von Nachrichten von Amazon MSK. Für die Integration von Amazon MSK in DynamoDB schreibt die Lambda-Funktion diese Nachrichten in DynamoDB.

Einrichten einer Integration von Amazon MSK in DynamoDB

> **Note**
>
> Sie können die in diesem Beispiel verwendeten Ressourcen im folgenden GitHub Repository herunterladen.

Die folgenden Schritte zeigen, wie Sie eine Beispielintegration von Amazon MSK in Amazon DynamoDB einrichten. Das Beispiel enthält Daten, die von IoT-Geräten (Internet der Dinge) generiert und in Amazon MSK erfasst wurden. In Amazon MSK erfasste Daten können in Analysedienste oder Tools von Drittanbietern integriert werden, die mit Apache Kafka kompatibel sind, was verschiedene Analyseanwendungsfälle ermöglicht. DynamoDB-Integration ermöglicht zudem auch die Suche nach Schlüssel-Wert-Paaren einzelner Gerätedatensätze.

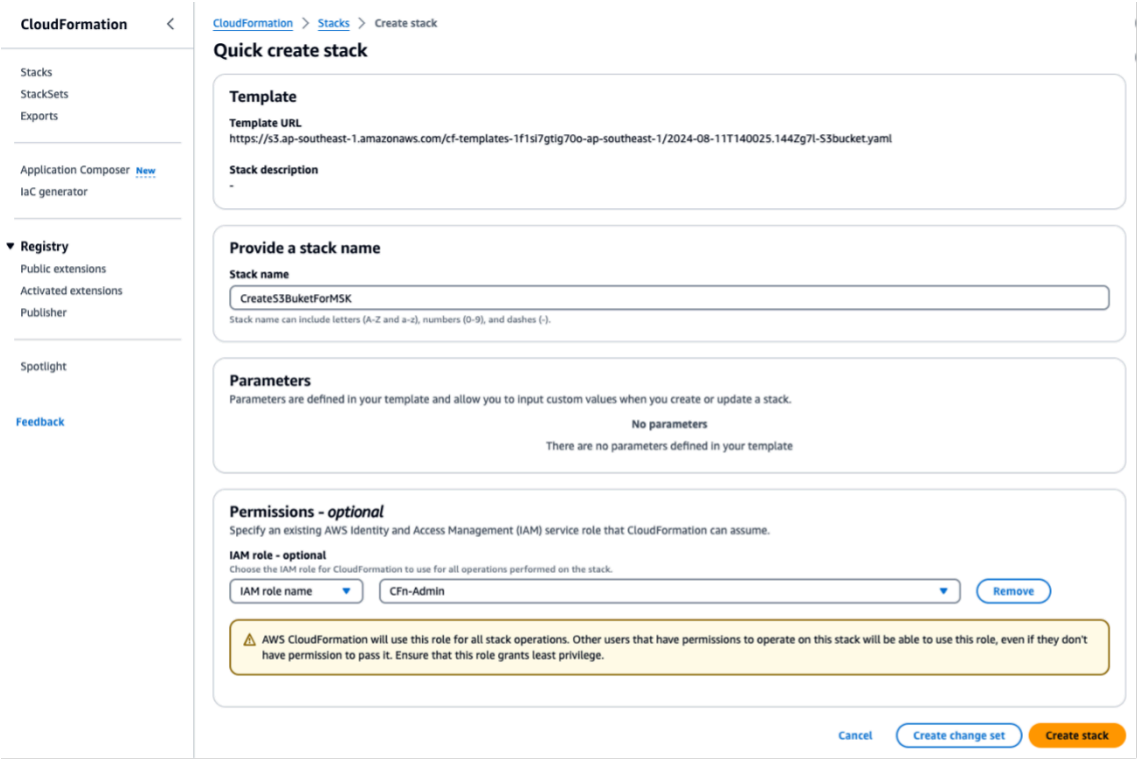
Dieses Beispiel zeigt, wie ein Python-Skript Daten eines IoT-Sensors in Amazon MSK schreibt. Anschließend werden durch eine Lambda-Funktion Elemente mit dem Partitionsschlüssel "`deviceid`" in DynamoDB geschrieben.

Die bereitgestellte CloudFormation Vorlage erstellt die folgenden Ressourcen: einen Amazon S3 S3-Bucket, eine Amazon VPC, einen Amazon MSK-Cluster und einen AWS CloudShell zum Testen von Datenoperationen.

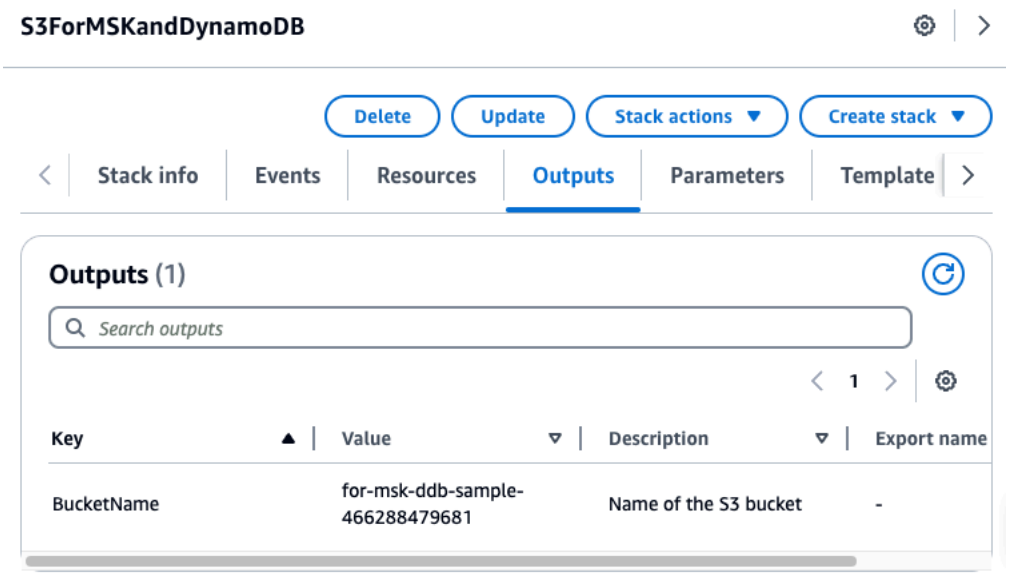
Um Testdaten zu generieren, erstellen Sie ein Amazon-MSK-Thema und anschließend eine DynamoDB-Tabelle. Sie können den Sitzungsmanager von der Managementkonsole aus verwenden, um sich beim Betriebssystem CloudShell des anzumelden und Python-Skripts auszuführen.

Nachdem Sie die CloudFormation Vorlage ausgeführt haben, können Sie die Erstellung dieser Architektur abschließen, indem Sie die folgenden Operationen ausführen.

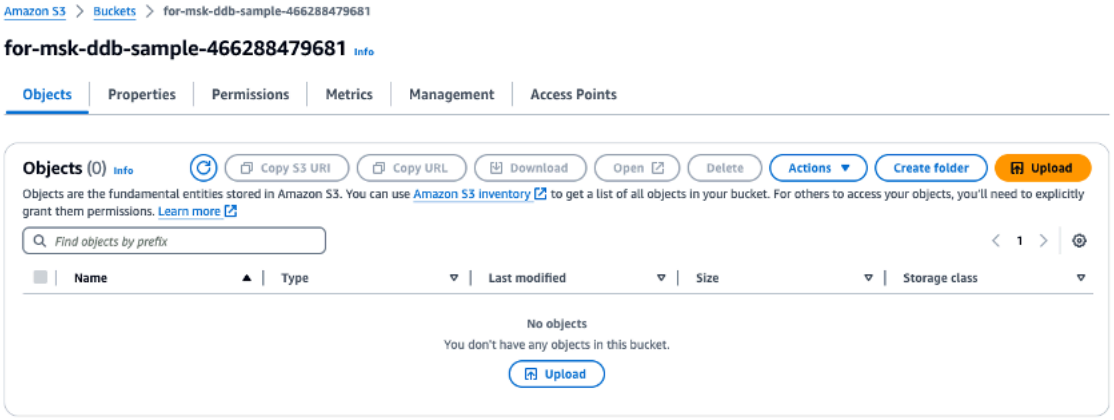
1. Führen Sie die CloudFormation Vorlage aus`S3bucket.yaml`, um einen S3-Bucket zu erstellen. Führen Sie sie für alle nachfolgenden Skripts oder Operationen in derselben Region aus. Geben Sie `ForMSKTestS3` den Namen des CloudFormation Stacks ein.



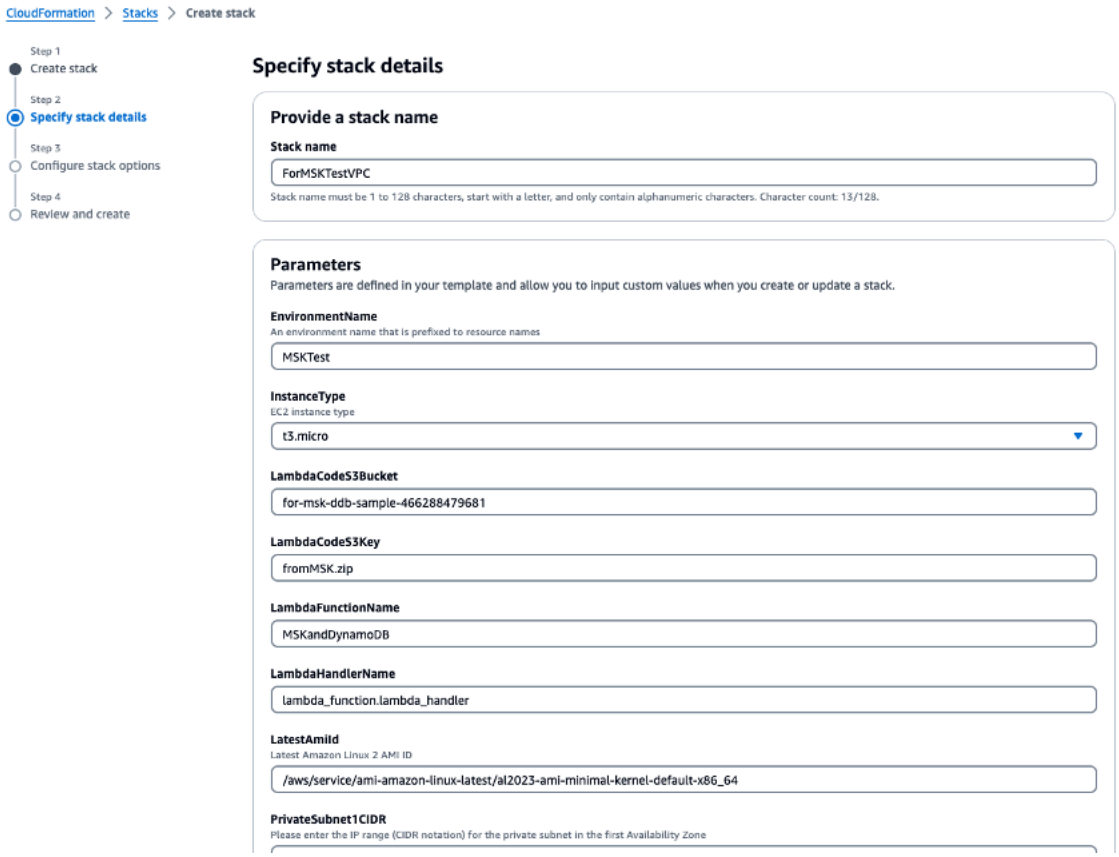
   Wenn der Vorgang abgeschlossen ist, notieren Sie sich den unter Ausgaben angezeigten Namen des S3-Buckets. Sie benötigen den Namen in Schritt 3.



2. Laden Sie die heruntergeladene ZIP-Datei `fromMSK.zip` in den soeben erstellten S3-Bucket hoch.

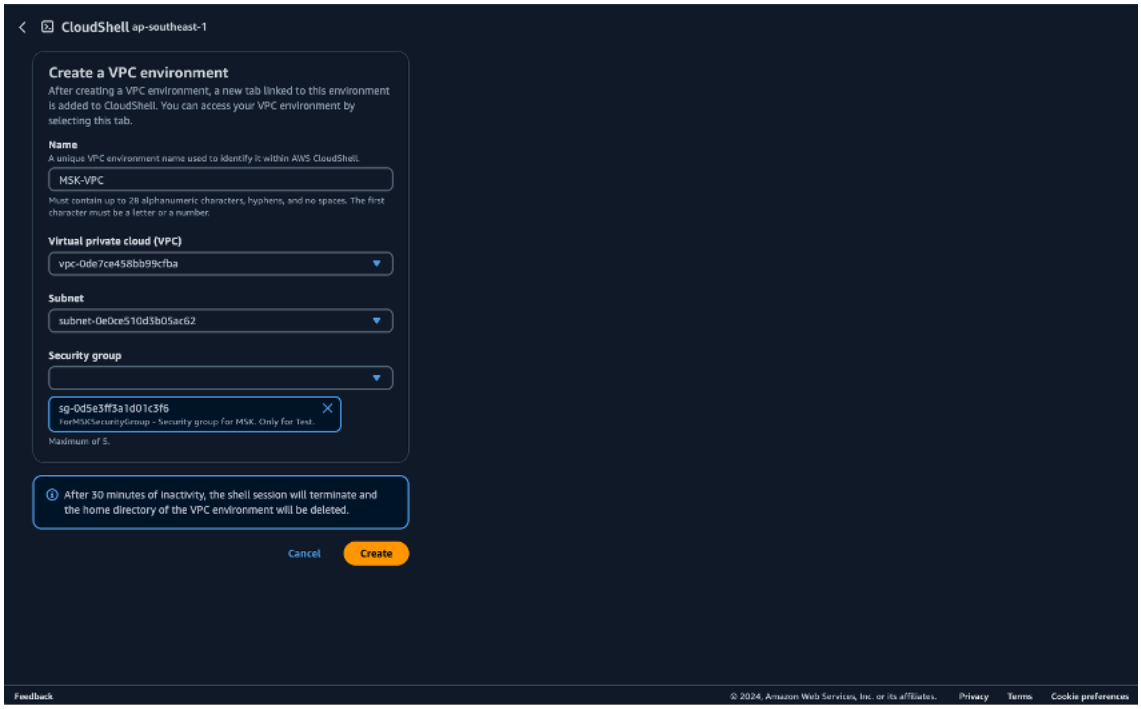


3. Führen Sie die CloudFormation Vorlage aus`VPC.yaml`, um eine VPC, einen Amazon MSK-Cluster und eine Lambda-Funktion zu erstellen. Geben Sie auf dem Bildschirm für die Parametereingabe auf Anforderung den Namen des S3-Buckets ein, den Sie in Schritt 1 erstellt haben. Setzen Sie den CloudFormation Stack-Namen auf. `ForMSKTestVPC`



4. Bereiten Sie die Umgebung für die Ausführung von Python-Skripten vor CloudShell. Sie können CloudShell auf dem verwenden AWS-Managementkonsole. Weitere Informationen zur Verwendung CloudShell finden Sie unter Erste Schritte mit AWS CloudShell. Erstellen Sie nach dem Start eine CloudShell, CloudShell die zu der VPC gehört, die Sie gerade erstellt haben, um eine Verbindung zum Amazon MSK-Cluster herzustellen. Erstellen Sie das CloudShell in einem privaten Subnetz. Füllen Sie die folgenden Felder aus:

    1. Name: Ein beliebiger Name. Zum Beispiel: MSK-VPC.
    2. VPC — auswählen MSKTest
    3. Subnetz — wählen Sie MSKTest Privates Subnetz () AZ1
    4. SecurityGroup- Wählen Sie „Für Gruppe“ MSKSecurity



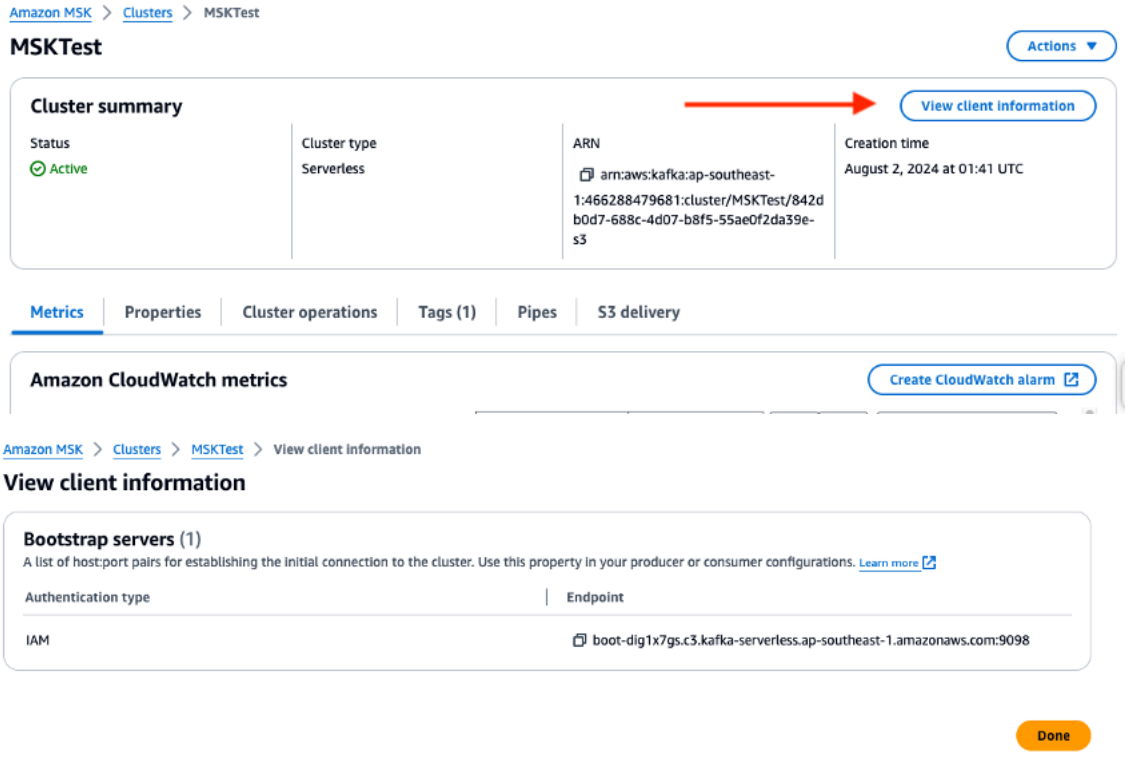
Sobald die CloudShell Zugehörigkeit zum privaten Subnetz gestartet wurde, führen Sie den folgenden Befehl aus:

```
pip install boto3 kafka-python aws-msk-iam-sasl-signer-python
```

5. Laden Sie Python-Skripte aus dem S3-Bucket herunter.

```
aws s3 cp s3://[YOUR-BUCKET-NAME]/pythonScripts.zip ./
unzip pythonScripts.zip
```

6. Überprüfen Sie die Management-Konsole und legen Sie die Umgebungsvariablen für die Broker-URL und den Regionswert in den Python-Skripten fest. Überprüfen Sie den Endpunkt des Amazon-MSK-Clusters in der Management-Konsole.



7. Stellen Sie die Umgebungsvariablen auf dem CloudShell ein. Bei Verwendung der Region „USA West (Oregon)“:

```
export AWS_REGION="us-west-2"
export MSK_BROKER="boot-YOURMSKCLUSTER.c3.kafka-serverless.ap-
southeast-1.amazonaws.com:9098"
```

8. Führen Sie die folgenden Python-Skripte aus.

   Erstellen Sie ein Amazon-MSK-Thema.

```
python ./createTopic.py
```

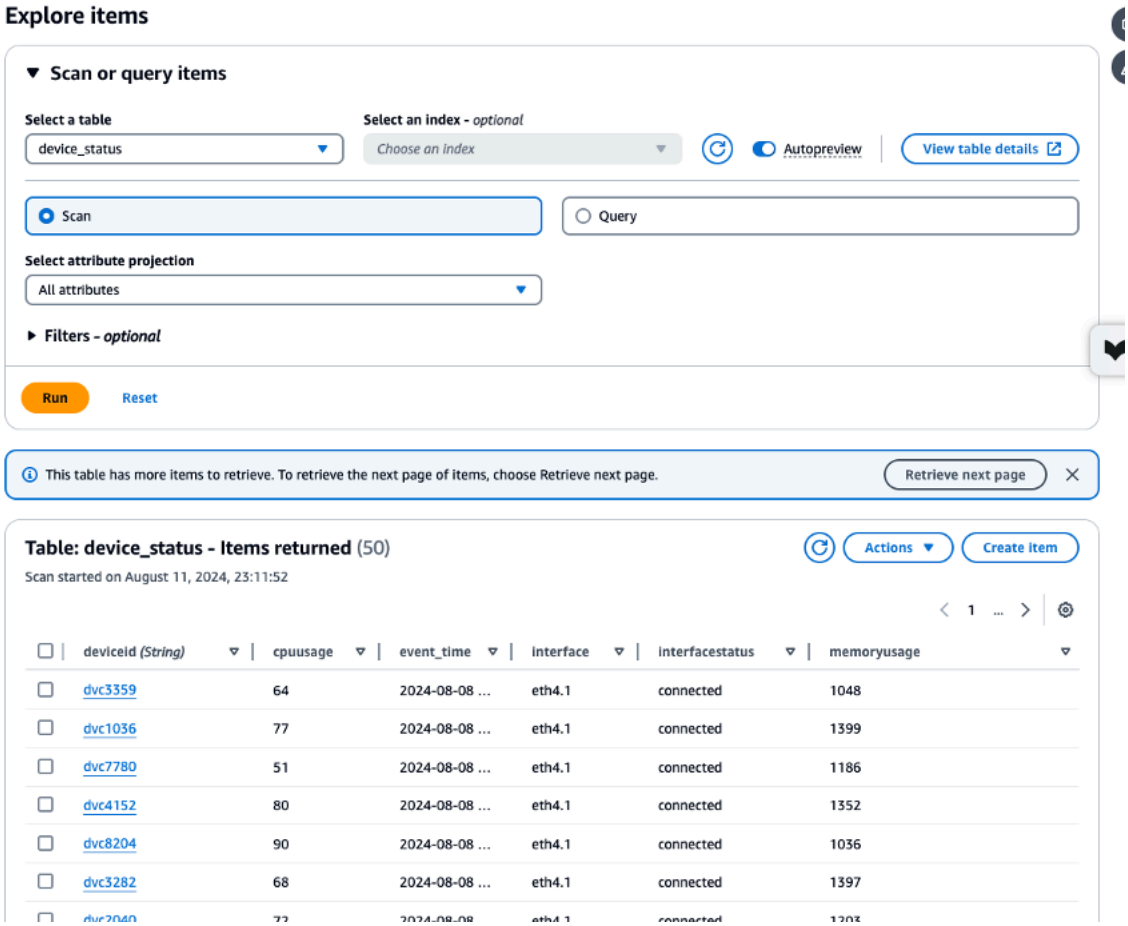
   Erstellen Sie eine DynamoDB-Tabelle.

```
python ./createTable.py
```

Schreiben Sie Testdaten in das Amazon MSK-Thema:

```
python ./kafkaDataGen.py
```

9. Überprüfen Sie die CloudWatch Metriken für die erstellten Amazon MSK-, Lambda- und DynamoDB-Ressourcen und überprüfen Sie die in der `device_status` Tabelle gespeicherten Daten mithilfe des DynamoDB-Daten-Explorers, um sicherzustellen, dass alle Prozesse korrekt ausgeführt wurden. Wenn jeder Prozess ohne Fehler ausgeführt wird, können Sie überprüfen, ob die Testdaten, die von Amazon MSK geschrieben wurden CloudShell , auch in DynamoDB geschrieben werden.



10. Wenn Sie mit diesem Beispiel fertig sind, löschen Sie die in diesem Tutorial erstellten Ressourcen. Löschen Sie die beiden CloudFormation Stapel: und. `ForMSKTestS3` `ForMSKTestVPC` Wenn das Löschen des Stacks erfolgreich war, werden alle Ressourcen gelöscht.

Nächste Schritte

> **Note**
>
> Wenn Sie im Rahmen dieses Beispiels Ressourcen erstellt haben, denken Sie daran, diese wieder zu löschen, damit keine unerwarteten Gebühren anfallen.

Die Integration hat eine Architektur identifiziert, die Amazon MSK und DynamoDB miteinander verbindet, sodass Stream-Daten OLTP-Workloads unterstützen können. Von hier aus können komplexere Suchen realisiert werden, indem DynamoDB mit OpenSearch Service verknüpft wird. Erwägen Sie die Integration mit EventBridge für komplexere ereignisgesteuerte Anforderungen und Erweiterungen wie Amazon Managed Service für Apache Flink für höheren Durchsatz und geringere Latenzanforderungen.

Bewährte Methoden für die Integration mit DynamoDB

Bei der Integration von DynamoDB mit anderen Diensten sollten Sie stets die bewährten Methoden für die Verwendung der einzelnen Services befolgen. Es gibt jedoch einige bewährte Methoden speziell für die Integration, die Sie berücksichtigen sollten.

Themen

- Erstellen eines Snapshots in DynamoDB
- Erfassung von Datenänderungen in DynamoDB

Erstellen eines Snapshots in DynamoDB

- Im Allgemeinen empfehlen wir, den Export zu Amazon S3 zu verwenden, um Snapshots für die erste Replikation zu erstellen. Dies ist kostengünstig und konkurriert außerdem nicht mit dem Datenverkehr Ihrer Anwendung um den Durchsatz. Sie können auch das Backup und die Wiederherstellung in einer neuen Tabelle in Betracht ziehen, gefolgt von einem Scanvorgang. Dadurch wird vermieden, dass Ihre Anwendung um den Durchsatz konkurriert, dies ist aber in der Regel deutlich weniger kostengünstig als ein Export.
- Legen Sie bei einem Export immer eine `StartTime` fest. Auf diese Weise können Sie leicht bestimmen, ab wann Sie mit der Erfassung von Änderungsdaten (CDC) beginnen werden.

- Legen Sie beim Exportieren zu S3 eine Lebenszyklusaktion für den S3-Bucket fest. In der Regel ist eine auf 7 Tage festgelegte Ablaufaktion sicher, Sie sollten jedoch alle Richtlinien Ihres Unternehmens befolgen. Selbst wenn Sie Ihre Elemente nach der Aufnahme explizit löschen, kann diese Aktion dazu beitragen, Probleme zu erkennen, wodurch unnötige Kosten reduziert und Richtlinienverstöße verhindert werden.

Erfassung von Datenänderungen in DynamoDB

- Wenn Sie CDC nahezu in Echtzeit benötigen, verwenden Sie DynamoDB Streams oder Amazon Kinesis Data Streams (KDS). Überlegen Sie bei der Entscheidung allgemein, welche Option am einfachsten mit dem Downstream-Service zu verwenden ist. Wenn Sie die Ereignisverarbeitung in der richtigen Reihenfolge auf Partitionsschlüsselebene bereitstellen müssen oder wenn Sie über außergewöhnlich große Elemente verfügen, verwenden Sie DynamoDB Streams.
- Wenn Sie CDC nicht nahezu in Echtzeit benötigen, können Sie den Export nach Amazon S3 mit inkrementellen Exporten verwenden, um nur die Änderungen zu exportieren, die zwischen zwei Zeitpunkten vorgenommen wurden.

  Wenn Sie den Export zu S3 zum Generieren eines Snapshots verwendet haben, kann dies besonders hilfreich sein, da Sie ähnlichen Code verwenden können, um inkrementelle Exporte zu verarbeiten. In der Regel ist der Export zu S3 etwas günstiger als die vorherigen Streaming-Optionen, aber die Kosten sind normalerweise nicht der Hauptfaktor für die Auswahl der zu verwendenden Option.
- Im Allgemeinen können Sie nur zwei gleichzeitige Verbraucher eines DynamoDB-Streams haben. Berücksichtigen Sie dies bei der Planung Ihrer Integrationsstrategie.
- Verwenden Sie keine Scans, um Änderungen zu erkennen. Das mag in kleinem Maßstab funktionieren, wird aber relativ schnell unpraktisch.

Verwenden von generative KI mit DynamoDB

Amazon DynamoDB ist eine serverlose, vollständig verwaltete, verteilte NoSQL-Datenbank mit einer Leistung im einstelligen Millisekundenbereich in jeder Größenordnung. DynamoDB ist für Workloads mit hohem Durchsatz optimiert, wobei die Funktionen durch die Integration generativer KI-Modelle erweitert werden können. Mithilfe generativer KI-Modelle können Sie in Echtzeit mit Daten arbeiten, die in DynamoDB-Tabellen gespeichert sind, und Anwendungen erstellen, die kontextsensitiv und hochgradig personalisiert sind. Sie können auch die Erfahrung der Endbenutzer verbessern, indem Sie Ihre Geschäfts-, Benutzer- und Anwendungsdaten optimal nutzen, um Ihre generativen KI-Lösungen individuell anzupassen.

Weitere Informationen über Gen AI und die Lösungen, die für die Entwicklung von Gen-KI-Anwendungen AWS bereitgestellt werden, finden Sie unter Transformieren Sie Ihr Unternehmen mit generativer KI.

Themen

- Anwendungsfälle für die Nutzung generativer KI in DynamoDB
- Generative KI-Blogs für DynamoDB
- Nutzung der DynamoDB-Zero-ETL-Integration mit Service OpenSearch

Anwendungsfälle für die Nutzung generativer KI in DynamoDB

DynamoDB kommt häufig in KI-gestützten Konversationsanwendungen wie Chatbots und Call-Centern zum Einsatz, die auf einem Basismodell (FM) basieren. Sie können FMs über Amazon Bedrock, Amazon SageMaker AI oder andere Modellanbieter darauf zugreifen. In solchen wird DynamoDB häufig eingesetzt, um die Personalisierung zu erhöhen und die Benutzererfahrung anhand von drei Datenmustern zu verbessern: Anwendungsdaten, Geschäftsdaten und Benutzerdaten. Im Folgenden einige Beispiele für diese Datenmuster:

- Speicherung von Anwendungsdaten, wie z. B. dem Verlauf von Chat-Nachrichten, durch Integrationen mit LangChainLlamaIndex, oder durch einen benutzerdefinierten Code. Dieser Kontext verbessert die Benutzererfahrung, da das Modell mit dem Benutzer in beide Richtungen kommunizieren kann.
- Schaffung eines maßgeschneiderten Benutzererlebnisses durch Nutzung von Geschäftsdaten wie Inventar, Preisgestaltung und Dokumentation.

- Verwendung von Benutzerdaten wie Webverlauf, vergangenen Bestellungen und Benutzereinstellungen, um personalisierte Antworten zu geben.

Beispielsweise kann ein Versicherungsunternehmen mithilfe von DynamoDB einen Chatbot erstellen, um seinem auf Retrieval-Augmented Generation (RAG) basierenden generativen KI-Modell Zugriff auf Daten nahezu in Echtzeit zu gewähren. Beispiele für solche Daten sind Hypothekenzinsen in Echtzeit, Produktpreise, compliant/standard Vertragskopie, Benutzerverlauf und Benutzereinstellungen. Durch die Kombination von DynamoDB mit RAG werden detaillierte und aktuelle Informationen zu Versicherungsprodukten und Benutzerdaten hinzugefügt. Dadurch werden die Eingabeaufforderungen und Antworten erweitert, sodass Endbenutzer ein genaues, personalisiertes Erlebnis nahezu in Echtzeit erhalten.

In ähnlicher Weise verwenden Kunden aus der Finanzdienstleistungsbranche DynamoDB, Amazon Bedrock Knowledge Bases und Amazon Bedrock-Agents, um RAG-basierte generative KI-Anwendungen zu entwickeln. Solche Anwendungen können Open-Source-Gewinnberichte und -Anrufprotokolle verwenden. Sie können auch die benutzerspezifische Portfolio- und Transaktionshistorie verwenden, um auf Abruf eine Zusammenfassung des Portfolios einschließlich eines Ausblicks für die Zukunft zu erstellen.

Generative KI-Blogs für DynamoDB

Die folgenden Artikel bieten detaillierte Anwendungsfälle, bewährte Methoden und step-by-step Anleitungen, die Ihnen helfen, die Funktionen von DynamoDB bei der Entwicklung fortschrittlicher KI-gestützter Anwendungen zu nutzen.

- Amazon DynamoDB-Datenmodelle für generative KI-Chatbots
- Erstellen Sie einen skalierbaren, kontextsensitiven Chatbot mit Amazon DynamoDB, Amazon Bedrock und LangChain

Nutzung der DynamoDB-Zero-ETL-Integration mit Service OpenSearch

Sie können Amazon Bedrock mit DynamoDB verwenden, um serverlosen Zugriff auf grundlegende Modelle (FMs) wie Amazon Titan und andere Modelle von Drittanbietern bereitzustellen. Sie können die Zero-ETL-Integration mit Amazon OpenSearch Service nutzen, um Vektorsuchfunktionen bei der Entwicklung generativer KI-Anwendungen zu aktivieren. Der Workshop Generative KI mit DynamoDB

Zero-ETL to OpenSearch Integration und Amazon Bedrock bietet Ihnen praktische Erfahrung bei der Einrichtung der DynamoDB Zero-ETL-Integration mit. OpenSearch Dieser Workshop umfasst die folgenden Aufgaben:

- Erstellt eine Pipeline von Ihrer DynamoDB-Tabelle zu. OpenSearch
- Erstellt einen Amazon Bedrock Connector in OpenSearch.
- Fragt Amazon Bedrock ab und nutzt OpenSearch es als Vektorspeicher.
- Verwendet Claude FM in Amazon Bedrock, um eine schriftliche Antwort in einfachem Englisch zu erstellen, in der die von OpenSearch zurückgegebenen Suchergebnisse erklärt werden.

In diesem Workshop können Sie DynamoDB integrieren, um generative KI-Anwendungen OpenSearch zu erstellen. Es demonstriert auch die flexiblen Abfragefunktionen für Datenbank-Engines, um Ihnen bei der Integration von DynamoDB und OpenSearch für herkömmliche Anwendungsfälle zu helfen. Dieser Workshop ist eines der sieben Module des Amazon DynamoDB Immersion Day. Sie können diesen Workshop in jedem beliebigen Format durchführen. AWS-Konto

Im folgenden Blogbeitrag erfahren Sie auch, wie Sie eine Zero-ETL-Integration zwischen DynamoDB und Service einrichten. OpenSearch In diesem Blogbeitrag wird auch beschrieben, wie Sie Modell-Konnektoren in OpenSearch Service einrichten, um mithilfe von Amazon Bedrock automatisch Einbettungen für eingehende Daten zu generieren. Vektorsuche für Amazon DynamoDB ohne ETL für Amazon OpenSearch Service.

Kontingente und Beschränkungen für Amazon DynamoDB

In diesem Abschnitt werden die aktuellen Kontingente (früher Limits genannt) in Amazon DynamoDB beschrieben. In diesem Thema wird ebenfalls beschrieben, wie Sie Aufgaben zur Kontingentverwaltung durchführen können (zum Beispiel Ihre aktuellen Kontingente anzeigen und eine Erhöhung beantragen).

Themen

- Aufgaben zur Kontingentverwaltung in DynamoDB ausführen
- Beantragen einer Kontingentserhöhung in DynamoDB
- Kontingente in Amazon DynamoDB
- Einschränkungen in Amazon DynamoDB

Aufgaben zur Kontingentverwaltung in DynamoDB ausführen

Amazon DynamoDB verfügt über mehrere Servicekomponenten wie Tabellen, Streams, Indizes und mehr. Wenn Sie Ihre AWS-Konto erstellen, werden Standardkontingente (früher Limits genannt) für diese Komponenten festgelegt. Wenn nicht anders angegeben, gilt jedes Kontingent spezifisch für eine Region. Für einige Kontingente können Sie Erhöhungen beantragen. Nachdem das Kontingent für eine Ressource erreicht wurde, schlagen zusätzliche Anfragen zum Erstellen dieser Ressource mit einer Ausnahme fehl.

Zugreifen auf DynamoDB-Kontingente

Sie können folgendermaßen mit DynamoDB-Service Quotas arbeiten:

- AWS-Managementkonsole

  Die Service Quotas-Konsole ist eine browserbasierte Oberfläche, mit der Sie Ihre Service Quotas verwalten können. Sie können von jeder AWS-Managementkonsole-Seite aus auf Service Quotas zugreifen, indem Sie sie in der oberen Navigationsleiste auswählen oder indem Sie unter AWS-Managementkonsole nach Service Quotas suchen.
- AWS Command Line Interface-Tools

Wenn Sie AWS Command Line Interface-Tools verwenden, können Sie Befehle in der Befehlszeile Ihres Systems eingeben, um Service-Quotas-Aufgaben auszuführen. Die Befehlszeilen-Tools können auch beim Erstellen von Skripts für die Ausführung von AWS-Aufgaben hilfreich sein.

- AWS SDKs

  Sie können die AWS-SDKs für verschiedene Programmiersprachen und Plattformen (z. B. Java, Python, Ruby, .NET, iOS, Android und andere) verwenden, um Service Quotas-Aufgaben durchzuführen.

Wenn ein einstellbares Kontingent in der Service Quotas-Konsole nicht verfügbar ist, verwenden Sie AWS Support Center Console, um einen Fall zur Erhöhung der Service Quotas zu erstellen.

Anzeigen Ihrer aktuellen Kontingente in der Konsole

So zeigen Sie Ihre aktuellen Kontingente mit der Service-Quotas-Konsole an

1. Öffnen Sie die Service Quotas-Konsole unter https://console.aws.amazon.com/servicequotas/home/services/dynamodb/quotas/
2. Wählen Sie in der Navigationsleiste (oben auf dem Bildschirm) eine Region aus.
3. In der Konsole werden Details zum DynamoDB-Kontingentnamen, zum angewendeten Kontingentwert auf Kontoebene, zum AWS-Standardkontingentwert, zur Auslastung und zur Anpassbarkeit des Kontingents auf Konto- oder Ressourcenebene angezeigt.

   Wenn der angewendete Kontingentwert oder die Auslastung nicht verfügbar ist, wird in der Konsole Nicht verfügbar angezeigt. Sie können Ihr angewendetes Kontingent über die Support-Center-Konsole anfordern.
4. Wählen Sie einen bestimmten Kontingentnamen, um die Detailseite aufzurufen, auf der die Beschreibung, der Kontingentcode, der Kontingent-ARN, die Auslastung, der angewendete Kontingentwert auf Kontoebene, die Anpassbarkeit und der AWS-Standardkontingentwert angezeigt werden.

   Falls zutreffend, werden auf der Detailseite auch alle Überwachungsoptionen, Alarme, der Anforderungsverlauf und alle Tags des Kontingents angezeigt.

Anzeigen der aktuellen Kontingente mithilfe der AWS CLI

So zeigen Sie die Standardwerte für DynamoDB-Kontingente an:

- Ruf Sie die Operation `ListDefaultServiceQuotas` mit dem Servicecode (dynamodb) auf, um die Standardwerte für die Service Quotas von Amazon DynamoDB abzurufen.

```
$ aws service-quotas list-aws-default-service-quotas \
        --service-code dynamodb

 {
    "Quotas": [
        {
            "ServiceCode": "dynamodb",
            "ServiceName": "Amazon DynamoDB",
            "QuotaArn": "arn:aws:servicequotas:us-east-1::dynamodb/L-F7858A77",
            "QuotaCode": "L-F7858A77",
            "QuotaName": "Global Secondary Indexes per table",
            "Value": 20.0,
            "Unit": "None",
            "Adjustable": true,
            "GlobalQuota": false
        },
        {
            "ServiceCode": "dynamodb",
            "ServiceName": "Amazon DynamoDB",
            "QuotaArn": "arn:aws:servicequotas:us-east-1::dynamodb/L-AB614373",
            "QuotaCode": "L-AB614373",
            "QuotaName": "Table-level write throughput limit",
            "Value": 40000.0,
            "Unit": "None",
            "Adjustable": true,
            "GlobalQuota": false
        }......
    ]
}
```

So zeigen Sie die angewendeten Kontingentwerte an:

- Rufen Sie den Vorgang `ListServiceQuotas` mit dem DynamoDB-Servicecode (dynamodb) auf, um alle angewendeten Kontingentwerte entweder auf Kontoebene, Ressourcenebene oder

auf allen Ebenen abzurufen, indem Sie als Parameterwert jeweils `ACCOUNT`, `RESOURCE`, `ALL` oder `QuotaAppliedAtLevel` übergeben. Im folgenden CLI-Beispiel werden Kontingentwerte abgerufen, die auf Kontoebene angewendet wurden.

```
$ aws service-quotas list-service-quotas \
        --service-code dynamodb \
        --quota-applied-at-level ACCOUNT


{
    "Quotas": [
        {
            "ServiceCode": "dynamodb",
            "ServiceName": "Amazon DynamoDB",
            "QuotaArn": "arn:aws:servicequotas:us-east-1:303935678045:dynamodb/L-
F7858A77",
            "QuotaCode": "L-F7858A77",
            "QuotaName": "Global Secondary Indexes per table",
            "Value": 20.0,
{
    "Quotas": [
        {
            "ServiceCode": "dynamodb",
            "ServiceName": "Amazon DynamoDB",
            "QuotaArn": "arn:aws:servicequotas:us-east-1:303935678045:dynamodb/L
-F7858A77",
            "QuotaCode": "L-F7858A77",
            "QuotaName": "Global Secondary Indexes per table",
            "Value": 20.0,
            "Unit": "None",
            "Adjustable": true,
            "GlobalQuota": false,
            "QuotaAppliedAtLevel": "ACCOUNT"
        }.....

        }
    ]
}
```

Beantragen einer Kontingentserhöhung in DynamoDB

Sie können mithilfe der Service-Quotas-Konsole, AWS CLI oder einem Support-Fall für jede Region eine Kontingenterhöhung anfordern. Wenn ein anpassbares Kontingent in der Service Quotas-Konsole nicht verfügbar ist, verwenden Sie AWS Support Center Console, um einen Service Quota-Fall zur Erhöhung des Kontingents zu erstellen.

Support kann Ihre Kontingenterhöhungsanträge genehmigen, ablehnen oder teilweise genehmigen. Erhöhungen werden nicht sofort gewährt. Es kann ein paar Tage dauern, bis sie wirksam werden.

So fordern Sie eine Erhöhung über die Service-Quotas-Konsole an

1. Öffnen Sie die Service-Quotas-Konsole unter https://console.aws.amazon.com/servicequotas/home/services/dynamodb/quotas/
2. Wählen Sie in der Navigationsleiste (oben auf dem Bildschirm) eine Region aus.
3. Filtern Sie die Liste nach Ressourcennamen. Geben Sie beispielsweise On-Demand ein, um die Kontingente für On-Demand-Instances zu ermitteln.
4. Wenn das Kontingent anpassbar ist, wählen Sie das Kontingent aus und wählen Sie dann Kontingenterhöhung anfordern.
5. Geben Sie unter Kontingentwert ändern den neuen Kontingentwert ein.
6. Wählen Sie Request (Anfrage).
7. Um ausstehende oder kürzlich genehmigte Anfragen in der Konsole anzuzeigen, wählen Sie im Navigationsbereich die Option Dashboard . Wählen Sie für ausstehende Anfragen den Status der Anfrage, um die Anfrage zu öffnen. Der Anfangsstatus einer Anfrage ist Pending (Ausstehend). Nachdem sich der Status in Kontingent angefordert geändert hat, wird die Fallnummer bei Support angezeigt. Wählen Sie die Fallnummer, um das Ticket für Ihre Anfrage zu öffnen.

Weitere Informationen, einschließlich der Verwendung der AWS CLI oder SDKs zum Beantragen einer Kontingenterhöhung, finden Sie unter Beantragen einer Kontingenterhöhung im Benutzerhandbuch von Service Quotas.

Kontingente in Amazon DynamoDB

In diesem Abschnitt werden die aktuellen Kontingente – früher als Grenzwerte bezeichnet – in Amazon DynamoDB beschrieben. Jedes Kontingent gilt pro Region, sofern nicht anders angegeben.

> **Note**
>
> Alle Größenmessungen in DynamoDB verwenden binäre Einheiten. DynamoDB steht für 1 KB = 1024 Byte, 1 MB = 1024 KB, 1 GB = 1024 MB, 1 TB = 1024 GB.

Themen

- Lese- und Schreibdurchsatz
- Reservierte Kapazität
- Tabellen
- Globale Tabellen
- Sekundäre Indizes
- Projizierte sekundäre Indexattribute
- DynamoDB Streams
- Importieren aus Amazon S3
- Exportieren von Tabellen zu Amazon S3
- Backup und Wiederherstellung
- Contributor Insights

Lese- und Schreibdurchsatz

Standardkontingente für den Durchsatz

AWS legt einige Standardkontingente für den Durchsatz fest, den Ihr Konto innerhalb einer Region bereitstellen und nutzen kann.

Die Kontingente für den Lese- und den Schreibdurchsatz auf Kontoebene werden auf Kontoebene angewendet. Diese Kontingente auf Kontoebene gelten für die Summe der bereitgestellten Durchsatzkapazität für alle Tabellen und globalen sekundären Indizes Ihres Kontos in einer bestimmten Region. Der gesamte verfügbare Durchsatz des Kontos kann für eine einzige Tabelle oder für mehrere Tabellen bereitgestellt werden. Diese Kontingente gelten nur für Tabellen, die den Modus für bereitgestellte Kapazität verwenden.

Die Kontingente für den Lese- und den Schreibdurchsatz auf Tabellenebene gelten unterschiedlich für Tabellen, die den Modus mit bereitgestellter Kapazität verwenden, und Tabellen, die den Modus mit On-Demand-Kapazität verwenden.

Bei Tabellen mit bereitgestelltem Kapazitätsmodus und GSIs ist das Kontingent die maximale Anzahl an Lese- und Schreibkapazitätseinheiten, die für jede Tabelle oder jede Tabelle GSIs in der Region bereitgestellt werden können. Die Summe aller einzelnen Tabellen und ihrer Gesamtheit GSIs muss ebenfalls unter dem Kontingent für den Lese- und Schreibdurchsatz auf Kontoebene bleiben. Dies gilt zusätzlich zu der Anforderung, dass die Summe aller bereitgestellten Tabellen und ihrer Tabellen unter der Quote für den Lese- und Schreibdurchsatz auf Kontoebene bleiben GSIs muss.

Bei Tabellen und GSIs im On-Demand-Kapazitätsmodus entspricht das Kontingent auf Tabellenebene den maximalen Lese- und Schreibkapazitätseinheiten, die für jede Tabelle oder jede einzelne GSI innerhalb dieser Tabelle verfügbar sind. Im On-Demand-Modus werden keine Kontingente für den Lese- und den Schreibdurchsatz auf Kontoebene auf Tabellen angewendet.

Im Folgenden sehen Sie die Durchsatzkontingente, die standardmäßig für Ihr Konto gelten.

> **Note**
>
> Alle Kontingente für Kapazitätseinheiten und Anforderungseinheiten werden pro Sekunde gemessen. Ein Kontingent von 40.000 Lesekapazitätseinheiten bedeutet beispielsweise 40.000 Lesevorgänge pro Sekunde.

> **Note**
>
> Sie können im Rahmen einer Erhöhung der Servicekontingente eine beliebige Anzahl von Lesekapazitätseinheiten (RCU) oder Schreibkapazitätseinheiten (WCU) für Ihre DynamoDB-Tabellen anfordern. Die in der folgenden Tabelle aufgeführten Werte stellen die anfänglichen Standardkontingente dar. Dies sind keine Höchstgrenzen für Ihre Tabellen.

Name des Durchsatzkontingents	On-Demand	Bereitgestellt	Einstellbar
Pro Tabelle	40 000 Leseanforderungseinheiten und	40.000 Lesekapazitätseinheiten und	Ja

Name des Durchsatzkontingents	On-Demand	Bereitgestellt	Einstellbar
	40 000 Schreibanforderungseinheiten	40.000 Schreibkapazitätseinheiten	
Pro Konto	Nicht zutreffend	80.000 Lesekapazitätseinheiten und 80.000 Schreibkapazitätseinheiten	Ja
Minimaler Durchsatz für jede Tabelle oder globalen sekundären Index	Nicht zutreffend	1 Lesekapazitätseinheit und 1 Schreibkapazitätseinheit	Ja

Erhöhen oder Verringern des Durchsatzes (für Tabellen im Modus bereitgestellter Kapazität)

Erhöhen des bereitgestellten Durchsatzes

Sie können `ReadCapacityUnits` oder `WriteCapacityUnits` nach Bedarf mithilfe der AWS-Managementkonsole oder der Operation `UpdateTable` erhöhen. In einem einzigen Aufruf können Sie den bereitgestellten Durchsatz für eine Tabelle, für jeden globalen sekundären Index in dieser Tabelle oder für eine beliebige Kombination dieser Komponenten erhöhen. Die neuen Einstellungen werden erst wirksam, wenn die `UpdateTable`-Operation abgeschlossen ist.

Sie können die Kontingente pro Konto nicht überschreiten, wenn Sie bereitgestellte Kapazität hinzufügen. DynamoDB erlaubt nicht, die bereitgestellte Kapazität sehr schnell zu erhöhen. Abgesehen von diesen Einschränkungen können Sie die bereitgestellte Kapazität für Ihre Tabellen nach Bedarf erhöhen. Weitere Informationen zu diesen Kontingenten auf Kontoebene finden Sie im vorhergehenden Abschnitt, Standardkontingente für den Durchsatz.

Senken des bereitgestellten Durchsatzes

Für jede Tabelle und jeden globalen sekundären Index in einer `UpdateTable`-Operation können Sie `ReadCapacityUnits` oder `WriteCapacityUnits` (oder beide) reduzieren. Die neuen Einstellungen werden erst wirksam, wenn die `UpdateTable`-Operation abgeschlossen ist.

Es gibt ein Standardkontingent für die Anzahl der Abnahmen bereitgestellter Kapazität, die Sie pro Tag für Ihre DynamoDB-Tabelle ausführen können. Tage sind entsprechend der koordinierten Weltzeit (Universal Time Coordinated, UTC) definiert. Sie beginnen jeden Tag mit 4 verfügbaren Kürzungen. Jede Stunde wird 1 zusätzliche Ermäßigung verfügbar, bis zu einem Maximum von 4, die zu einem beliebigen Zeitpunkt verfügbar sind. Über einen vollen 24-Stunden-Tag können Sie auf diese Weise bis zu 27-mal reduzieren (4 in der ersten Stunde plus 1 für jede der verbleibenden 23 Stunden).

> **Important**
>
> Die Tabellen und die Limits für Verringerungen von globalen sekundären Indizes sind entkoppelt, sodass die globalen sekundären Indizes für eine bestimmte Tabelle ihre eigenen Limits für Verringerungen haben. Wenn jedoch eine einzelne Anforderung den Durchsatz für eine Tabelle und einen globalen sekundären Index verringert, wird dies abgelehnt, wenn das aktuelle Limit für einen davon überschritten wird. Anforderungen werden nicht teilweise verarbeitet.

Example

Eine Tabelle mit einem globalen sekundären Index kann in den ersten 4 Stunden des Tages wie folgt geändert werden:

- Die `WriteCapacityUnits` oder die `ReadCapacityUnits` (oder beide) der Tabelle können viermal verringert werden.
- Die `WriteCapacityUnits` oder die `ReadCapacityUnits` (oder beide) des globalen sekundären Index können viermal verringert werden.

Am Ende eines Tages kann der Durchsatz der Tabelle und des globalen sekundären Index potentiell insgesamt jeweils 27 Mal verringert werden.

Reservierte Kapazität

AWS legt ein Standardkontingent für die Menge an aktiver reservierter Kapazität fest, die Ihr Konto erwerben kann. Das Kontingentlimit ist eine Kombination aus reservierter Kapazität für Schreibkapazitätseinheiten (WCUs) und Lesekapazitätseinheiten (RCUs).

Kontingent für reservierte Kapazität	Aktive reservierte Kapazität	Einstellbar
Pro Konto	1.000.000 bereitgestellte Kapazitätseinheiten (_) WCUs RCUs	Ja

Wenn Sie versuchen, mehr als 1.000.000 bereitgestellte Kapazitätseinheiten in einem einzigen Kauf zu erwerben, erhalten Sie eine Fehlermeldung aufgrund der Servicekontingentbeschränkung. Wenn Sie über aktive reservierte Kapazität verfügen und versuchen, zusätzliche reservierte Kapazität zu erwerben, was zu mehr als 1.000.000 aktiven bereitgestellten Kapazitätseinheiten führen würde, erhalten Sie eine Fehlermeldung aufgrund der Servicekontingentbeschränkung.

Tabellen

Tabellengröße

Es gibt praktisch kein Limit für die Tabellengröße. Tabellen sind in Bezug auf die Anzahl von Elementen oder die Anzahl von Bytes unbeschränkt.

Maximale Anzahl von Tabellen pro Konto und Region

Für jedes AWS Konto gibt es ein anfängliches Kontingent von 2.500 Tabellen pro AWS Region.

Wenn Sie mehr als 2 500 Tabellen für ein einzelnes Konto benötigen, wenden Sie sich bitte an Ihr AWS -Kontoteam, um eine Erhöhung auf maximal 10 000 Tabellen zu prüfen. Bei mehr als 10 000 Tabellen wird empfohlen, mehrere Konten einzurichten, von denen jedes bis zu 10 000 Tabellen bearbeiten kann.

Globale Tabellen

Bei der Verwendung globaler Tabellen gelten die folgenden Standardkontingente.

Standardkontingente für globale Tabellen	On-Demand	Bereitgestellt
Anzahl der globalen MRSC-Tabellen (siehe the section called “Konsistenzmodi”)	Insgesamt 400 globale MRSC-Tabellen in einem beliebigen Kapazitätsmodus	Insgesamt 400 globale MRSC-Tabellen in einem beliebigen Kapazitätsmodus
Durchsatz pro Tabelle, konfiguriert für Multi-Region Eventual Consistency (MREC)	40 000 Leseanforderungseinheiten und 40 000 Schreibanforderungseinheiten	40.000 Lesekapazitätseinheiten und 40.000 Schreibkapazitätseinheiten
Durchsatz pro Tabelle, konfiguriert für Multi-Region Strong Consistency (MRSC)	40 000 Leseanforderungseinheiten und 40 000 Schreibanforderungseinheiten	40.000 Lesekapazitätseinheiten und 40.000 Schreibkapazitätseinheiten
Vervollständigte Daten für neue Replikate pro Konto, pro Region, pro Tag	10 TB	10 TB

Note

Es kann vorkommen, dass Sie eine Erhöhung des Kontingentlimits beantragen müssen AWS Support. Wenn einer der folgenden Punkte auf Sie zutrifft, wenden Sie sich bitte an https://aws.amazon.com/support:

- Die globalen Kontingente für den Tabellendurchsatz müssen mindestens den Durchsatzkontingenten pro Tabelle entsprechen, damit die Replikaterstellung erfolgreich ist. Für globale MREC- und MRSC-Tabellen gibt es separate Kontingente für den globalen Tabellendurchsatz.
- Wenn Sie innerhalb eines Zeitraums von 24 Stunden ein Replikat oder Replikate zu einer Zielregion mit einer kombinierten Summe von mehr als 10 TB hinzufügen, müssen Sie eine Erhöhung des Servicekontingents für das Backfill-Kontingent zum Hinzufügen von Replikaten anfordern.
- Wenn Sie eine Fehlermeldung erhalten, die etwa wie folgt lautet:

- Es kann kein Replikat der Tabelle 'example_table' in der Region 'example_region_A' erstellt werden, da damit Ihr aktuelles Kontolimit in der Region 'example_region_B' überschritten wird.

Sekundäre Indizes

Sie können bis zu 5 globale Sekundärindizes pro Tabelle definieren.

Pro Tabelle besteht ein Standardkontingent von 20 globalen sekundären Indizes.

Projizierte sekundäre Indexattribute

Sie können insgesamt bis zu 100 Attribute in alle lokalen und globalen sekundären Indizes einer Tabelle projizieren. Dieses Kontingent gilt nur für vom Benutzer angegebene, projizierte Attribute.

Wenn Sie in einer `CreateTable`-Operation für `ProjectionType` von `INCLUDE` angeben, darf die Gesamtanzahl von in `NonKeyAttributes` angegebenen Attributen, die sich aus der Summe aller sekundären Indizes ergibt, 100 nicht überschreiten. Das Projizieren desselben Attributnamens in zwei verschiedene Indizes zählt im Rahmen des Kontingents als zwei separate Attribute.

Dieses Kontingent gilt nicht für sekundäre Indizes mit einem `ProjectionType` von `KEYS_ONLY` oder `ALL`.

DynamoDB Streams

Gleichzeitige Leser eines Shard in DynamoDB Streams

Für Einzelregionstabellen, bei denen es sich nicht um globale Tabellen handelt, können Sie bis zu zwei simultane Prozesse entwerfen, die gleichzeitig aus demselben DynamoDB-Streams-Shard lesen. Eine Überschreitung dieses Grenzwerts kann zu einer Anforderungsdrosselung führen. Für globale Tabellen empfehlen wir, die Anzahl der gleichzeitigen Lesevorgänge auf einen zu beschränken, um eine Anforderungsdrosselung zu vermeiden.

Maximale Schreibkapazität für eine Tabelle mit aktivierten DynamoDB-Streams

AWS legt einige Standardkontingente für die Schreibkapazität für DynamoDB-Tabellen mit aktivierten DynamoDB Streams fest. Diese Standardkontingente gelten nur für Tabellen im Modus „Bereitgestellte Kapazität“. read/write

- Pro Tabelle – 40 000 Schreibkapazitätseinheiten

Importieren aus Amazon S3

Der DynamoDB-Import aus Amazon S3 kann bis zu 50 gleichzeitige Importaufgaben mit einer Gesamtgröße von 15 TB gleichzeitig in den Regionen us-east-1, us-west-2 und eu-west-1 unterstützen. In allen anderen Regionen werden bis zu 50 gleichzeitige Importaufgaben mit einer Gesamtgröße von 1 TB unterstützt. Jede Importaufgabe kann bis zu 50 000 Amazon-S3-Objekte in allen Regionen aufnehmen. Weitere Informationen zum Importieren und Validieren finden Sie unter Importformatkontingente und Validierung.

Exportieren von Tabellen zu Amazon S3

Es können bis zu 300 gleichzeitige Exportaufgaben oder bis zu insgesamt 100 TB aus allen laufenden Tabellenexporten exportiert werden. Beide Grenzwerte werden überprüft, bevor ein Export in die Warteschlange gestellt wird.

Inkrementeller Export: Der inkrementelle DynamoDB-Export in Amazon S3 kann bis zu 300 gleichzeitige Exportaufgaben oder bis zu 100 TB aus allen in der Übertragung befindlichen Tabellenexporten unterstützen. Die Zeitfensterbegrenzung für den Exportzeitraum beträgt mindestens 15 Minuten und höchstens 24 Stunden.

Backup und Wiederherstellung

DynamoDB unterstützt bis zu 50 gleichzeitige Wiederherstellungen mit insgesamt 50 TB über DynamoDB-Backups auf Abruf oder kontinuierliche Backups. AWS Backup unterstützt bis zu 50 gleichzeitige Wiederherstellungen mit einer Gesamtkapazität von 25 TB.

Contributor Insights

Wenn Sie Customer Insights für Ihre DynamoDB-Tabelle aktivieren, unterliegen Sie weiterhin den Grenzwerten der Contributor Insights-Regeln. Weitere Informationen finden Sie unter CloudWatch - Servicekontingente.

Einschränkungen in Amazon DynamoDB

In diesem Abschnitt werden die aktuellen Einschränkungen – früher als Grenzwerte bezeichnet – in Amazon DynamoDB beschrieben.

> **Note**
>
> Alle Größenmessungen in DynamoDB verwenden binäre Einheiten. DynamoDB steht für 1 KB = 1024 Byte, 1 MB = 1024 KB, 1 GB = 1024 MB, 1 TB = 1024 GB.

Themen

- Lese-/Schreibkapazitätsmodus
- Sekundäre Indizes
- Partitions- und Sortierschlüssel
- Benennungsregeln
- Datentypen
- Items
- Attribute
- Ausdrucksparameter
- DynamoDB-Transaktionen
- DynamoDB Streams
- DynamoDB Accelerator (DAX)
- API-spezifische Einschränkungen
- Ruhende DynamoDB-Verschlüsselung

Lese-/Schreibkapazitätsmodus

Sie können Tabellen innerhalb eines fortlaufenden 24-stündigen Zeitfensters bis zu viermal vom Modus mit bereitgestellter Kapazität auf den On-Demand-Modus umstellen. Sie können Tabellen jederzeit vom On-Demand-Modus auf den Modus mit bereitgestellter Kapazität umstellen.

Weitere Informationen zum Wechseln zwischen Lese- und Schreibkapazitätsmodus finden Sie unter Überlegungen beim Umstellen der Kapazitätsmodi in DynamoDB.

Kapazitätseinheitsgrößen (für Tabellen mit bereitgestellter Kapazität)

Eine Lesekapazitätseinheit entspricht einem Strongly-Consistent-Lesevorgang pro Sekunde oder zwei Eventually-Consistent-Lesevorgängen pro Sekunde für Elemente mit einer Größe von bis zu 4 KB.

Eine Schreibkapazitätseinheit entspricht einem Schreibvorgang pro Sekunde für Elemente mit einer Größe von bis zu 1 KB.

Transactional-Leseanforderungen benötigen zwei Lesekapazitätseinheiten, um einen Lesevorgang pro Sekunde für Elemente mit einer Größe bis zu 4 KB durchzuführen.

Transaktionelle Schreibforderungen benötigen zwei Schreibkapazitätseinheiten, um einen Schreibvorgang pro Sekunde für Elemente mit einer Größe bis zu 1 KB durchzuführen.

Anforderungseinheitengrößen (für On-Demand-Tabellen)

Eine Leseanforderungseinheit entspricht einem strikt konsistenten Lesevorgang pro Sekunde oder zwei letztendlich konsistenten Lesevorgängen pro Sekunde für Elemente mit einer Größe von bis zu 4 KB.

Eine Schreibanforderungseinheit entspricht einem Schreibvorgang pro Sekunde für Elemente mit einer Größe von bis zu 1 KB.

Transaktionale Leseanforderungen benötigen zwei Leseanforderungseinheiten, um einen Lesevorgang pro Sekunde für Elemente mit einer Größe bis zu 4 KB durchführen zu können.

Transaktionale Schreibanforderungen benötigen zwei Schreibanforderungseinheiten, um einen Schreibvorgang pro Sekunde für Elemente mit einer Größe bis zu 1 KB durchführen zu können.

Sekundäre Indizes

Projizierte sekundäre Indexattribute pro Tabelle

Sie können insgesamt bis zu 100 Attribute in alle lokalen und globalen sekundären Indizes einer Tabelle projizieren. Dies gilt nur für vom Benutzer angegebene, projizierte Attribute.

Wenn Sie in einer `CreateTable`-Operation für `ProjectionType` von `INCLUDE` angeben, darf die Gesamtanzahl von in `NonKeyAttributes` angegebenen Attributen, die sich aus der Summe aller sekundären Indizes ergibt, 100 nicht überschreiten. Wenn Sie denselben Attributnamen in zwei verschiedene Indizes projizieren, werden beim Ermitteln der Gesamtanzahl zwei unterschiedliche Attribute gezählt.

Dieses Limit gilt nicht für sekundäre Indizes mit dem `ProjectionType` `KEYS_ONLY` oder `ALL`.

Partitions- und Sortierschlüssel

Partitionsschlüssellänge

Die Mindestlänge eines Partitionsschlüsselwerts beträgt 1 Byte. Die maximale Länge beträgt 2 048 Byte.

Partitionsschlüsselwerte

Es gibt praktisch keine Einschränkung in Bezug auf die Anzahl von eindeutigen Partitionsschlüsselwerten, weder für Tabellen noch für sekundäre Indizes.

Sortierschlüssellänge

Die Mindestlänge eines Sortierschlüsselwerts beträgt 1 Byte. Die maximale Länge beträgt 1 024 Byte.

Sortierschlüsselwerte

Im Prinzip gibt es praktisch keine Einschränkung in Bezug auf die Anzahl von eindeutigen Sortierschlüsselwerten pro Partitionsschlüsselwert.

Eine Ausnahme bilden Tabellen mit sekundären Indizes. Eine Elementauflistung ist ein Satz von Elementen, die als Partitionsschlüsselattribut den gleichen Wert haben. In einem globalen sekundären Index ist die Elementauflistung unabhängig von der Basistabelle (und kann ein anderes Partitionsschlüsselattribut haben). In einem lokalen sekundären Index befindet sich die indizierte Ansicht jedoch in derselben Partition (Co-Location) wie das Element in der Tabelle und verwendet dasselbe Partitionsschlüsselattribut. Aufgrund dieser Lokalität kann die Elementsammlung nicht auf mehrere Partitionen verteilt werden LSIs, wenn eine Tabelle eine oder mehrere Partitionen enthält.

Bei einer Tabelle mit einer oder mehreren LSIs Elementen dürfen Elementsammlungen eine Größe von 10 GB nicht überschreiten. Dies schließt alle Basistabellenelemente und alle projizierten LSI-Ansichten ein, die denselben Wert des Partitionsschlüsselattributs haben. 10 GB ist die maximale Größe einer Partition. Detailliertere Informationen erhalten Sie unter Größenlimit der Elementauflistung.

Benennungsregeln

Tabellennamen und sekundäre Indexnamen

Namen für Tabellen und sekundäre Indizes müssen mindestens 3 Zeichen und dürfen höchstens 255 Zeichen lang sein. Es sind die folgenden Zeichen zulässig:

- `A-Z`
- `a-z`
- `0-9`
- `_` (Unterstrich)
- `-` (Bindestrich)
- `.` (Punkt)

Attributnamen

Im Allgemeinen muss ein Attributname mindestens ein Zeichen lang sein und darf nicht größer als 64 KB sein.

Dabei gibt es die folgenden Ausnahmen. Diese Attributnamen dürfen maximal 255 Zeichen lang sein:

- Partitionsschlüsselnamen des Sekundärindexes
- Sortierschlüsselnamen des Sekundärindexes
- Die Namen aller vom Benutzer angegebenen, projizierten Attribute (gilt nur für lokale sekundäre Indizes). Wenn Sie in einer `CreateTable`-Operation den `ProjectionType` `INCLUDE` angeben, sind die Namen der Attribute des Parameters `NonKeyAttributes` längenbeschränkt. Die Projektionstypen `KEYS_ONLY` und `ALL` sind nicht betroffen.

Diese Attributnamen müssen mit UTF-8 kodiert werden und die Gesamtgröße der einzelnen Namen (nach der Kodierung) darf nicht größer als 255 Byte sein.

Datentypen

Zeichenfolge

Die Länge einer Zeichenfolge wird durch die maximale Elementgröße 400 KB beschränkt.

Die Zeichenfolgen sind Unicode mit binärer UTF-8-Kodierung. Da UTF-8 eine Kodierung mit variabler Breite ist, bestimmt DynamoDB die Länge einer Zeichenfolge unter Verwendung der UTF-8-Bytes.

Zahl

Eine Zahl kann bis zu 38 Nachkommastellen besitzen und positiv, negativ oder null sein.

- Positiver Bereich: `1E-130` bis `9.9999999999999999999999999999999999999E+125`
- Negativer Bereich: `-9.9999999999999999999999999999999999999E+125` bis `-1E-130`

DynamoDB verwendet JSON-Zeichenfolgen, um Zahlendaten in Anfragen und Antworten darzustellen. Weitere Informationen finden Sie unter DynamoDB Low-Level-API.

Wenn die Zahlengenauigkeit wichtig ist, sollten Sie Zahlen mithilfe von Zeichenfolgen, die Sie aus einem Number-Datentyp konvertieren, an DynamoDB übergeben.

Binär

Die Länge eines binären Werts wird durch die maximale Elementgröße 400 KB beschränkt.

Anwendungen, die binäre Attribute verwenden, müssen die Daten vor dem Senden an DynamoDB im base64-Format kodieren. Nach dem Empfang der Daten dekodiert DynamoDB sie in ein nicht signiertes Byte-Array und verwendet dieses Array für die Länge der Attribute.

Items

Elementgröße

Die maximale Elementgröße in DynamoDB beträgt 400 KB. Darin inbegriffen sind die binäre Länge (UTF-8-Länge) des Attributnamens und die Länge des Attributwerts (ebenfalls binäre Länge). Der Attributname wird bei der Größenbeschränkung mit eingerechnet.

Betrachten wir ein Element mit zwei Attributen als Beispiel: ein Attribut mit dem Namen „Shirtfarbe“ und dem Wert „R“ und ein weiteres Attribut mit dem Namen „Shirtgröße“ und dem Wert „M“. Die Gesamtgröße dieses Elements ist 23 Byte.

Elementgröße für Tabellen mit lokalen sekundären Indizes

Für jeden lokalen sekundären Index in einer Tabelle gilt ein Limit von 400 KB im Hinblick auf die Summe folgender Werte:

- Die Größe der Daten eines Elements in der Tabelle.
- Die Größe der entsprechenden Einträge (einschließlich der Schlüsselwerte und projizierten Attribute) in allen lokalen sekundären Indizes.

Attribute

Attribut-Namen-Wert-Paare pro Element

Die Gesamtgröße von Attributen pro Element muss der maximalen DynamoDB-Elementgröße (400 KB) entsprechen.

Anzahl der Werte in einer Liste, einer Zuordnung oder einem Satz

Es gibt keine Beschränkungen in Bezug auf die Anzahl der Werte in einer Liste, einer Zuordnung oder einem Satz, solange das Element, das die Werte enthält, das Limit der Elementgröße von 400 KB einhält.

Attributwerte

Leere Zeichenfolgen- und Binär-Attributwerte sind zulässig, wenn das Attribut nicht als Schlüsselattribut für eine Tabelle oder einen Index verwendet wird. Leere Zeichenfolgen- und Binärwerte sind innerhalb der Mengen, Listen und Zuordnungen zulässig. Ein Attributwert kann kein leerer Satz sein (Zeichenfolgensatz, Zahlensatz und Binärzahlensatz). Leere Listen und Zuordnungen sind jedoch zulässig.

Verschachtelte Attributtiefe

DynamoDB unterstützt bis zu 32 Ebenen verschachtelter Attribute.

Ausdrucksparameter

Ausdrucksparameter umfassen `ProjectionExpression`, `ConditionExpression`, `UpdateExpression` und `FilterExpression`.

Länge

Die maximale Länge der Ausdruckszeichenfolge ist 4 KB. Die Größe von `ConditionExpression` `a=b` beträgt z. B. 3 Byte.

Die maximale Länge jedes einzelnen Ausdrucksattributnamens oder Ausdrucksattributwerts beträgt 255 Byte. `#name` ist beispielsweise 5 Byte lang; `:val` 4 Byte.

Die maximale Länge aller Substitutionsvariablen in einem Ausdruck beträgt 2 MB. Dies ist die Summe der Längen aller `ExpressionAttributeNames` und `ExpressionAttributeValues`.

Operatoren und Operanden

Die maximale Anzahl von in `UpdateExpression` zulässigen Operatoren oder Funktionen ist 300. Beispielsweise UpdateExpression`SET a = :val1 + :val2 + :val3`enthält der zwei "+" - Operatoren.

Die maximale Anzahl von Operanden für den Vergleichsoperator `IN` ist 100.

Reservierte Wörter

DynamoDB verhindert nicht, dass Namen verwendet werden, die mit reservierten Wörtern in Konflikt stehen. (Eine vollständige Liste finden Sie unter Reservierte Wörter in DynamoDB.)

Wenn Sie jedoch ein reserviertes Wort in einem Ausdrucksparameter verwenden, müssen Sie auch `ExpressionAttributeNames` angeben. Weitere Informationen finden Sie unter Ausdrucksattributnamen (Aliasse) in DynamoDB.

DynamoDB-Transaktionen

Für transaktionale DynamoDB-API-Operationen gelten die folgenden Einschränkungen:

- Eine Transaktion darf nicht mehr als 100 eindeutige Elemente enthalten.
- Eine Transaktion darf nicht mehr als 4 MB Daten enthalten.
- Zwei Aktionen gleichzeitig dürfen nicht auf dasselbe Element in der derselben Tabelle einwirken. Beispielsweise können `ConditionCheck` und `Update` in einer Transaktion nicht beide auf dasselbe Element abzielen.
- Eine Transaktion kann nicht für Tabellen in mehr als einem AWS Konto oder einer Region ausgeführt werden.
- Transaktionsoperationen garantieren Atomarität, Konsistenz, Isolation und Dauerhaftigkeit (ACID) nur innerhalb der AWS Region, in der der Schreibvorgang ursprünglich vorgenommen wurde. Regionsübergreifende Transaktionen werden in globalen Tabellen nicht unterstützt. Angenommen, Sie haben eine globale Tabelle mit Replikaten in den Regionen USA Ost (Ohio) und USA West (Oregon) und Sie führen einen `TransactWriteItems`-Vorgang in der Region USA Ost (Nord-Virginia) aus. In diesem Fall können sind möglicherweise teilweise abgeschlossene Transaktionen in der Region USA West (Oregon) zu beobachten, während Änderungen repliziert werden. Die Änderungen werden erst dann in die anderen Regionen repliziert, nachdem sie in der Quellregion in die Datenbank eingetragen wurden.

DynamoDB Streams

Gleichzeitige Leser eines Shard in DynamoDB Streams

Für Einzelregionstabellen, bei denen es sich nicht um globale Tabellen handelt, können Sie bis zu zwei Prozesse entwerfen, die gleichzeitig aus demselben DynamoDB-Streams-Shard lesen. Eine Überschreitung dieses Grenzwerts kann zu einer Anforderungsdrosselung führen. Für globale Tabellen empfehlen wir, die Anzahl der gleichzeitigen Lesevorgänge auf einen zu beschränken, um eine Anforderungsdrosselung zu vermeiden.

DynamoDB Accelerator (DAX)

AWS Verfügbarkeit in der Region

Eine Liste der AWS Regionen, in denen DAX verfügbar ist, finden Sie unter DynamoDB Accelerator (DAX) in der. Allgemeine AWS-Referenz

Knoten

Ein DAX-Cluster besteht aus genau einem Primärknoten und zwischen null und zehn Lesereplikat-Knoten.

Die Gesamtzahl der Knoten (pro AWS Konto) darf 50 in einer einzelnen AWS Region nicht überschreiten.

Parametergruppen

Sie können bis zu 20 DAX-Parametergruppen pro Region erstellen.

Subnetzgruppen

Sie können bis zu 50 DAX-Subnetzgruppen pro Region erstellen.

Innerhalb einer Subnetzgruppe können Sie bis zu 20 Subnetze definieren.

> **Important**
>
> Ein DAX-Cluster unterstützt maximal 500 DynamoDB-Tabellen. Wenn Sie mehr als 500 DynamoDB-Tabellen verwenden, kann es bei Ihrem Cluster zu einer Verschlechterung der Verfügbarkeit und Leistung kommen.

API-spezifische Einschränkungen

`CreateTable`/`UpdateTable`/`DeleteTable`/`PutResourcePolicy`/`DeleteResourcePolicy`

Im Allgemeinen können bis zu 500 CreateTable,, UpdateTableDeleteTablePutResourcePolicy, und DeleteResourcePolicyAnfragen gleichzeitig in beliebiger Kombination ausgeführt werden. Folglich darf die Gesamtanzahl von Tabellen im Status `CREATING`, `UPDATING` oder `DELETING` nicht mehr als 500 betragen.

Sie können pro Sekunde bis zu 2 500 Anfragen veränderbarer Steuerebenen-API-Anfragen (`CreateTable`, `DeleteTable`, `UpdateTable`, `PutResourcePolicy` und `DeleteResourcePolicy`) in einer Tabellengruppe senden. Für `PutResourcePolicy`- und `DeleteResourcePolicy`-Anforderungen gelten jedoch niedrigere individuelle Grenzwerte. Weitere Informationen finden Sie in den folgenden Kontingentdetails für `PutResourcePolicy` und `DeleteResourcePolicy`.

`CreateTable`- und `PutResourcePolicy`-Anforderungen, die eine ressourcenbasierte Richtlinie beinhalten, zählen als zwei zusätzliche Anfragen pro KB der Richtlinie. Eine `PutResourcePolicy`- oder `CreateTable`-Anforderung mit einer Richtliniengröße von 5 KB wird beispielsweise als 11 Anforderungen gezählt. 1 für die `CreateTable`-Anforderung und 10 für die ressourcenbasierte Richtlinie (2 x 5 KB). Entsprechend wird eine Richtlinie mit einer Größe von 20 KB als 41 Anforderungen gezählt: 1 für die `CreateTable`-Anforderung und 40 für die ressourcenbasierte Richtlinie (2 x 20 KB).

`PutResourcePolicy`

Sie können bis zu 25 `PutResourcePolicy`-API-Anforderungen pro Sekunde für eine Gruppe von Tabellen senden. Nach einer erfolgreichen Anforderung für eine einzelne Tabelle werden in den folgenden 15 Sekunden keine neuen `PutResourcePolicy`-Anforderungen unterstützt.

Ein ressourcenbasiertes Richtliniendokument kann maximal 20 KB groß sein. Leerzeichen werden in DynamoDB beim Berechnen der Richtliniengröße in Bezug auf diesen Limit nicht mitgezählt.

`DeleteResourcePolicy`

Sie können bis zu 50 `DeleteResourcePolicy`-API-Anforderungen pro Sekunde für eine Gruppe von Tabellen senden. Nach einer erfolgreichen `PutResourcePolicy`-Anforderung für eine einzelne Tabelle werden in den folgenden 15 Sekunden keine neuen `DeleteResourcePolicy`-Anforderungen unterstützt.

`BatchGetItem`

Eine einzelne `BatchGetItem`-Operation kann maximal 100 Elemente abrufen. Die Gesamtgröße aller abgerufenen Elemente darf 16 MB nicht überschreiten.

`BatchWriteItem`

Eine einzelne `BatchWriteItem`-Operation kann bis zu 25 `PutItem`- oder `DeleteItem`-Anforderungen umfassen. Die Gesamtgröße aller geschriebenen Elemente darf 16 MB nicht überschreiten.

`DescribeStream`

Sie können `DescribeStream` maximal 10 Mal pro Sekunde aufrufen.

`DescribeTableReplicaAutoScaling`

Die Methode `DescribeTableReplicaAutoScaling` unterstützt nur zehn Anforderungen pro Sekunde.

`DescribeLimits`

`DescribeLimits` sollte nur gelegentlich aufgerufen werden. Sie müssen mit Drosselungsfehlern rechnen, wenn Sie die Anforderung mehr als einmal pro Minute aufrufen.

`DescribeContributorInsights`/`ListContributorInsights`/`UpdateContributorInsights`

`DescribeContributorInsights`, `ListContributorInsights` und `UpdateContributorInsights` sollten nur gelegentlich aufgerufen werden. DynamoDB unterstützt für jede dieser Anfragen bis zu fünf Anfragen pro Sekunde. APIs

`DescribeTable`/`ListTables`/`GetResourcePolicy`

Sie können pro Sekunde bis zu 2 500 Anforderungen an schreibgeschützten (`DescribeTable`, `ListTables` und `GetResourcePolicy`) Steuerebenen-API-Anforderungen in beliebiger Kombination senden. Für die `GetResourcePolicy`-API gilt ein niedrigeres individuelles Limit von 100 Anforderungen pro Sekunde.

`DescribeTimeToLive`

Die `DescribeTimeToLive`-Operation ist auf 10 Leseanforderungseinheiten pro Sekunde gedrosselt. Wenn Sie diese Grenze überschreiten, gibt DynamoDB einen `ThrottlingException`-Fehler zurück.

`Query`

Der Ergebnissatz einer `Query` ist auf 1 MB pro Aufruf beschränkt. Sie können den `LastEvaluatedKey` von der Query-Antwort verwenden, um weitere Ergebnisse abzurufen.

`Scan`

Der Ergebnissatz einer `Scan` ist auf 1 MB pro Aufruf beschränkt. Sie können den `LastEvaluatedKey` von der Scan-Antwort verwenden, um weitere Ergebnisse abzurufen.

`UpdateKinesisStreamingDestination`

Bei der Ausführung von `UpdateKinesisStreamingDestination`-Operationen können Sie innerhalb von 24 Stunden maximal dreimal einen neuen Wert für `ApproximateCreationDateTimePrecision` festlegen.

`UpdateTableReplicaAutoScaling`

Die Methode `UpdateTableReplicaAutoScaling` unterstützt nur zehn Anforderungen pro Sekunde.

`UpdateTableTimeToLive`

Die Methode `UpdateTableTimeToLive` unterstützt nur eine Anforderung zum Aktivieren oder Deaktivieren von `Time to Live (TTL)` pro angegebener Tabelle und Stunde. Es kann bis zu einer Stunde dauern, bis diese Änderung vollständig verarbeitet ist. Alle zusätzlichen `UpdateTimeToLive` Aufrufe für dieselbe Tabelle während dieser einstündigen Dauer führen zu einer. ValidationException

Ruhende DynamoDB-Verschlüsselung

Ab dem Zeitpunkt AWS-eigener Schlüssel, an dem die Tabelle erstellt wurde Von AWS verwalteter Schlüssel, können Sie innerhalb eines 24-Stunden-Fensters bis zu viermal pro Tabelle zwischen einem, einem und einem kundenverwalteten Schlüssel wechseln. Wenn in den vorhergehenden 6 Stunden kein Wechsel erfolgt ist, ist ein zusätzlicher Wechsel erlaubt. Dadurch wird die maximale Anzahl an Wechseln pro Tag faktisch auf 8 erhöht (4 Wechsel in den ersten 6 Stunden und ein weiterer Wechsel jeweils für alle folgenden Zeitfenster von 6 Stunden an einem Tag).

Sie können die Verschlüsselungsschlüssel AWS-eigener Schlüssel so oft wie nötig wechseln, um sie zu verwenden, auch wenn das oben genannte Kontingent ausgeschöpft ist.

Es gelten die folgenden Kontingente, sofern Sie keine höheren Werte beantragen. Informationen zur Beantragung einer Erhöhung des Servicekontingents finden Sie unter https://aws.amazon.com/support.

Referenz zur DynamoDB-API

Die Amazon DynamoDB API-Referenz enthält eine vollständige Liste der Operationen, die von folgenden unterstützt werden:

- DynamoDB
- DynamoDB-Streams
- DynamoDB Accelerator (DAX).

Fehlerbehebung bei Amazon DynamoDB

Die folgenden Themen enthalten Ratschläge zur Behebung von Fehlern und Problemen, die bei der Verwendung von Amazon DynamoDB auftreten können. Wenn Sie auf ein Problem stoßen, das hier nicht aufgeführt ist, können Sie die Schaltfläche „Feedback“ auf dieser Seite verwenden, um es zu melden.

Weitere Tipps zur Fehlerbehebung und Antworten auf häufig gestellte Supportfragen finden Sie im AWS -Wissenscenter.

Themen

- Beheben interner Serverfehler in Amazon DynamoDB
- Beheben von Latenzproblemen in Amazon DynamoDB
- Beheben von Drosselungsereignissen in Amazon DynamoDB

Beheben interner Serverfehler in Amazon DynamoDB

In DynamoDB deuten interne Serverfehler (500-Fehler) darauf hin, dass der Service die Anforderung nicht bearbeiten kann. Diese Fehler können aus verschiedenen Gründen auftreten, darunter z. B. vorübergehende Netzwerkprobleme in der Flotte, Infrastrukturprobleme, Probleme im Zusammenhang mit Speicherknoten und mehr.

Während des Lebenszyklus Ihrer DynamoDB-Tabelle können einige interne Serverfehler auftreten. Dies ist aufgrund des verteilten Charakters des Service zu erwarten und sollte in der Regel kein Grund zur Sorge sein. DynamoDB repariert und behebt automatisch alle vorübergehenden Probleme mit dem Service in Echtzeit, ohne dass Sie eingreifen müssen. Wenn Sie jedoch eine konstant hohe Anzahl interner Serverfehler bei Anfragen an Ihre Tabelle beobachten (wie in der the section called “SystemErrors”-Metrik dargestellt), sollten Sie weitere Untersuchungen durchführen.

Themen

- Untersuchen interner Serverfehler
- Minimieren der Auswirkungen interner Serverfehler
- Verbessern der operativen Einblicke

Untersuchen interner Serverfehler

Wenn Sie in Ihrer DynamoDB-Tabelle auf interne Serverfehler stoßen, sollten Sie die folgenden Optionen in Betracht ziehen:

1. Überprüfen Sie das AWS Health Dashboard.

   Um das Problem zu identifizieren, überprüfen Sie zunächst das AWS Service Health Dashboard und Ihr AWS Account Health Dashboard. Diese Dashboards bieten wertvolle Informationen zu allen serviceweiten Problemen, betroffenen Tabellen, andauernden Problemen sowie der Ursache, sobald das Problem behoben wurde.

   Wenn Sie die Details in diesen Dashboards überprüfen, erhalten Sie einen besseren Überblick über den aktuellen Status der von AWS-Services Ihnen verwendeten Software und über mögliche Probleme, die Ihr Konto betreffen. Anhand dieser Informationen können Sie die nächsten Schritte zur Behebung des Problems und zur Minimierung von Betriebsunterbrechungen festlegen.

2. Wenden Sie sich an. Support

   Wenn Sie in Ihren Anforderungen länger andauernde, anhaltende Fehler feststellen, kann dies auf ein Problem mit dem Service hinweisen. In der Regel gilt: Wenn Sie in den letzten 15 Minuten eine Gesamtausfallrate von 1% oder mehr feststellen, ist es ein geeigneter Zeitpunkt, das Problem an das AWS Support-Team weiterzuleiten. Weitere Informationen finden Sie unter DynamoDB Service Level Agreement.

   Wenn Sie einen Fall mit dem AWS Support-Team eröffnen, geben Sie die folgenden Informationen an, um die Fehlerbehebung zu beschleunigen:

   - Betroffene DDB; Tabellen oder sekundäre Indizes
   - Zeitfenster, in dem die Fehler festgestellt wurden
   - DynamoDB-Anfrage IDs, z. B. `4KBNVRGD25RG1KE09UT4V3FQDJVV4KQNSO5AEMVJF66Q9ASUAAJG`, die Sie in Ihren Anwendungsprotokollen finden.

   Wenn Sie diese Details in den Support-Fall aufnehmen, kann das AWS Team das Problem besser verstehen und es schneller lösen. Wenn Ihnen die Anfrage nicht vorliegt IDs, sollten Sie den Fall trotzdem mit den anderen verfügbaren Details protokollieren.

Minimieren der Auswirkungen interner Serverfehler

Wenn bei der Verwendung von DynamoDB interne Serverfehler auftreten, minimieren Sie deren Auswirkungen auf Ihre Anwendung. Beachten Sie dabei die folgenden bewährten Methoden:

- Backoffs und Wiederholversuche verwenden: Das SDK-Standardverhalten von DynamoDB ist so konzipiert, dass es für die meisten Anwendungen die richtige Balance zwischen Backoff- und Wiederholungsstrategie findet. Sie können diese Einstellungen jedoch an die Toleranz Ihrer Anwendung in Bezug auf Ausfallzeiten und Leistungsanforderungen anpassen. Erfahren Sie mehr über Backoffs und Wiederholversuche, um nachzuvollziehen, wie Sie diese Einstellungen für Wiederholversuche optimieren können.
- Letztendlich konsistente Lesevorgänge verwenden: Wenn Ihre Anwendung keine strikt konsistenten Lesevorgänge erfordert, sollten Sie die Verwendung letztendlich konsistenter Lesevorgänge in Betracht ziehen. Diese Lesevorgänge sind kostengünstiger und es ist weniger wahrscheinlich, dass vorübergehende Probleme aufgrund interner Serverfehler auftreten, da sie von einem der verfügbaren Speicherknoten aus verarbeitet würden. Weitere Informationen finden Sie unter DynamoDB-Lesekonsistenz.

Verbessern der operativen Einblicke

Die Aufrechterhaltung einer hohen Verfügbarkeit und Zuverlässigkeit Ihrer Anwendungen ist in der digitalen Landschaft von heute von entscheidender Bedeutung. Ein wichtiger Aspekt dabei ist die proaktive Überwachung Ihrer DynamoDB-Tabellen und globalen Sekundärindizes (ISEs) auf interne Serverfehler (). GSIs Durch die Einrichtung von CloudWatch Alarmen zur Überwachung dieser Fehler können Sie einen besseren Überblick über den Betrieb gewinnen und sich vor potenziellen Problemen warnen lassen, bevor sie sich auf Ihre Endbenutzer auswirken. Dieser Ansatz steht im Einklang mit dem Grundpfeiler Operational Excellence des AWS Well-Architected Framework und stellt sicher, dass Ihr DynamoDB-Workload im Hinblick auf Leistung, Sicherheit und Zuverlässigkeit optimiert ist.

CloudWatch Alarme erstellen

Sie sollten in Ihren DynamoDB-Tabellen CloudWatch Alarme eingerichtet haben, um Benachrichtigungen für eine konstant hohe Anzahl interner Serverfehler zu erhalten, anstatt die Metriken manuell zu beobachten. Dies steht im Einklang mit der Säule Operational Excellence des Well-Architected-Frameworks für alle Workloads. AWS Weitere Informationen zum Aufbau einer

optimalen Architektur Ihrer DynamoDB-Tabellen finden Sie unter Verwenden der DynamoDB Well-Architected Lens zur Optimierung Ihres DynamoDB-Workloads.

Diese Alarme verwenden benutzerdefinierte Metrikberechnungen, um den Prozentsatz fehlgeschlagener Anforderungen für ein 5-Minuten-Fenster zu berechnen. Es wird empfohlen, den Alarm so zu konfigurieren, dass er in den ALARM-Status wechselt, wenn drei aufeinanderfolgende Datenpunkte den Schwellenwert von 1 % überschreiten, was bedeutet, dass insgesamt 1 % der Anforderungen innerhalb eines Zeitraums von 15 Minuten fehlschlagen.

Das folgende Beispiel ist eine CloudFormation Vorlage, mit der Sie CloudWatch Alarme für Ihre Tabelle und GSI für die Tabelle erstellen können.

```
AWSTemplateFormatVersion: "2010-09-09"
Description: Sample template for monitoring DynamoDB
Parameters:
 DynamoDBProvisionedTableName:
    Description: Name of DynamoDB Provisioned Table to create
    Type: String
    MinLength: 3
    MaxLength: 255
    ConstraintDescription : https://docs.aws.amazon.com/amazondynamodb/latest/
developerguide/Limits.html#limits-naming-rules
  DynamoDBSNSEmail:
    Description : Email Address subscribed to newly created SNS Topic
    Type: String
    AllowedPattern: "^[a-zA-Z0-9_.+-]+@[a-zA-Z0-9-]+\\.[a-zA-Z0-9-.]+$"
    MinLength: 1
    MaxLength: 255

Resources:
  DynamoDBMonitoringSNSTopic:
    Type: AWS::SNS::Topic
    Properties:
      DisplayName: DynamoDB Monitoring SNS Topic
      Subscription:
        - Endpoint: !Ref DynamoDBSNSEmail
          Protocol: email
      TopicName: dynamodb-monitoring

  DynamoDBTableSystemErrorAlarm:
    Type: 'AWS::CloudWatch::Alarm'
    Properties:
      AlarmName: 'DynamoDBTableSystemErrorAlarm'
```

```
      AlarmDescription: 'Alarm when system errors exceed 1% of total number of requests
for 15 minutes'
      AlarmActions:
        - !Ref DynamoDBMonitoringSNSTopic
      Metrics:
        - Id: 'e1'
          Expression: 'm1/(m1+m2+m3)'
          Label: SystemErrorsOverTotalRequests
        - Id: 'm1'
          MetricStat:
            Metric:
              Namespace: 'AWS/DynamoDB'
              MetricName: 'SystemErrors'
              Dimensions:
                - Name: 'TableName'
                  Value: !Ref DynamoDBProvisionedTableName
            Period: 300
            Stat: 'SampleCount'
            Unit: 'Count'
          ReturnData: False
        - Id: 'm2'
          MetricStat:
            Metric:
              Namespace: 'AWS/DynamoDB'
              MetricName: 'ConsumedReadCapacityUnits'
              Dimensions:
                - Name: 'TableName'
                  Value: !Ref DynamoDBProvisionedTableName
            Period: 300
            Stat: 'SampleCount'
            Unit: 'Count'
          ReturnData: False
        - Id: 'm3'
          MetricStat:
            Metric:
              Namespace: 'AWS/DynamoDB'
              MetricName: 'ConsumedWriteCapacityUnits'
              Dimensions:
                - Name: 'TableName'
                  Value: !Ref DynamoDBProvisionedTableName
            Period: 300
            Stat: 'SampleCount'
            Unit: 'Count'
          ReturnData: False
```

```
      EvaluationPeriods: 3
      Threshold: 1.0
      ComparisonOperator: 'GreaterThanThreshold'
  DynamoDBGSISystemErrorAlarm:
    Type: 'AWS::CloudWatch::Alarm'
    Properties:
      AlarmName: 'DynamoDBGSISystemErrorAlarm'
      AlarmDescription: 'Alarm when GSI system errors exceed 2% of total number of
requests for 15 minutes'
      AlarmActions:
        - !Ref DynamoDBMonitoringSNSTopic
      Metrics:
        - Id: 'e1'
          Expression: 'm1/(m1+m2+m3)'
          Label: GSISystemErrorsOverTotalRequests
        - Id: 'm1'
          MetricStat:
            Metric:
              Namespace: 'AWS/DynamoDB'
              MetricName: 'SystemErrors'
              Dimensions:
                - Name: 'TableName'
                  Value: !Ref DynamoDBProvisionedTableName
                - Name: 'GlobalSecondaryIndexName'
                  Value: !Join [ '-', [!Ref DynamoDBProvisionedTableName, 'gsi1'] ]
            Period: 300
            Stat: 'SampleCount'
            Unit: 'Count'
          ReturnData: False
        - Id: 'm2'
          MetricStat:
            Metric:
              Namespace: 'AWS/DynamoDB'
              MetricName: 'ConsumedReadCapacityUnits'
              Dimensions:
                - Name: 'TableName'
                  Value: !Ref DynamoDBProvisionedTableName
                - Name: 'GlobalSecondaryIndexName'
                  Value: !Join [ '-', [!Ref DynamoDBProvisionedTableName, 'gsi1'] ]
            Period: 300
            Stat: 'SampleCount'
            Unit: 'Count'
          ReturnData: False
        - Id: 'm3'
```

```
          MetricStat:
            Metric:
              Namespace: 'AWS/DynamoDB'
              MetricName: 'ConsumedWriteCapacityUnits'
              Dimensions:
                - Name: 'TableName'
                  Value: !Ref DynamoDBProvisionedTableName
                - Name: 'GlobalSecondaryIndexName'
                  Value: !Join [ '-', [!Ref DynamoDBProvisionedTableName, 'gsi1'] ]
            Period: 300
            Stat: 'SampleCount'
            Unit: 'Count'
          ReturnData: False
      EvaluationPeriods: 3
      Threshold: 1.0
      ComparisonOperator: 'GreaterThanThreshold'
```

Beheben von Latenzproblemen in Amazon DynamoDB

Wenn Ihr Workload eine hohe Latenz aufweist, können Sie die CloudWatch-Metrik `SuccessfulRequestLatency` analysieren und die durchschnittliche und mittlere Latenz anhand von Perzentilmetriken (p50) überprüfen, um zu ermitteln, ob das Problem mit DynamoDB zusammenhängt. Gewisse Schwankungen im gemeldeten Wert zu `SuccessfulRequestLatency` sind normal und gelegentliche Spitzen (insbesondere in der `Maximum`-Statistik und bei hohen Perzentilwerten) sollten Sie nicht beunruhigen. Wenn die `Average`-Statistik oder p50 (Medianwert) jedoch einen anhaltenden starken Anstieg zeigt, sollten Sie das AWS-Servicestatus-Dashboard und Ihr Persönliches Servicestatus-Dashboard überprüfen, um mehr zu erfahren. Zu den möglichen Ursachen gehören die Größe des Elements in Ihrer Tabelle (ein Element mit 1 KB und ein Element mit 400 KB weisen unterschiedliche Latenzzeiten auf) oder die Größe der Abfrage (10 Elemente gegenüber 100 Elementen).

Die Perzentilmetriken (p99, p90 usw.) können Ihnen dabei helfen, Ihre Latenzverteilung besser zu verstehen. Zum Beispiel:

- p50 (Medianwert) gibt die typische Latenz für Ihren Workload an.
- p90 zeigt, dass 90 % der Anfragen schneller sind als dieser Wert.
- p99 hilft dabei, die Latenz im Schlimmstfall zu identifizieren, von der 1 % der Anforderungen betroffen sind.

Hohe p99-Werte bei normalen p50-Werten können auf sporadische Probleme hinweisen, die einen kleinen Teil der Anforderungen betreffen, wohingegen konstant erhöhte p50-Werte ggf. auf Leistungseinbußen hindeuten.

> **Note**
>
> Um benutzerdefinierte Perzentilwerte (z. B. p99,9) zu analysieren, können Sie das gewünschte Perzentil (z. B. p99,9) manuell in das CloudWatch-Metrikstatistikfeld eingeben. So können Sie Latenzverteilungen auswerten, die über die in der Dropdown-Liste aufgeführten Standardperzentilwerte hinausgehen.

Es ist mit gewissen Schwankungen bei den Latenzmetriken zu rechnen, insbesondere bei höheren Perzentilwerten. Diese sind auf DynamoDB-gesteuerte Hintergrundvorgänge zurückzuführen, die zur Aufrechterhaltung einer hohen Verfügbarkeit und Beständigkeit Ihrer in DynamoDB-Tabellen gespeicherten Daten beitragen, oder auf vorübergehende Infrastrukturprobleme.

Falls erforderlich, können Sie einen Supportfall beim AWS Support öffnen und weiterhin alle verfügbaren Fallback-Optionen für Ihre Anwendung (z. B. Leerung einer Region, wenn Sie über eine Architektur mit mehreren Regionen verfügen) gemäß Ihren Runbooks bewerten. Für langsame Anforderungen sollten Sie die Anforderungs-IDs protokollieren, damit Sie diese IDs AWS Support bereitstellen können, wenn Sie einen Supportfall öffnen.

Die Metrik `SuccessfulRequestLatency` misst nur die interne Latenz des DynamoDB-Service – clientseitige Aktivitäten und Netzwerklaufzeiten sind darin nicht enthalten. Wenn Sie mehr über die allgemeine Latenz für Aufrufe von Ihrem Client an den DynamoDB-Service erfahren möchten, können Sie die Protokollierung von Latenzmetriken im AWS-SDK aktivieren.

> **Note**
>
> Für die meisten Singleton-Operationen (Operationen, die sich auf ein einzelnes Element beziehen, indem der Wert des Primärschlüssels vollständig angegeben wird) stellt DynamoDB `Average SuccessfulRequestLatency` im einstelligen Millisekundenbereich bereit. Dieser Wert beinhaltet nicht den Transport-Overhead für den Aufrufer-Code, der auf den DynamoDB-Endpunkt zugreift. Bei Datenoperationen mit mehreren Elementen hängt die Latenz von Faktoren wie der Größe der Ergebnismenge, der Komplexität der zurückgegebenen Datenstrukturen und allen angewendeten Bedingungs- und Filterausdrücken ab. Bei wiederholten Operationen mit mehreren

Elementen für denselben Datensatz mit denselben Parametern stellt DynamoDB `Average SuccessfulRequestLatency` mit hoher Konsistenz bereit.

Ziehen Sie eine oder mehrere der folgenden Strategien in Erwägung, um die Latenz zu reduzieren:

- Verbindungen wiederverwenden: DynamoDB-Anforderungen werden über eine authentifizierte Sitzung gestellt, die standardmäßig HTTPS verwendet. Das Initiieren der Verbindung nimmt mehrere Roundtrips in Anspruch und dauert einige Zeit, sodass die Latenz der ersten Anforderung höher als bei nachfolgenden Anforderungen ist, die die Verbindung wiederverwenden. Anforderungen über eine bereits initialisierte Verbindung stellen die gleichbleibend niedrige Latenz von DynamoDB bereit. Um Latenz beim Aufbau neuer Verbindungen zu vermeiden, können Sie einen „Keep-Alive“-Mechanismus implementieren, indem Sie alle 30 Sekunden eine `GetItem`-Anforderung senden, wenn keine anderen Anforderungen gestellt werden.
- Letztendlich konsistente Lesevorgänge verwenden: Wenn Ihre Anwendung keine strikt konsistenten Lesevorgänge erfordert, sollten Sie die Verwendung der standardmäßigen letztendlich konsistenten Lesevorgänge in Betracht ziehen. Letztendlich konsistente Lesevorgänge sind kostengünstiger und können aus mehreren Availability Zones stammen, sodass eine Availability Zone ausgewählt werden kann, die sich am Standort des Anforderers befindet, wodurch die Latenz verringert wird. Weitere Informationen finden Sie unter DynamoDB-Lesekonsistenz.
- Request Hedging implementieren: Für Anforderungen mit sehr niedriger p99-Latenz sollten Sie die Implementierung von Request Hedging in Betracht ziehen. Beim Request Hedging gilt: Wenn die erste Anforderung nicht schnell genug beantwortet wird, wird eine zweite gleichwertige Anforderung gesendet – beide Anforderungen treten also gegeneinander an, wobei die mit der ersten Antwort gewinnt. Dadurch wird die Latenz verbessert, dies erfordert allerdings einige zusätzliche Anforderungen. Sie können entscheiden, wie lange Sie warten möchten, bevor die zweite Anforderung gesendet wird. Hedging ist für Lesevorgänge einfacher. Verwenden Sie für Schreibvorgänge eine zeitstempelbasierte Reihenfolge, um sicherzustellen, dass abgesicherte Anforderungen so behandelt werden, als ob sie zum Zeitpunkt des ersten Versuchs erfolgt wären, was Aktualisierungen in falscher Reihenfolge verhindert. Dieser Ansatz wurde in Timestamp writes for write hedging in Amazon DynamoDB erörtert.
- Anfrage-Timeout und Wiederholungsverhalten anpassen: Der Pfad von Ihrem Client zu DynamoDB durchläuft viele Komponenten, von denen jede auf Redundanz ausgelegt ist. Berücksichtigen Sie die folgenden Aspekte:
  - Ausfallsicherheit des Netzwerks
  - TCP-Paket-Timeouts

- Verteilte Architektur von DynamoDB

Das Standardverhalten des SDK ist für die meisten Anwendungen optimiert. Sie können jedoch eine Fail-Fast-Strategie implementieren und die Timeout-Einstellungen anpassen. Anforderungen, die deutlich länger als normal dauern, haben eine geringere Wahrscheinlichkeit, letztendlich erfolgreich zu sein. Durch schnelles Scheitern („Fail Fast“) und erneutes Versuchen können Sie auf einem anderen Weg zügig zum Erfolg gelangen. Dies ähnelt dem Request Hedging, die erste Anforderung wird jedoch beendet, anstatt ihr Fortsetzen zu erlauben.

Vermeiden Sie es, Timeout-Werte zu niedrig festzulegen. Zu niedrige Timeouts können zu kundenbedingten Verfügbarkeitsproblemen führen. So kann etwa ein Socket-Timeout von 50 Millisekunden bei Netzwerklatenzspitzen zu Verbindungsfehlern führen, zum Beispiel, wenn die Bandbreitenbeschränkungen von Amazon-EC2-Instances für Single-Flow-Datenverkehr erreicht werden. Wägen Sie die Vorteile niedrigerer Timeouts sorgfältig gegen potenzielle Risiken für die Anwendungsverfügbarkeit ab und ziehen Sie eine Absicherung (Hedging) kurzen Timeouts vor.

Eine hilfreiche Diskussion zu diesem Thema finden Sie unter Tuning AWS Java SDK HTTP request settings for latency-aware Amazon DynamoDB applications.

- Die Distanz zwischen dem Client und dem DynamoDB-Endpunkt verringern: Wenn Ihre Benutzer global verteilt sind, sollten Sie die Verwendung von Globale Tabellen: multiaktive, regionsübergreifende Replikation in Erwägung ziehen. Wenn Sie globale Tabellen verwenden, können Sie Ihre Tabelle in bestimmte AWS-Regionen replizieren, in denen die Tabellen verfügbar sein sollen. Sie können eine Kopie der Daten näher am Endbenutzer platzieren, um die Netzwerklatenz beim Lesen und Schreiben zu reduzieren. Weitere Informationen zur effektiven Verwendung globaler DynamoDB-Tabellen finden Sie unter Verwenden der globalen Amazon-DynamoDB-Tabellen in AWS Prescriptive Guidance.
- Caching verwenden: Wenn Ihr Datenverkehr leseintensiv ist, sollten Sie die Verwendung eines dieser Caching-Service in Betracht ziehen:
  - DynamoDB Accelerator (DAX): ein vollständig verwalteter In-Memory-Cache mit Hochverfügbarkeit für DynamoDB, der eine bis zu 10-fache Leistungssteigerung von Millisekunden zu Mikrosekunden bietet, selbst bei Millionen von Anforderungen pro Sekunde. Weitere Informationen zu DAX finden Sie unter In-Memory-Beschleunigung mit DynamoDB Accelerator (DAX):
  - Amazon ElastiCache: ein vollständig verwalteter In-Memory-Caching-Service, der in DynamoDB integriert werden kann, um die Leseleistung mithilfe des Cache-Aside-Musters zu verbessern.

Weitere Informationen finden Sie unter Integrating Amazon DynamoDB and Amazon ElastiCache by using read-through caching in AWS Prescriptive Guidance.

Beheben von Drosselungsereignissen in Amazon DynamoDB

DynamoDB implementiert Drosselungen für zwei Hauptzwecke: Aufrechterhaltung der allgemeinen Serviceleistung und Kostenkontrolle. Die Drosselung dient entweder als absichtliche Schutzmaßnahme, um Leistungseinbußen zu verhindern, wenn die Nutzungsraten die Kapazität überschreiten, oder als Kostenkontrollmechanismus, wenn Sie die maximalen Durchsatz- oder Service-Quota-Limits erreichen. Wenn eine Drosselung auftritt, gibt DynamoDB spezifische Ausnahmen mit detaillierten Informationen darüber zurück, warum die Anforderung gedrosselt wurde und welche Ressource betroffen war. Jeder Grund für die Drosselung entspricht bestimmten CloudWatch Kennzahlen, die zusätzliche Einblicke in die Häufigkeit und Muster von Drosselungsereignissen bieten.

Das folgende Diagramm veranschaulicht die vier Hauptszenarien, in denen DynamoDB Drosselung als Schutzmaßnahme implementiert:

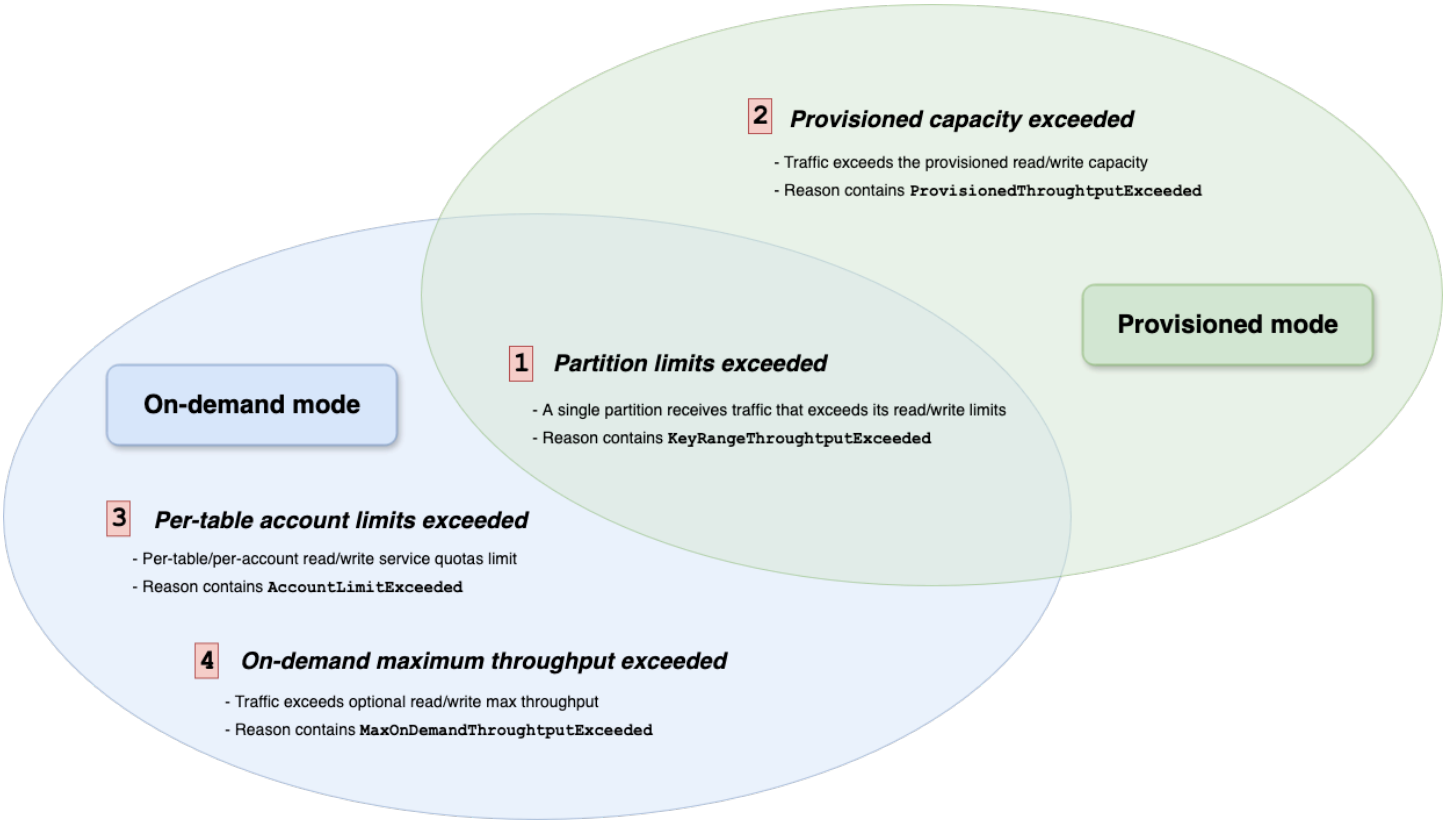
1. Der Durchsatz im Schlüsselbereich wurde überschritten (in beiden Modi):

   Der Verbrauch, der auf bestimmte Partitionen gerichtet ist, überschreitet die internen Durchsatzlimits auf Partitionsebene.
2. Der bereitgestellte Durchsatz wurde überschritten (im Bereitstellungsmodus):

   Der Verbrauch übersteigt die bereitgestellten Kapazitätseinheiten (Lesen oder Schreiben), die für eine Tabelle oder einen globalen sekundären Index (GSI) konfiguriert sind.
3. Die Service Quotas auf Kontoebene wurden überschritten (im On-Demand-Modus):

   Der Verbrauch führt dazu, dass eine Tabelle oder GSI die auf Kontoebene festgelegten Servicekontingente für den Durchsatz pro Tabelle in der aktuellen Region überschreitet. read/write AWS Diese Quotas dienen als Backstop-Schutzmaßnahmen und können erhöht werden.
4. Der maximale On-Demand-Durchsatz wurde überschritten (im On-Demand-Modus):

   Der Verbrauch überschreitet die konfigurierten maximalen Durchsatzlimits, die für eine Tabelle oder einen globalen sekundären Index festgelegt wurden. Dies sind Grenzwerte, die Sie speziell für Zwecke der Kostenkontrolle konfigurieren.



Dieser Leitfaden soll Ihnen dabei helfen, die Drosselung in DynamoDB zu verstehen und damit zu arbeiten. Zunächst helfen wir Ihnen mithilfe eines Diagnose-Frameworks, die spezifische Art der Drosselung zu identifizieren, die sich auf Ihren Workload auswirkt.

Anschließend bietet der Abschnitt Lösungsleitfaden spezifische Anleitungen für jedes Drosselungsszenario, einschließlich CloudWatch Messwerte zur Überwachung zur Erkennung und Analyse sowie empfohlene Optimierungsschritte. Mit diesem strukturierten Ansatz können Sie die Ursache der Drosselung besser diagnostizieren und die entsprechende Lösung implementieren, um sicherzustellen, dass Ihre DynamoDB-Tabellen effizient funktionieren.

Folgen Sie zunächst the section called “Drosselungsdiagnose”, um zu erfahren, wie Sie feststellen können, welche Art von Drosselung sich auf Ihren Workload auswirkt und wie Sie die empfohlene Lösungsstrategie umsetzen können.

Themen

- Drosselungsdiagnose
- Lösungsleitfaden für Drosselungen in DynamoDB
- Grundlegendes zu Schreibdrosselung und Gegendruck im globalen sekundären Index (GSI) in DynamoDB

- CloudWatch-Drosselungsmetriken

Drosselungsdiagnose

Wenn in Ihrer Anwendung eine Drosselung auftritt, stellt DynamoDB detaillierte Ausnahmeinformationen und gezielte CloudWatch Metriken bereit, um Sie bei der Diagnose dieser Ereignisse zu unterstützen.

In diesem Abschnitt wird ein systematischer Ansatz zum Verständnis von Drosselungsereignissen in Ihren DynamoDB-Anwendungen vorgestellt. Es zeigt, wie Sie Drosselungsausnahmen interpretieren, sie mit CloudWatch Metriken korrelieren, um tiefere Einblicke zu erhalten, und zu verstehen, welche Änderungen die Drosselung in Ihren DynamoDB-Anwendungen reduzieren würden.

Übersicht über Drosselungsausnahmen

Wenn DynamoDB eine Anfrage drosselt, gibt es spezifische Ausnahmen mit detaillierten Diagnoseinformationen zurück. In Java gehören dazu beispielsweise `ProvisionedThroughputExceededException`, `RequestLimitExceeded` oder `ThrottlingException`.

Jede Ausnahme umfasst `ThrottlingReasons`, eine Reihe einzelner `ThrottlingReason` mit zwei Schlüsselfeldern, mit deren Hilfe Sie Drosselungen identifizieren und verstehen können:

- ein Grund – ein verkettetes Feld mit dem Format `<ResourceType><OperationType><LimitType>`
- ein Amazon-Ressourcenname (ARN) – der Amazon-Ressourcenname (ARN) der betroffenen Tabelle oder des betroffenen Indexes

Das Feld mit dem Grund folgt einem konsistenten Muster, das Ihnen hilft, das Ereignis genau zu verstehen:

- ResourceType(Was wird gedrosselt): oder `Table` `Index`
- OperationType(Welche Art von Operation): oder `Read` `Write`
- LimitType(Warum die Drosselung aufgetreten ist):
  - `KeyRangeThroughputExceeded`: Dies tritt auf, wenn eine bestimmte Partition, die Ihrer Tabelle oder Ihrem Index zugrunde liegt, Lese- oder Schreibkapazität verbraucht hat, die die internen Durchsatzlimits pro Partition überschreitet.

- `ProvisionedThroughputExceeded`: Dies passiert bei einer bereitgestellten Tabelle oder einem bereitgestellten globalen sekundären Index, wenn die Lese- oder Schreibverbrauchsrate die bereitgestellte Menge überschritten hat.
- `AccountLimitExceeded`: Dies passiert bei einer On-Demand-Tabelle oder einem On-Demand-Index, wenn die Lese- oder Schreibverbrauchsrate die auf Kontoebene festgelegte maximale Verbrauchsrate für eine Tabelle und ihre Indizes überschritten hat. Sie können eine Erhöhung dieses Kontingents anfordern.
- `MaxOnDemandThroughputExceeded`: Dies passiert bei einer On-Demand-Tabelle oder einem On-Demand-Index, wenn die Lese- oder Schreibverbrauchsrate die vom Nutzer konfigurierte maximale Verbrauchsrate für die Tabelle und den Index überschritten hat. Sie können diesen Wert selbst auf einen beliebigen Wert bis zum Kontolimit erhöhen oder auf -1 setzen, um anzugeben, dass kein vom Nutzer festgelegtes Limit vorhanden ist.

Der Ressourcenname (ARN) gibt genau an, welche Tabelle oder welcher Index gedrosselt wird:

- Für Tabellen: `arn:aws:dynamodb:[region]:[account-id]:table/[table-name]`
- Für Indizes: `arn:aws:dynamodb:[region]:[account-id]:table/[table-name]/index/[index-name]`

Beispiele für vollständige Drosselungsgründe:

- `TableReadProvisionedThroughputExceeded`
- `IndexWriteAccountLimitExceeded`

Damit können Sie genau ermitteln, welche Ressource gedrosselt wurde, durch welche Art von Vorgang die Drosselung verursacht wurde und warum die Drosselung erfolgt ist.

Beispielausnahmen

Beispiel 1: Überschreitung der bereitgestellten Kapazität eines GSI

```
{
    "ThrottlingReasons": [
        {
            "reason": "IndexWriteProvisionedThroughputExceeded",
            "resource": "arn:aws:dynamodb:us-west-2:123456789012:table/CustomerOrders/
index/OrderDateIndex"
```

```
        }
    ],
    "awsErrorDetails": {
        "errorCode": "ProvisionedThroughputExceeded",
        "errorMessage": "The level of configured provisioned throughput for the index
 was exceeded",
        "serviceName": "DynamoDB",
        "sdkHttpResponse": {
            "statusText": "Bad Request",
            "statusCode": 400
        }
    }
}
```

In diesem Beispiel erhält die Anwendung eine `ProvisionedThroughputExceededException` mit dem Grund `IndexWriteProvisionedThroughputExceeded`. Schreibvorgänge in den `OrderDateIndex` werden gedrosselt, weil der Schreibverbrauch die konfigurierte bereitgestellte Schreibkapazität des GSI überschritten hat.

Beispiel 2: Überschreitung des maximalen On-Demand-Durchsatzes

```
{
    "ThrottlingReasons": [
        {
            "reason": "TableReadMaxOnDemandThroughputExceeded",
            "resource": "arn:aws:dynamodb:us-east-1:123456789012:table/UserSessions"
        }
    ],
    "awsErrorDetails": {
        "errorMessage": "Throughput exceeds the maximum OnDemandThroughput configured
 on table or index",
        "errorCode": "ThrottlingException",
        "serviceName": "DynamoDB",
        "sdkHttpResponse": {
            "statusText": "Bad Request",
            "statusCode": 400
        }
    }
}
```

In diesem Beispiel werden Lesevorgänge aus der `UserSessions`-Tabelle gedrosselt, weil sie den für die Tabelle konfigurierten maximalen On-Demand-Durchsatz überschreiten.

Diagnose-Framework für DynamoDB-Drosselungen

Wenn in Ihrer Anwendung eine Drosselung auftritt, gehen Sie wie folgt vor, um das Problem zu diagnostizieren und zu beheben.

Schritt 1 – Analysieren der **`ThrottlingReason`**-Details

1. Überprüfen Sie das Feld „reason“, um den spezifischen Grund für die Drosselung zu ermitteln. Zu den Angaben gehören die Art der gedrosselten Ressource (Tabelle oder Index), die Art des Vorgangs, der das Drosselungsereignis verursacht hat (Lesen oder Schreiben), und die Art des Limits, das überschritten wurde (Partition, bereitgestellter Durchsatz, Kontolimit).
2. Überprüfen Sie das Feld resourceArn, um herauszufinden, welche Ressource (Tabelle oder GSI) gedrosselt wird.
3. Mithilfe dieser kombinierten Informationen können Sie den vollständigen Kontext des Drosselungsproblems bestimmen.

   Nehmen Sie zum Beispiel dieses Szenario, in dem Sie die folgende Ausnahme `ProvisionedThroughputExceededException` mit dem Drosselungsgrund `TableWriteKeyRangeThroughputExceeded` erhalten. Der betroffene resourceARN ist `arn:aws:dynamodb:us-west-2:123456789012:table/CustomerOrders`.

   Diese Kombination informiert Sie darüber, dass Schreibvorgänge in Ihre `CustomerOrders`-Tabelle gedrosselt werden. Die Drosselung erfolgt auf Partitionsebene (nicht auf Tabellenebene, was als `TableWriteProvisionedThroughputExceeded` angezeigt werden würde). Die Ursache des Problems besteht darin, dass die maximale Durchsatzkapazität für einen bestimmten Schlüsselwert oder -bereich einer Partition überschritten wurde. Dies deutet auf ein Problem mit einer Hot-Partition hin.

   Wenn Sie diesen Zusammenhang zwischen den Ausnahmeelementen verstehen, können Sie geeignete Abhilfemaßnahmen ergreifen. In diesem Fall muss das Problem mit der Hot-Partition behoben werden, anstatt die bereitgestellte Gesamtkapazität der Tabelle zu erhöhen.

Schritt 2 — Identifizieren und analysieren Sie die zugehörigen Metriken CloudWatch

1. Identifizieren Sie Ihre Metriken: Jeder Drosselungsgrund in DynamoDB entspricht direkt bestimmten CloudWatch Metriken, die Sie überwachen können, um Drosselungsereignisse zu verfolgen und zu analysieren. Sie können die entsprechenden CloudWatch Metriknamen systematisch aus dem Grund der Drosselung ableiten.

2. Ordnen Sie anhand dieser Referenztabelle den Grund für die Drosselung den entsprechenden CloudWatch Kennzahlen zu:

   Vollständige Referenz zu den Gründen und Kennzahlen für die Drosselung CloudWatch

Kategorie	Drosselungsgrund	Primäre Metriken CloudWatch
Überschreitung der bereitgestellten Kapazität	TableReadProvisionedThroughputExceeded	ReadProvisionedThroughputThrottleEvents
	TableWriteProvisionedThroughputExceeded	WriteProvisionedThroughputThrottleEvents
	IndexReadProvisionedThroughputExceeded	ReadProvisionedThroughputThrottleEvents (GSI)
	IndexWriteProvisionedThroughputExceeded	WriteProvisionedThroughputThrottleEvents (GSI)
Überschreitung der Partitionslimits	TableReadKeyRangeThroughputExceeded	ReadKeyRangeThroughputThrottleEvents
	TableWriteKeyRangeThroughputExceeded	WriteKeyRangeThroughputThrottleEvents
	IndexReadKeyRangeThroughputExceeded	ReadKeyRangeThroughputThrottleEvents (GSI)
	IndexWriteKeyRangeThroughputExceeded	WriteKeyRangeThroughputThrottleEvents (GSI)

Kategorie	Drosselungsgrund	Primäre Metriken CloudWatch
Überschreitung des maximalen On-Demand-Durchsatzes	TableReadMaxOnDemandThroughputExceeded	ReadMaxOnDemandThroughputThrottleEvents
	TableWriteMaxOnDemandThroughputExceeded	WriteMaxOnDemandThroughputThrottleEvents
	IndexReadMaxOnDemandThroughputExceeded	ReadMaxOnDemandThroughputThrottleEvents (GSI)
	IndexWriteMaxOnDemandThroughputExceeded	WriteMaxOnDemandThroughputThrottleEvents (GSI)
Überschreitung der Kontolimits	TableReadAccountLimitExceeded	ReadAccountLimitThrottleEvents
	TableWriteAccountLimitExceeded	WriteAccountLimitThrottleEvents
	IndexReadAccountLimitExceeded	ReadAccountLimitThrottleEvents (GSIs)
	IndexWriteAccountLimitExceeded	WriteAccountLimitThrottleEvents (GSIs)

Wenn Sie beispielsweise mindestens die Metrik für den spezifischen Index erhalten `IndexWriteProvisionedThroughputExceeded` haben, sollten Sie die `WriteProvisionedThroughputThrottleEvents` CloudWatch Metrik überwachen, der in der `ResourceArn` identifiziert wurde.

3. Überwachen Sie diese Kennzahlen CloudWatch , um die Häufigkeit und den Zeitpunkt der Drosselung zu verstehen, zwischen Lese- und Schreibdrosselung zu unterscheiden, Zeitmuster zu identifizieren, in denen die Drosselung zunimmt, und Ihre Kapazitätsauslastungstrends zu verfolgen.

   DynamoDB veröffentlicht detaillierte Metriken für jede Tabelle und jeden globalen sekundären Index. Die Metriken (`ReadThrottleEvents`, `WriteThrottleEvents` und`ThrottledRequests`) fassen alle Drosselungsereignisse in Ihrer Tabelle und ihren Indizes zusammen.

Schritt 3: Identifizieren Sie mithilfe von Contributor Insights (für partitionsbedingte Drosselung) Ihre gedrosselten Schlüssel und Ihre hohen Zugriffsraten CloudWatch

Wenn Sie in Schritt 1 partitionsbezogene Probleme festgestellt haben (z. B. `KeyRangeThroughputExceeded` Fehler), kann CloudWatch Contributor Insights for DynamoDB Ihnen dabei helfen, zu diagnostizieren, welche spezifischen Schlüssel Ihren Traffic antreiben und welche Drosselungsereignisse in Ihrer Tabelle oder Ihrem Index auftreten.

1. Aktivieren Sie CloudWatch Contributor Insights für Ihre gedrosselte Tabelle oder Ihren Index auf der Grundlage Ihrer. `ResourceARN`

   Sie können den Modus Gedrosselte Schlüssel wählen, um sich ausschließlich auf die am stärksten gedrosselten Schlüssel zu konzentrieren. Dieser Modus eignet sich ideal für die kontinuierliche Überwachung, da er nur Ereignisse verarbeitet, bei denen eine Drosselung erfolgt. Alternativ hilft Ihnen der Modus für Schlüssel mit Zugriffen und Drosselungen dabei, nach Mustern in den am häufigsten aufgerufenen Schlüsseln zu suchen.
2. Analysieren Sie die Berichte, um problematische Muster zu identifizieren. Suchen Sie nach Schlüsseln mit unverhältnismäßig hohen Zugriffs- oder Drosselungsraten und setzen Sie Drosselung und Datenverkehrsmuster miteinander in Verbindung. Sie können integrierte Dashboards erstellen, die Contributor Insights-Diagramme und CloudWatch DynamoDB-Metriken kombinieren.

Ausführliche Informationen zur Aktivierung und Verwendung von CloudWatch Contributor Insights finden Sie unter CloudWatch Contributor Insights for DynamoDB verwenden.

Schritt 4 – Ermitteln der passenden Lösung

Nachdem Sie die konkrete Ursache der Drosselung bestimmt haben, implementieren Sie die empfohlene Lösung für Ihren spezifischen Kontext. Die geeignete Lösung hängt von mehreren Faktoren ab. Dazu gehören das Drosselungsszenario, der Kapazitätsmodus der Tabelle, Entscheidungen zum Tabellen- und Schlüsseldesign, Zugriffsmuster und Abfrageeffizienz, die Konfiguration des globalen sekundären Indexes sowie die allgemeine Systemarchitektur und die Integrationspunkte.

Detaillierte Lösungen für Ihre spezifischen Drosselungsszenarien finden Sie im Abschnitt the section called “Lösungsleitfaden”. Diese Ressource bietet gezielte Abhilfemaßnahmen, die auf Ihren speziellen Drosselungsgrund und die Konfiguration des Kapazitätsmodus zugeschnitten sind.

Schritt 5 – Überwachen des Fortschritts

1. Verfolgen Sie Ihre CloudWatch Metriken, die Ihrem Drosselungsszenario entsprechen.
2. Prüfen Sie, ob Ihre Abhilfemaßnahmen wirksam sind und ein Rückgang der Drosselungsereignisse zu beobachten ist.

Lösungsleitfaden für Drosselungen in DynamoDB

Dieser Abschnitt bietet gezielte Lösungsansätze für jeden spezifischen Drosselungsgrund, den DynamoDB zurückgeben kann. Jeder Eintrag enthält Vorschläge für Lösungsansätze, die auf bewährten Verfahren basieren, und entsprechende CloudWatch Kennzahlen, die überwacht werden müssen.

DynamoDB implementiert 16 verschiedene Drosselungsgründe in vier Hauptkategorien. Verwenden Sie die Drosselungsgründe in der Anwendungsausnahme, um direkt zu den entsprechenden Anleitungen zu gelangen.

Überschreitung des Durchsatzes in Schlüsselbereichen (Hot-Partitionen)

Diese Drosselungsgründe treten auf, wenn einzelne Partitionen ihre Durchsatzlimits überschreiten. Dies betrifft sowohl den bereitgestellten als auch den On-Demand-Modus:

- TableReadKeyRangeThroughputExceeded
- TableWriteKeyRangeThroughputExceeded
- IndexReadKeyRangeThroughputExceeded
- IndexWriteKeyRangeThroughputExceeded

Überschreitung des bereitgestellten Durchsatzes

Diese Drosselungsgründe treten auf, wenn die Nutzungsraten die bereitgestellten Kapazitätslimits im bereitgestellten Modus überschreiten:

- TableReadProvisionedThroughputExceeded
- TableWriteProvisionedThroughputExceeded
- IndexReadProvisionedThroughputExceeded
- IndexWriteProvisionedThroughputExceeded

Überschreitung der Kontolimits

Diese Drosselungsgründe treten auf, wenn die Nutzungsraten die Durchsatzquoten auf Kontoebene in Ihrer Region überschreiten: AWS

- TableReadAccountLimitExceeded
- TableWriteAccountLimitExceeded
- IndexReadAccountLimitExceeded
- IndexWriteAccountLimitExceeded

Überschreitung des maximalen On-Demand-Durchsatzes

Diese Drosselungsgründe treten auf, wenn die Nutzungsraten die konfigurierten maximalen Durchsatzlimits im On-Demand-Modus überschreiten:

- TableReadMaxOnDemandThroughputExceeded
- TableWriteMaxOnDemandThroughputExceeded
- IndexReadMaxOnDemandThroughputExceeded
- IndexWriteMaxOnDemandThroughputExceeded

1- Überschreitung des Durchsatzes in Schlüsselbereichen (Hot-Partitionen)

Amazon DynamoDB erzwingt bestimmte Durchsatzlimits auf Partitionsebene für Tabellen und den globalen sekundären Index (GSI). Jede Partition hat eine maximale Anzahl von Lesekapazitätseinheiten (RCUs) und Schreibkapazitätseinheiten (WCUs) pro Sekunde. Wenn Partitionen konzentrierten Datenverkehr erhalten, der diese Limits überschreitet, kommt es zu

einer Drosselung, während andere Operationen möglicherweise nicht ausgelastet sind. Dadurch entstehen „Hot-Partitionen“, also Partitionen, die stark beansprucht sind. Die Drosselung auf Partitionsebene von DynamoDB erfolgt unabhängig von Lese- und Schreibvorgängen – eine Partition kann Lesevorgänge drosseln, während Schreibvorgänge normal fortgesetzt werden, oder umgekehrt. Diese Drosselung kann auch dann auftreten, wenn die Tabelle oder der GSI über eine ausreichende Gesamtkapazität verfügt. Weitere Hilferessourcen:

- Informationen zu DynamoDB-Partitionslimits und einem effektiven Design von Partitionsschlüsseln zur Verhinderung von Hot-Partitionen finden Sie unter Bewährte Methoden für das Entwerfen von Partitionsschlüsseln und ihre effektive Verwendung in DynamoDB.
- Informationen zu allgemeinen Partitionskonzepten und zur Datenverteilung finden Sie unter Partitionen in DynamoDB.
- Weitere Hinweise und reale Szenarien für die Verwaltung von Partitionsschlüsseln und Durchsatz finden Sie unter the section called “Weitere Ressourcen”.

Wenn einzelne Partitionen ihre Durchsatzlimits überschreiten, gibt DynamoDB in der Drosselungsausnahme einen Drosselungsgrund der Art `KeyRangeThroughputExceeded` zurück. Die Informationen geben an, dass auf einer Partition viel Datenverkehr herrscht. Außerdem ist ersichtlich, welcher Operationstyp (Lesen oder Schreiben) das Problem verursacht.

Durchsatz des Schlüsselbereichs übersteigt Abhilfemaßnahmen

Dieser Abschnitt bietet Anleitungen zur Behebung von Drosselungsszenarien auf Partitionsebene. Bevor Sie diese Anleitung verwenden, sollten Sie die spezifischen Drosselungsgründe anhand der Ausnahmebehandlung Ihrer Anwendung identifizieren und den Amazon-Ressourcennamen (ARN) der betroffenen Ressource ermitteln. Informationen dazu, wie Sie die Gründe für die Drosselung abrufen und gedrosselte Ressourcen identifizieren, finden Sie unter the section called “Diagnose-Framework”.

Bevor Sie sich eingehender mit bestimmten Drosselungsszenarien befassen, überprüfen Sie zuerst, ob das Problem automatisch behoben wird:

- DynamoDB passt sich durch seinen automatischen split-for-heat Mechanismus häufig an heiße Partitionen an. Wenn Sie Drosselungsereignisse sehen, die nach kurzer Zeit aufhören, hat sich Ihre Tabelle möglicherweise bereits angepasst, indem die Hot-Partition aufgeteilt wurde. Wenn Partitionen aufgeteilt werden, verarbeitet jede neue Partition einen kleineren Teil des Keyspace, sodass die Last gleichmäßiger verteilt werden kann. In vielen Fällen ist keine weitere Aktion erforderlich, da DynamoDB das Problem automatisch behoben hat.

Weitere Hinweise zu diesem split-for-heatMechanismus finden Sie unter. the section called “Weitere Ressourcen”

Falls die Drosselung weiterhin besteht, finden Sie in den unten aufgeführten spezifischen Drosselungsszenarien gezielte Abhilfemaßnahmen:

- the section called “TableReadKeyRangeThroughputExceeded”
- the section called “TableWriteKeyRangeThroughputExceeded”
- the section called “IndexReadKeyRangeThroughputExceeded”
- the section called “IndexWriteKeyRangeThroughputExceeded”

TableReadKeyRangeThroughputExceeded

Wenn dies auftritt

Die Verbrauchsrate einer oder mehrerer Partitionen in Ihrer DynamoDB-Tabelle überschreitet das Limit für den Lesedurchsatz der Partition. Diese Drosselung erfolgt unabhängig von der bereitgestellten Gesamtkapazität Ihrer Tabelle und betrifft sowohl bereitgestellte als auch On-Demand-Tabellen. Sie können die CloudWatch Messwerte überwachenthe section called “Allgemeine Diagnose und Überwachung”, um Ihr Drosselungsereignis zu analysieren.

Mögliche Abhilfemaßnahmen

Erwägen Sie die folgenden Schritte, um die Drosselungsereignisse zu beheben:

Für den Modus „Bereitgestellt“ und „On-Demand“:

- Vorwärmkapazität: Wenn die Drosselung weiterhin besteht, überprüfen Sie, ob Ihre Tabelle durch ihre the section called “Warmdurchsatz”-Kapazität begrenzt ist. Verwenden Sie Warmdurchsatz oder erhöhen Sie die bereitgestellte Lesekapazität vor einem erwarteten Anstieg des Datenverkehrs. Durch die Erhöhung des Warmdurchsatzes ist Ihre Tabelle besser in der Lage, plötzliche Datenverkehrsspitzen zu bewältigen, bevor es zu einer Drosselung kommt. Im Laufe der Zeit kann DynamoDB ausgelastete Partitionen auf der Grundlage von beobachteten Verwendungsmustern aufteilen, wenn sich der tatsächliche Durchsatz konstant dem Warmdurchsatz nähert.
- Identifizieren von Hotkeys: Wenn die Tabelle das Problem nicht automatisch gelöst hat und Ihr Warmdurchsatz hoch ist oder eine Erhöhung des Durchsatzes nicht geholfen hat, müssen

Sie bestimmte Hotkeys identifizieren. Verwenden Sie the section called “Identifizieren von Tastenkombinationen mithilfe von Contributor Insights CloudWatch”, um festzustellen, ob bestimmte Partitions-Schlüsselwerte stark ausgelastet („hot“) sind. Dies ist ein erster Schritt, um Ihre Abhilfemaßnahmen effektiv auszurichten. Beachten Sie, dass die Identifizierung möglicherweise nicht immer einfach ist, insbesondere bei rollierenden Hot-Partitionen (bei denen verschiedene Partitionen im Laufe der Zeit ausgelastet werden) oder wenn die Drosselung durch Operationen wie Scans ausgelöst wird. Für diese komplexen Szenarien müssen Sie ggf. die Zugriffsmuster Ihrer Anwendung analysieren und sie mit dem Zeitpunkt der Drosselung korrelieren.

- Je nach Anwendungsfall sollten Sie eventuell konsistente Lesevorgänge in Betracht ziehen: Wechseln Sie von stark konsistenten Lesevorgängen zu eventuell konsistenten Lesevorgängen, die nur die Hälfte verbrauchen RCUs und Ihre effektive Lesekapazität sofort verdoppeln können. Bewährte Methoden zur Implementierung von letztendlich konsistenten Lesevorgängen zur Reduzierung des Lesekapazitätsverbrauchs finden Sie unter the section called “DynamoDB-Lesekonsistenz”.
- Verbessern des Designs von Partitionsschlüsseln: Als langfristige Lösung sollten Sie das the section called “Verbessern des Designs von Partitionsschlüsseln” erwägen, um den Zugriff gleichmäßiger auf die Partitionen zu verteilen. Dieser Ansatz bietet oft die umfassendste Lösung für Probleme mit Hot-Partitionen, da die eigentliche Ursache behoben wird. Er erfordert jedoch eine sorgfältige Planung, da er mit erheblichen Migrationsherausforderungen verbunden ist.

TableWriteKeyRangeThroughputExceeded

Wenn dies auftritt

Die Verbrauchsrate einer oder mehrerer Partitionen in Ihrer DynamoDB-Tabelle überschreitet das Limit für den Schreibdurchsatz der Partition. Diese Drosselung erfolgt unabhängig von der bereitgestellten Gesamtkapazität Ihrer Tabelle und betrifft sowohl bereitgestellte als auch On-Demand-Tabellen. Sie können die CloudWatch Messwerte überwachenthe section called “Allgemeine Diagnose und Überwachung”, um Ihr Drosselungsereignis zu analysieren.

Mögliche Abhilfemaßnahmen

Erwägen Sie die folgenden Schritte, um die Drosselungsereignisse zu beheben:

Für den Modus „Bereitgestellt“ und „On-Demand“:

- Vorwärmkapazität: Wenn die Drosselung weiterhin besteht, überprüfen Sie, ob Ihre Tabelle durch ihre Grundlegendes zum Warmdurchsatz von DynamoDB-Kapazität begrenzt ist. Verwenden

Sie Warmdurchsatz oder erhöhen Sie die bereitgestellte Schreibkapazität vor einem erwarteten Anstieg des Datenverkehrs. Durch die Erhöhung des Warmdurchsatzes ist Ihre Tabelle besser in der Lage, plötzliche Datenverkehrsspitzen zu bewältigen, bevor es zu einer Drosselung kommt. Im Laufe der Zeit kann DynamoDB ausgelastete Partitionen auf der Grundlage von beobachteten Verwendungsmustern aufteilen, wenn sich der tatsächliche Durchsatz konstant dem Warmdurchsatz nähert.

- Identifizieren von Hotkeys: Wenn die Tabelle das Problem nicht automatisch gelöst hat und Ihr Warmdurchsatz hoch ist oder eine Erhöhung des Durchsatzes nicht geholfen hat, müssen Sie bestimmte Hotkeys identifizieren. Verwenden Sie the section called “Identifizieren von Tastenkombinationen mithilfe von Contributor Insights CloudWatch”, um festzustellen, ob bestimmte Partitions-Schlüsselwerte stark ausgelastet („hot“) sind. Dies ist ein erster Schritt, um Ihre Abhilfemaßnahmen effektiv auszurichten. Beachten Sie die folgenden gängigen Muster:
  - Wenn Sie feststellen, dass derselbe Partitionsschlüssel häufig in Ihren Drosselungsdaten vorkommt, deutet dies auf einen konzentrierten Hotkey hin.
  - Wenn Sie keine wiederholten Schlüssel sehen, die Daten aber in geordneter Weise schreiben (z. B. als sequenzielle Zeitstempel oder scanbasierte Operationen, die der Reihenfolg der Keyspaces folgen), haben Sie wahrscheinlich rollierende Hot-Partitionen, bei denen verschiedene Schlüssel im Laufe der Zeit stark beansprucht werden, wenn Ihre Schreibvorgänge den Keyspace durchlaufen.

  Beachten Sie, dass Schreibdrosselung auch bei Operationen wie `BatchWriteItem` oder bei Transaktionen auftreten können, die sich auf mehrere Elemente gleichzeitig auswirken. Wenn einzelne Elemente innerhalb einer `BatchWriteItem`-Anforderung gedrosselt werden, überträgt DynamoDB diese Drosselungsfehler nicht an den Anwendungscode. Stattdessen gibt DynamoDB Informationen über die unverarbeiteten Elemente in der Antwort zurück, die Ihre Anwendung durch einen erneuten Versuch verarbeiten muss. Bei Transaktionen schlägt die gesamte Operation mit einer `TransactionCanceledException` fehl, wenn bei einem Element eine Drosselung auftritt. Bei solch komplexen Szenarien müssen Sie möglicherweise die Schreibmuster Ihrer Anwendung und die Datenerfassungsworkflows analysieren, sie mit dem Zeitpunkt von Drosselungsereignissen korrelieren und geeignete Strategien zu Wiederholungsversuchen implementieren.
- Verbessern des Designs von Partitionsschlüsseln: Als langfristige Lösung sollten Sie das the section called “Verbessern des Designs von Partitionsschlüsseln” erwägen, um den Zugriff gleichmäßiger auf die Partitionen zu verteilen. Dieser Ansatz bietet oft die umfassendste Lösung für Probleme mit Hot-Partitionen, da die eigentliche Ursache behoben wird. Er erfordert jedoch eine sorgfältige Planung, da er mit erheblichen Migrationsherausforderungen verbunden ist.

IndexReadKeyRangeThroughputExceeded

Wenn dies auftritt

Die Verbrauchsrate einer oder mehrerer Partitionen in Ihrem DynamoDB-GSI überschreitet das Limit für den Lesedurchsatz der Partition. Diese Drosselung erfolgt unabhängig von der bereitgestellten Gesamtkapazität Ihres GSI und betrifft sowohl bereitgestellte als auch On-Demand-Tabellen. Sie können die CloudWatch Messwerte überwachen, um Ihr the section called “Allgemeine Diagnose und Überwachung” Drosselungsereignis zu analysieren.

Mögliche Abhilfemaßnahmen

Erwägen Sie die folgenden Schritte, um die Drosselungsereignisse zu beheben:

- Vorwärmkapazität: Wenn die Drosselung weiterhin besteht, überprüfen Sie, ob Ihr GSI durch seine Grundlegendes zum Warmdurchsatz von DynamoDB-Kapazität begrenzt ist. Verwenden Sie Warmdurchsatz oder erhöhen Sie die bereitgestellte Lesekapazität vor einem erwarteten Anstieg des Datenverkehrs. Durch die Erhöhung des Warmdurchsatzes ist Ihr GSI besser in der Lage, plötzliche Datenverkehrsspitzen zu bewältigen, bevor es zu einer Drosselung kommt. Im Laufe der Zeit kann DynamoDB ausgelastete Partitionen auf der Grundlage von beobachteten Verwendungsmustern aufteilen, wenn sich der tatsächliche Durchsatz konstant dem Warmdurchsatz nähert.
- Hotkeys identifizieren: Wenn der GSI das Problem nicht automatisch gelöst hat und Ihr Warmdurchsatz hoch ist oder eine Erhöhung des Durchsatzes nicht geholfen hat, müssen Sie bestimmte Hotkeys identifizieren. Verwenden Sie the section called “Identifizieren von Tastenkombinationen mithilfe von Contributor Insights CloudWatch”, um festzustellen, ob bestimmte Partitions-Schlüsselwerte stark ausgelastet („hot“) sind. Dies ist ein erster Schritt, um Ihre Abhilfemaßnahmen effektiv auszurichten. Beachten Sie GSIs, dass sich die Verteilung der Partitionsschlüssel erheblich von Ihrer Basistabelle unterscheiden kann, wodurch unterschiedliche Tastenkombinationen entstehen.
- Neugestalten von GSI-Partitionsschlüsseln: Überlegen Sie, ob Ihr GSI-Schlüsseldesign möglicherweise künstliche kritische Punkte (wie Status-Flags, reine Datumsschlüssel oder boolesche Attribute) erzeugt, die Lesevorgänge auf eine kleine Anzahl von Partitionen konzentrieren. Erwägen Sie die Verwendung zusammengesetzter Schlüssel, die das Attribut mit niedriger Kardinalität mit einem Attribut mit hoher Kardinalität kombinieren (z. B. „ACTIVE#customer123" statt nur „ACTIVE“), oder wenden Sie Techniken wie unter the section called “Schreib-Sharding” beschrieben auf die Basistabellenelemente an, die sich auf die GSI-Verteilung auswirken, um Schreibvorgänge auf mehrere Partitionen zu verteilen. Für die

Abfrage von Shard-Daten ist zwar zusätzliche Anwendungslogik erforderlich, um die Ergebnisse zu aggregieren, dieser Ansatz verhindert Drosselungen jedoch, indem die Zugriffsmuster gleichmäßiger verteilt werden.

IndexWriteKeyRangeThroughputExceeded

Wenn dies auftritt

Die Verbrauchsrate einer oder mehrerer Partitionen in Ihrem DynamoDB-GSI überschreitet das Limit für den Schreibdurchsatz der Partition. Diese Drosselung erfolgt unabhängig von der bereitgestellten Gesamtkapazität Ihres GSI und betrifft sowohl bereitgestellte als auch On-Demand-Tabellen. Sie können die CloudWatch Messwerte überwachenthe section called “Allgemeine Diagnose und Überwachung”, um Ihr Drosselungsereignis zu analysieren.

Mögliche Abhilfemaßnahmen

Erwägen Sie die folgenden Schritte, um die Drosselungsereignisse zu beheben:

- Neugestalten von GSI-Partitionsschlüsseln: Überprüfen Sie das Design Ihres GSI-Partitionsschlüssels, um sicherzustellen, dass er über eine ausreichende Kardinalität (Eindeutigkeit) verfügt, um Schreibvorgänge gleichmäßig zu verteilen. Eine häufige Ursache für GSI-Schreibdrosselungen ist die Verwendung von Attributen mit niedriger Kardinalität als GSI-Partitionsschlüssel (z. B. Status-Flags mit nur wenigen möglichen Werten). Selbst wenn Ihre Basistabelle über gut verteilte Partitionsschlüssel verfügt, kann es beim GSI dennoch zu Hot-Partitionen kommen, wenn der Partitionsschlüssel Schreibvorgänge auf eine kleine Anzahl von Werten konzentriert. Wenn beispielsweise 80 % Ihrer Elemente den Status „Aktiv“ (status="ACTIVE") aufweisen, führt dies zu einer ausgeprägten Hot-Partition in einem statusbasierten GSI. Erwägen Sie die Verwendung zusammengesetzter Schlüssel, die das Attribut mit niedriger Kardinalität mit einem Attribut mit hoher Kardinalität kombinieren (z. B. „ACTIVE#customer123" statt nur „ACTIVE“), oder wenden Sie Techniken wie unter the section called “Schreib-Sharding” beschrieben auf die Basistabellenelemente an, die sich auf die GSI-Verteilung auswirken, um Schreibvorgänge auf mehrere Partitionen zu verteilen. Für die Abfrage von Shard-Daten ist zwar zusätzliche Anwendungslogik erforderlich, um die Ergebnisse zu aggregieren, dieser Ansatz verhindert Drosselungen jedoch, indem die Zugriffsmuster gleichmäßiger verteilt werden.
- Vorwärmkapazität: Überprüfen Sie, ob Ihr GSI durch seine Grundlegendes zum Warmdurchsatz von DynamoDB-Kapazität begrenzt ist. Verwenden Sie Warmdurchsatz oder erhöhen Sie die bereitgestellte Lesekapazität vor einem erwarteten Anstieg des Datenverkehrs. Durch die

Erhöhung des Warmdurchsatzes ist Ihr GSI besser in der Lage, plötzliche Datenverkehrsspitzen zu bewältigen, bevor es zu einer Drosselung kommt. Im Laufe der Zeit kann DynamoDB ausgelastete Partitionen auf der Grundlage von beobachteten Verwendungsmustern aufteilen, wenn sich der tatsächliche Durchsatz konstant dem Warmdurchsatz nähert.

- Optimieren Sie die GSI-Prognosen: Wenden Sie the section called “Optimieren von GSI-Projektionen” Techniken an, um das Schreibvolumen auf zu reduzieren. GSIs Werden weniger Attribute projiziert, kann die Schreibkapazität, die durch jede GSI-Aktualisierung verbraucht wird, erheblich reduziert werden.

Allgemeine Diagnose und Überwachung

Bei der Behebung von Drosselungen auf Partitionsebene können verschiedene CloudWatch Metriken dabei helfen, heiße Partitionen zu identifizieren und die Ursache zu bestätigen.

CloudWatch Wesentliche Metriken

Überwachen Sie diese wichtigen Metriken, um eine Drosselung auf Partitionsebene zu diagnostizieren:

- Drosselungsereignisse auf Partitionsebene: **`ReadKeyRangeThroughputThrottleEvents`** und `WriteKeyRangeThroughputThrottleEvents` verfolgen, wann einzelne Partitionen ihre Durchsatzlimits überschreiten. `ReadThrottleEvents` und `WriteThrottleEvents` verfolgen, wann Lese- oder Schreibanforderungen die bereitgestellte Kapazität überschreiten.
- Kapazitätsverbrauch: **`ConsumedReadCapacityUnits`** und `ConsumedWriteCapacityUnits` zeigen allgemeine Nutzungsmuster an.

Fehlerbehebung

Identifizieren von Tastenkombinationen mithilfe von Contributor Insights CloudWatch

Gehen Sie wie folgt vor, um zu ermitteln, welche Partitionsschlüssel die Drosselung verursachen.

1. Aktivieren Sie CloudWatch Contributor Insights für Ihre Tabelle oder GSI, um die am häufigsten gedrosselten Schlüssel zu verfolgen. Erwägen Sie, CloudWatch Contributor Insights kontinuierlich zu aktivieren, um Drosselungswarnungen in Echtzeit zu erhalten, indem Sie den Modus für gedrosselte Tasten verwenden. Dieser Modus konzentriert sich ausschließlich auf gedrosselte Anfragen, indem Ereignisse nur dann verarbeitet werden, wenn eine Drosselung

eintritt. Diese gezielte Überwachung ist eine kostengünstige Methode, um Drosselungsprobleme kontinuierlich im Blick zu behalten.

2. Ermitteln Sie, welche Schlüssel die Probleme mit der Hot-Partition verursachen.
3. (Wenn der vollständige Modus für Schlüssel mit Zugriff und Drosselungen aktiviert ist) Analysieren Sie die Zugriffsmuster im Zeitverlauf, um festzustellen, ob Hotkeys konsistent sind oder nur zu bestimmten Zeiten auftreten.

Verbessern des Designs von Partitionsschlüsseln

Verwenden Sie diesen Ansatz, wenn Sie Ihr Tabellenschema ändern können, um den Datenverkehr besser auf die Partitionen zu verteilen. Sollte dies möglich sein, ist dies die effektivste langfristige Lösung für Probleme mit Hot-Partitionen. Im Idealfall sollte das Design der Partitionsschlüssel in der ersten Phase des Tabellenentwurfs sorgfältig geprüft werden.

Die Neugestaltung von Partitionsschlüsseln stellt eine grundlegende Änderung Ihres Datenmodells dar, die sich auf Ihre gesamte Anwendungsumgebung auswirkt. Bevor Sie mit diesem Ansatz fortfahren, sollten Sie die folgenden wesentlichen Einschränkungen sorgfältig abwägen:

- Komplexität der Datenmigration: Die Neugestaltung von Partitionsschlüsseln erfordert die Migration aller vorhandenen Daten, was bei großen Tabellen ressourcenintensiv und zeitaufwändig sein kann.
- Änderungen am Anwendungscode: Der gesamte Anwendungscode, der Lese- oder Schreibvorgänge in die Tabelle durchführt, muss aktualisiert werden, damit er die neue Schlüsselstruktur verwenden kann.
- Auswirkungen auf die Produktion: Die Migration zu einem neuen Schlüsseldesign erfordert während der Umstellung häufig Ausfallzeiten oder komplexe Dual-Write-Strategien.

Umfassende Anleitungen und Prinzipien zum Design von Partitionsschlüsseln finden Sie unter the section called “Entwerfen von Partitionsschlüsseln” und the section called “Verteilung von Workloads”.

Optimieren von GSI-Projektionen

Überprüfen Sie die Abfragemuster Ihrer Anwendung, um genau zu bestimmen, welche Attribute bei der Abfrage des GSI verfügbar sein müssen, und beschränken Sie Ihre Projektionen nur auf diese Attribute. Wenn Sie Attribute aktualisieren, die nicht in einen GSI projiziert werden, findet auf diesem GSI kein Schreibvorgang statt, sodass der Schreibdurchsatzverbrauch bei Aktualisierungen reduziert

wird. Diese gezielte Projektionsstrategie optimiert sowohl die Leistung als auch die Kosten und unterstützt zugleich die Abfrageanforderungen Ihrer Anwendung. Beachten Sie, dass die Projektion von weniger Attributen den Schreibkapazitätsverbrauch reduziert, aber möglicherweise zusätzliche Lesevorgänge in der Basistabelle erforderlich macht.

Weitere Informationen zu effizienten Projektionsstrategien finden Sie unter Bewährte Methoden für die Verwendung sekundärer Indexe in DynamoDB.

Weitere Ressourcen

Die folgenden Blog-Beiträge enthalten praktische Beispiele und ausführliche praxisorientierte Informationen zu den in diesem Handbuch behandelten Konzepten:

- Praktische Anleitungen zur Skalierung von DynamoDB und zur Verwaltung von Hot-Partitionen finden Sie in Teil 1: Skalierung von DynamoDB: Wie sich Partitionen, Hotkeys und Split for Heat auf die Leistung auswirken.
- Ausführliche Informationen zur Funktionsweise und den Vorteilen des split-for-heat DynamoDB-Mechanismus sowie Einzelheiten zur Implementierung finden Sie in Teil 3: Zusammenfassung und bewährte Methoden.
- Ausführliche Schreib-Sharding-Strategien finden Sie unter the section called “Schreib-Sharding”.

2. Überschreitung des bereitgestellten Durchsatzes

Die Drosselung der bereitgestellten Kapazität erfolgt, wenn die Nutzungsrate Ihrer Anwendung die für Ihre Tabellen oder globalen sekundären Indizes konfigurierten Lese- oder Schreibkapazitätseinheiten (RCUs/WCUs) überschreitet. DynamoDB bietet zwar Burst-Kapazität, um gelegentliche Datenverkehrsspitzen zu bewältigen, aber anhaltende Anfragen, die Ihre bereitgestellten Limits überschreiten, führen zu einer Drosselung. In diesem Fall gibt DynamoDB in der Drosselungsausnahme einen Drosselungsgrund vom Typ `ProvisionedThroughputExceeded` zurück. Dieser zeigt an, ob das Problem bei Lese- oder Schreibvorgängen liegt und ob es die Basistabelle oder einen globalen sekundären Index betrifft.

Die Drosselung kann unabhängig davon erfolgen, ob Auto Scaling aktiviert ist. Auto Scaling passt sich zwar dem steigenden Verbrauch an, reagiert aber nicht sofort. Zudem ist es durch die von Ihnen konfigurierten maximalen Kapazitätslimits eingeschränkt. Daher kann es immer noch zu Drosselungen kommen, wenn plötzliche Verkehrsspitzen auftreten oder der Verbrauch Ihre maximalen Auto-Scaling-Limits überschreitet.

Bereitgestellter Durchsatz übersteigt Abhilfemaßnahmen

Dieser Abschnitt bietet Anleitungen zur Lösung von Drosselungsszenarien bei bereitgestellten Kapazitäten. Bevor Sie diese Anleitung verwenden, sollten Sie den spezifischen Drosselungsgrund anhand der Ausnahmebehandlung Ihrer Anwendung identifizieren und den Amazon-Ressourcennamen (ARN) der betroffenen Ressource ermitteln. Informationen dazu, wie Sie die Gründe für die Drosselung abrufen und gedrosselte Ressourcen identifizieren, finden Sie unter the section called “Diagnose-Framework”.

Bevor Sie sich mit bestimmten Drosselungsszenarien befassen, sollten Sie zunächst prüfen, ob es sich bei der Drosselung tatsächlich um ein Problem handelt, das ausgeräumt werden muss:

- Eine gelegentliche Drosselung ist normal und wird in gut optimierten DynamoDB-Anwendungen erwartet. Drosselung bedeutet einfach, dass Sie 100 % dessen, was Sie bereitgestellt haben, verbrauchen. Wenn Ihre Anwendung ordnungsgemäß mit Wiederholungsversuchen auf die Drosselung reagiert und die Gesamtleistung den Anforderungen entspricht, sind möglicherweise keine sofortigen Gegenmaßnahmen erforderlich.
- Wenn die Drosselung jedoch zu einer inakzeptablen clientseitigen Latenz führt, die Nutzererfahrung beeinträchtigt oder verhindert, dass kritische Vorgänge rechtzeitig abgeschlossen werden, fahren Sie mit den unten aufgeführten Abhilfemöglichkeiten fort.

Wenn Sie Drosselungsprobleme beheben müssen, stellen Sie zunächst fest, ob folgende Ursachen vorliegen:

- Temporäre Datenverkehrsspitzen: Kurzfristiger Anstieg des Datenverkehrs, der die bereitgestellte Kapazität übersteigt, aber nicht dauerhaft anhält. Hierbei sind andere Strategien erforderlich als bei kontinuierlich hohem Datenverkehr.
- Kontinuierlich hoher Datenverkehr: Dauerhafte Workloads, die die bereitgestellte Kapazität ständig überschreiten.

Bei Datenverkehrsspitzen sollten Sie die Strategien aus dem Blog Handle traffic spikes with Amazon DynamoDB provisioned capacity unter the section called “Weitere Ressourcen” in Betracht ziehen.

Bei anhaltend hohem Datenverkehr haben Sie folgende Möglichkeiten zur Kapazitätsanpassung:

- the section called “TableReadProvisionedThroughputExceeded”
- the section called “TableWriteProvisionedThroughputExceeded”

- the section called “IndexReadProvisionedThroughputExceeded”
- the section called “IndexWriteProvisionedThroughputExceeded”

TableReadProvisionedThroughputExceeded

Wenn dies auftritt

Die Leseverbrauchsrate Ihrer Anwendung übersteigt die bereitgestellten Lesekapazitätseinheiten (RCUs), die für Ihre Tabelle konfiguriert sind. Sie können die CloudWatch Messwerte überwachen, the section called “Allgemeine Diagnose und Überwachung” um Ihr Drosselungsereignis zu analysieren.

Lösungsansatz

Ziehen Sie die folgenden Strategien in Betracht, um die Drosselung der Lesekapazität zu beheben:

- In den On-Demand-Kapazitätsmodus wechseln: Falls häufig Drosselungen aufgrund von Datenverkehrsspitzen auftreten, können Sie Ihre Tabelle auf den On-Demand-Modus umstellen. Der On-Demand-Modus beseitigt Probleme bei der Bereitstellung und passt sich automatisch Ihrer Workload an.
- Wenn der bereitgestellte Modus beibehalten wird und Auto Scaling nicht aktiviert ist:
  - Erwägen Sie, die Lesekapazität für die Tabelle zu erhöhen.
  - Aktivieren Sie Auto Scaling für die Lesekapazität bei der Tabelle.
- Wenn Auto Scaling aktiviert ist (Standard für Tabellen, die in der Konsole erstellt wurden):
  - Optimieren Sie die Auto-Scaling-Parameter für Lesevorgänge in der Tabelle.

TableWriteProvisionedThroughputExceeded

Wenn dies auftritt

Die Schreibverbrauchsrate Ihrer Anwendung übersteigt die bereitgestellten Schreibkapazitätseinheiten (WCUs), die für Ihre Tabelle konfiguriert sind. Sie können die CloudWatch Messwerte überwachen, the section called “Allgemeine Diagnose und Überwachung” um Ihr Drosselungsereignis zu analysieren.

Lösungsansatz

Ziehen Sie die folgenden Strategien in Betracht, um die Drosselung der Schreibkapazität zu beheben:

- In den On-Demand-Kapazitätsmodus wechseln: Falls häufig Drosselungen aufgrund von Datenverkehrsspitzen auftreten, können Sie Ihre Tabelle auf den On-Demand-Modus umstellen. Der On-Demand-Modus beseitigt Probleme bei der Bereitstellung und passt sich automatisch Ihrer Workload an.
- Wenn der bereitgestellte Modus beibehalten wird und Auto Scaling nicht aktiviert ist:
  - Erwägen Sie, die Schreibkapazität für die Tabelle zu erhöhen.
  - Aktivieren Sie Auto Scaling für die Schreibkapazität bei der Tabelle.
- Wenn Auto Scaling aktiviert ist (Standard für Tabellen, die in der Konsole erstellt wurden):
  - Optimieren Sie die Auto-Scaling-Parameter für Schreibvorgänge in der Tabelle.

IndexReadProvisionedThroughputExceeded

Wenn dies auftritt

Der Leseverbrauch auf einem Global Secondary Index (GSI) übersteigt die vom GSI bereitgestellten Lesekapazitätseinheiten (). RCUs Sie können die CloudWatch Messwerte überwachen, um Ihr the section called “Allgemeine Diagnose und Überwachung” Drosselungsereignis zu analysieren.

Lösungsansatz

Ziehen Sie diese Strategien in Betracht, um die Drosselung der GSI-Lesekapazität zu beheben:

- In den On-Demand-Kapazitätsmodus wechseln: Falls häufig Drosselungen aufgrund von Datenverkehrsspitzen auftreten, können Sie die Basistabelle auf den On-Demand-Modus umstellen. Der On-Demand-Modus beseitigt Probleme bei der Bereitstellung und passt sich automatisch Ihrer Workload an.
- Wenn der bereitgestellte Modus beibehalten wird und Auto Scaling nicht aktiviert ist:
  - Erwägen Sie, die GSI-Lesekapazität zu erhöhen.
  - Aktivieren Sie Auto Scaling für die Lesekapazität bei dem GSI.
- Wenn Auto Scaling aktiviert ist (Standard für Tabellen, die in der Konsole erstellt wurden):
  - Optimieren Sie die Auto-Scaling-Parameter für Lesevorgänge im GSI.

IndexWriteProvisionedThroughputExceeded

Wenn dies auftritt

Aktualisierungen von Elementen in der Basistabelle lösen Schreibvorgänge in einen GSI aus, die die bereitgestellte Schreibkapazität des GSI überschreiten. Dies führt zu einer Gegendruckdrosselung bei Schreibvorgängen in der Basistabelle. Sie können die CloudWatch Messwerte überwachen, um Ihr the section called “Allgemeine Diagnose und Überwachung” Drosselungsereignis zu analysieren.

Lösungsansatz

Ziehen Sie die folgenden Strategien in Betracht, um die Drosselung der GSI-Schreibkapazität zu beheben:

- In den On-Demand-Kapazitätsmodus wechseln: Falls häufig Drosselungen aufgrund von Datenverkehrsspitzen auftreten, können Sie die Basistabelle auf den On-Demand-Modus umstellen. Der On-Demand-Modus beseitigt Probleme bei der Bereitstellung und passt sich automatisch Ihrer Workload an.
- Wenn der bereitgestellte Modus beibehalten wird und Auto Scaling nicht aktiviert ist:
  - Erwägen Sie, die Schreibkapazität für den GSI zu erhöhen.
  - Aktivieren Sie Auto Scaling für die Schreibkapazität bei dem GSI.
- Wenn Auto Scaling aktiviert ist (Standard für Tabellen, die in der Konsole erstellt wurden):
  - Optimieren Sie die Auto-Scaling-Parameter für Schreibvorgänge im GSI.

Allgemeine Diagnose und Überwachung

Bei der Behebung von Durchsatzfehlern können verschiedene CloudWatch Metriken helfen, die Ursache zu ermitteln.

Wesentliche CloudWatch Metriken

Überwachen Sie diese wichtigen Kennzahlen, um die Drosselung der bereitgestellten Kapazität zu diagnostizieren:

- Drosselungsereignisse: `ReadProvisionedThroughputThrottleEvents` und `WriteProvisionedThroughputThrottleEvents` verfolgen, wann Anfragen aus diesem Grund gedrosselt werden. `ReadThrottleEvents` und `WriteThrottleEvents` verfolgen, wann Lese- oder Schreibanfragen die bereitgestellte Kapazität überschreiten.
- Kapazitätsverbrauch: `ConsumedReadCapacityUnits` und `ConsumedWriteCapacityUnits` zeigen die tatsächliche Nutzung an.
- Bereitgestellte Kapazität: `ProvisionedReadCapacityUnits` und `ProvisionedWriteCapacityUnits` zeigen die konfigurierten Limits an.

Fehlerbehebung

Erhöhen der Durchsatzkapazität einer Tabelle

Verwenden Sie dieses Verfahren, wenn Auto Scaling nicht aktiviert ist und Sie eine sofortige Kapazitätserhöhung benötigen.

1. Aktualisieren Sie die bereitgestellte Kapazität Ihrer Tabelle mithilfe der DynamoDB-Konsole oder des SDK AWS CLI:
   - Für die Lesekapazität: Erhöhen Sie den Parameter `ReadCapacityUnits`, der die maximale Anzahl strikt konsistenter Lesevorgänge angibt, die pro Sekunde verbraucht werden, bevor DynamoDB Anfragen drosselt.
   - Für die Schreibkapazität: Erhöhen Sie den Parameter `WriteCapacityUnits`, der die maximale Anzahl der Schreibvorgänge angibt, die pro Sekunde verbraucht werden, bevor DynamoDB Anfragen drosselt.
2. Achten Sie darauf, dass die neuen Kapazitätseinstellungen die Durchsatzkontingente pro Tabelle nicht überschreiten und dass Ihr gesamter Kontoverbrauch unter den Durchsatzkontingenten pro Konto für Ihre Region bleibt. Wenn Sie an diese Limits herankommen, sollten Sie stattdessen in den On-Demand-Kapazitätsmodus wechseln.

Konfigurieren von Auto Scaling zum Anpassen der Lese- oder Schreibkapazität einer Tabelle oder eines GSI

Konfigurieren Sie DynamoDB Auto Scaling so, dass die Lese- oder Schreibkapazität automatisch an die Datenverkehrsmuster angepasst wird. Sie können Auto Scaling unabhängig für beide Tabellen und GSIs mit separaten Steuerungen für Lese- und Schreibkapazitätseinheiten konfigurieren.

1. Aktivieren Sie Auto Scaling für die Lese- oder die Schreibkapazität oder für beides bei der Tabelle oder dem GSI.
2. Legen Sie einen Zielauslastungsprozentsatz mit Spielraum für Datenverkehrsspitzen fest.

> **Note**
>
> Eine geringere Zielauslastung erhöht die Kosten und die Skalierungshäufigkeit. Ziele unter 40 % können zu einer übermäßigen Bereitstellung führen. Überwachen Sie Nutzungsmuster und Kosten, um ein ausgewogenes Verhältnis zwischen Leistung und Effizienz zu finden.

3. Legen Sie Kapazitätsgrenzen fest:

   - Minimum RCUs/WCUs: Sorgt für eine ausreichende Kapazität in Zeiten mit geringem Datenaufkommen.
   - Maximum RCUs/WCUs: Nimmt hohe Datenverkehrsanforderungen auf und schützt vor unkontrollierten Skalierungsereignissen.

Anleitungen zum Konfigurieren und Verwalten von DynamoDB Auto Scaling finden Sie unter Managing throughput capacity automatically with DynamoDB Auto Scaling.

> **Note**
>
> Auto Scaling benötigt in der Regel mehrere Minuten, um auf Datenverkehrsänderungen zu reagieren. Bei plötzlichen Datenverkehrsspitzen bietet die Burst-Kapazität der Tabelle sofortigen Schutz, während Auto Scaling die nötigen Anpassungen vornimmt. Konfigurieren Sie die Zielauslastung mit ausreichend Spielraum, um Zeit für solche Skalierungsvorgänge zu lassen und die Burst-Kapazität für unerwarteten Bedarf aufzusparen.

Optimieren der Auto-Scaling-Einstellungen für Lese- oder Schreibvorgänge bei einer Tabelle oder einem Index

Verwenden Sie dieses Verfahren, wenn Auto Scaling aktiviert ist, aber weiterhin eine Drosselung erfolgt. Sie können Auto Scaling unabhängig sowohl für Tabellen als auch für globale Sekundärindizes (GSIs) mit separaten Steuerungen für Lese- und Schreibkapazitätseinheiten optimieren.

- Passen Sie die Zielauslastung an: Erwägen Sie, die Zielauslastung für Ihre Tabelle GSIs zu verringern oder die Skalierung früher auszulösen, bevor eine Drosselung erfolgt. Nachdem Sie diese Anpassungen vorgenommen haben, sollten Sie den Datenverkehr überwachen. Weitere Informationen zum Kapazitätsverbrauch und den Auswirkungen auf die Kosten finden Sie unter the section called “Konfigurieren von Auto Scaling zum Anpassen der Lese- oder Schreibkapazität einer Tabelle oder eines GSI”.
- Kapazitätsgrenzen überprüfen: Stellen Sie sicher, dass die Einstellungen für minimale und maximale Kapazität den tatsächlichen Workload-Mustern entsprechen.

Wechseln in den On-Demand-Kapazitätsmodus

Allgemeine Informationen zum Wechseln der Kapazitätsmodi finden Sie unter the section called “Umstellen der Kapazitätsmodi”. Informationen zu spezifischen Einschränkungen beim Moduswechsel finden Sie in den Service Quotas.

Erhöhen der GSI-Durchsatzkapazität

Verwenden Sie dieses Verfahren, wenn Auto Scaling bei Ihrem GSI nicht aktiviert ist oder Sie eine sofortige Kapazitätserhöhung benötigen.

1. Aktualisieren Sie die bereitgestellte Kapazität der GSI mithilfe der DynamoDB-Konsole oder des SDK: AWS CLI

   - Für die Lesekapazität: Erhöhen Sie den Parameter `ReadCapacityUnits`, der die maximale Anzahl an Lesevorgängen angibt, die der GSI pro Sekunde verbrauchen kann, bevor DynamoDB Anfragen drosselt. Beachten Sie, dass GSIs nur eventuell konsistente Lesevorgänge unterstützt werden.
   - Für die Schreibkapazität: Erhöhen Sie den Parameter `WriteCapacityUnits`, der die maximale Anzahl an Schreibvorgängen angibt, die der GSI pro Sekunde verbrauchen kann, bevor DynamoDB Anfragen drosselt.

2. Achten Sie darauf, dass die für den GSI bereitgestellte Durchsatzkapazität innerhalb der Durchsatzkontingente pro Konto und Tabelle bleibt.

Weitere Ressourcen

- Ausführliche Informationen zum Umgang mit Datenverkehrsspitzen in DynamoDB-Tabellen mit bereitgestellter Kapazität, einschließlich Strategien von Auto Scaling und Burst-Kapazität bis zu strategischem Drosselungsmanagement, finden Sie unter Handle traffic spikes with Amazon DynamoDB provisioned capacity.
- Informationen zur Verwendung eines Cron-Ausdrucks zum Planen einer Skalierungsrichtlinie finden Sie unter Optimize costs by scheduling provisioned capacity for DynamoDB.
- Praktische Informationen zum Überwachen und Analysieren von Durchsatznutzungsmustern für DynamoDB-Tabellen im bereitgestellten Kapazitätsmodus finden Sie unter How to evaluate throughput utilization for Amazon DynamoDB tables in provisioned mode.

3- Überschreitung der Kontolimits

Für On-Demand-Tabellen gibt es keine bereitgestellten Kapazitätsstufen, die verwaltet werden müssen, aber DynamoDB setzt Durchsatzlimits auf Kontoebene durch, um eine unkontrollierte Ausführung zu verhindern und eine faire Ressourcennutzung für alle Kunden sicherzustellen. Diese Kontolimits pro Tabelle dienen als einstellbare Sicherheitsvorkehrungen, die für jede Kombination aus Konto und Region festgelegt werden. Wenn Ihre Lese- oder Schreibverbrauchsrate diese Grenzwerte überschreitet, gibt DynamoDB in der Drosselungsausnahme den Drosselungsgrund `AccountLimitExceeded` zurück. Die standardmäßigen Kontolimits pro Tabelle gelten automatisch, wenn für Tabellen keine benutzerdefinierten Einstellungen für den maximalen Durchsatz konfiguriert sind. Sie können optional die Einstellungen für den maximalen Durchsatz konfigurieren, um die Kosten besser kontrollieren und präziser vorhersagen zu können, oder über die the section called “Kontingente”-Konsole Kontingenterhöhungen anfordern, falls Ihre Anwendungsanforderungen die Standardgrenzwerte überschreiten.

Das Kontolimit hat die Abhilfemaßnahmen überschritten

Dieser Abschnitt bietet Anleitungen zur Lösung von Drosselungsszenarien mit Kontolimits. Bevor Sie diese Anleitung verwenden, sollten Sie die spezifischen Drosselungsgründe anhand der Ausnahmebehandlung Ihrer Anwendung identifizieren und den Amazon-Ressourcennamen (ARN) der betroffenen Ressource ermitteln. Informationen dazu, wie Sie die Gründe für die Drosselung abrufen und gedrosselte Ressourcen identifizieren, finden Sie unter the section called “Diagnose-Framework”.

Bevor Sie sich mit bestimmten Drosselungsszenarien befassen, sollten Sie zunächst ermitteln, ob tatsächlich Maßnahmen erforderlich sind:

- Beurteilen der Auswirkungen auf die Leistung: Überprüfen Sie, ob Ihre Anwendung trotz der Drosselung die Leistungsanforderungen weiter erfüllt. Viele Anwendungen werden erfolgreich an oder nahe den Kontolimits ausgeführt, insbesondere bei Massenoperationen oder Datenmigrationen.
- Überprüfen der Drosselungsmuster: Wenn die Drosselung nur sporadisch erfolgt und Ihre Anwendung Wiederholversuche effektiv verarbeitet, können die aktuellen Grenzwerte für Ihren Workload ausreichend sein.

Wenn Ihre Anwendung auch bei gelegentlichem Überschreiten der Kontolimits eine akzeptable Leistung erbringt, können Sie die Situation auch einfach überwachen, anstatt unmittelbar Änderungen vorzunehmen.

Wenn Sie feststellen, dass die Drosselung zu inakzeptablen Problemen mit der Leistung oder Zuverlässigkeit führt, wählen Sie unten einen bestimmten Drosselungsgrund aus, um die empfohlenen Abhilfemaßnahmen zu finden:

- the section called “TableReadAccountLimitExceeded”
- the section called “TableWriteAccountLimitExceeded”
- the section called “IndexReadAccountLimitExceeded”
- the section called “IndexWriteAccountLimitExceeded”

TableReadAccountLimitExceeded

Wenn dies auftritt

Der Leseverbrauch Ihrer Tabelle hat das Kontingent für den Lesedurchsatz pro Tabelle auf Kontoebene für Ihre Region überschritten. Sie können die CloudWatch Messwerte im Auge behalten, um Ihr the section called “Allgemeine Diagnose und Überwachung” Drosselungsereignis zu analysieren.

Lösungsansatz

Führen Sie zur Behebung dieser Drosselung die folgenden Schritte aus:

- Anfordern einer Kontingenterhöhung:

  Beantragen Sie eine Erhöhung des Grenzwerts für den Lesedurchsatz pro Tabelle (Quotencode L-). CF0 CBE56 Eine ausführliche Anleitung zum Einreichen der Anfrage finden Sie unter Requesting per-table quota increases.

TableWriteAccountLimitExceeded

Wenn dies auftritt

Der Schreibverbrauch Ihrer Tabelle hat das Kontingent für den Schreibdurchsatz pro Tabelle auf Kontoebene für Ihre Region überschritten. Sie können die CloudWatch Messwerte überwachen, the section called “Allgemeine Diagnose und Überwachung” um Ihr Drosselungsereignis zu analysieren.

Lösungsansatz

Führen Sie zur Behebung dieser Drosselung die folgenden Schritte aus:

- Erhöhung der Quote anfordern: Fordern Sie eine Erhöhung des Schreibdurchsatzlimits pro Tabelle an (Quotencode L-). AB614373 Eine ausführliche Anleitung zum Einreichen der Anfrage finden Sie unter Requesting per-table quota increases.

IndexReadAccountLimitExceeded

Wenn dies auftritt

Die Leseoperationen, die auf einen globalen sekundären Index (GSI) abzielen, verbrauchen mehr Durchsatz, als das Lesekontingent Ihres Kontos pro Tabelle in Ihrer aktuellen AWS -Region zulässt. Die Quote für den Lesedurchsatz pro Tabelle auf Kontoebene gilt sowohl für eine Tabelle als auch für die gesamte Tabelle zusammen. GSIs Sie können die CloudWatch Messwerte überwachen, um Ihr the section called “Allgemeine Diagnose und Überwachung” Drosselungsereignis zu analysieren.

Lösungsansatz

Wählen Sie die passende Lösung auf der Grundlage der Kapazitätsverteilung Ihres Kontos:

- Erhöhung des Kontingents anfordern: Beantragen Sie eine Erhöhung des Grenzwerts für den Lesedurchsatz pro Tabelle (Quotencode L-). CF0 CBE56 Eine ausführliche Anleitung zum Einreichen der Anfrage finden Sie unter Requesting per-table quota increases.
- Optimieren der GSI-Nutzung: Überprüfen Sie das GSI-Design und die Abfragemuster, um unnötigen Lesekapazitätsverbrauch zu reduzieren.

IndexWriteAccountLimitExceeded

Wenn dies auftritt

Schreibvorgänge in Ihre Basistabelle führen zu entsprechenden Aktualisierungen für einen globalen Index, die zusammen die Quote für den Schreibdurchsatz pro Tabelle auf Kontoebene für Ihre Region überschreiten. AWS Jede Schreiboperation in ein Basistabellenelement, das Attribute enthält, die von einem GSI-Trigger indiziert wurden, löst einen entsprechenden Schreibvorgang für diesen GSI aus. Diese kombinierten Schreibvorgänge werden auf Ihr Schreibdurchsatzkontingent pro Tabelle angerechnet. Sie können die CloudWatch Messwerte überwachenthe section called “Allgemeine Diagnose und Überwachung”, um die Muster und den Zeitpunkt dieser Drosselungsereignisse zu analysieren und festzustellen, welche Operationen die übermäßige GSI-Schreibaktivität verursachen.

Lösungsansatz

Wählen Sie die passende Lösung auf der Grundlage der Kapazitätsverteilung Ihres Kontos:

- Erhöhung des Anforderungskontingents: Fordern Sie eine Erhöhung des Grenzwerts für den Schreibdurchsatz pro Tabelle (Quotencode L-AB614373) an, um dem höheren GSI-Schreibdatenverkehr aus Basistabellenoperationen Rechnung zu tragen. Die Quote für den Schreibdurchsatz pro Tabelle gilt für die gesamte Tabelle, einschließlich aller Tabellen. GSIs Eine ausführliche Anleitung zum Einreichen der Anfrage finden Sie unter Requesting per-table quota increases.
- Optimieren Sie GSI-Prognosen: Überprüfen Sie die GSI-Prognosen und das Design, um das Schreibvolumen auf zu reduzieren. GSIs

Allgemeine Diagnose und Überwachung

Bei der Fehlerbehebung bei Überschreitung der Drosselung des Kontolimits können Sie anhand verschiedener CloudWatch Messwerte ermitteln, ob Sie die Grenzen pro Tabelle oder für das gesamte Konto überschreiten, und Ihre Kapazitätsverteilungsmuster besser nachvollziehen.

CloudWatch Wesentliche Messwerte

Überwachen Sie diese wichtigen Kennzahlen, um die Drosselung von Kontolimits zu diagnostizieren:

- Drosselungsereignisse auf Kontoebene: `ReadAccountLimitThrottleEvents` und `WriteAccountLimitThrottleEvents` verfolgen, wann Anforderungen aufgrund von Limits auf Kontoebene gedrosselt werden. `ReadThrottleEvents` und `WriteThrottleEvents` verfolgen, wann Lese- oder Schreibanforderungen die bereitgestellte Kapazität überschreiten.
- Kapazitätsverbrauch nach Ressource: **`ConsumedReadCapacityUnits`** und `ConsumedWriteCapacityUnits` für jede Tabelle und jeden GSI zeigen die individuelle Ressourcennutzung an.
- Kontoweiter Verbrauch: Verwenden Sie CloudWatch metrische mathematische Ausdrücke, um die verbrauchte Kapazität in allen Tabellen zu summieren und GSIs die gesamte Kontonutzung zu überwachen.

Fehlerbehebung

Anfordern einer Kontingenterhöhung pro Tabelle

Wenn Ihre Anwendungen aus operativen Gründen die aktuellen Durchsatzlimits pro Tabelle überschreiten müssen, sollten Sie mit dem unten beschriebenen Verfahren eine Erhöhung des Kontingents beantragen. Jede DynamoDB-Tabelle in Ihrem AWS Konto (zusammen mit allen zugehörigen Tabellen GSIs) unterliegt diesen Durchsatzquoten innerhalb einer bestimmten Region.

Diese Kontingente stellen die maximale Lese- oder Schreibkapazität dar, die eine einzelne Tabelle zusammen verbrauchen GSIs kann, und sie gelten unabhängig für jede Tabelle und nicht als Summe für alle Tabellen in Ihrem Konto.

Optional können Sie auch niedrigere Limits pro Tabelle oder GSI festlegen, indem Sie deren Einstellungen für den maximalen On-Demand-Durchsatz konfigurieren.

1. Identifizieren Sie das spezifische Kontingent, das erhöht werden muss:
   - Limit für den Lesedurchsatz pro Tabelle (Quotencode L- CF0CBE56): Standardmäßig 40.000 RCUs pro Tabelle
   - Limit für den Schreibdurchsatz pro Tabelle (Quota-Code L-AB614373): Standardmäßig 40.000 pro WCUs Tabelle
2. Verwenden Sie zum Anfordern einer Erhöhung die AWS -Konsole für Service Quotas.
   - Navigieren Sie in Service Quotas zum DynamoDB-Service.
   - Suchen Sie mithilfe des Kontingentcodes nach dem entsprechenden Kontingent.
   - Fordern Sie eine Erhöhung auf der Grundlage Ihrer voraussichtlichen Spitzennutzung an.
3. Begründen Sie die Erhöhung, einschließlich folgender Informationen:
   - Aktuelle Nutzungsmuster und Anforderungen an den Spitzenverkehr
   - Geschäftliche Begründung für die Kapazitätserhöhung
   - Zeitplan für den Zeitpunkt, zu dem die erhöhte Kapazität benötigt wird

> **Note**
>
> Die Bearbeitung von Kontingenterhöhungen nimmt in der Regel 24 bis 48 Stunden in Anspruch. Planen Sie Ihre Anforderungen entsprechend und ziehen Sie vorübergehende Abhilfemaßnahmen in Betracht, während Sie auf die Genehmigung warten.

Optimieren von GSI-Projektionen und -Design

Optimieren Sie Ihre GSI-Projektionen und das Design, um den Kapazitätsverbrauch zu reduzieren und die Leistung zu verbessern.

Selektive Projektionsstrategien

Wenn Ihre Abfragen nur auf einige wenige Attribute zugreifen müssen, reduziert die Projektion nur dieser Attribute die Datenmenge, die in den GSI geschrieben wird, wenn sich die Elemente der Basistabelle ändern. Einzelheiten zu den Projektionstypen finden Sie unter Projections for Global Secondary Indexes.

1. Analysieren der Abfragemuster: Überprüfen Sie die Abfragemuster Ihrer Anwendung, um festzustellen, auf welche Attribute tatsächlich über den GSI zugegriffen wird.
2. Verwenden selektiver Prognosen: Projizieren Sie nur Attribute, die in Abfragen tatsächlich benötigt werden, um das Schreibvolumen zu reduzieren.
3. Erwägen von `KEYS_ONLY`: Wenn Ihre Abfragen nur die Schlüsselattribute benötigen, verwenden Sie die `KEYS_ONLY`-Projektion, um das Schreibvolumen zu minimieren.
4. Abwägen zwischen Lese-/Schreibvorgängen: Die Projektion von weniger Attributen reduziert zwar den Schreibkapazitätsverbrauch, macht aber möglicherweise zusätzliche Lesevorgänge in der Basistabelle erforderlich.

Sparse-GSI-Implementierung

Sparse GSIs enthält nur Elemente, die das indizierte Attribut haben, und nicht alle Elemente aus Ihrer Basistabelle. Dadurch wird die Partitionsdichte reduziert und die Leistung verbessert, wenn Sie häufig bestimmte Teilmengen von Daten abfragen.

1. Design GSIs , das nur Elemente mit bestimmten Attributwerten enthält.
2. Implementieren Sie die bedingte Indizierung, indem Sie das Schlüsselattribut der GSI-Partition nur für Elemente festlegen, die indiziert werden sollen.
3. Verwenden Sie zusammengesetzte Schlüssel mit geringer Dichte GSIs (z. B. status #timestamp), um den Verkehr innerhalb der Teilmenge der indizierten Elemente weiter zu verteilen.

Weitere Informationen zur Umsetzung dieser Strategien finden Sie unter Bewährte Methoden für die Verwendung sekundärer Indexe in DynamoDB.

4. Überschreitung des maximalen On-Demand-Durchsatzes

Wenn Sie eine On-Demand-Tabelle oder einen On-Demand-GSI konfigurieren, können Sie optional ein maximales Durchsatzlimit (`MaxReadRequestUnits` und `MaxWriteRequestUnits`) auf Tabellen- oder Indexebene festlegen. So lassen sich unkontrollierte Kosten vermeiden oder

nachgelagerte Systeme vor Überlastung schützen. Weitere Informationen zu maximalen Durchsätzen finden Sie unter the section called “Maximaler DynamoDB-Durchsatz für On-Demand-Tabellen”.

Wenn Ihr Lese- oder Schreibverbrauch diese selbst festgelegten Grenzwerte überschreitet, erhalten weitere Anfragen gedrosselte Schnellantworten. DynamoDB gibt Ausnahmen mit einem Drosselungsgrund der Art `MaxOnDemandThroughputExceeded` zurück, der angibt, welche Ressource ihre Durchsatzgrenze erreicht hat.

Drosselung wegen Überschreitens des maximalen On-Demand-Durchsatzes

Dieser Abschnitt enthält Anleitungen zur Lösung von Drosselungsszenarien, in denen der maximale On-Demand-Durchsatz überschritten wurde. Bevor Sie diese Anleitung verwenden, sollten Sie die spezifischen Drosselungsgründe anhand der Ausnahmebehandlung Ihrer Anwendung identifizieren und den Amazon-Ressourcennamen (ARN) der betroffenen Ressource ermitteln. Informationen dazu, wie Sie die Gründe für die Drosselung abrufen und gedrosselte Ressourcen identifizieren, finden Sie unter the section called “Diagnose-Framework”.

Bevor Sie sich mit bestimmten Drosselungsszenarien befassen, sollten Sie zunächst überlegen, ob tatsächlich Maßnahmen erforderlich sind:

- Einstellungen für den maximalen Durchsatz prüfen: Diese Limits wurden bewusst konfiguriert, um Kosten zu kontrollieren oder nachgelagerte Systeme zu schützen. Wenn Sie `MaxOnDemandThroughputExceeded`-Drosselungsereignisse erhalten, funktionieren Ihre Limits wie vorgesehen. Überlegen Sie, ob eine Erhöhung dieser Limits Ihren ursprünglichen Zielen in Bezug auf Kostenkontrolle oder Systemschutz entspricht.
- Auswirkungen auf die Anwendung bewerten: Ermitteln Sie, ob die Drosselung tatsächlich Probleme für Ihre Anwendungen oder Nutzer verursacht. Wenn Ihre Anwendungen Wiederholungsversuche effektiv verarbeiten und ihre Leistungsanforderungen trotz gelegentlicher Drosselung erfüllen, ist die Beibehaltung der aktuellen Limits möglicherweise die richtige Wahl.
- Datenverkehrsmuster überprüfen: Analysieren Sie, ob es sich bei der Drosselung um ein erwartetes Verkehrsmuster oder um einen ungewöhnlichen Anstieg handelt. Bei vorhersehbaren, wiederkehrenden Datenverkehrsmustern, die Ihre Limits ständig überschreiten, kann eine Anpassung der Einstellungen für den maximalen Durchsatz gerechtfertigt sein. Bei vorübergehenden Spitzenwerten sind bessere Strategien zur Verteilung von Anfragen möglicherweise besser geeignet als die Erhöhung der Limits.

Wenn Sie nach eingehender Prüfung der Ansicht sind, dass Ihre Einstellungen für den maximalen Durchsatz angepasst werden müssen, finden Sie in den unten aufgeführten spezifischen Drosselungsszenarien gezielte Abhilfemaßnahmen:

- the section called “TableReadMaxOnDemandThroughputExceeded”
- the section called “TableWriteMaxOnDemandThroughputExceeded”
- the section called “IndexReadMaxOnDemandThroughputExceeded”
- the section called “IndexWriteMaxOnDemandThroughputExceeded”

TableReadMaxOnDemandThroughputExceeded

Wenn dies auftritt

Ihre On-Demand-Tabelle hat die konfigurierte maximale Lesedurchsatzkapazität überschritten. Sie können die CloudWatch Messwerte überwachenthe section called “Allgemeine Diagnose und Überwachung”, um Ihr Drosselungsereignis zu analysieren.

Mögliche Abhilfemaßnahmen

Erwägen Sie die folgenden Schritte, um die Drosselungsereignisse zu beheben:

- Maximales Durchsatzlimit erhöhen: Verwenden Sie die DynamoDB-Konsole, AWS CLI oder die DynamoDB `UpdateTable`-API, um den Wert `MaxReadRequestUnits` für die betroffene Tabelle zu erhöhen. Überwachen Sie ihn anschließend und passen Sie ihn an. Dadurch kann die Tabelle einen höheren Lesedurchsatz bewältigen, bevor es zu einer Drosselung kommt.
- Das maximale Limit entfernen: Stellen Sie `MaxReadRequestUnits` auf `-1` ein, um die Obergrenze aufzuheben und eine bedarfsgerechte Skalierung bis zu Ihren Durchsatzkontingenten auf Kontoebene zu ermöglichen. Dadurch wird Ihr benutzerdefiniertes Limit entfernt, die Sicherheitsvorkehrungen auf Kontoebene von AWS bleiben aber erhalten. Nach der Aufhebung des Limits sollten Sie jedoch die Kosten genau im Blick behalten, da Ihre Tabelle jetzt deutlich mehr Kapazität verbrauchen kann, bevor die Kontingente auf Kontoebene erreicht werden.

TableWriteMaxOnDemandThroughputExceeded

Wenn dies auftritt

Ihre On-Demand-Tabelle hat die konfigurierte maximale Schreibdurchsatzkapazität überschritten. Sie können die CloudWatch Messwerte überwachen, um Ihr the section called “Allgemeine Diagnose und Überwachung” Drosselungsereignis zu analysieren.

Mögliche Abhilfemaßnahmen

Erwägen Sie die folgenden Schritte, um die Drosselungsereignisse zu beheben:

- Maximales Durchsatzlimit erhöhen: Verwenden Sie die DynamoDB-Konsole, AWS CLI oder die DynamoDB `UpdateTable`-API, um den Wert `MaxWriteRequestUnits` für die betroffene Tabelle zu erhöhen. Überwachen Sie ihn anschließend und passen Sie ihn an.
- Das maximale Limit entfernen: Stellen Sie `MaxWriteRequestUnits` auf `-1` ein, um die Obergrenze aufzuheben und eine bedarfsgerechte Skalierung bis zu Ihren Durchsatzkontingenten auf Kontoebene zu ermöglichen. Dadurch wird Ihr benutzerdefiniertes Limit entfernt, die Sicherheitsvorkehrungen auf Kontoebene von AWS bleiben aber erhalten. Nach der Aufhebung des Limits sollten Sie jedoch die Kosten genau im Blick behalten, da Ihre Tabelle jetzt deutlich mehr Kapazität verbrauchen kann, bevor die Kontingente auf Kontoebene erreicht werden.

IndexReadMaxOnDemandThroughputExceeded

Wenn dies auftritt

Leseanfragen an einen GSI im On-Demand-Modus haben die konfigurierte maximale Lesedurchsatzkapazität des GSI überschritten. Sie können die CloudWatch Messwerte überwachen, um Ihr the section called “Allgemeine Diagnose und Überwachung” Drosselungsereignis zu analysieren.

Mögliche Abhilfemaßnahmen

Erwägen Sie die folgenden Schritte, um die Drosselungsereignisse zu beheben:

- Maximales GSI-Durchsatzlimit erhöhen: Verwenden Sie die DynamoDB-Konsole, AWS CLI oder die DynamoDB `UpdateTable`-API, um den Wert `MaxReadRequestUnits` für den betroffenen GSI zu erhöhen. Überwachen Sie ihn anschließend und passen Sie ihn an.
- Das maximale GSI-Limit entfernen: Stellen Sie `MaxReadRequestUnits` auf `-1` ein, um die Obergrenze aufzuheben und eine bedarfsgerechte Skalierung bis zu Ihren Durchsatzkontingenten auf Kontoebene zu ermöglichen. Dadurch wird Ihr benutzerdefiniertes Limit aufgehoben, die Sicherheitsvorkehrungen auf Kontoebene AWS bleiben jedoch erhalten. Nach der Aufhebung des Limits sollten Sie jedoch die Kosten genau im Blick behalten.

IndexWriteMaxOnDemandThroughputExceeded

Wenn dies auftritt

Aktualisierungen von Elementen in der Basistabelle lösen im On-Demand-Modus Schreibvorgänge in einen GSI aus, die dessen konfigurierte maximale Schreibdurchsatzkapazität überschreiten. Dies führt zu einer Gegendruckdrosselung. Sie können die CloudWatch Messwerte überwachen, um Ihr the section called “Allgemeine Diagnose und Überwachung” Drosselungsereignis zu analysieren.

Mögliche Abhilfemaßnahmen

Erwägen Sie die folgenden Schritte, um die Drosselungsereignisse zu beheben:

- Maximales GSI-Durchsatzlimit erhöhen: Verwenden Sie die DynamoDB-Konsole, AWS CLI oder die DynamoDB `UpdateTable`-API, um den Wert `MaxWriteRequestUnits` für den betroffenen GSI zu erhöhen. Überwachen Sie ihn anschließend und passen Sie ihn an.
- Das maximale GSI-Limit entfernen: Stellen Sie `MaxWriteRequestUnits` auf `-1` ein, um die Obergrenze aufzuheben und eine bedarfsgerechte Skalierung bis zu Ihren Durchsatzkontingenten auf Kontoebene zu ermöglichen. Dadurch wird Ihr benutzerdefiniertes Limit aufgehoben, die Sicherheitsvorkehrungen auf Kontoebene AWS bleiben jedoch erhalten. Nach der Aufhebung des Limits sollten Sie jedoch die Kosten genau im Blick behalten.

Allgemeine Diagnose und Überwachung

Bei der On-Demand-Fehlerbehebung, bei der der maximale Durchsatz Drosselungsereignisse überschritten hat, können verschiedene CloudWatch Metriken dabei helfen, die Ursache und Skalierungsmuster zu identifizieren.

Wesentliche Metriken CloudWatch

Überwachen Sie diese wichtigen Kennzahlen, um die Ursachen für die Drosselung wegen des Überschreitens des maximalen On-Demand-Durchsatzes zu finden:

- Ereignisse zur Drosselung des maximalen Durchsatzes: `ReadMaxOnDemandThroughputThrottleEvents` und `WriteMaxOnDemandThroughputThrottleEvents` verfolgen, wann Anfragen aufgrund einer Überschreitung der Höchstgrenzen gedrosselt werden. `ReadThrottleEvents` und `WriteThrottleEvents` verfolgen, wann Lese- oder Schreibanforderungen die bereitgestellte Kapazität überschreiten.

- Aktuell konfigurierter maximaler Durchsatz für eine Tabelle oder einen globalen sekundären Index: `OnDemandMaxReadRequestUnits` und `OnDemandMaxWriteRequestUnits` zeigen die aktuellen maximalen Kapazitätslimits an.
- Tatsächlicher Kapazitätsverbrauch: `ConsumedReadCapacityUnits` und `ConsumedWriteCapacityUnits` zeigen die tatsächlichen Nutzungsmuster.

Analyseansatz

Gehen Sie wie folgt vor, um die Diagnose „Maximaler On-Demand-Durchsatz überschritten“ zu bestätigen:

1. Vergleichen Sie die verbrauchte Kapazität mit den maximalen Kapazitätslimits. Prüfen Sie, ob sich der Verbrauch kontinuierlich den Höchstgrenzen nähert oder diese überschreitet.
2. Überprüfen Sie die Häufigkeit und den Zeitpunkt von Drosselungsereignissen, um Muster zu identifizieren. Achten Sie auf plötzliche Steigerungen der verbrauchten Kapazität, die mit dem Drosselungsereignis zusammenfallen.
3. Verwenden Sie CloudWatch Contributor Insights, um zu ermitteln, welche Elemente oder Partitionsschlüssel die meiste Kapazität verbrauchen.

Grundlegendes zu Schreibdrosselung und Gegendruck im globalen sekundären Index (GSI) in DynamoDB

Die GSI-Gegendruckdrosselung stellt eines der komplexesten Drosselungsszenarien in DynamoDB dar, da sie eine indirekte Beziehung zwischen Schreibvorgängen und Drosselung herstellt: Ihre Anwendung schreibt in eine Basistabelle, wird jedoch aufgrund von Kapazitätsbeschränkungen bei einem oder mehreren Indizes gedrosselt.

Grundlegendes zur GSI-Gegendruckdrosselung

Wenn Sie in eine DynamoDB-Tabelle schreiben, werden alle globalen Sekundärindizes (GSIs) für diese Tabelle asynchron unter Verwendung eines eventuell konsistenten Modells aktualisiert. Wenn ein GSI nicht über genügend Kapazität verfügt, um diese Aktualisierungen zu verarbeiten, drosselt DynamoDB Schreibvorgänge in die Basistabelle, um Datenkonsistenz aufrechtzuerhalten. Dies wird als GSI-Gegendruck bezeichnet. Weitere Informationen zur GSIs Funktionsweise finden Sie unter Globale Sekundärindizes in DynamoDB.

Im Gegensatz zur direkten Tabellendrosselung, bei der die Ressource, auf die zugegriffen wird, auch die Drosselung verursacht, erzeugt der GSI-Gegendruck eine Abhängigkeit zwischen der Basistabelle und ihren Indizes. Selbst wenn Ihre Basistabelle über ausreichend Kapazität verfügt, werden Schreibvorgänge gedrosselt, wenn ein zugeordneter GSI das Aktualisierungsvolumen nicht verarbeiten kann. Es ist besonders wichtig, diesen Zusammenhang zu verstehen, da Einschränkungen auf Partitionsebene unabhängig sowohl für die Basistabelle als auch für jeden GSI gelten — jede bzw. jeder hat ihre eigene Partitionsstruktur und entsprechende Durchsatzlimits.

Die GSI-Partitionierung basiert auf dem Partitionsschlüssel des GSI, der sich häufig vom Partitionsschlüssel der Basistabelle unterscheidet. Selbst wenn der Zugriff auf die Basistabelle perfekt auf die Partitionen verteilt ist, konzentrieren sich Ihre GSI-Aktualisierungen möglicherweise trotzdem auf bestimmte Partitionen, wodurch kritische Punkte im GSI entstehen. Allgemeine bewährte Methoden für das Design von Partitionsschlüsseln für beide Tabellen und GSIs finden Sie unter DynamoDB-Partitionsschlüsseldesign.

Wenn Ihre Basistabelle beispielsweise `customerId` als Partitionsschlüssel (gleichmäßig verteilt) verwendet, Ihr GSI jedoch `status` (mit begrenzten möglichen Werten wie „aktiv“, „ausstehend“, „geschlossen“), können Aktualisierungen von Elementen mit gängigen Statuswerten GSI-Hot-Partitionen erzeugen, selbst wenn der Zugriff auf die Basistabelle ausgewogen ist. Dies führt zu einem besonders schwierigen Szenario, in dem Ihre Anwendung aufgrund von GSI-Hot-Partitionen möglicherweise gedrosselt wird, obwohl sowohl die Basistabelle als auch der GSI über ausreichende Gesamtkapazität verfügen und das Zugriffsmuster der Basistabelle gut verteilt zu sein scheint.

Obwohl die Drosselungsausnahme auf den GSI (über `ResourceArn`) verweist, ist die eigentliche Operation, die gedrosselt wird, das Schreiben in die Basistabelle. Dies kann verwirrend sein, weil Ihre Anwendung zwar in die Basistabelle schreibt, aber eine Ausnahme bezüglich des GSI erhält.

Arten von GSI-Drosselung

Die Drosselung des GSI-Gegendrucks äußert sich je nach der spezifischen Kapazitätsbeschränkung in unterschiedlichen Arten von Ausnahmen:

- Die vom GSI bereitgestellte Kapazität wurde überschritten: Tritt auf, wenn der GSI nicht über ausreichende Schreibkapazitätseinheiten verfügt, um Aktualisierungen aus Basistabellenoperationen zu verarbeiten. Dies führt zu einer `ProvisionedThroughputExceededException` mit dem Grund `IndexWriteProvisionedThroughputExceeded` und der `ResourceArn` weist direkt auf den GSI hin, bei dem Kapazitätsbeschränkungen vorliegen.

- Maximaler GSI-On-Demand-Durchsatz wurde überschritten: Tritt auf, wenn GSI-Schreibvorgänge die konfigurierten Höchstgrenzen für On-Demand-Tabellen überschreiten. Dies führt zu einer `ThrottlingException` mit dem Grund `IndexWriteMaxOnDemandThroughputExceeded`, die den spezifischen GSI mit konfigurierten Durchsatzbeschränkungen angibt.
- Die GSI-Partitionslimits wurden überschritten: Tritt auf, wenn einzelne GSI-Partitionen ihre Durchsatzlimits überschreiten (Hot-Partitionen), auch wenn die gesamte GSI-Kapazität ausreichend erscheint. Dadurch wird eine `ThrottlingException` mit dem Grund `IndexWriteKeyRangeThroughputExceeded` generiert, die auf Probleme mit Hot-Partitionen bei dem GSI hinweist, der im `ResourceArn` identifiziert wurde. Dies ist besonders wichtig, da sich die GSI-Partitionsverteilung erheblich von der Partitionsverteilung der Basistabelle unterscheiden kann, was ggf. zu kritischen Punkten im GSI führt, selbst wenn der Zugriff auf die Basistabelle gleichmäßig verteilt ist.
- Die GSI-Kontolimits wurden überschritten: Wird ausgelöst, wenn Schreibvorgänge in einen bestimmten GSI die auf Kontoebene festgelegten regionalen Durchsatzgrenzen pro Tabelle (oder jedes einzelnen GSI innerhalb dieser Tabelle) überschreiten. DynamoDB gibt eine `ThrottlingException` mit dem Grund `IndexWriteAccountLimitExceeded` zurück und verweist auf den GSI, der die Nutzung über die Kontolimits hinaus verursacht hat. Diese Drosselung erfolgt unabhängig für jeden GSI, der das Limit überschreitet. Informationen zu Service Quotas pro Tabelle, Konto und Region finden Sie unter the section called “Kontingente”.

CloudWatch-Drosselungsmetriken

Diese Seite bietet einen umfassenden Überblick über CloudWatch-Metriken, die speziell darauf ausgelegt sind, Sie bei der Identifizierung, Diagnose und Lösung von Drosselungsproblemen in Ihren DynamoDB-Tabellen und -Indizes zu unterstützen.

Allgemeine Drosselungsmetriken

- `ThrottledRequests`
  - Werden um eins erhöht, wenn ein Ereignis innerhalb einer Anforderung gedrosselt wird, unabhängig davon, wie viele einzelne Ereignisse innerhalb dieser Anforderung gedrosselt werden. Wenn z. B. ein Element in einer Tabelle mit globalen sekundären Indizes (GSIs) aktualisiert wird, liegen mehrere Ereignisse vor: eine Schreiboperation in die Basistabelle und eine Schreiboperation in jeden Index. Wenn eines dieser einzelnen Ereignisse gedrosselt wird, wird die Metrik `ThrottledRequests` nur einmal erhöht.

Es ist wichtig, dieses Verhalten bei der Überwachung und Problembehandlung der DynamoDB-Leistung zu verstehen, da es das wahre Ausmaß der Drosselung verschleiern kann. Um umfassendere Einblicke zu erhalten, vergleichen Sie die Metrik `ThrottledRequests` mit spezifischen Metriken auf Ereignisebene wie `ReadThrottleEvents` und `WriteThrottleEvents` sowie zielgerichteten Metriken wie zum Beispiel `ReadKeyRangeThroughputThrottleEvents`. Eine vollständige Liste dieser ursachenspezifischen Metriken finden Sie auf dieser Seite. Jede Metrik entspricht bestimmten Drosselungsgründen, die in der Drosselungsausnahme erfasst werden. Anleitungen zum Abrufen und Interpretieren dieser Gründe bei Drosselungsereignissen finden Sie im Abschnitt the section called “Drosselungsdiagnose”, der Anleitungen zur Identifizierung und Behebung der Hauptursachen von Drosselungsproblemen enthält.

- `ReadThrottleEvents`
  - Achten Sie auf Anforderungen, die die bereitgestellte RCU für eine Tabelle oder einen globalen sekundären Index überschreiten.
- `WriteThrottleEvents`
  - Achten Sie auf Anforderungen, die die bereitgestellte WCU für eine Tabelle oder einen globalen sekundären Index überschreiten.

Detaillierte Drosselungsmetriken nach Ursache

On-Demand-Durchsatzdrosselung

- `ReadMaxOnDemandThroughputThrottleEvents`
  - Anzahl der Leseanforderungen, die aufgrund des maximalen On-Demand-Durchsatzes gedrosselt wurden.
- `WriteMaxOnDemandThroughputThrottleEvents`
  - Anzahl der Schreibanforderungen, die aufgrund des maximalen On-Demand-Durchsatzes gedrosselt wurden.

Drosselung auf Kontoebene

- `ReadAccountLimitThrottleEvents`
  - Anzahl der Leseanforderungen, die aufgrund von Kontobeschränkungen gedrosselt wurden.
- `WriteAccountLimitThrottleEvents`
  - Anzahl der Schreibanforderungen, die aufgrund von Kontobeschränkungen gedrosselt wurden.

Drosselung auf Partitionsebene

- `ReadKeyRangeThroughputThrottleEvents`
  - Anzahl der Leseanforderungen, die aufgrund von Partitionsbeschränkungen gedrosselt wurden.
- `WriteKeyRangeThroughputThrottleEvents`
  - Anzahl der Schreibanforderungen, die aufgrund von Partitionsbeschränkungen gedrosselt wurden.

Metriken zur Kapazitätsanalyse

- `OnlineIndexConsumedWriteCapacity`
  - Wenn Sie einer vorhandenen Tabelle einen neuen GSI hinzufügen, führt DynamoDB eine Backfill-Operation durch, bei der Daten aus der Basistabelle in den neuen Index kopiert werden. Diese Operation verbraucht Schreibkapazitätseinheiten. Die Metrik `OnlineIndexConsumedWriteCapacity` verfolgt diesen spezifischen Verbrauch.

    Dieser Verbrauch erfolgt unabhängig von und zusätzlich zu den regulären Schreibvorgängen, die von `ConsumedWriteCapacityUnits` verfolgt werden. Die reguläre Metrik `ConsumedWriteCapacityUnits` für einen GSI enthält nicht die während der ersten Indexerstellung verbrauchte Schreibkapazität.
- `ProvisionedReadCapacityUnits` und `ProvisionedWriteCapacityUnits`
  - Sehen Sie sich an, wie viele bereitgestellte Lese- oder Schreibkapazitätseinheiten im angegebenen Zeitraum für eine Tabelle oder einen bestimmten globalen sekundären Index verbraucht wurden.
  - Beachten Sie, dass die `TableName`-Dimension standardmäßig nur für die Tabelle `ProvisionedReadCapacityUnits` zurückgibt. Um die Anzahl der bereitgestellten Lese- oder Schreibkapazitätseinheiten für einen globalen sekundären Index anzuzeigen, müssen Sie sowohl `TableName` als auch `GlobalSecondaryIndexName` angeben.
- `ConsumedReadCapacityUnits` und `ConsumedWriteCapacityUnits`
  - Sehen Sie sich an, wie viele Lese- oder Schreibkapazitätseinheiten im angegebenen Zeitraum verbraucht wurden. `ConsumedWriteCapacityUnits` enthält nicht die während der ersten Indexerstellung verbrauchte Schreibkapazität.

Weitere Informationen zu DynamoDB-CloudWatch-Metriken finden Sie unter DynamoDB-Metriken und -Dimensionen.

Anhang DynamoDB

Themen

- Behebung von Problemen SSL/TLS beim Verbindungsaufbau mit DynamoDB
- Beispieltabellen und -daten zur Verwendung in DynamoDB
- Erstellen von Beispieltabellen und Hochladen von Daten in DynamoDB
- DynamoDB-Beispielanwendung mit dem: AWS SDK für Python (Boto) Tic-tac-toe
- Reservierte Wörter in DynamoDB
- AWS SDK-Beispiele für Java 1.x
- AWS Beispiele für SDK for Go 1.x
- AWS Beispiele für SDK für Node.js 2.x

Behebung von Problemen SSL/TLS beim Verbindungsaufbau mit DynamoDB

Amazon DynamoDB ist dabei, unsere Endpunkte auf sichere Zertifikate umzustellen, die von der Amazon-Trust-Services-(ATS)-Zertifizierungsstelle anstelle einer Drittanbieter-Zertifizierungsstelle signiert wurden. Im Dezember 2017 haben wir die Region EU-WEST-3 (Paris) mit den von Amazon Trust Services ausgestellten sicheren Zertifikaten ins Leben gerufen. Alle neuen Regionen, die nach Dezember 2017 gestartet wurden, verfügen über Endpunkte mit den von Amazon Trust Services ausgestellten Zertifikaten. In diesem Handbuch erfahren Sie, wie Sie SSL/TLS-Verbindungsprobleme überprüfen und beheben.

Testen Ihrer Anwendung oder Ihres Services

Die meisten AWS SDKs Befehlszeilenschnittstellen (CLIs) unterstützen die Amazon Trust Services Certificate Authority. Wenn Sie eine Version des AWS SDK für Python oder CLI verwenden, die vor dem 29. Oktober 2013 veröffentlicht wurde, müssen Sie ein Upgrade durchführen. .NET, Java, PHP JavaScript, Go SDKs und C++ bieten CLIs keine Bündelung von Zertifikaten, ihre Zertifikate stammen aus dem zugrunde liegenden Betriebssystem. Das Ruby-SDK enthält CAs seit dem 10. Juni 2015 mindestens eines der erforderlichen. Vor diesem Datum hat das Ruby V2 SDK keine Zertifikate gebündelt. Wenn Sie eine nicht unterstützte, benutzerdefinierte oder modifizierte Version des AWS SDK verwenden oder wenn Sie einen benutzerdefinierten Trust Store verwenden, verfügen Sie

möglicherweise nicht über den Support, den Sie für die Amazon Trust Services Certificate Authority benötigen.

Um den Zugriff auf DynamoDB Endpunkte zu validieren, müssen Sie einen Test entwickeln, der auf die DynamoDB-API oder die DynamoDB Streams-API in der Region EU-WEST-3 zugreift und überprüfen, ob der TLS-Handshake erfolgreich ist. Die spezifischen Endpunkte, auf die Sie in einem solchen Test zugreifen müssen, sind:

- DynamoDB: https://dynamodb.eu-west-3.amazonaws.com
- DynamoDB Streams: https://streams.dynamodb.eu-west-3.amazonaws.com

Wenn Ihre Anwendung die Amazon-Trust-Services-Zertifizierungsstelle nicht unterstützt, wird einer der folgenden Fehler angezeigt:

- SSL/TLS Verhandlungsfehler
- Es dauert lange, bis Ihre Software einen Fehler erhält, der auf ein Fehlschlagen der SSL/TLS Verhandlungen hindeutet. Die Verzögerungszeit hängt von der Wiederholungsstrategie und der Timeout-Konfiguration Ihres Clients ab.

Testen des Client-Browsers

Um zu überprüfen, ob Ihr Browser eine Verbindung zu Amazon DynamoDB herstellen kann, öffnen Sie die folgende URL:. https://dynamodb.eu-west-3.amazonaws.com Wenn der Test erfolgreich ist, sehen Sie eine Meldung wie diese:

```
healthy: dynamodb.eu-west-3.amazonaws.com
```

Wenn der Test nicht erfolgreich ist, wird ein Fehler ähnlich dem folgenden angezeigt:https://untrusted-root.badssl.com/.

Aktualisieren des Software-Anwendungsclients

Anwendungen, die auf DynamoDB- oder DynamoDB Streams-API-Endpunkte zugreifen (ob über Browser oder programmgesteuert), müssen die Liste der vertrauenswürdigen Zertifizierungsstellen auf den Client-Computern aktualisieren, sofern sie nicht bereits eine der folgenden Funktionen unterstützen: CAs

- Amazon Root CA 1

- Starfield Services Root Certificate Authority – G2
- Starfield Class 2 Certification Authority

Wenn die Clients bereits EINEM der oben genannten drei CAs vertrauen, vertrauen diese den von DynamoDB verwendeten Zertifikaten, und es ist keine Aktion erforderlich. Wenn Ihre Clients jedoch noch keinem der oben genannten Punkte vertrauen CAs, schlagen die HTTPS-Verbindungen zu den DynamoDB- oder DynamoDB Streams fehl. APIs Weitere Informationen finden Sie in diesem Blogbeitrag: https://aws.amazon.com/blogs/how-to-prepare-forsecurity/ - -/. aws-move-to-its own-certificate-authority

Aktualisieren Ihres Clientbrowsers

Sie können das Zertifikatspaket in Ihrem Browser einfach aktualisieren, indem Sie Ihren Browser aktualisieren. Anweisungen für die gängigsten Browser finden Sie auf den Webseiten der Browser:

- Chrome: https://support.google.com/chrome/Antwort/95414? hl=de
- Firefox: https://support.mozilla. org/en-US/kb/update- firefox-latest-version
- Safari: https://support.apple.com/en-us/HT204416
- Internet Explorer: https://support.microsoft.com/en-us/hilfe/17295/ -version #ie =andere windows-internet-explorer-which

Manuelles Aktualisieren Ihres Zertifikatpakets

Wenn Sie nicht auf die DynamoDB-API oder die DynamoDB-Streams-API zugreifen können, müssen Sie Ihr Zertifikatpaket aktualisieren. Dazu müssen Sie mindestens eine der erforderlichen Dateien importieren. CAs Sie finden diese unter https://www.amazontrust.com/repository/.

Die folgenden Betriebssysteme und Programmiersprachen unterstützen Amazon-Trust-Services-Zertifikate:

- Windows Versionen, auf denen Updates vom Januar 2005 oder höher installiert sind, Windows Vista, Windows 7, Windows Server 2008 oder neuere Versionen.
- MacOS X 10.4 mit Java für MacOS X 10.4 Release 5, MacOS X 10.5 und neuere Versionen.
- Red Hat Enterprise Linux 5 (März 2007), Linux 6 und Linux 7 und CentOS 5, CentOS 6 und CentOS 7
- Ubuntu 8.10

- Debian 5.0
- Amazon Linux (Alle Versionen)
- Java 1.4.2_12, Java 5 Update 2 und alle neueren Versionen, einschließlich Java 6, Java 7 und Java 8

Wenn Sie immer noch keine Verbindung herstellen können, schlagen Sie in der Softwaredokumentation oder beim Betriebssystemanbieter nach, oder wenden Sie sich an AWS Support https://aws.amazon.com/Support, um weitere Unterstützung zu erhalten.

Beispieltabellen und -daten zur Verwendung in DynamoDB

Im Amazon-DynamoDB-Entwicklerhandbuch werden die verschiedene Aspekte von DynamoDB anhand von Beispieltabellen veranschaulicht.

Tabellenname	Primärschlüssel
ProductCatalog	Einfacher Primärschlüssel: • `Id` (Zahl)
Forum	Einfacher Primärschlüssel: • `Name` (Zeichenfolge)
Thread	Zusammengesetzter Primärschlüssel: • `ForumName` (Zeichenfolge) • `Subject` (Zeichenfolge)
Antworten	Zusammengesetzter Primärschlüssel: • `Id` (Zeichenfolge) • `ReplyDateTime` (Zeichenfolge)

Die Tabelle Antwort verfügt über einen globalen sekundären Index mit dem Namen PostedBy-Message-Index. Dieser Index vereinfacht Abfragen für zwei Nicht-Schlüsselattribute der Tabelle Reply.

Indexname	Primärschlüssel
PostedBy-Message-Index	Zusammengesetzter Primärschlüssel: • `PostedBy` (Zeichenfolge) • `Message` (Zeichenfolge)

Weitere Informationen zu diesen Tabellen finden Sie unter Schritt 1: Erstellen einer Tabelle in DynamoDB und Schritt 2: Schreiben von Daten in eine DynamoDB-Tabelle.

Beispieldatendateien

Themen

- ProductCatalog – Beispieldaten
- Forum – Beispieldaten
- Thread – Beispieldaten
- Reply – Beispieldaten

In den folgenden Abschnitten werden die Beispieldatendateien für das Laden der Tabellen ProductCatalog, Forum, Thread und Reply veranschaulicht.

Jede Datendatei enthält mehrere `PutRequest`-Elemente mit jeweils einem einzelnen Element. Diese `PutRequest`-Elemente dienen als Eingabe für die `BatchWriteItem`-Operation unter Verwendung der AWS Command Line Interface (AWS CLI).

ProductCatalog – Beispieldaten

```
{
    "ProductCatalog": [
        {
            "PutRequest": {
                "Item": {
                    "Id": {
                        "N": "101"
                    },
                    "Title": {
                        "S": "Book 101 Title"
                    },
```

```
                "ISBN": {
                    "S": "111-1111111111"
                },
                "Authors": {
                    "L": [
                        {
                            "S": "Author1"
                        }
                    ]
                },
                "Price": {
                    "N": "2"
                },
                "Dimensions": {
                    "S": "8.5 x 11.0 x 0.5"
                },
                "PageCount": {
                    "N": "500"
                },
                "InPublication": {
                    "BOOL": true
                },
                "ProductCategory": {
                    "S": "Book"
                }
            }
        }
    },
    {
        "PutRequest": {
            "Item": {
                "Id": {
                    "N": "102"
                },
                "Title": {
                    "S": "Book 102 Title"
                },
                "ISBN": {
                    "S": "222-2222222222"
                },
                "Authors": {
                    "L": [
                        {
                            "S": "Author1"
```

```
                        },
                        {
                            "S": "Author2"
                        }
                    ]
                },
                "Price": {
                    "N": "20"
                },
                "Dimensions": {
                    "S": "8.5 x 11.0 x 0.8"
                },
                "PageCount": {
                    "N": "600"
                },
                "InPublication": {
                    "BOOL": true
                },
                "ProductCategory": {
                    "S": "Book"
                }
            }
        }
    },
    {
        "PutRequest": {
            "Item": {
                "Id": {
                    "N": "103"
                },
                "Title": {
                    "S": "Book 103 Title"
                },
                "ISBN": {
                    "S": "333-3333333333"
                },
                "Authors": {
                    "L": [
                        {
                            "S": "Author1"
                        },
                        {
                            "S": "Author2"
                        }
```

```
                    ]
                },
                "Price": {
                    "N": "2000"
                },
                "Dimensions": {
                    "S": "8.5 x 11.0 x 1.5"
                },
                "PageCount": {
                    "N": "600"
                },
                "InPublication": {
                    "BOOL": false
                },
                "ProductCategory": {
                    "S": "Book"
                }
            }
        }
    },
    {
        "PutRequest": {
            "Item": {
                "Id": {
                    "N": "201"
                },
                "Title": {
                    "S": "18-Bike-201"
                },
                "Description": {
                    "S": "201 Description"
                },
                "BicycleType": {
                    "S": "Road"
                },
                "Brand": {
                    "S": "Mountain A"
                },
                "Price": {
                    "N": "100"
                },
                "Color": {
                    "L": [
                        {
```

```
                            "S": "Red"
                        },
                        {
                            "S": "Black"
                        }
                    ]
                },
                "ProductCategory": {
                    "S": "Bicycle"
                }
            }
        }
    },
    {
        "PutRequest": {
            "Item": {
                "Id": {
                    "N": "202"
                },
                "Title": {
                    "S": "21-Bike-202"
                },
                "Description": {
                    "S": "202 Description"
                },
                "BicycleType": {
                    "S": "Road"
                },
                "Brand": {
                    "S": "Brand-Company A"
                },
                "Price": {
                    "N": "200"
                },
                "Color": {
                    "L": [
                        {
                            "S": "Green"
                        },
                        {
                            "S": "Black"
                        }
                    ]
                },
```

```
                "ProductCategory": {
                    "S": "Bicycle"
                }
            }
        }
    },
    {
        "PutRequest": {
            "Item": {
                "Id": {
                    "N": "203"
                },
                "Title": {
                    "S": "19-Bike-203"
                },
                "Description": {
                    "S": "203 Description"
                },
                "BicycleType": {
                    "S": "Road"
                },
                "Brand": {
                    "S": "Brand-Company B"
                },
                "Price": {
                    "N": "300"
                },
                "Color": {
                    "L": [
                        {
                            "S": "Red"
                        },
                        {
                            "S": "Green"
                        },
                        {
                            "S": "Black"
                        }
                    ]
                },
                "ProductCategory": {
                    "S": "Bicycle"
                }
            }
```

```
            }
        },
        {
            "PutRequest": {
                "Item": {
                    "Id": {
                        "N": "204"
                    },
                    "Title": {
                        "S": "18-Bike-204"
                    },
                    "Description": {
                        "S": "204 Description"
                    },
                    "BicycleType": {
                        "S": "Mountain"
                    },
                    "Brand": {
                        "S": "Brand-Company B"
                    },
                    "Price": {
                        "N": "400"
                    },
                    "Color": {
                        "L": [
                            {
                                "S": "Red"
                            }
                        ]
                    },
                    "ProductCategory": {
                        "S": "Bicycle"
                    }
                }
            }
        },
        {
            "PutRequest": {
                "Item": {
                    "Id": {
                        "N": "205"
                    },
                    "Title": {
                        "S": "18-Bike-204"
```

```
                    },
                    "Description": {
                        "S": "205 Description"
                    },
                    "BicycleType": {
                        "S": "Hybrid"
                    },
                    "Brand": {
                        "S": "Brand-Company C"
                    },
                    "Price": {
                        "N": "500"
                    },
                    "Color": {
                        "L": [
                            {
                                "S": "Red"
                            },
                            {
                                "S": "Black"
                            }
                        ]
                    },
                    "ProductCategory": {
                        "S": "Bicycle"
                    }
                }
            }
        }
    ]
}
```

Forum – Beispieldaten

```
{
    "Forum": [
        {
            "PutRequest": {
                "Item": {
                    "Name": {"S":"Amazon DynamoDB"},
                    "Category": {"S":"Amazon Web Services"},
                    "Threads": {"N":"2"},
                    "Messages": {"N":"4"},
```

```
                    "Views": {"N":"1000"}
                }
            }
        },
        {
            "PutRequest": {
                "Item": {
                    "Name": {"S":"Amazon S3"},
                    "Category": {"S":"Amazon Web Services"}
                }
            }
        }
    ]
}
```

Thread – Beispieldaten

```
{
    "Thread": [
        {
            "PutRequest": {
                "Item": {
                    "ForumName": {
                        "S": "Amazon DynamoDB"
                    },
                    "Subject": {
                        "S": "DynamoDB Thread 1"
                    },
                    "Message": {
                        "S": "DynamoDB thread 1 message"
                    },
                    "LastPostedBy": {
                        "S": "User A"
                    },
                    "LastPostedDateTime": {
                        "S": "2015-09-22T19:58:22.514Z"
                    },
                    "Views": {
                        "N": "0"
                    },
                    "Replies": {
                        "N": "0"
                    },
```

```
                "Answered": {
                    "N": "0"
                },
                "Tags": {
                    "L": [
                        {
                            "S": "index"
                        },
                        {
                            "S": "primarykey"
                        },
                        {
                            "S": "table"
                        }
                    ]
                }
            }
        }
    },
    {
        "PutRequest": {
            "Item": {
                "ForumName": {
                    "S": "Amazon DynamoDB"
                },
                "Subject": {
                    "S": "DynamoDB Thread 2"
                },
                "Message": {
                    "S": "DynamoDB thread 2 message"
                },
                "LastPostedBy": {
                    "S": "User A"
                },
                "LastPostedDateTime": {
                    "S": "2015-09-15T19:58:22.514Z"
                },
                "Views": {
                    "N": "3"
                },
                "Replies": {
                    "N": "0"
                },
                "Answered": {
```

```
                    "N": "0"
                },
                "Tags": {
                    "L": [
                        {
                            "S": "items"
                        },
                        {
                            "S": "attributes"
                        },
                        {
                            "S": "throughput"
                        }
                    ]
                }
            }
        }
    },
    {
        "PutRequest": {
            "Item": {
                "ForumName": {
                    "S": "Amazon S3"
                },
                "Subject": {
                    "S": "S3 Thread 1"
                },
                "Message": {
                    "S": "S3 thread 1 message"
                },
                "LastPostedBy": {
                    "S": "User A"
                },
                "LastPostedDateTime": {
                    "S": "2015-09-29T19:58:22.514Z"
                },
                "Views": {
                    "N": "0"
                },
                "Replies": {
                    "N": "0"
                },
                "Answered": {
                    "N": "0"
```

```
                    },
                    "Tags": {
                        "L": [
                            {
                                "S": "largeobjects"
                            },
                            {
                                "S": "multipart upload"
                            }
                        ]
                    }
                }
            }
        }
    ]
}
```

Reply – Beispieldaten

```
{
    "Reply": [
        {
            "PutRequest": {
                "Item": {
                    "Id": {
                        "S": "Amazon DynamoDB#DynamoDB Thread 1"
                    },
                    "ReplyDateTime": {
                        "S": "2015-09-15T19:58:22.947Z"
                    },
                    "Message": {
                        "S": "DynamoDB Thread 1 Reply 1 text"
                    },
                    "PostedBy": {
                        "S": "User A"
                    }
                }
            }
        },
        {
            "PutRequest": {
                "Item": {
                    "Id": {
```

```
                    "S": "Amazon DynamoDB#DynamoDB Thread 1"
                },
                "ReplyDateTime": {
                    "S": "2015-09-22T19:58:22.947Z"
                },
                "Message": {
                    "S": "DynamoDB Thread 1 Reply 2 text"
                },
                "PostedBy": {
                    "S": "User B"
                }
            }
        }
    },
    {
        "PutRequest": {
            "Item": {
                "Id": {
                    "S": "Amazon DynamoDB#DynamoDB Thread 2"
                },
                "ReplyDateTime": {
                    "S": "2015-09-29T19:58:22.947Z"
                },
                "Message": {
                    "S": "DynamoDB Thread 2 Reply 1 text"
                },
                "PostedBy": {
                    "S": "User A"
                }
            }
        }
    },
    {
        "PutRequest": {
            "Item": {
                "Id": {
                    "S": "Amazon DynamoDB#DynamoDB Thread 2"
                },
                "ReplyDateTime": {
                    "S": "2015-10-05T19:58:22.947Z"
                },
                "Message": {
                    "S": "DynamoDB Thread 2 Reply 2 text"
                },
```

```
                    "PostedBy": {
                        "S": "User A"
                    }
                }
            }
        }
    ]
}
```

Erstellen von Beispieltabellen und Hochladen von Daten in DynamoDB

Dieser Anhang enthält Code, um sowohl die Tabellen zu erstellen als auch Daten programmgesteuert hinzuzufügen.

Themen

- Erstellen von Beispieltabellen und Hochladen von Daten mit dem AWS SDK für Java
- Erstellen von Beispieltabellen und Hochladen von Daten mit dem AWS SDK für .NET

Erstellen von Beispieltabellen und Hochladen von Daten mit dem AWS SDK für Java

Das folgende Java-Codebeispiel erstellt Tabellen und lädt Daten in die Tabellen hoch. step-by-stepAnweisungen zur Ausführung dieses Codes mit Eclipse finden Sie unter. Java-Codebeispiele

```
package com.amazonaws.codesamples;

import java.text.SimpleDateFormat;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.Date;
import java.util.HashSet;
import java.util.TimeZone;

import com.amazonaws.services.dynamodbv2.AmazonDynamoDB;
import com.amazonaws.services.dynamodbv2.AmazonDynamoDBClientBuilder;
import com.amazonaws.services.dynamodbv2.document.DynamoDB;
import com.amazonaws.services.dynamodbv2.document.Item;
```

```
import com.amazonaws.services.dynamodbv2.document.Table;
import com.amazonaws.services.dynamodbv2.model.AttributeDefinition;
import com.amazonaws.services.dynamodbv2.model.CreateTableRequest;
import com.amazonaws.services.dynamodbv2.model.KeySchemaElement;
import com.amazonaws.services.dynamodbv2.model.KeyType;
import com.amazonaws.services.dynamodbv2.model.LocalSecondaryIndex;
import com.amazonaws.services.dynamodbv2.model.Projection;
import com.amazonaws.services.dynamodbv2.model.ProjectionType;
import com.amazonaws.services.dynamodbv2.model.ProvisionedThroughput;

public class CreateTablesLoadData {

    static AmazonDynamoDB client = AmazonDynamoDBClientBuilder.standard().build();
    static DynamoDB dynamoDB = new DynamoDB(client);

    static SimpleDateFormat dateFormatter = new SimpleDateFormat("yyyy-MM-
dd'T'HH:mm:ss.SSS'Z'");

    static String productCatalogTableName = "ProductCatalog";
    static String forumTableName = "Forum";
    static String threadTableName = "Thread";
    static String replyTableName = "Reply";

    public static void main(String[] args) throws Exception {

        try {

            deleteTable(productCatalogTableName);
            deleteTable(forumTableName);
            deleteTable(threadTableName);
            deleteTable(replyTableName);

            // Parameter1: table name
            // Parameter2: reads per second
            // Parameter3: writes per second
            // Parameter4/5: partition key and data type
            // Parameter6/7: sort key and data type (if applicable)

            createTable(productCatalogTableName, 10L, 5L, "Id", "N");
            createTable(forumTableName, 10L, 5L, "Name", "S");
            createTable(threadTableName, 10L, 5L, "ForumName", "S", "Subject", "S");
            createTable(replyTableName, 10L, 5L, "Id", "S", "ReplyDateTime", "S");

            loadSampleProducts(productCatalogTableName);
```

```
            loadSampleForums(forumTableName);
            loadSampleThreads(threadTableName);
            loadSampleReplies(replyTableName);

        } catch (Exception e) {
            System.err.println("Program failed:");
            System.err.println(e.getMessage());
        }
        System.out.println("Success.");
    }

    private static void deleteTable(String tableName) {
        Table table = dynamoDB.getTable(tableName);
        try {
            System.out.println("Issuing DeleteTable request for " + tableName);
            table.delete();
            System.out.println("Waiting for " + tableName + " to be deleted...this may
take a while...");
            table.waitForDelete();

        } catch (Exception e) {
            System.err.println("DeleteTable request failed for " + tableName);
            System.err.println(e.getMessage());
        }
    }

    private static void createTable(String tableName, long readCapacityUnits, long
writeCapacityUnits,
            String partitionKeyName, String partitionKeyType) {

        createTable(tableName, readCapacityUnits, writeCapacityUnits, partitionKeyName,
partitionKeyType, null, null);
    }

    private static void createTable(String tableName, long readCapacityUnits, long
writeCapacityUnits,
            String partitionKeyName, String partitionKeyType, String sortKeyName,
String sortKeyType) {

        try {

            ArrayList<KeySchemaElement> keySchema = new ArrayList<KeySchemaElement>();
```

```
            keySchema.add(new
KeySchemaElement().withAttributeName(partitionKeyName).withKeyType(KeyType.HASH)); //
Partition

                            // key

            ArrayList<AttributeDefinition> attributeDefinitions = new
ArrayList<AttributeDefinition>();
            attributeDefinitions
                    .add(new AttributeDefinition().withAttributeName(partitionKeyName)
                            .withAttributeType(partitionKeyType));

            if (sortKeyName != null) {
                keySchema.add(new
KeySchemaElement().withAttributeName(sortKeyName).withKeyType(KeyType.RANGE)); // Sort

                            // key
                attributeDefinitions
                        .add(new
AttributeDefinition().withAttributeName(sortKeyName).withAttributeType(sortKeyType));
            }

            CreateTableRequest request = new
CreateTableRequest().withTableName(tableName).withKeySchema(keySchema)
                    .withProvisionedThroughput(new
ProvisionedThroughput().withReadCapacityUnits(readCapacityUnits)
                            .withWriteCapacityUnits(writeCapacityUnits));

            // If this is the Reply table, define a local secondary index
            if (replyTableName.equals(tableName)) {

                attributeDefinitions
                        .add(new
AttributeDefinition().withAttributeName("PostedBy").withAttributeType("S"));

                ArrayList<LocalSecondaryIndex> localSecondaryIndexes = new
ArrayList<LocalSecondaryIndex>();
                localSecondaryIndexes.add(new
LocalSecondaryIndex().withIndexName("PostedBy-Index")
                        .withKeySchema(
                                new
KeySchemaElement().withAttributeName(partitionKeyName).withKeyType(KeyType.HASH), //
Partition
```

```
                                // key
                                 new
KeySchemaElement().withAttributeName("PostedBy").withKeyType(KeyType.RANGE)) // Sort

                           // key
                          .withProjection(new
Projection().withProjectionType(ProjectionType.KEYS_ONLY)));

                request.setLocalSecondaryIndexes(localSecondaryIndexes);
            }

            request.setAttributeDefinitions(attributeDefinitions);

            System.out.println("Issuing CreateTable request for " + tableName);
            Table table = dynamoDB.createTable(request);
            System.out.println("Waiting for " + tableName + " to be created...this may
take a while...");
            table.waitForActive();

        } catch (Exception e) {
            System.err.println("CreateTable request failed for " + tableName);
            System.err.println(e.getMessage());
        }
    }

    private static void loadSampleProducts(String tableName) {

        Table table = dynamoDB.getTable(tableName);

        try {

            System.out.println("Adding data to " + tableName);

            Item item = new Item().withPrimaryKey("Id", 101).withString("Title", "Book
101 Title")
                    .withString("ISBN", "111-1111111111")
                    .withStringSet("Authors", new
HashSet<String>(Arrays.asList("Author1"))).withNumber("Price", 2)
                    .withString("Dimensions", "8.5 x 11.0 x
0.5").withNumber("PageCount", 500)
                    .withBoolean("InPublication", true).withString("ProductCategory",
"Book");
            table.putItem(item);
```

```
            item = new Item().withPrimaryKey("Id", 102).withString("Title", "Book 102
 Title")
                    .withString("ISBN", "222-2222222222")
                    .withStringSet("Authors", new
 HashSet<String>(Arrays.asList("Author1", "Author2")))
                    .withNumber("Price", 20).withString("Dimensions", "8.5 x 11.0 x
 0.8").withNumber("PageCount", 600)
                    .withBoolean("InPublication", true).withString("ProductCategory",
 "Book");
            table.putItem(item);

            item = new Item().withPrimaryKey("Id", 103).withString("Title", "Book 103
 Title")
                    .withString("ISBN", "333-3333333333")
                    .withStringSet("Authors", new
 HashSet<String>(Arrays.asList("Author1", "Author2")))
                    // Intentional. Later we'll run Scan to find price error. Find
                    // items > 1000 in price.
                    .withNumber("Price", 2000).withString("Dimensions", "8.5 x 11.0 x
 1.5").withNumber("PageCount", 600)
                    .withBoolean("InPublication", false).withString("ProductCategory",
 "Book");
            table.putItem(item);

            // Add bikes.

            item = new Item().withPrimaryKey("Id", 201).withString("Title", "18-
Bike-201")
                    // Size, followed by some title.
                    .withString("Description", "201
 Description").withString("BicycleType", "Road")
                    .withString("Brand", "Mountain A")
                    // Trek, Specialized.
                    .withNumber("Price", 100).withStringSet("Color", new
 HashSet<String>(Arrays.asList("Red", "Black")))
                    .withString("ProductCategory", "Bicycle");
            table.putItem(item);

            item = new Item().withPrimaryKey("Id", 202).withString("Title", "21-
Bike-202")
                    .withString("Description", "202
 Description").withString("BicycleType", "Road")
                    .withString("Brand", "Brand-Company A").withNumber("Price", 200)
```

```
                    .withStringSet("Color", new HashSet<String>(Arrays.asList("Green",
 "Black")))
                    .withString("ProductCategory", "Bicycle");
            table.putItem(item);

            item = new Item().withPrimaryKey("Id", 203).withString("Title", "19-
Bike-203")
                    .withString("Description", "203
 Description").withString("BicycleType", "Road")
                    .withString("Brand", "Brand-Company B").withNumber("Price", 300)
                    .withStringSet("Color", new HashSet<String>(Arrays.asList("Red",
 "Green", "Black")))
                    .withString("ProductCategory", "Bicycle");
            table.putItem(item);

            item = new Item().withPrimaryKey("Id", 204).withString("Title", "18-
Bike-204")
                    .withString("Description", "204
 Description").withString("BicycleType", "Mountain")
                    .withString("Brand", "Brand-Company B").withNumber("Price", 400)
                    .withStringSet("Color", new HashSet<String>(Arrays.asList("Red")))
                    .withString("ProductCategory", "Bicycle");
            table.putItem(item);

            item = new Item().withPrimaryKey("Id", 205).withString("Title", "20-
Bike-205")
                    .withString("Description", "205
 Description").withString("BicycleType", "Hybrid")
                    .withString("Brand", "Brand-Company C").withNumber("Price", 500)
                    .withStringSet("Color", new HashSet<String>(Arrays.asList("Red",
 "Black")))
                    .withString("ProductCategory", "Bicycle");
            table.putItem(item);

        } catch (Exception e) {
            System.err.println("Failed to create item in " + tableName);
            System.err.println(e.getMessage());
        }

    }

    private static void loadSampleForums(String tableName) {

        Table table = dynamoDB.getTable(tableName);
```

```
        try {

            System.out.println("Adding data to " + tableName);

            Item item = new Item().withPrimaryKey("Name", "Amazon DynamoDB")
                    .withString("Category", "Amazon Web
Services").withNumber("Threads", 2).withNumber("Messages", 4)
                    .withNumber("Views", 1000);
            table.putItem(item);

            item = new Item().withPrimaryKey("Name", "Amazon
S3").withString("Category", "Amazon Web Services")
                    .withNumber("Threads", 0);
            table.putItem(item);

        } catch (Exception e) {
            System.err.println("Failed to create item in " + tableName);
            System.err.println(e.getMessage());
        }
    }

    private static void loadSampleThreads(String tableName) {
        try {
            long time1 = (new Date()).getTime() - (7 * 24 * 60 * 60 * 1000); // 7
            // days
            // ago
            long time2 = (new Date()).getTime() - (14 * 24 * 60 * 60 * 1000); // 14
            // days
            // ago
            long time3 = (new Date()).getTime() - (21 * 24 * 60 * 60 * 1000); // 21
            // days
            // ago

            Date date1 = new Date();
            date1.setTime(time1);

            Date date2 = new Date();
            date2.setTime(time2);

            Date date3 = new Date();
            date3.setTime(time3);

            dateFormatter.setTimeZone(TimeZone.getTimeZone("UTC"));
```

```
            Table table = dynamoDB.getTable(tableName);

            System.out.println("Adding data to " + tableName);

            Item item = new Item().withPrimaryKey("ForumName", "Amazon DynamoDB")
                    .withString("Subject", "DynamoDB Thread 1").withString("Message",
"DynamoDB thread 1 message")
                    .withString("LastPostedBy", "User
A").withString("LastPostedDateTime", dateFormatter.format(date2))
                    .withNumber("Views", 0).withNumber("Replies",
0).withNumber("Answered", 0)
                    .withStringSet("Tags", new HashSet<String>(Arrays.asList("index",
"primarykey", "table")));
            table.putItem(item);

            item = new Item().withPrimaryKey("ForumName", "Amazon
DynamoDB").withString("Subject", "DynamoDB Thread 2")
                    .withString("Message", "DynamoDB thread 2
message").withString("LastPostedBy", "User A")
                    .withString("LastPostedDateTime",
dateFormatter.format(date3)).withNumber("Views", 0)
                    .withNumber("Replies", 0).withNumber("Answered", 0)
                    .withStringSet("Tags", new HashSet<String>(Arrays.asList("index",
"partitionkey", "sortkey")));
            table.putItem(item);

            item = new Item().withPrimaryKey("ForumName", "Amazon
S3").withString("Subject", "S3 Thread 1")
                    .withString("Message", "S3 Thread 3
message").withString("LastPostedBy", "User A")
                    .withString("LastPostedDateTime",
dateFormatter.format(date1)).withNumber("Views", 0)
                    .withNumber("Replies", 0).withNumber("Answered", 0)
                    .withStringSet("Tags", new
HashSet<String>(Arrays.asList("largeobjects", "multipart upload")));
            table.putItem(item);

        } catch (Exception e) {
            System.err.println("Failed to create item in " + tableName);
            System.err.println(e.getMessage());
        }

    }
```

```
    private static void loadSampleReplies(String tableName) {
        try {
            // 1 day ago
            long time0 = (new Date()).getTime() - (1 * 24 * 60 * 60 * 1000);
            // 7 days ago
            long time1 = (new Date()).getTime() - (7 * 24 * 60 * 60 * 1000);
            // 14 days ago
            long time2 = (new Date()).getTime() - (14 * 24 * 60 * 60 * 1000);
            // 21 days ago
            long time3 = (new Date()).getTime() - (21 * 24 * 60 * 60 * 1000);

            Date date0 = new Date();
            date0.setTime(time0);

            Date date1 = new Date();
            date1.setTime(time1);

            Date date2 = new Date();
            date2.setTime(time2);

            Date date3 = new Date();
            date3.setTime(time3);

            dateFormatter.setTimeZone(TimeZone.getTimeZone("UTC"));

            Table table = dynamoDB.getTable(tableName);

            System.out.println("Adding data to " + tableName);

            // Add threads.

            Item item = new Item().withPrimaryKey("Id", "Amazon DynamoDB#DynamoDB
Thread 1")
                    .withString("ReplyDateTime", (dateFormatter.format(date3)))
                    .withString("Message", "DynamoDB Thread 1 Reply 1
text").withString("PostedBy", "User A");
            table.putItem(item);

            item = new Item().withPrimaryKey("Id", "Amazon DynamoDB#DynamoDB Thread 1")
                    .withString("ReplyDateTime", dateFormatter.format(date2))
                    .withString("Message", "DynamoDB Thread 1 Reply 2
text").withString("PostedBy", "User B");
            table.putItem(item);
```

```
            item = new Item().withPrimaryKey("Id", "Amazon DynamoDB#DynamoDB Thread 2")
                    .withString("ReplyDateTime", dateFormatter.format(date1))
                    .withString("Message", "DynamoDB Thread 2 Reply 1
 text").withString("PostedBy", "User A");
            table.putItem(item);

            item = new Item().withPrimaryKey("Id", "Amazon DynamoDB#DynamoDB Thread 2")
                    .withString("ReplyDateTime", dateFormatter.format(date0))
                    .withString("Message", "DynamoDB Thread 2 Reply 2
 text").withString("PostedBy", "User A");
            table.putItem(item);

        } catch (Exception e) {
            System.err.println("Failed to create item in " + tableName);
            System.err.println(e.getMessage());

        }
    }

}
```

Erstellen von Beispieltabellen und Hochladen von Daten mit dem AWS SDK für .NET

Das folgende C#-Codebeispiel erstellt Tabellen und lädt Daten in die Tabellen hoch. step-by-stepAnweisungen zur Ausführung dieses Codes in Visual Studio finden Sie unter. .NET-Codebeispiele

```
using System;
using System.Collections.Generic;
using Amazon.DynamoDBv2;
using Amazon.DynamoDBv2.DocumentModel;
using Amazon.DynamoDBv2.Model;
using Amazon.Runtime;
using Amazon.SecurityToken;

namespace com.amazonaws.codesamples
{
    class CreateTablesLoadData
    {
```

```
        private static AmazonDynamoDBClient client = new AmazonDynamoDBClient();

        static void Main(string[] args)
        {
            try
            {
                //DeleteAllTables(client);
                DeleteTable("ProductCatalog");
                DeleteTable("Forum");
                DeleteTable("Thread");
                DeleteTable("Reply");

                // Create tables (using the AWS SDK for .NET low-level API).
                CreateTableProductCatalog();
                CreateTableForum();
                CreateTableThread(); // ForumTitle, Subject */
                CreateTableReply();

                // Load data (using the .NET SDK document API)
                LoadSampleProducts();
                LoadSampleForums();
                LoadSampleThreads();
                LoadSampleReplies();
                Console.WriteLine("Sample complete!");
                Console.WriteLine("Press ENTER to continue");
                Console.ReadLine();
            }
            catch (AmazonServiceException e) { Console.WriteLine(e.Message); }
            catch (Exception e) { Console.WriteLine(e.Message); }
        }

        private static void DeleteTable(string tableName)
        {
            try
            {
                var deleteTableResponse = client.DeleteTable(new DeleteTableRequest()
                {
                    TableName = tableName
                });
                WaitTillTableDeleted(client, tableName, deleteTableResponse);
            }
            catch (ResourceNotFoundException)
            {
                // There is no such table.
```

```
            }
        }

        private static void CreateTableProductCatalog()
        {
            string tableName = "ProductCatalog";

            var response = client.CreateTable(new CreateTableRequest
            {
                TableName = tableName,
                AttributeDefinitions = new List<AttributeDefinition>()
                            {
                                new AttributeDefinition
                                {
                                    AttributeName = "Id",
                                    AttributeType = "N"
                                }
                            },
                KeySchema = new List<KeySchemaElement>()
                            {
                                new KeySchemaElement
                                {
                                    AttributeName = "Id",
                                    KeyType = "HASH"
                                }
                            },
                ProvisionedThroughput = new ProvisionedThroughput
                {
                    ReadCapacityUnits = 10,
                    WriteCapacityUnits = 5
                }
            });

            WaitTillTableCreated(client, tableName, response);
        }

        private static void CreateTableForum()
        {
            string tableName = "Forum";

            var response = client.CreateTable(new CreateTableRequest
            {
                TableName = tableName,
                AttributeDefinitions = new List<AttributeDefinition>()
```

```
                        {
                            new AttributeDefinition
                            {
                                AttributeName = "Name",
                                AttributeType = "S"
                            }
                        },
            KeySchema = new List<KeySchemaElement>()
                        {
                            new KeySchemaElement
                            {
                                AttributeName = "Name", // forum Title
                                KeyType = "HASH"
                            }
                        },
            ProvisionedThroughput = new ProvisionedThroughput
            {
                ReadCapacityUnits = 10,
                WriteCapacityUnits = 5
            }
        });

        WaitTillTableCreated(client, tableName, response);
    }

    private static void CreateTableThread()
    {
        string tableName = "Thread";

        var response = client.CreateTable(new CreateTableRequest
        {
            TableName = tableName,
            AttributeDefinitions = new List<AttributeDefinition>()
                        {
                            new AttributeDefinition
                            {
                                AttributeName = "ForumName", // Hash attribute
                                AttributeType = "S"
                            },
                            new AttributeDefinition
                            {
                                AttributeName = "Subject",
                                AttributeType = "S"
                            }
```

```
                        },
            KeySchema = new List<KeySchemaElement>()
                        {
                            new KeySchemaElement
                            {
                                AttributeName = "ForumName", // Hash attribute
                                KeyType = "HASH"
                            },
                            new KeySchemaElement
                            {
                                AttributeName = "Subject", // Range attribute
                                KeyType = "RANGE"
                            }
                        },
            ProvisionedThroughput = new ProvisionedThroughput
            {
                ReadCapacityUnits = 10,
                WriteCapacityUnits = 5
            }
        });

        WaitTillTableCreated(client, tableName, response);
    }

    private static void CreateTableReply()
    {
        string tableName = "Reply";
        var response = client.CreateTable(new CreateTableRequest
        {
            TableName = tableName,
            AttributeDefinitions = new List<AttributeDefinition>()
                        {
                            new AttributeDefinition
                            {
                                AttributeName = "Id",
                                AttributeType = "S"
                            },
                            new AttributeDefinition
                            {
                                AttributeName = "ReplyDateTime",
                                AttributeType = "S"
                            },
                            new AttributeDefinition
                            {
```

```
                                    AttributeName = "PostedBy",
                                    AttributeType = "S"
                                }
                            },
                KeySchema = new List<KeySchemaElement>()
                            {
                                new KeySchemaElement()
                                {
                                    AttributeName = "Id",
                                    KeyType = "HASH"
                                },
                                new KeySchemaElement()
                                {
                                    AttributeName = "ReplyDateTime",
                                    KeyType = "RANGE"
                                }
                            },
                LocalSecondaryIndexes = new List<LocalSecondaryIndex>()
                            {
                                new LocalSecondaryIndex()
                                {
                                    IndexName = "PostedBy_index",


                                    KeySchema = new List<KeySchemaElement>() {
                                        new KeySchemaElement() {
                                            AttributeName = "Id", KeyType = "HASH"
                                        },
                                        new KeySchemaElement() {
                                            AttributeName = "PostedBy", KeyType =
"RANGE"
                                        }
                                    },
                                    Projection = new Projection() {
                                        ProjectionType = ProjectionType.KEYS_ONLY
                                    }
                                }
                            },
                ProvisionedThroughput = new ProvisionedThroughput
                {
                    ReadCapacityUnits = 10,
                    WriteCapacityUnits = 5
                }
            });
```

```
            WaitTillTableCreated(client, tableName, response);
        }

        private static void WaitTillTableCreated(AmazonDynamoDBClient client, string
tableName,
                                 CreateTableResponse response)
        {
            var tableDescription = response.TableDescription;

            string status = tableDescription.TableStatus;

            Console.WriteLine(tableName + " - " + status);

            // Let us wait until table is created. Call DescribeTable.
            while (status != "ACTIVE")
            {
                System.Threading.Thread.Sleep(5000); // Wait 5 seconds.
                try
                {
                    var res = client.DescribeTable(new DescribeTableRequest
                    {
                        TableName = tableName
                    });
                    Console.WriteLine("Table name: {0}, status: {1}",
res.Table.TableName,
                                      res.Table.TableStatus);
                    status = res.Table.TableStatus;
                }
                // Try-catch to handle potential eventual-consistency issue.
                catch (ResourceNotFoundException)
                { }
            }
        }

        private static void WaitTillTableDeleted(AmazonDynamoDBClient client, string
tableName,
                                 DeleteTableResponse response)
        {
            var tableDescription = response.TableDescription;

            string status = tableDescription.TableStatus;

            Console.WriteLine(tableName + " - " + status);
```

```
            // Let us wait until table is created. Call DescribeTable
            try
            {
                while (status == "DELETING")
                {
                    System.Threading.Thread.Sleep(5000); // wait 5 seconds

                    var res = client.DescribeTable(new DescribeTableRequest
                    {
                        TableName = tableName
                    });
                    Console.WriteLine("Table name: {0}, status: {1}",
res.Table.TableName,
                              res.Table.TableStatus);
                    status = res.Table.TableStatus;
                }
            }
            catch (ResourceNotFoundException)
            {
                // Table deleted.
            }
        }

        private static void LoadSampleProducts()
        {
            Table productCatalogTable = Table.LoadTable(client, "ProductCatalog");
            // ********** Add Books ********************
            var book1 = new Document();
            book1["Id"] = 101;
            book1["Title"] = "Book 101 Title";
            book1["ISBN"] = "111-1111111111";
            book1["Authors"] = new List<string> { "Author 1" };
            book1["Price"] = -2; // *** Intentional value. Later used to illustrate
scan.
            book1["Dimensions"] = "8.5 x 11.0 x 0.5";
            book1["PageCount"] = 500;
            book1["InPublication"] = true;
            book1["ProductCategory"] = "Book";
            productCatalogTable.PutItem(book1);

            var book2 = new Document();

            book2["Id"] = 102;
```

```
            book2["Title"] = "Book 102 Title";
            book2["ISBN"] = "222-2222222222";
            book2["Authors"] = new List<string> { "Author 1", "Author 2" }; ;
            book2["Price"] = 20;
            book2["Dimensions"] = "8.5 x 11.0 x 0.8";
            book2["PageCount"] = 600;
            book2["InPublication"] = true;
            book2["ProductCategory"] = "Book";
            productCatalogTable.PutItem(book2);

            var book3 = new Document();
            book3["Id"] = 103;
            book3["Title"] = "Book 103 Title";
            book3["ISBN"] = "333-3333333333";
            book3["Authors"] = new List<string> { "Author 1", "Author2", "Author
3" }; ;
            book3["Price"] = 2000;
            book3["Dimensions"] = "8.5 x 11.0 x 1.5";
            book3["PageCount"] = 700;
            book3["InPublication"] = false;
            book3["ProductCategory"] = "Book";
            productCatalogTable.PutItem(book3);

            // ************ Add bikes. ******************
            var bicycle1 = new Document();
            bicycle1["Id"] = 201;
            bicycle1["Title"] = "18-Bike 201"; // size, followed by some title.
            bicycle1["Description"] = "201 description";
            bicycle1["BicycleType"] = "Road";
            bicycle1["Brand"] = "Brand-Company A"; // Trek, Specialized.
            bicycle1["Price"] = 100;
            bicycle1["Color"] = new List<string> { "Red", "Black" };
            bicycle1["ProductCategory"] = "Bike";
            productCatalogTable.PutItem(bicycle1);

            var bicycle2 = new Document();
            bicycle2["Id"] = 202;
            bicycle2["Title"] = "21-Bike 202Brand-Company A";
            bicycle2["Description"] = "202 description";
            bicycle2["BicycleType"] = "Road";
            bicycle2["Brand"] = "";
            bicycle2["Price"] = 200;
            bicycle2["Color"] = new List<string> { "Green", "Black" };
            bicycle2["ProductCategory"] = "Bicycle";
```

```
            productCatalogTable.PutItem(bicycle2);

            var bicycle3 = new Document();
            bicycle3["Id"] = 203;
            bicycle3["Title"] = "19-Bike 203";
            bicycle3["Description"] = "203 description";
            bicycle3["BicycleType"] = "Road";
            bicycle3["Brand"] = "Brand-Company B";
            bicycle3["Price"] = 300;
            bicycle3["Color"] = new List<string> { "Red", "Green", "Black" };
            bicycle3["ProductCategory"] = "Bike";
            productCatalogTable.PutItem(bicycle3);

            var bicycle4 = new Document();
            bicycle4["Id"] = 204;
            bicycle4["Title"] = "18-Bike 204";
            bicycle4["Description"] = "204 description";
            bicycle4["BicycleType"] = "Mountain";
            bicycle4["Brand"] = "Brand-Company B";
            bicycle4["Price"] = 400;
            bicycle4["Color"] = new List<string> { "Red" };
            bicycle4["ProductCategory"] = "Bike";
            productCatalogTable.PutItem(bicycle4);

            var bicycle5 = new Document();
            bicycle5["Id"] = 205;
            bicycle5["Title"] = "20-Title 205";
            bicycle4["Description"] = "205 description";
            bicycle5["BicycleType"] = "Hybrid";
            bicycle5["Brand"] = "Brand-Company C";
            bicycle5["Price"] = 500;
            bicycle5["Color"] = new List<string> { "Red", "Black" };
            bicycle5["ProductCategory"] = "Bike";
            productCatalogTable.PutItem(bicycle5);
        }

        private static void LoadSampleForums()
        {
            Table forumTable = Table.LoadTable(client, "Forum");

            var forum1 = new Document();
            forum1["Name"] = "Amazon DynamoDB"; // PK
            forum1["Category"] = "Amazon Web Services";
            forum1["Threads"] = 2;
```

```
            forum1["Messages"] = 4;
            forum1["Views"] = 1000;

            forumTable.PutItem(forum1);

            var forum2 = new Document();
            forum2["Name"] = "Amazon S3"; // PK
            forum2["Category"] = "Amazon Web Services";
            forum2["Threads"] = 1;

            forumTable.PutItem(forum2);
        }

        private static void LoadSampleThreads()
        {
            Table threadTable = Table.LoadTable(client, "Thread");

            // Thread 1.
            var thread1 = new Document();
            thread1["ForumName"] = "Amazon DynamoDB"; // Hash attribute.
            thread1["Subject"] = "DynamoDB Thread 1"; // Range attribute.
            thread1["Message"] = "DynamoDB thread 1 message text";
            thread1["LastPostedBy"] = "User A";
            thread1["LastPostedDateTime"] = DateTime.UtcNow.Subtract(new TimeSpan(14,
0, 0, 0));
            thread1["Views"] = 0;
            thread1["Replies"] = 0;
            thread1["Answered"] = false;
            thread1["Tags"] = new List<string> { "index", "primarykey", "table" };

            threadTable.PutItem(thread1);

            // Thread 2.
            var thread2 = new Document();
            thread2["ForumName"] = "Amazon DynamoDB"; // Hash attribute.
            thread2["Subject"] = "DynamoDB Thread 2"; // Range attribute.
            thread2["Message"] = "DynamoDB thread 2 message text";
            thread2["LastPostedBy"] = "User A";
            thread2["LastPostedDateTime"] = DateTime.UtcNow.Subtract(new TimeSpan(21,
0, 0, 0));
            thread2["Views"] = 0;
            thread2["Replies"] = 0;
            thread2["Answered"] = false;
            thread2["Tags"] = new List<string> { "index", "primarykey", "rangekey" };
```

```
            threadTable.PutItem(thread2);

            // Thread 3.
            var thread3 = new Document();
            thread3["ForumName"] = "Amazon S3"; // Hash attribute.
            thread3["Subject"] = "S3 Thread 1"; // Range attribute.
            thread3["Message"] = "S3 thread 3 message text";
            thread3["LastPostedBy"] = "User A";
            thread3["LastPostedDateTime"] = DateTime.UtcNow.Subtract(new TimeSpan(7, 0,
0, 0));
            thread3["Views"] = 0;
            thread3["Replies"] = 0;
            thread3["Answered"] = false;
            thread3["Tags"] = new List<string> { "largeobjects", "multipart upload" };
            threadTable.PutItem(thread3);
        }

        private static void LoadSampleReplies()
        {
            Table replyTable = Table.LoadTable(client, "Reply");

            // Reply 1 - thread 1.
            var thread1Reply1 = new Document();
            thread1Reply1["Id"] = "Amazon DynamoDB#DynamoDB Thread 1"; // Hash
attribute.
            thread1Reply1["ReplyDateTime"] = DateTime.UtcNow.Subtract(new TimeSpan(21,
0, 0, 0)); // Range attribute.
            thread1Reply1["Message"] = "DynamoDB Thread 1 Reply 1 text";
            thread1Reply1["PostedBy"] = "User A";

            replyTable.PutItem(thread1Reply1);

            // Reply 2 - thread 1.
            var thread1reply2 = new Document();
            thread1reply2["Id"] = "Amazon DynamoDB#DynamoDB Thread 1"; // Hash
attribute.
            thread1reply2["ReplyDateTime"] = DateTime.UtcNow.Subtract(new TimeSpan(14,
0, 0, 0)); // Range attribute.
            thread1reply2["Message"] = "DynamoDB Thread 1 Reply 2 text";
            thread1reply2["PostedBy"] = "User B";

            replyTable.PutItem(thread1reply2);
```

```
            // Reply 3 - thread 1.
            var thread1Reply3 = new Document();
            thread1Reply3["Id"] = "Amazon DynamoDB#DynamoDB Thread 1"; // Hash
 attribute.
            thread1Reply3["ReplyDateTime"] = DateTime.UtcNow.Subtract(new TimeSpan(7,
 0, 0, 0)); // Range attribute.
            thread1Reply3["Message"] = "DynamoDB Thread 1 Reply 3 text";
            thread1Reply3["PostedBy"] = "User B";

            replyTable.PutItem(thread1Reply3);

            // Reply 1 - thread 2.
            var thread2Reply1 = new Document();
            thread2Reply1["Id"] = "Amazon DynamoDB#DynamoDB Thread 2"; // Hash
 attribute.
            thread2Reply1["ReplyDateTime"] = DateTime.UtcNow.Subtract(new TimeSpan(7,
 0, 0, 0)); // Range attribute.
            thread2Reply1["Message"] = "DynamoDB Thread 2 Reply 1 text";
            thread2Reply1["PostedBy"] = "User A";


            replyTable.PutItem(thread2Reply1);

            // Reply 2 - thread 2.
            var thread2Reply2 = new Document();
            thread2Reply2["Id"] = "Amazon DynamoDB#DynamoDB Thread 2"; // Hash
 attribute.
            thread2Reply2["ReplyDateTime"] = DateTime.UtcNow.Subtract(new TimeSpan(1,
 0, 0, 0)); // Range attribute.
            thread2Reply2["Message"] = "DynamoDB Thread 2 Reply 2 text";
            thread2Reply2["PostedBy"] = "User A";

            replyTable.PutItem(thread2Reply2);
        }
    }
}
```

DynamoDB-Beispielanwendung mit dem: AWS SDK für Python (Boto) Tic-tac-toe

Das Tic-Tac-Toe Spiel ist eine Beispiel-Webanwendung, die auf Amazon DynamoDB basiert. Die Anwendung verwendet die AWS SDK für Python (Boto) , um die erforderlichen DynamoDB-

Aufrufe durchzuführen, um Spieldaten in einer DynamoDB-Tabelle zu speichern, und das Python-Webframework Flask, um die end-to-end Anwendungsentwicklung in DynamoDB zu veranschaulichen, einschließlich der Modellierung von Daten. Außerdem zeigt sie bewährte Methoden hinsichtlich des Modellierens von Daten in DynamoDB, einschließlich der Tabelle, die Sie für die Spielanwendung erstellt haben, dem von Ihnen definierten Primärschlüssel, zusätzlichen Indizes, basierend auf Ihren Abfrageanforderungen und der Nutzung von verketteten Wertattributen.

Sie spielen die Anwendung im Internet wie folgt ab Tic-Tac-Toe:

1. Sie melden sich bei der Homepage der Anwendung an.
2. Anschließend laden Sie einen anderen Benutzer ein, das Spiel als Ihr Gegner zu spielen.

   Der Spielstatus lautet weiterhin `PENDING`, bis ein anderer Benutzer Ihre Einladung annimmt. Nachdem ein Gegner die Einladung angenommen hat, ändert sich der Status in `IN_PROGRESS`.
3. Das Spiel beginnt, nachdem sich der Gegner angemeldet und die Einladung akzeptiert hat.
4. Die Anwendung speichert alle Spielzüge und Statusinformationen in einer DynamoDB-Tabelle.
5. Das Spiel endet mit einem Sieg oder einem Unentschieden, wodurch der Spielstatus auf `FINISHED` gesetzt wird.

Die Übung end-to-end zur Anwendungserstellung wird in Schritten beschrieben:

- Schritt 1: Lokales Bereitstellen und Testen – In diesem Abschnitt wird die Anwendung von Ihnen auf Ihrem lokalen Computer heruntergeladen, bereitgestellt und getestet. Sie erstellen die erforderlichen Tabellen in der herunterladbaren Version von DynamoDB.
- Schritt 2: Überprüfen des Datenmodells und der Implementierungsdetails – Dieser Abschnitt beschreibt im Einzelnen das Datenmodell, einschließlich der Indizes und der Nutzung der verketteten Wertattribute. Dann wird im Abschnitt erklärt, wie die Anwendung funktioniert.
- Schritt 3: Bereitstellen in Produktion mit dem DynamoDB-Service – Dieser Abschnitt konzentriert sich auf Bereitstellungsüberlegungen in der Produktion. In diesem Schritt erstellen Sie eine Tabelle mit dem Amazon-DynamoDB-Service und stellen die Anwendung mithilfe von AWS Elastic Beanstalk bereit. Wenn Sie die Anwendung in der Produktion haben, gewähren sie auch entsprechende Berechtigungen, damit die Anwendung auf die DynamoDB-Tabelle zugreifen kann. Die Anweisungen in diesem Abschnitt führen Sie durch die end-to-end Produktionsbereitstellung.
- Schritt 4: Bereinigen von Ressourcen – Dieser Abschnitt hebt Bereiche hervor, die nicht von diesem Beispiel abgedeckt wurden. In diesem Abschnitt finden Sie auch Schritte zum Entfernen

der AWS Ressourcen, die Sie in den vorherigen Schritten erstellt haben, sodass Ihnen keine Kosten entstehen.

Schritt 1: Lokales Bereitstellen und Testen

Themen

- 1.1: Herunterladen und Installieren der erforderlichen Pakete
- 1.2: Testen der Spielanwendung

In diesem Schritt laden Sie die Tic-Tac-Toe Spielanwendung herunter, stellen sie bereit und testen sie auf Ihrem lokalen Computer. Anstatt den Amazon-DynamoDB-Webservice zu nutzen, laden Sie DynamoDB auf den Computer herunter und erstellen dort die erforderliche Tabelle.

1.1: Herunterladen und Installieren der erforderlichen Pakete

Zum lokalen Testen dieser Anwendung benötigen Sie Folgendes:

- Python
- Flask (ein Microframework für Python)
- AWS SDK für Python (Boto)
- Ausführen von DynamoDB auf Ihrem Computer
- Git

Um diese Tools zu erhalten, führen Sie die folgenden Schritte aus:

1. Installieren Sie Python. step-by-stepAnweisungen finden Sie unter Python herunterladen.

   Die Tic-Tac-Toe Anwendung wurde mit Python Version 2.7 getestet.
2. Installieren Sie Flask und AWS SDK für Python (Boto) verwenden Sie den Python Package Installer (PIP):
   - Installieren Sie PIP.

     Anweisungen finden Sie unter Installieren von PIP. Klicken Sie auf der Installationsseite auf den Link get-pip.py und speichern Sie die Datei. Öffnen Sie anschließend ein Befehlsterminal als Administrator und geben Sie Folgendes bei der Eingabeaufforderung ein.

```
python.exe get-pip.py
```

Auf Linux geben sie die Erweiterung `.exe` nicht an. Sie geben ausschließlich `python get-pip.py` an.

- Installieren Sie unter Verwendung von PIP die Flask- und Boto-Pakete mit folgendem Code:

```
pip install Flask
pip install boto
pip install configparser
```

3. Laden Sie DynamoDB auf den Computer herunter. Anweisungen zum Ausführen finden Sie unter Lokale Einrichtung von DynamoDB (herunterladbare Version).
4. Laden Sie die Anwendung herunter: Tic-Tac-Toe
   a. Installieren Sie Git. Anweisungen finden Sie unter git Downloads.
   b. Führen Sie den folgenden Code aus, um die Anwendung herunterzuladen.

```
git clone https://github.com/awslabs/dynamodb-tictactoe-example-app.git
```

1.2: Testen der Spielanwendung

Um die Tic-Tac-Toe Anwendung zu testen, müssen Sie DynamoDB lokal auf Ihrem Computer ausführen.

Um die Anwendung auszuführen tic-tac-toe

1. Starten Sie DynamoDB.
2. Starten Sie den Webserver für die Tic-Tac-Toe Anwendung.

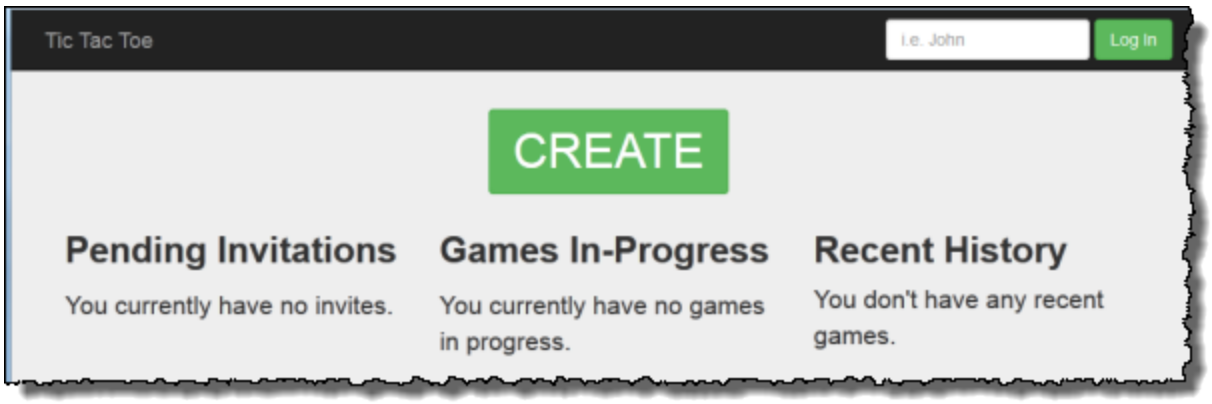
   Öffnen Sie dazu ein Befehlsterminal, navigieren Sie zu dem Ordner, in den Sie die Tic-Tac-Toe Anwendung heruntergeladen haben, und führen Sie die Anwendung lokal mit dem folgenden Code aus.

```
python.exe application.py --mode local --serverPort 5000 --port 8000
```

   Auf Linux geben sie die Erweiterung `.exe` nicht an.
3. Öffnen Sie Ihren Webbrowser und geben Sie Folgendes ein.

```
http://localhost:5000/
```

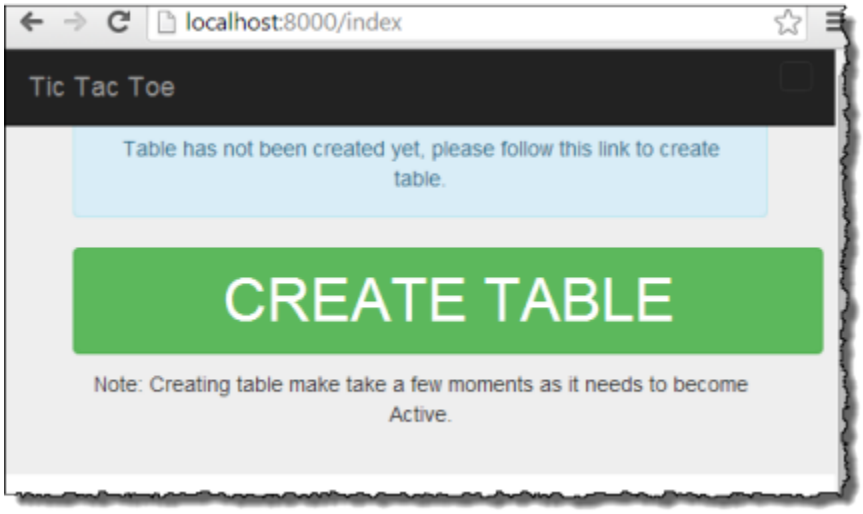
Der Browser zeigt die Homepage an.



4. Geben Sie **`user1`** in das Feld Log in (Anmelden) ein, um sich als user1 anzumelden.

> **Note**
>
> Diese Beispielanwendung führt keine Benutzerauthentifizierung durch. Die Benutzer-ID wird nur verwendet, um Spieler zu identifizieren. Wenn sich zwei Spieler mit identischem Alias anmelden, arbeitet die Anwendung so, als ob Sie in zwei verschiedenen Browsern spielen.

5. Wenn Sie das Spiel zum ersten Mal spielen, wird eine Seite angezeigt, die Sie auffordert, die erforderliche Tabelle (`Games`) in DynamoDB zu erstellen. Wählen Sie CREATE TABLE aus.



6. Wähle CREATE, um das erste tic-tac-toe Spiel zu erstellen.

7. Geben Sie **`user2`** in das Feld Choose an Opponent (Gegner auswählen) ein und wählen Sie Create Game! (Spiel erstellen!) aus.

   Dadurch wird das Spiel erstellt, indem der Tabelle `Games` ein Element hinzugefügt wird. Es setzt den Spielstatus auf `PENDING`.

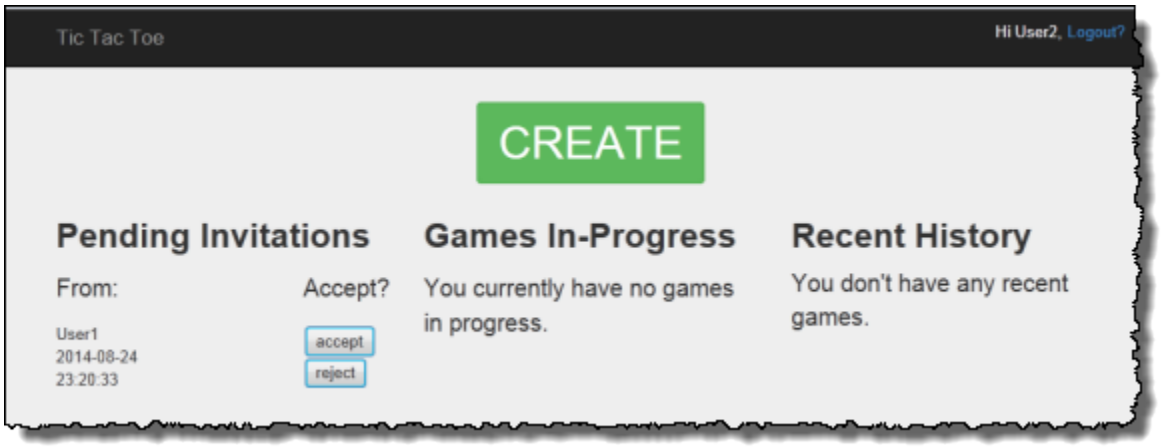
8. Öffnen Sie ein anderes Browserfenster und geben Sie Folgendes ein.

```
http://localhost:5000/
```

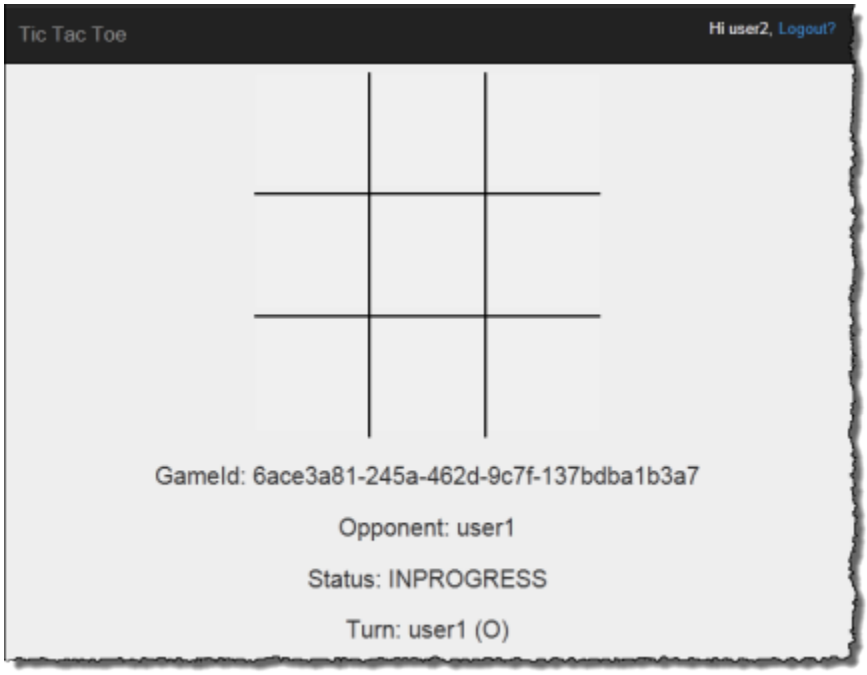
   Der Browser leitet Informationen über Cookies weiter, daher sollten Sie den Inkognitomodus oder privates Browsen nutzen, damit Ihre Cookies nicht übertragen.

9. Melden Sie sich als Benutzer2 an.

   Eine Seite erscheint, die eine ausstehende Einladung von Benutzer1 anzeigt.



10. Wählen Sie accept (akzeptieren) aus, um die Einladung zu akzeptieren.



Die Spieleseite wird mit einem leeren tic-tac-toe Raster angezeigt. Die Seite zeigt auch relevante Informationen wie die Spiel-ID, wer am Zug ist und den Spielstatus an.

11. Spielen Sie das Spiel.

Für jeden Zug des Benutzers sendet der Webservice eine Anforderung an DynamoDB, um das Spielelement in der Tabelle `Games` bedingt zu aktualisieren. Die Bedingungen stellen beispielsweise sicher, dass der Zug gültig ist, dass das Quadrat, das der Benutzer ausgewählt hat, verfügbar ist und dass der Benutzer, der den Zug gemacht hat, an der Reihe war. Für eine gültigen Zug fügt die Aktualisierungsoperation ein neues Attribut, entsprechend der Auswahl auf dem Brett, hinzu. Die Aktualisierungsoperation setzt auch den Wert des vorhandenen Attributs auf den Benutzer, der den nächsten Zug machen kann.

Auf der Spieleseite führt die Anwendung jede Sekunde asynchrone JavaScript Aufrufe für bis zu 5 Minuten durch, um zu überprüfen, ob sich der Spielstatus in DynamoDB geändert hat. Wenn dies der Fall ist, aktualisiert die Anwendung die Seite mit neuen Informationen. Nach 5 Minuten stellt die Anwendung keine weiteren Anfragen mehr und Sie müssen die Seite aktualisieren, um aktualisierte Informationen zu erhalten.

Schritt 2: Überprüfen des Datenmodells und der Implementierungsdetails

Themen

- 2.1: Grundlegendes Datenmodell
- 2.2: Anwendung in Aktion (Anleitung des Codes)

2.1: Grundlegendes Datenmodell

Diese Beispielanwendung hebt folgende DynamoDB-Datenmodellkonzepte hervor:

- Tabelle – In DynamoDB ist eine Tabelle eine Sammlung von Elementen (das bedeutet, Datensätzen) und jedes Element ist eine Sammlung von Name-Wert-Paaren, die Attribute genannt werden.

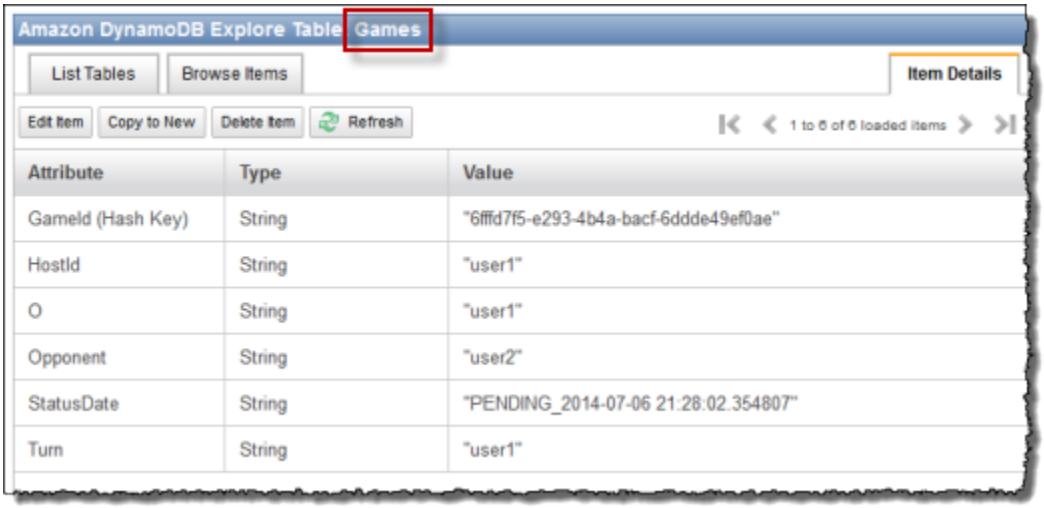
  In diesem Tic-Tac-Toe Beispiel speichert die Anwendung alle Spieldaten in einer Tabelle. `Games` Die Anwendung erstellt pro Spiel ein Element in der Tabelle und speichert alle Spieldaten als Attribute. Ein tic-tac-toe Spiel kann bis zu neun Züge haben. Da DynamoDB-Tabellen über kein Schema in Fällen verfügen, in denen nur der Primärschlüssel das erforderliche Attribut ist, kann die Anwendung eine unterschiedliche Anzahl von Attributen pro Spiel speichern.

  Die Tabelle `Games` verfügt über einen einfachen Primärschlüssel, der aus einem Attribut des Typs Zeichenfolge, `GameId`, besteht. Die Anwendung weist jedem Spiel eine eindeutige ID zu. Weitere Informationen zu DynamoDB-Primärschlüsseln finden Sie unter Primärschlüssel.

  Wenn ein Benutzer ein tic-tac-toe Spiel startet, indem er einen anderen Benutzer zum Spielen einlädt, erstellt die Anwendung ein neues Element in der `Games` Tabelle mit Attributen, in denen Spielmetadaten gespeichert werden, wie z. B. die folgenden:

  - `HostId`, der Benutzer, der das Spiel gestartet hat.
  - `Opponent`, der Benutzer, der zum Spielen eingeladen wurde.
  - Der Benutzer, der an der Reihe ist zu spielen. Der Benutzer, der das Spiel gestartet hat, beginnt als erster zu spielen.
  - Der Benutzer, der das Symbol O auf dem Brett verwendet. Der Benutzer, der die Spiele gestartet hat, verwendet das Symbol O.

  Zudem erstellt die Anwendung ein `StatusDate`-verkettetes Attribut, das den ersten Spielstatus als `PENDING` markiert. Der folgende Screenshot zeigt ein Beispielelement, wie es in der DynamoDB-Konsole angezeigt wird:



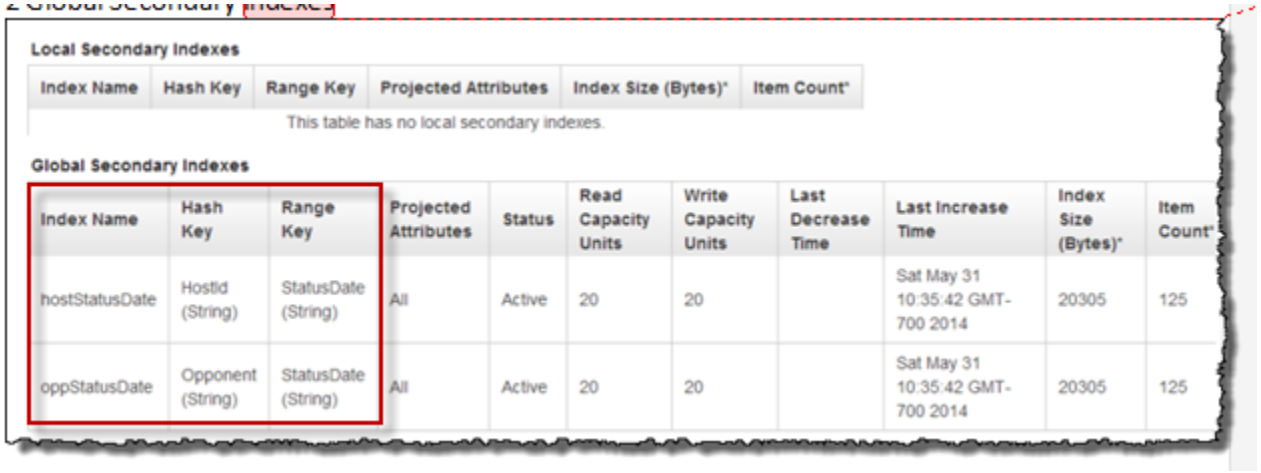
Während das Spiel voranschreitet, fügt die Anwendung der Tabelle ein Attribut für jeden Spielzug hinzu. Der Attributname ist die Brettposition, zum Beispiel `TopLeft` oder `BottomRight`. Ein Zug verfügt beispielsweise über ein `TopLeft`-Attribut mit dem Wert 0, ein `TopRight`-Attribut mit dem Wert 0 und ein `BottomRight`-Attribut mit dem Wert `X`. Der Attributwert ist entweder 0 oder `X`, abhängig davon, welcher Benutzer den Zug gemacht hat. Betrachten Sie beispielsweise das folgende Board:

- Verkettete Wertattribute – Das `StatusDate`-Attribut zeigt ein verkettetes Wertattribut. Anstatt bei diesem Ansatz getrennte Attribute zu erstellen, um den Spielstatus (`PENDING`, `IN_PROGRESS` und `FINISHED`) und das Datum (wann der letzte Zug erfolgte) zu speichern, kombinieren Sie sie als ein einzelnes Attribut. Zum Beispiel `IN_PROGRESS_2014-04-30 10:20:32`.

  Die Anwendung nutzt dann das `StatusDate`-Attribut bei der Erstellung von sekundären Indizes, indem `StatusDate` als Sortierschlüssel für den Index angegeben wird. Der Vorteil eines

`StatusDate`-verketteten Wertattributs wird in den Indizes, die im nächsten Abschnitt beschrieben werden, weiter dargestellt.

- Globale sekundäre Indizes – Sie können den Primärschlüssel `GameId` der Tabelle nutzen, um die Tabelle effizient nach Spielelementen abzufragen. Um die Tabelle nach anderen Attributen als den Primärschlüsselattributen abzufragen, unterstützt DynamoDB die Erstellung von sekundären Indizes. In dieser Beispielanwendung erstellen Sie die folgenden zwei sekundären Indizes:



Local Secondary Indexes

Index Name	Hash Key	Range Key	Projected Attributes	Index Size (Bytes)*	Item Count*

This table has no local secondary indexes.

Global Secondary Indexes

Index Name	Hash Key	Range Key	Projected Attributes	Status	Read Capacity Units	Write Capacity Units	Last Decrease Time	Last Increase Time	Index Size (Bytes)*	Item Count*
hostStatusDate	HostId (String)	StatusDate (String)	All	Active	20	20		Sat May 31 10:35:42 GMT-700 2014	20305	125
oppStatusDate	Opponent (String)	StatusDate (String)	All	Active	20	20		Sat May 31 10:35:42 GMT-700 2014	20305	125

  - HostId- StatusDate -index. Dieser Index hat `HostId` als Partitionsschlüssel und `StatusDate` als Sortierschlüssel. Sie können diesen Index für die Abfrage von `HostId` verwenden, um zum Beispiel Spiele zu finden, die von einem bestimmten Benutzer gehostet wurden.
  - OpponentId- StatusDate -index. Dieser Index hat `OpponentId` als Partitionsschlüssel und `StatusDate` als Sortierschlüssel. Sie können diesen Index für die Abfrage von `Opponent` verwenden, um zum Beispiel Spiele zu finden, in denen ein bestimmter Benutzer der Gegner ist.

  Diese Indizes werden globale sekundäre Indizes genannt, weil der Partitionsschlüssel in diesen Indizes nicht identisch mit dem Partitionsschlüssel (`GameId`) ist, der in dem Primärschlüssel der Tabelle verwendet wird.

  Beachten Sie, dass beide Indizes `StatusDate` als Sortierschlüssel angeben. Dies ermöglicht Folgendes:

  - Sie können Abfragen mithilfe des `BEGINS_WITH`-Vergleichsoperators durchführen. Sie können beispielsweise alle Spiele mit dem `IN_PROGRESS`-Attribut, das von einem bestimmten Benutzer gehostet wurde, finden. In diesem Fall überprüft der `BEGINS_WITH`-Operator den `StatusDate`-Wert, der mit `IN_PROGRESS` beginnt.
  - DynamoDB speichert die Elemente in sortierter Reihenfolge nach Sortierschlüssel in den Index. Wenn also alle Statuspräfixe identisch sind (zum Beispiel `IN_PROGRESS`), wird das ISO-Format,

das für den Datumsteil verwendet wird, die Elemente vom ältesten zum neuesten sortiert haben. Dieser Ansatz ermöglicht das effiziente Durchführen bestimmter Abfragen, z. B. die Folgenden:

- Abrufen von bis zu zehn der letzten `IN_PROGRESS`-Spiele, die der angemeldete Benutzer gehostet hat. Sie geben den `HostId-StatusDate-index`-Index für diese Abfrage an.
- Abrufen von bis zu 10 der letzten `IN_PROGRESS`-Spiele, in denen der angemeldete Benutzer der Gegner ist. Sie geben den `OpponentId-StatusDate-index`-Index für diese Abfrage an.

Weitere Informationen zu den sekundären Indizes finden Sie unter Verbessern des Datenzugriffs mit sekundären Indizes in DynamoDB.

2.2: Anwendung in Aktion (Anleitung des Codes)

Diese Anwendung hat zwei Hauptseiten:

- Startseite — Diese Seite bietet dem Benutzer eine einfache Anmeldung, eine CREATE-Taste, um ein neues tic-tac-toe Spiel zu erstellen, eine Liste der laufenden Spiele, den Spielverlauf und alle aktiven ausstehenden Spieleinladungen.

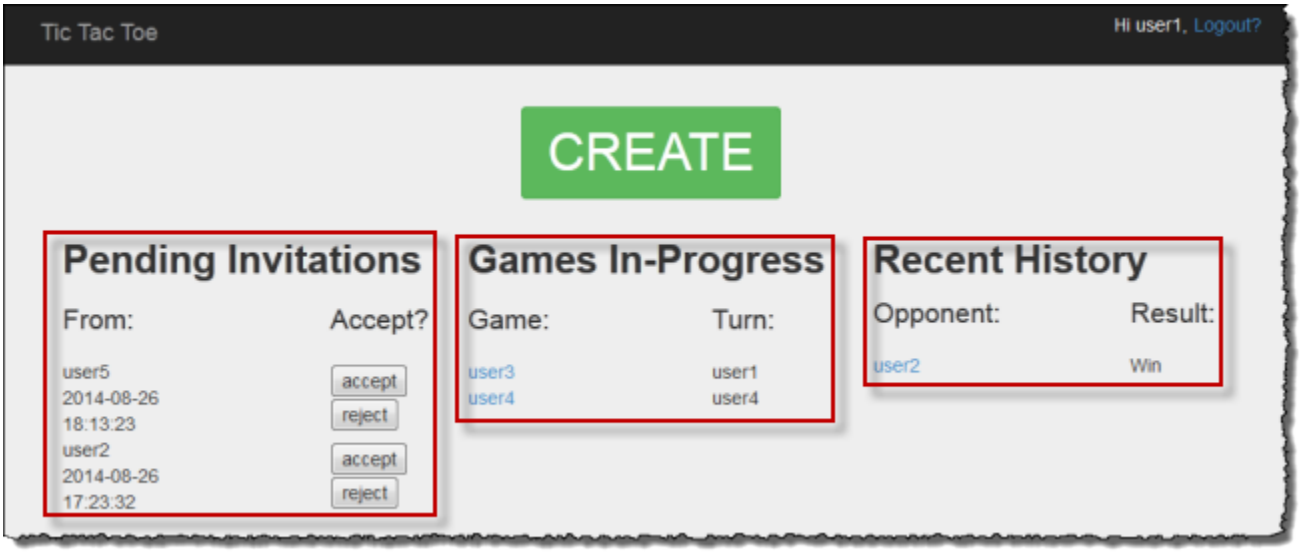
  Die Homepage wird nicht automatisch aktualisiert. Sie müssen die Seite aktualisieren, um die Listen zu aktualisieren.
- Spielseite — Auf dieser Seite wird das Spielfeld angezeigt tic-tac-toe, in dem die Nutzer spielen.

  Die Anwendung aktualisiert automatisch die Spielseite im Sekundentakt. Der JavaScript in Ihrem Browser ruft jede Sekunde den Python-Webserver auf, um die Spieltabelle abzufragen, ob sich die Spielelemente in der Tabelle geändert haben. Wenn dies der Fall ist, JavaScript wird eine Seitenaktualisierung ausgelöst, sodass der Benutzer das aktualisierte Board sieht.

Lassen Sie uns im Detail anschauen, wie die Anwendung funktioniert.

Homepage

Nachdem sich der Benutzer anmeldet, zeigt die Anwendung die folgenden drei Informationslisten.



- Einladungen – Die Liste zeigt bis zu zehn der letzten Einladungen von anderen an, die der angemeldete Benutzer noch nicht angenommen hat. Im vorherigen Screenshot hat Benutzer1 ausstehende Einladungen von Benutzer5 und Benutzer2.
- Laufende Spiele – Diese Liste zeigt bis zu zehn der letzten Spiele an, die im Gange sind. Das sind Spiele, die der Benutzer aktiv spielt, die den Status `IN_PROGRESS` haben. Im Screenshot spielt Benutzer1 aktiv ein tic-tac-toe Spiel mit Benutzer3 und Benutzer4.
- Aktueller Verlauf – Die Liste zeigt bis zu zehn der letzten Spiele an, die der Benutzer beendet hat, die den Status `FINISHED` haben. In dem im Screenshot gezeigten Spiel hat Benutzer1 zuvor mit Benutzer2 gespielt. Die Liste zeigt für jedes abgeschlossene Spiel die Spielergebnis an.

In dem Code führt die `index`-Funktion (in `application.py`) die folgenden drei Aufrufe durch, um Informationen über den Spielstatus abzurufen:

```
inviteGames     = controller.getGameInvites(session["username"])
inProgressGames = controller.getGamesWithStatus(session["username"], "IN_PROGRESS")
finishedGames   = controller.getGamesWithStatus(session["username"], "FINISHED")
```

Jeder dieser Aufrufe gibt eine Liste von Elementen von DynamoDB zurück, die von den `Game`-Objekten umschlossen sind. Es ist einfach, Daten aus diesen Objekten in der Ansicht zu extrahieren. Die Indexfunktion übergibt diese Objektlisten an die Ansicht, um die HTML zu rendern.

```
return render_template("index.html",
                       user=session["username"],
                       invites=inviteGames,
```

```
                        inprogress=inProgressGames,
                        finished=finishedGames)
```

Die Tic-Tac-Toe Anwendung definiert die `Game` Klasse hauptsächlich zum Speichern von Spieldaten, die von DynamoDB abgerufen wurden. Diese Funktionen geben Listen von `Game`-Objekten zurück, die es Ihnen ermöglichen, den Rest der Anwendung von dem Code zu isolieren, der zu Amazon-DynamoDB-Elementen gehört. Daher helfen diese Funktionen Ihnen dabei, Ihren Anwendungscode von den Details der Datenspeicherebene abzukoppeln.

Das hier beschriebene Architekturmuster wird auch als model-view-controller (MVC) UI-Muster bezeichnet. In diesem Fall sind die `Game`-Objekt-Instances (die Daten repräsentieren) das Modell und die HTML-Seite ist die Ansicht. Der Controller ist in zwei Dateien aufgeteilt. Die Datei `application.py` hat die Controllerlogik für das Flask-Framework und die Geschäftslogik wird in der Datei `gameController.py` isoliert. Das bedeutet, dass die Anwendung alles speichert, was mit DynamoDB-SDK zu tun hat, in ihrer eigenen separaten Datei im Ordner `dynamodb`.

Betrachten wir die drei Funktionen und wie sie die Spiele-Tabelle mithilfe der globalen sekundären Indizes abfragen, um relevante Daten abzurufen.

Wird verwendet getGameInvites , um die Liste der ausstehenden Spieleinladungen abzurufen

Die `getGameInvites`-Funktion ruft die Liste der zehn letzten ausstehenden Einladungen ab. Diese Spiele wurden von Benutzern erstellt, aber die Gegner haben die Spieleinladungen nicht angenommen. Für diese Spiele lautet der Status weiterhin `PENDING`, bis der Gegner die Einladung annimmt. Wenn der Gegner die Einladung ablehnt, entfernt die Anwendung das entsprechende Element aus der Tabelle.

Die Funktion gibt die Abfrage wie folgt an:

- Sie gibt den Index `OpponentId-StatusDate-index` an, der mit den folgenden Indexschlüsselwerten und Vergleichsoperatoren verwendet wird:
  - Der Partitionsschlüssel ist `OpponentId` und übernimmt den Indexschlüssel *`user ID`*.
  - Der Sortierschlüssel ist `StatusDate` und übernimmt den Vergleichsoperator und den Indexschlüsselwert `beginswith="PENDING_"`.

  Sie nutzen den `OpponentId-StatusDate-index`-Index, um Spiele abzurufen, für die der angemeldete Benutzer eingeladen ist – das bedeutet, in denen der angemeldete Benutzer der Gegner ist.
- Die Abfrage begrenzt das Ergebnis auf zehn Elemente.

```
gameInvitesIndex = self.cm.getGamesTable().query(
                                                Opponent__eq=user,
                                                StatusDate__beginswith="PENDING_",
                                                index="OpponentId-StatusDate-index",
                                                limit=10)
```

Für jede `OpponentId` (Partitionsschlüssel) im Index behält DynamoDB die Sortierung der Elemente nach `StatusDate` (Sortierschlüssel) bei. Daher werden ausschließlich die zehn letzten Spiele von der Abfrage zurückgegeben.

Verwenden von getGamesWith Status, um die Liste der Spiele mit einem bestimmten Status abzurufen

Nachdem ein Gegner eine Spieleinladung angenommen hat, ändert sich der Status in `IN_PROGRESS`. Nach dem Beenden des Spiels ändert sich der Status in `FINISHED`.

Abfragen für das Finden von Spielen, die entweder in Bearbeitung oder abgeschlossen sind, sind mit Ausnahme des unterschiedlichen Statuswerts identisch. Daher definiert die Anwendung die `getGamesWithStatus`-Funktion, die den Statuswert als Parameter übernimmt.

```
inProgressGames = controller.getGamesWithStatus(session["username"], "IN_PROGRESS")
finishedGames   = controller.getGamesWithStatus(session["username"], "FINISHED")
```

Der folgende Abschnitt erläutert laufende Spiele, aber die gleiche Beschreibung gilt ebenso für beendete Spiele.

Eine Liste der laufenden Spiele für einen bestimmten Benutzer umfasst die beiden Folgenden:

- Laufende Spiele, die von dem Benutzer gehostet werden
- Laufende Spiele, in denen der Benutzer der Gegner ist

Die `getGamesWithStatus`-Funktion führt die folgenden zwei Abfragen aus und verwendet jedes Mal den entsprechenden sekundären Index.

- Die Funktion fragt die Tabelle `Games` mithilfe des Index `HostId-StatusDate-index` ab. Die Abfrage gibt für den Index Primärschlüsselwert an – die Partitionsschlüssel- (`HostId`) und die Sortierschlüsselwerte (`StatusDate`), zusammen mit Vergleichsoperatoren.

  ```
  hostGamesInProgress = self.cm.getGamesTable ().query(HostId__eq=user,
  ```

```
                                                    StatusDate__beginswith=status,
                                                    index="HostId-StatusDate-index",
                                                    limit=10)
```

Beachten Sie die Python-Syntax für Vergleichsoperatoren:

- `HostId__eq=user` gibt den Gleichheitsvergleichsoperator an.
- `StatusDate__beginswith=status` gibt den `BEGINS_WITH`-Vergleichsoperator an.

- Die Funktion fragt die Tabelle `Games` mithilfe des Index `OpponentId-StatusDate-index` ab.

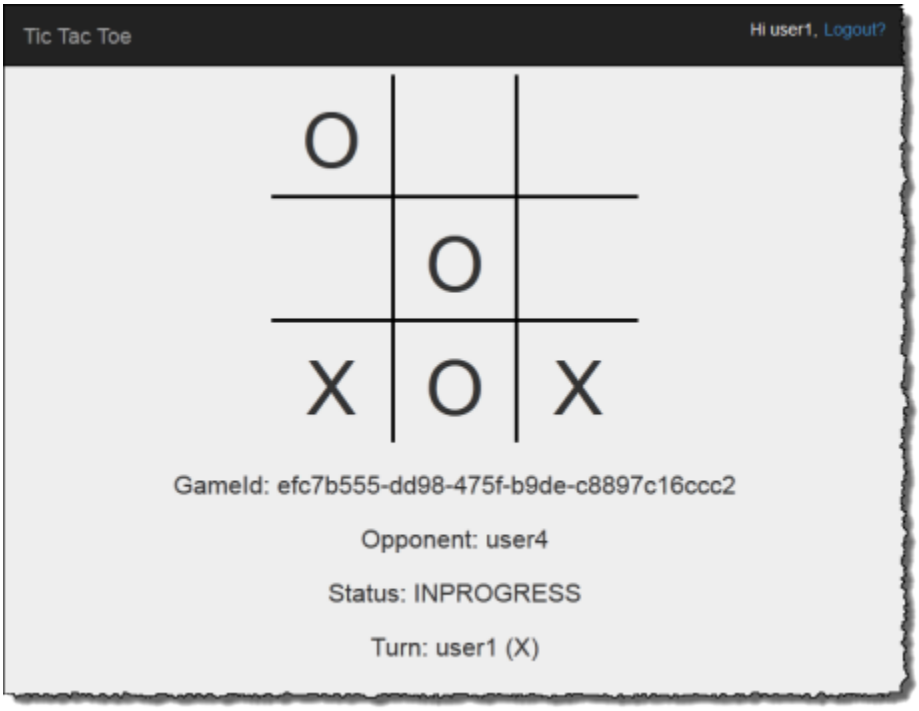
```
oppGamesInProgress = self.cm.getGamesTable().query(Opponent__eq=user,
                                                   StatusDate__beginswith=status,
                                                   index="OpponentId-StatusDate-index",
                                                   limit=10)
```

- Die Funktion kombiniert dann die beiden Listen, sortiert sie und erstellt für die ersten 0 bis 10 Elemente eine Liste der `Game`-Objekte. Diese Liste gibt sie der aufrufenden Funktion (das bedeutet, dem Index) zurück.

```
games = self.mergeQueries(hostGamesInProgress,
                          oppGamesInProgress)
return games
```

Spielseite

Auf der Spieleseite spielt der Benutzer tic-tac-toe Spiele. Sie zeigt das Spielraster zusammen mit spielrelevanten Informationen an. Der folgende Screenshot zeigt ein laufendes Beispielspiel:



Die Anwendung zeigt die Spielseite in den folgenden Situationen an:

- Der Benutzer erstellt ein Spiel, um einen anderen Benutzer zum Spielen einzuladen.

  In diesem Fall zeigt die Seite den Benutzer als Host und den Spielstatus als `PENDING` an, während darauf gewartet wird, dass der Gegner akzeptiert.
- Der Benutzer akzeptiert eine der ausstehenden Einladungen auf der Homepage.

  In diesem Fall zeigt die Seite den Benutzer als Gegner und den Spielstatus als `IN_PROGRESS` an.

Eine Benutzerauswahl auf dem Board generiert die Formularanforderung `POST` für die Anwendung. Das bedeutet, dass Flask die `selectSquare`-Funktion (in `application.py`) mit den HTML-Formulardaten aufruft. Diese Funktion ruft wiederum die `updateBoardAndTurn`-Funktion (in `gameController.py`) auf, um das Spielelement wie folgt zu aktualisieren:

- Sie fügt ein neues Attribut hinzu, das für den Zug spezifisch ist.
- Sie aktualisiert den Attributwert `Turn` für den Benutzer, der an der Reihe ist.

```
controller.updateBoardAndTurn(item, value, session["username"])
```

Die Funktion gibt True zurück, wenn das Element erfolgreich aktualisiert wurde; andernfalls False. Beachten Sie Folgendes zu der `updateBoardAndTurn`-Funktion:

- Die Funktion ruft die `update_item`-Funktion des SDK for Python auf, um eine begrenzte Reihe von Aktualisierungen für ein vorhandenes Element durchzuführen. Die Funktion wird der Operation `UpdateItem` in DynamoDB zugeordnet. Weitere Informationen finden Sie unter UpdateItem.

> **Note**
>
> Der Unterschied zwischen der `UpdateItem`- und der `PutItem`-Operation besteht darin, dass `PutItem` das gesamte Element ersetzt. Weitere Informationen finden Sie unter PutItem.

Für den `update_item`-Aufruf identifiziert der Code Folgendes:

- Den Primärschlüssel der Tabelle `Games` (d. h. `ItemId`).

```
key = { "GameId" : { "S" : gameId } }
```

- Das neue hinzuzufügende Attribut, das für den aktuellen Benutzerzug spezifisch ist und sein Wert (zum Beispiel `TopLeft="X"`).

```
attributeUpdates = {
    position : {
        "Action" : "PUT",
        "Value" : { "S" : representation }
    }
}
```

- Bedingungen, die erfüllt sein müssen, damit die Aktualisierung ausgeführt werden kann:
  - Das Spiel muss in Bearbeitung sein. Das bedeutet, dass der `StatusDate`-Attributwert mit `IN_PROGRESS` beginnen muss.
  - Der aktuelle Zug muss, wie von dem `Turn`-Attribut angegeben, ein gültiger Benutzerzug sein.
  - Das Quadrat, das der Benutzer ausgewählt hat, muss verfügbar sein. Das bedeutet, dass das Attribut, das dem Quadrat entspricht, nicht existieren muss.

```
expectations = {"StatusDate" : {"AttributeValueList": [{"S" : "IN_PROGRESS_"}],
    "ComparisonOperator": "BEGINS_WITH"},
```

```
    "Turn" : {"Value" : {"S" : current_player}},
    position : {"Exists" : False}}
```

Jetzt ruft die Funktion `update_item` auf, um das Element zu aktualisieren.

```
self.cm.db.update_item("Games", key=key,
    attribute_updates=attributeUpdates,
    expected=expectations)
```

Nach der Rückgabe der Funktion werden die `selectSquare`-Funktionsaufrufe wie im folgenden Beispiel gezeigt umgeleitet.

```
redirect("/game="+gameId)
```

Dieser Aufruf bewirkt ein Aktualisieren des Browsers. Als Teil dieser Aktualisierung prüft die Anwendung, ob das Spiel mit einem Sieg oder einem Unentschieden beendet wurde. Wenn dies der Fall ist, aktualisiert die Anwendung das Spielelement entsprechend.

Schritt 3: Bereitstellen in Produktion mit dem DynamoDB-Service

Themen

- 3.1: Erstellen einer IAM-Rolle für Amazon EC2
- 3.2: Erstellen der Spiele-Tabelle in Amazon DynamoDB
- 3.3: Den Anwendungscode bündeln und bereitstellen tic-tac-toe
- 3.4: Richten Sie die AWS Elastic Beanstalk Umgebung ein

In den vorherigen Abschnitten haben Sie die Tic-Tac-Toe Anwendung mithilfe von DynamoDB local lokal auf Ihrem Computer bereitgestellt und getestet. Jetzt können Sie die Anwendung in der Produktion wie folgt bereitstellen:

- Stellen Sie die Anwendung mithilfe AWS Elastic Beanstalk eines easy-to-use Dienstes für die Bereitstellung und Skalierung von Webanwendungen und -diensten bereit. Weitere Informationen finden Sie unter Bereitstellen einer Flask-Anwendung auf AWS Elastic Beanstalk.

  Elastic Beanstalk startet eine oder mehrere Amazon Elastic Compute Cloud (Amazon EC2) - Instances, die Sie über Elastic Beanstalk konfigurieren und auf denen Ihre Anwendung ausgeführt wird. Tic-Tac-Toe

- Erstellen Sie mit dem Amazon-DynamoDB-Service eine `Games`Tabelle, die in AWS anstatt lokal auf Ihrem Computer existiert.

Außerdem müssen Sie auch Berechtigungen konfigurieren. Alle AWS Ressourcen, die Sie erstellen, wie z. B. die `Games` Tabelle in DynamoDB, sind standardmäßig privat. Nur der Besitzer der Ressourcen, also das AWS-Konto, das die Tabelle `Games` erstellt hat, kann auf diese Tabelle zugreifen. Daher kann Ihre Tic-Tac-Toe Anwendung die Tabelle standardmäßig nicht aktualisieren. `Games`

Um die erforderlichen Berechtigungen zu erteilen, erstellen Sie eine AWS Identity and Access Management (IAM-) Rolle und gewähren dieser Rolle Berechtigungen für den Zugriff auf die `Games` Tabelle. Ihre Amazon-EC2-Instance übernimmt diese Rolle zum ersten Mal. AWS Gibt als Antwort temporäre Sicherheitsanmeldedaten zurück, die die Amazon EC2 EC2-Instance verwenden kann, um die `Games` Tabelle im Namen der Tic-Tac-Toe Anwendung zu aktualisieren. Wenn Sie die Elastic-Beanstalk-Anwendung konfigurieren, geben Sie die IAM-Rolle an, die die Amazon-EC2-Instance oder Instances übernehmen können. Weitere Informationen zu IAM-Rollen finden Sie unter IAM-Rollen für Amazon EC2 im Amazon-EC2-Benutzerhandbuch.

> **Note**
>
> Bevor Sie Amazon EC2 EC2-Instances für die Tic-Tac-Toe Anwendung erstellen, müssen Sie zunächst die AWS Region festlegen, in der Elastic Beanstalk die Instances erstellen soll. Nachdem Sie die Elastic-Beanstalk-Anwendung erstellen, geben Sie denselben Regionsnamen und Endpunkt in einer Konfigurationsdatei an. Die Tic-Tac-Toe Anwendung verwendet die Informationen in dieser Datei, um die `Games` Tabelle zu erstellen und nachfolgende Anfragen in einer bestimmten Region zu senden. AWS Die DynamoDB `Games`-Tabelle und die Amazon-EC2-Instances, die Elastic Beanstalk startet, müssen sich in derselben Region befinden. Eine Liste von verfügbaren Regionen finden Sie unter Amazon DynamoDB in der Allgemeine Amazon Web Services-Referenz.

Zusammenfassend gehen Sie wie folgt vor, um die Tic-Tac-Toe Anwendung in der Produktion bereitzustellen:

1. Erstellen Sie eine IAM-Rolle mit dem IAM-Service. Sie fügen dieser Rolle eine Richtlinie an, die Berechtigungen für DynamoDB-Aktionen erteilt, auf die Tabelle `Games` zuzugreifen.

2. Bündeln Sie den Tic-Tac-Toe Anwendungscode und eine Konfigurationsdatei und erstellen Sie eine `.zip` Datei. Sie verwenden diese `.zip` Datei, um den Tic-Tac-Toe Anwendungscode an Elastic Beanstalk weiterzugeben, damit er ihn auf Ihren Servern platzieren kann. Weitere Informationen zur Erstellung eines Pakets finden Sie unter Erstellen eines Anwendungsquellenbündel im AWS Elastic Beanstalk -Entwicklerhandbuch.

   In der Konfigurationsdatei (`beanstalk.config`) geben Sie Informationen zu AWS -Regionen und -Endpunkten an. Die Tic-Tac-Toe Anwendung verwendet diese Informationen, um zu bestimmen, mit welcher DynamoDB-Region kommuniziert werden soll.

3. Einrichten der Elastic Beanstalk-Umgebung. Elastic Beanstalk startet eine oder Amazon EC2 EC2-Instances und stellt Ihr Tic-Tac-Toe Anwendungspaket darauf bereit. Nachdem die Elastic Beanstalk-Umgebung bereit ist, geben Sie den Konfigurationsdateinamen an, indem Sie die `CONFIG_FILE`-Umgebungsvariable hinzufügen.
4. Erstellen einer DynamoDB-Tabelle. Mit dem Amazon DynamoDB-Service erstellen Sie die `Games` Tabelle nicht lokal AWS, sondern auf Ihrem Computer. Denken Sie daran, dass diese Tabelle einen einfachen Primärschlüssel hat, der aus dem `GameId`-Partitionsschlüssel, des Typs Zeichenfolge, besteht.
5. Testen Sie das Spiel in der Produktion.

3.1: Erstellen einer IAM-Rolle für Amazon EC2

Durch das Erstellen einer IAM-Rolle des Typs Amazon EC2 kann die Amazon EC2 EC2-Instance, auf der Ihre Tic-Tac-Toe Anwendung ausgeführt wird, die richtige Rolle übernehmen und Anwendungsanfragen für den Zugriff auf die Tabelle stellen. `Games` Wählen Sie beim Erstellen der Rolle die Option Custom Policy (Benutzerdefinierte Richtlinie) aus, kopieren Sie die folgende Richtlinie und fügen Sie sie ein.

JSON

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Action":[
            "dynamodb:ListTables"
         ],
         "Effect":"Allow",
```

```
            "Resource":"*"
         },
         {
            "Action":[
               "dynamodb:*"
            ],
            "Effect":"Allow",
            "Resource":[
               "arn:aws:dynamodb:us-west-2:922852403271:table/Games",
               "arn:aws:dynamodb:us-west-2:922852403271:table/Games/index/*"
            ]
         }
      ]
   }
```

Weitere Anweisungen finden Sie unter Erstellen einer Rolle für den AWS -Service (AWS-Managementkonsole) im IAM-Benutzerhandbuch.

3.2: Erstellen der Spiele-Tabelle in Amazon DynamoDB

Die `Games`-Tabelle in DynamoDB speichert Spieldaten. Wenn die Tabelle nicht vorhanden ist, erstellt die Anwendung die Tabelle für Sie. Lassen Sie in diesem Fall die Anwendung die `Games`-Tabelle erstellen.

3.3: Den Anwendungscode bündeln und bereitstellen tic-tac-toe

Wenn Sie die Schritte dieses Beispiels befolgt haben, haben Sie die Tic-Tac-Toe Anwendung bereits heruntergeladen. Wenn nicht, laden sie die Anwendung herunter und entpacken Sie alle Dateien in einen Ordner auf Ihrem lokalen Computer. Detaillierte Anweisungen finden Sie unter Schritt 1: Lokales Bereitstellen und Testen.

Nachdem Sie alle Dateien extrahiert haben, verfügen Sie über einen `code`-Ordner. Um diesen Ordner an Elastic Beanstalk zu übergeben, bündeln Sie die Inhalte dieses Ordners als `.zip`-Datei. Zuerst müssen Sie diesem Ordner eine Konfigurationsdatei hinzufügen. Ihre Anwendung verwendet die Region- und Endpunktinformationen zum Erstellen einer DynamoDB-Tabelle in der angegebenen Region und führt nachfolgende Tabellenoperationsanforderungen mithilfe des angegebenen Endpunktes durch.

1. Wechseln Sie zu dem Ordner, in den Sie die Tic-Tac-Toe Anwendung heruntergeladen haben.

2. Erstellen Sie im Stammordner der Anwendung eine Textdatei mit dem Namen `beanstalk.config` mit folgendem Inhalt:

```
[dynamodb]
region=<AWS region>
endpoint=<DynamoDB endpoint>
```

Beispielsweise können Sie den folgenden Inhalt verwenden.

```
[dynamodb]
region=us-west-2
endpoint=dynamodb.us-west-2.amazonaws.com
```

Eine Liste der verfügbaren Regionen finden Sie unter Amazon DynamoDB in der Allgemeinen Amazon-Web-Services-Referenz..

Important

Die in der Konfigurationsdatei angegebene Region ist der Ort, an dem die Tic-Tac-Toe Anwendung die `Games` Tabelle in DynamoDB erstellt. Sie müssen die Elastic-Beanstalk-Anwendung, die im nächsten Abschnitt behandelt wird, in derselben Region erstellen.

Note

Wenn Sie die Elastic-Beanstalk-Anwendung erstellen, werden Sie den Start einer Umgebung anfordern, in der Sie den Umgebungstyp auswählen können. Um die Tic-Tac-Toe Beispielanwendung zu testen, können Sie den Umgebungstyp Single Instance wählen, den folgenden Vorgang überspringen und mit dem nächsten Schritt fortfahren. Beachten Sie jedoch, dass der Umgebungstyp Load balancing, autoscaling eine hoch verfügbare und skalierbare Umgebung bietet. Dies sollten Sie berücksichtigen, wenn Sie andere Anwendungen erstellen und bereitstellen. Wenn Sie diesen Umgebungstyp auswählen, müssen Sie auch eine UUID generieren und diese der Konfigurationsdatei hinzufügen, wie nachfolgend gezeigt.

```
[dynamodb]
region=us-west-2
endpoint=dynamodb.us-west-2.amazonaws.com
```

```
[flask]
secret_key= 284e784d-1a25-4a19-92bf-8eeb7a9example
```

Wenn der Server bei der Client-Server-Kommunikation eine Antwort sendet, verwendet er aus Sicherheitsgründen ein signiertes Cookie, das der Client in der nächsten Anforderung zurück an den Server sendet. Wenn nur ein Server vorhanden ist, kann dieser beim Start lokal einen Verschlüsselungsschlüssel generieren. Wenn viele Server vorhanden sind, müssen alle den gleichen Verschlüsselungsschlüssel kennen. Andernfalls sind sie nicht in der Lage, Cookies zu lesen, die von den Peerservern festgelegt werden. Durch das Hinzufügen von `secret_key` in die Konfigurationsdatei informieren Sie alle Server darüber, dass dieser Verschlüsselungsschlüssel verwendet werden soll.

3. Zippen Sie den Inhalt des Stammordners der Anwendung (einschließlich der `beanstalk.config`-Datei) – zum Beispiel `TicTacToe.zip`.
4. Laden Sie die `.zip`-Datei in einem Amazon-Simple-Storage-Service-(Amazon-S3)-Bucket hoch. Im nächsten Abschnitt stellen Sie diese `.zip`-Datei für Elastic Beanstalk bereit, um sie auf den oder die Server hochzuladen.

   Anleitungen zum Hochladen einer Datei in einen Amazon S3-Bucket finden Sie unter Erstellen eines Buckets und Hinzufügen eines Objekts zu einem Bucket im Amazon Simple Storage Service-Benutzerhandbuch.

3.4: Richten Sie die AWS Elastic Beanstalk Umgebung ein

In diesem Schritt erstellen Sie eine Elastic-Beanstalk-Anwendung, die aus einer Sammlung von Komponenten einschließlich Umgebungen besteht. In diesem Beispiel starten Sie eine Amazon EC2 EC2-Instance, um Ihre Tic-Tac-Toe Anwendung bereitzustellen und auszuführen.

1. Geben Sie die folgende benutzerdefinierte URL ein, um eine Elastic-Beanstalk-Konsole zum Einrichten der Umgebung einzurichten.

```
https://console.aws.amazon.com/elasticbeanstalk/?region=<AWS-Region>#/
newApplication
?applicationName=TicTacToeyour-name
&solutionStackName=Python
&sourceBundleUrl=https://s3.amazonaws.com/<bucket-name>/TicTacToe.zip
&environmentType=SingleInstance
```

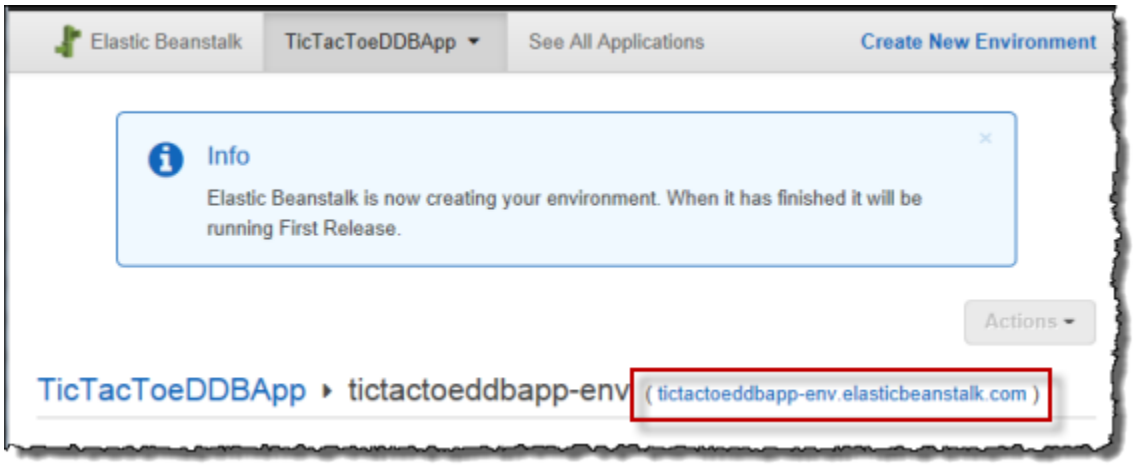
```
&instanceType=t1.micro
```

Weitere Informationen zu benutzerdefinierten URLs Einstellungen finden Sie unter Constructing a Launch Now URL im AWS Elastic Beanstalk Developer Guide. Bitte beachten Sie für die URL Folgendes:

- Sie müssen einen AWS Regionsnamen (derselbe, den Sie in der Konfigurationsdatei angegeben haben), einen Amazon S3 S3-Bucket-Namen und den Objektnamen angeben.
- Zum Testen fordert die URL den SingleInstanceUmgebungstyp und `t1.micro` den Instanztyp an.
- Der Anwendungsname muss eindeutig sein. Demnach haben wir in der vorherigen URL vorgeschlagen, dass Sie Ihren Namen dem `applicationName` voranstellen.

Dies öffnet die Elastic-Beanstalk-Konsole. In einigen Fällen müssen Sie sich möglicherweise anmelden.

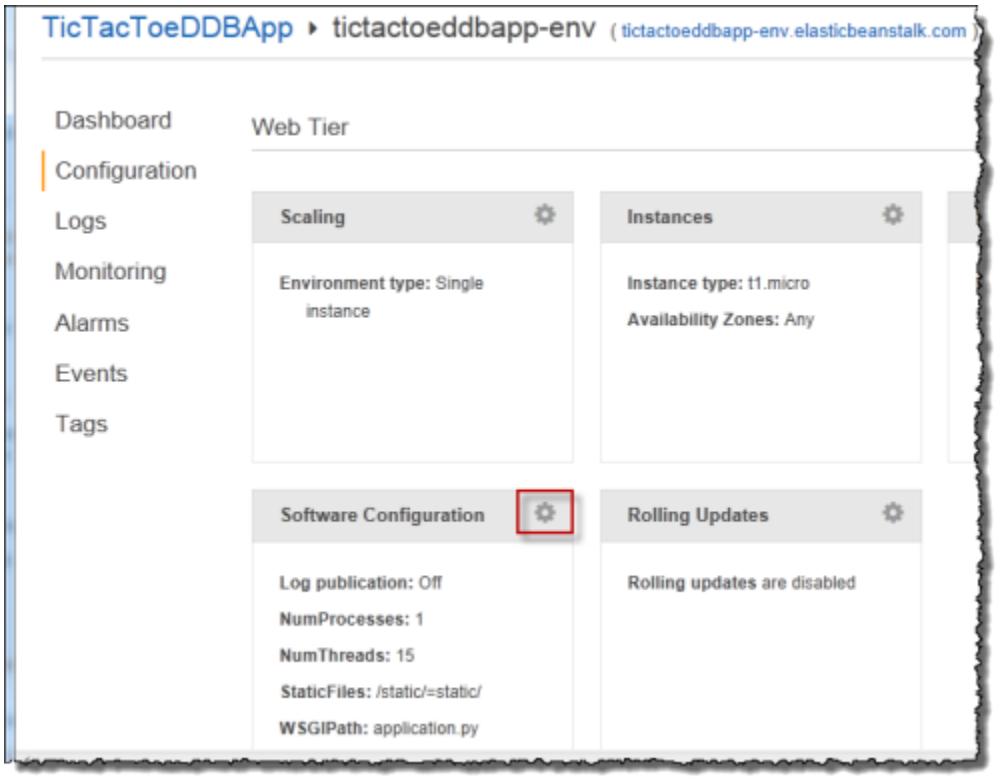
2. Wählen Sie in der Elastic Beanstalk-Konsole Review and Launch (Überprüfen und starten) und anschließend Launch (Starten) aus.
3. Notieren Sie die URL für die spätere Verwendung. Diese URL öffnet die Startseite Ihrer Tic-Tac-Toe Anwendung.



4. Konfigurieren Sie die Tic-Tac-Toe Anwendung so, dass sie den Speicherort der Konfigurationsdatei kennt.

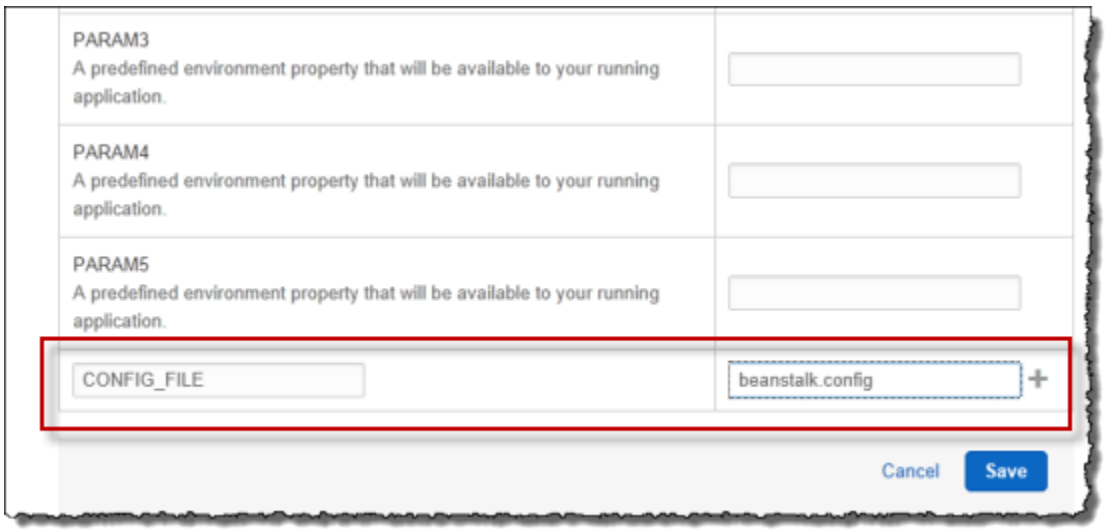
   Nachdem Elastic Beanstalk die Anwendung erstellt hat, wählen Sie Configuration (Konfiguration) aus.

a. Wählen Sie das Zahnradsymbol neben Software Configuration (Softwarekonfiguration) aus, wie in folgendem Screenshot gezeigt.



b. Geben Sie am Ende des Abschnitts Environment Properties (Umgebungseigenschaften) **`CONFIG_FILE`** und dafür den Wert **`beanstalk.config`** ein, und wählen Sie anschließend Save (Speichern) aus.

   Es kann ein paar Minuten dauern, bis die Aktualisierung der Umgebung abgeschlossen ist.

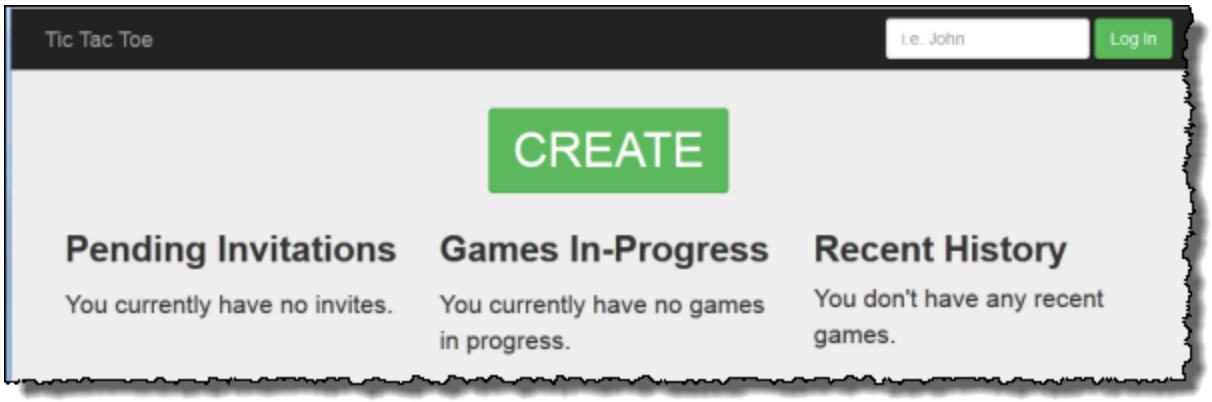


Nachdem die Aktualisierung abgeschlossen ist, können Sie das Spiel spielen.

5. Geben Sie im Browser die URL ein, die Sie im vorherigen Schritt kopiert haben, wie im folgenden Beispiel gezeigt.

```
http://<pen-name>.elasticbeanstalk.com
```

Dadurch wird die Startseite der Anwendung geöffnet.



6. Melde dich als testuser1 an und wähle CREATE, um ein neues tic-tac-toe Spiel zu starten.
7. Geben Sie **`testuser2`** in das Feld Choose an Opponent (Gegner wählen) ein.

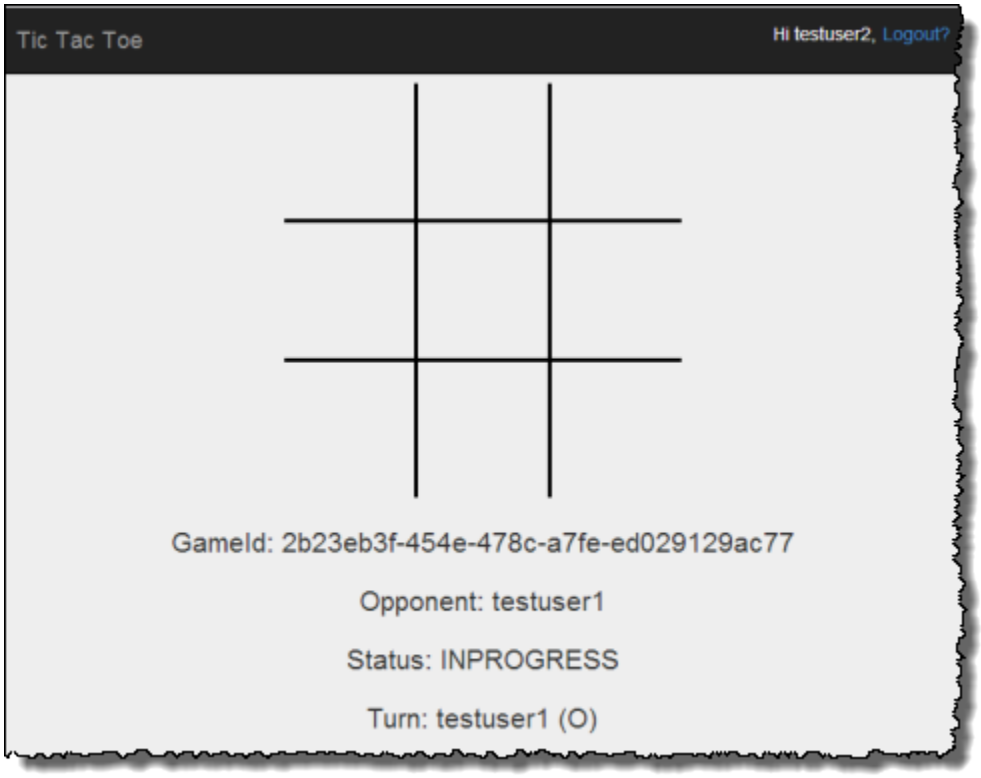
8. Öffnen Sie ein anderes Browserfenster.

   Stellen Sie sicher, dass Sie alle Cookies in Ihrem Browser-Fenster löschen, damit Sie nicht als identischer Benutzer angemeldet werden.
9. Geben Sie dieselbe URL ein, um die Startseite der Anwendung zu öffnen, wie in dem folgenden Beispiel gezeigt.

```
http://<env-name>.elasticbeanstalk.com
```

10. Melden Sie sich als Testbenutzer2 an.
11. Wählen Sie für die Einladung von Testbenutzer1 in der Liste der ausstehenden Einladungen accept (akzeptieren) aus.

12. Jetzt wird die Spielseite angezeigt..



Testbenutzer1 und Testbenutzer2 können das Spiel spielen. Die Anwendung speichert jeden Zug in dem entsprechenden Element in der `Games`-Tabelle.

Schritt 4: Bereinigen von Ressourcen

Jetzt haben Sie die Bereitstellung und das Testen der Tic-Tac-Toe Anwendung abgeschlossen. Die Anwendung deckt die Entwicklung von end-to-end Webanwendungen auf Amazon DynamoDB ab, mit Ausnahme der Benutzerauthentifizierung. Die Anwendung verwendet die Anmeldeinformationen auf der Homepage nur, um einen Spielernamen beim Erstellen eines Spiels hinzuzufügen. In einer Produktionsanwendung fügen Sie den erforderlichen Code hinzu, um die Benutzeranmeldung und Authentifizierung durchzuführen.

Wenn Sie mit dem Testen fertig sind, können Sie die Ressourcen, die Sie zum Testen der Tic-Tac-Toe Anwendung erstellt haben, entfernen, um Gebühren zu vermeiden.

So löschen sie Ressourcen, die Sie erstellt haben:

1. Entfernen Sie die `Games`-Tabelle, die Sie in DynamoDB erstellt haben.
2. Beenden Sie die Elastic-Beanstalk-Umgebung, um die Amazon-EC2-Instances freizugeben.
3. Löschen Sie die IAM-Rolle, die Sie erstellt haben.
4. Entfernen Sie das Objekt, das Sie in Amazon S3 erstellt haben.

Reservierte Wörter in DynamoDB

Die folgenden Schlüsselwörter sind für die Verwendung von DynamoDB reserviert. Verwenden Sie keine dieser Wörter als Attributnamen in Ausdrücken. Bei dieser Liste wird nicht zwischen Groß- und Kleinschreibung unterschieden.

Wenn Sie einen Ausdruck mit einem Attributnamen schreiben möchten, der mit einem DynamoDB-reservierten Wort in Konflikt steht, können Sie einen Namen für ein Ausdrucksattribut definieren, der anstelle des reservierten Worts verwendet wird. Weitere Informationen finden Sie unter Ausdrucksattributnamen (Aliasse) in DynamoDB.

```
ABORT
ABSOLUTE
ACTION
```

```
ADD
AFTER
AGENT
AGGREGATE
ALL
ALLOCATE
ALTER
ANALYZE
AND
ANY
ARCHIVE
ARE
ARRAY
AS
ASC
ASCII
ASENSITIVE
ASSERTION
ASYMMETRIC
AT
ATOMIC
ATTACH
ATTRIBUTE
AUTH
AUTHORIZATION
AUTHORIZE
AUTO
AVG
BACK
BACKUP
BASE
BATCH
BEFORE
BEGIN
BETWEEN
BIGINT
BINARY
BIT
BLOB
BLOCK
BOOLEAN
BOTH
BREADTH
BUCKET
```

```
BULK
BY
BYTE
CALL
CALLED
CALLING
CAPACITY
CASCADE
CASCADED
CASE
CAST
CATALOG
CHAR
CHARACTER
CHECK
CLASS
CLOB
CLOSE
CLUSTER
CLUSTERED
CLUSTERING
CLUSTERS
COALESCE
COLLATE
COLLATION
COLLECTION
COLUMN
COLUMNS
COMBINE
COMMENT
COMMIT
COMPACT
COMPILE
COMPRESS
CONDITION
CONFLICT
CONNECT
CONNECTION
CONSISTENCY
CONSISTENT
CONSTRAINT
CONSTRAINTS
CONSTRUCTOR
CONSUMED
```

```
CONTINUE
CONVERT
COPY
CORRESPONDING
COUNT
COUNTER
CREATE
CROSS
CUBE
CURRENT
CURSOR
CYCLE
DATA
DATABASE
DATE
DATETIME
DAY
DEALLOCATE
DEC
DECIMAL
DECLARE
DEFAULT
DEFERRABLE
DEFERRED
DEFINE
DEFINED
DEFINITION
DELETE
DELIMITED
DEPTH
DEREF
DESC
DESCRIBE
DESCRIPTOR
DETACH
DETERMINISTIC
DIAGNOSTICS
DIRECTORIES
DISABLE
DISCONNECT
DISTINCT
DISTRIBUTE
DO
DOMAIN
```

```
DOUBLE
DROP
DUMP
DURATION
DYNAMIC
EACH
ELEMENT
ELSE
ELSEIF
EMPTY
ENABLE
END
EQUAL
EQUALS
ERROR
ESCAPE
ESCAPED
EVAL
EVALUATE
EXCEEDED
EXCEPT
EXCEPTION
EXCEPTIONS
EXCLUSIVE
EXEC
EXECUTE
EXISTS
EXIT
EXPLAIN
EXPLODE
EXPORT
EXPRESSION
EXTENDED
EXTERNAL
EXTRACT
FAIL
FALSE
FAMILY
FETCH
FIELDS
FILE
FILTER
FILTERING
FINAL
```

```
FINISH
FIRST
FIXED
FLATTERN
FLOAT
FOR
FORCE
FOREIGN
FORMAT
FORWARD
FOUND
FREE
FROM
FULL
FUNCTION
FUNCTIONS
GENERAL
GENERATE
GET
GLOB
GLOBAL
GO
GOTO
GRANT
GREATER
GROUP
GROUPING
HANDLER
HASH
HAVE
HAVING
HEAP
HIDDEN
HOLD
HOUR
IDENTIFIED
IDENTITY
IF
IGNORE
IMMEDIATE
IMPORT
IN
INCLUDING
INCLUSIVE
```

```
INCREMENT
INCREMENTAL
INDEX
INDEXED
INDEXES
INDICATOR
INFINITE
INITIALLY
INLINE
INNER
INNTER
INOUT
INPUT
INSENSITIVE
INSERT
INSTEAD
INT
INTEGER
INTERSECT
INTERVAL
INTO
INVALIDATE
IS
ISOLATION
ITEM
ITEMS
ITERATE
JOIN
KEY
KEYS
LAG
LANGUAGE
LARGE
LAST
LATERAL
LEAD
LEADING
LEAVE
LEFT
LENGTH
LESS
LEVEL
LIKE
LIMIT
```

```
LIMITED
LINES
LIST
LOAD
LOCAL
LOCALTIME
LOCALTIMESTAMP
LOCATION
LOCATOR
LOCK
LOCKS
LOG
LOGED
LONG
LOOP
LOWER
MAP
MATCH
MATERIALIZED
MAX
MAXLEN
MEMBER
MERGE
METHOD
METRICS
MIN
MINUS
MINUTE
MISSING
MOD
MODE
MODIFIES
MODIFY
MODULE
MONTH
MULTI
MULTISET
NAME
NAMES
NATIONAL
NATURAL
NCHAR
NCLOB
NEW
```

```
NEXT
NO
NONE
NOT
NULL
NULLIF
NUMBER
NUMERIC
OBJECT
OF
OFFLINE
OFFSET
OLD
ON
ONLINE
ONLY
OPAQUE
OPEN
OPERATOR
OPTION
OR
ORDER
ORDINALITY
OTHER
OTHERS
OUT
OUTER
OUTPUT
OVER
OVERLAPS
OVERRIDE
OWNER
PAD
PARALLEL
PARAMETER
PARAMETERS
PARTIAL
PARTITION
PARTITIONED
PARTITIONS
PATH
PERCENT
PERCENTILE
PERMISSION
```

```
PERMISSIONS
PIPE
PIPELINED
PLAN
POOL
POSITION
PRECISION
PREPARE
PRESERVE
PRIMARY
PRIOR
PRIVATE
PRIVILEGES
PROCEDURE
PROCESSED
PROJECT
PROJECTION
PROPERTY
PROVISIONING
PUBLIC
PUT
QUERY
QUIT
QUORUM
RAISE
RANDOM
RANGE
RANK
RAW
READ
READS
REAL
REBUILD
RECORD
RECURSIVE
REDUCE
REF
REFERENCE
REFERENCES
REFERENCING
REGEXP
REGION
REINDEX
RELATIVE
```

```
RELEASE
REMAINDER
RENAME
REPEAT
REPLACE
REQUEST
RESET
RESIGNAL
RESOURCE
RESPONSE
RESTORE
RESTRICT
RESULT
RETURN
RETURNING
RETURNS
REVERSE
REVOKE
RIGHT
ROLE
ROLES
ROLLBACK
ROLLUP
ROUTINE
ROW
ROWS
RULE
RULES
SAMPLE
SATISFIES
SAVE
SAVEPOINT
SCAN
SCHEMA
SCOPE
SCROLL
SEARCH
SECOND
SECTION
SEGMENT
SEGMENTS
SELECT
SELF
SEMI
```

```
SENSITIVE
SEPARATE
SEQUENCE
SERIALIZABLE
SESSION
SET
SETS
SHARD
SHARE
SHARED
SHORT
SHOW
SIGNAL
SIMILAR
SIZE
SKEWED
SMALLINT
SNAPSHOT
SOME
SOURCE
SPACE
SPACES
SPARSE
SPECIFIC
SPECIFICTYPE
SPLIT
SQL
SQLCODE
SQLERROR
SQLEXCEPTION
SQLSTATE
SQLWARNING
START
STATE
STATIC
STATUS
STORAGE
STORE
STORED
STREAM
STRING
STRUCT
STYLE
SUB
```

```
SUBMULTISET
SUBPARTITION
SUBSTRING
SUBTYPE
SUM
SUPER
SYMMETRIC
SYNONYM
SYSTEM
TABLE
TABLESAMPLE
TEMP
TEMPORARY
TERMINATED
TEXT
THAN
THEN
THROUGHPUT
TIME
TIMESTAMP
TIMEZONE
TINYINT
TO
TOKEN
TOTAL
TOUCH
TRAILING
TRANSACTION
TRANSFORM
TRANSLATE
TRANSLATION
TREAT
TRIGGER
TRIM
TRUE
TRUNCATE
TTL
TUPLE
TYPE
UNDER
UNDO
UNION
UNIQUE
UNIT
```

```
UNKNOWN
UNLOGGED
UNNEST
UNPROCESSED
UNSIGNED
UNTIL
UPDATE
UPPER
URL
USAGE
USE
USER
USERS
USING
UUID
VACUUM
VALUE
VALUED
VALUES
VARCHAR
VARIABLE
VARIANCE
VARINT
VARYING
VIEW
VIEWS
VIRTUAL
VOID
WAIT
WHEN
WHENEVER
WHERE
WHILE
WINDOW
WITH
WITHIN
WITHOUT
WORK
WRAPPED
WRITE
YEAR
ZONE
```

AWS SDK-Beispiele für Java 1.x

Dieser Abschnitt enthält Beispielcode für DAX-Anwendungen, die SDK for Java 1.x verwenden.

Themen

- Verwenden von DAX mit AWS SDK for Java 1.x
- Verwenden einer vorhandenen SDK for Java 1.x Anwendung zur Nutzung von DAX
- Abfragen von globalen sekundären Indizes mit SDK for Java 1.x

Verwenden von DAX mit AWS SDK for Java 1.x

Um das Java-Beispiel für Amazon DynamoDB Accelerator (DAX) auf der Amazon-EC2-Instance auszuführen, gehen Sie wie folgt vor.

> **Note**
>
> Diese Anweisungen gelten für Anwendungen, die AWS SDK for Java 1.x verwenden. Für Anwendungen, die AWS SDK for Java 2.x verwenden, siehe Java und DAX.

So führen Sie das Java-Beispiel für DAX aus

1. Installieren des Java Development Kits (JDK).

   ```
   sudo yum install -y java-devel
   ```

2. Laden Sie AWS SDK für Java (`.zip`-Datei) herunter und extrahieren Sie die Datei.

   ```
   wget http://sdk-for-java.amazonwebservices.com/latest/aws-java-sdk.zip

   unzip aws-java-sdk.zip
   ```

3. Laden Sie die neueste Version des DAX-Java-Clients (`.jar`-Datei) herunter:

   ```
   wget http://dax-sdk.s3-website-us-west-2.amazonaws.com/java/DaxJavaClient-
   latest.jar
   ```

> **Note**
>
> Der Client für das DAX SDK for Java ist auf Apache Maven verfügbar. Weitere Informationen finden Sie unter Verwenden eines Clients als Apache Maven-Abhängigkeit.

4. Legen Sie Ihre CLASSPATH-Variable fest. Ersetzen Sie in diesem Beispiel *sdkVersion* durch die tatsächliche Versionsnummer von AWS SDK für Java, zum Beispiel `1.11.112`.

```
export SDKVERSION=sdkVersion

export CLASSPATH=$(pwd)/TryDax/java:$(pwd)/DaxJavaClient-latest.jar:$(pwd)/
aws-java-sdk-$SDKVERSION/lib/aws-java-sdk-$SDKVERSION.jar:$(pwd)/aws-java-sdk-
$SDKVERSION/third-party/lib/*
```

5. Downloaden Sie den Quellcode des Beispielprogramms (`.zip`-Datei):

```
wget http://docs.aws.amazon.com/amazondynamodb/latest/developerguide/samples/
TryDax.zip
```

   Wenn der Download abgeschlossen ist, extrahieren Sie die Quelldateien.

```
unzip TryDax.zip
```

6. Navigieren Sie zum Java-Code-Verzeichnis und kompilieren Sie den Code wie folgt.

```
cd TryDax/java/
javac TryDax*.java
```

7. Führen Sie das Programm aus.

```
java TryDax
```

   Die Ausgabe sollte in etwa wie folgt aussehen:

```
Creating a DynamoDB client

Attempting to create table; please wait...
Successfully created table.  Table status: ACTIVE
```

```
Writing data to the table...
Writing 10 items for partition key: 1
Writing 10 items for partition key: 2
Writing 10 items for partition key: 3
Writing 10 items for partition key: 4
Writing 10 items for partition key: 5
Writing 10 items for partition key: 6
Writing 10 items for partition key: 7
Writing 10 items for partition key: 8
Writing 10 items for partition key: 9
Writing 10 items for partition key: 10

Running GetItem, Scan, and Query tests...
First iteration of each test will result in cache misses
Next iterations are cache hits

GetItem test - partition key 1 and sort keys 1-10
 Total time: 136.681 ms - Avg time: 13.668 ms
 Total time: 122.632 ms - Avg time: 12.263 ms
 Total time: 167.762 ms - Avg time: 16.776 ms
 Total time: 108.130 ms - Avg time: 10.813 ms
 Total time: 137.890 ms - Avg time: 13.789 ms
Query test - partition key 5 and sort keys between 2 and 9
 Total time: 13.560 ms - Avg time: 2.712 ms
 Total time: 11.339 ms - Avg time: 2.268 ms
 Total time: 7.809 ms - Avg time: 1.562 ms
 Total time: 10.736 ms - Avg time: 2.147 ms
 Total time: 12.122 ms - Avg time: 2.424 ms
Scan test - all items in the table
 Total time: 58.952 ms - Avg time: 11.790 ms
 Total time: 25.507 ms - Avg time: 5.101 ms
 Total time: 37.660 ms - Avg time: 7.532 ms
 Total time: 26.781 ms - Avg time: 5.356 ms
 Total time: 46.076 ms - Avg time: 9.215 ms

Attempting to delete table; please wait...
Successfully deleted table.
```

Beachten Sie die Zeitinformationen – die Anzahl der benötigten Millisekunden für den `GetItem`-, `Query`- und `Scan`-Test.

8. Im vorherigen Schritt führten Sie das Programm über den DynamoDB-Endpunkt aus. Führen Sie das Programm jetzt erneut aus. Dieses Mal werden die `GetItem`-, `Query`- und `Scan`-Operationen jedoch vom DAX-Cluster verarbeitet.

   Um den Endpunkt für Ihren DAX-Cluster zu bestimmen, wählen Sie einen der folgenden Schritte aus:

   - Using the DynamoDB console (Verwenden der DynamoDB-Konsole) — Wählen Sie Ihren DAX-Cluster aus. Der Cluster-Endpunkt wird auf der Konsole angezeigt, wie im folgenden Beispiel gezeigt.

     ```
     dax://my-cluster.l6fzcv.dax-clusters.us-east-1.amazonaws.com
     ```

   - Verwenden von AWS CLI – Geben Sie den folgenden Befehl ein.

     ```
     aws dax describe-clusters --query "Clusters[*].ClusterDiscoveryEndpoint"
     ```

     Der Cluster-Endpunkt wird in der Ausgabe angezeigt, wie im folgenden Beispiel gezeigt.

     ```
     {
         "Address": "my-cluster.l6fzcv.dax-clusters.us-east-1.amazonaws.com",
         "Port": 8111,
         "URL": "dax://my-cluster.l6fzcv.dax-clusters.us-east-1.amazonaws.com"
     }
     ```

   Führen Sie jetzt das Programm erneut aus. Geben Sie dieses Mal jedoch den Cluster-Endpunkt als Befehlszeilenparameter an.

   ```
   java TryDax dax://my-cluster.l6fzcv.dax-clusters.us-east-1.amazonaws.com
   ```

   Sehen Sie sich den Rest der Ausgabe an und notieren Sie die Zeitinformationen. Die verstrichene Zeit sollte für `GetItem`, `Query` und `Scan` mit DAX deutlich kürzer sein als mit DynamoDB.

Weitere Informationen zu diesem Programm finden Sie in folgenden Abschnitten:

- TryDax.java

- TryDaxHelper.java
- TryDaxTests.java

Verwenden eines Clients als Apache Maven-Abhängigkeit

Führen Sie die folgenden Schritte aus, um den Client für das DAX SDK for Java als Abhängigkeit in Ihrer Anwendung zu verwenden.

So verwenden Sie den Client als Maven-Abhängigkeit

1. Laden Sie Apache Maven herunter und installieren Sie es. Weitere Informationen finden Sie unter Downloading Apache Maven und Installing Apache Maven.
2. Fügen Sie die Client-Maven-Abhängigkeit der POM-Datei (Project Object Model) Ihrer Anwendung hinzu. Ersetzen Sie in diesem Beispiel `x.x.x.x` durch die tatsächliche Versionsnummer des Clients (z. B. `1.0.200704.0`).

```
<!--Dependency:-->
<dependencies>
    <dependency>
     <groupId>com.amazonaws</groupId>
     <artifactId>amazon-dax-client</artifactId>
     <version>x.x.x.x</version>
    </dependency>
</dependencies>
```

TryDax.java

Die `TryDax.java`-Datei enthält die `main`-Methode. Wenn Sie das Programm ohne Befehlszeilenparameter ausführen, erstellt es einen Amazon-DynamoDB-Client und verwendet diesen für alle API-Operationen. Wenn Sie einen DynamoDB-Accelerator-(DAX)-Cluster-Endpunkt in der Befehlszeile angeben, erstellt das Programm auch einen DAX-Client und verwendet diesen für `GetItem`-, `Query`- und `Scan`-Operationen.

Sie können das Programm auf verschiedene Arten ändern.

- Verwenden Sie den DAX-Client anstelle des DynamoDB-Clients. Weitere Informationen finden Sie unter Java und DAX.
- Wählen Sie einen anderen Namen für die Testtabelle aus.

- Ändern Sie die Anzahl der Elemente, indem Sie die Parameter `helper.writeData` ändern. Der zweite Parameter ist die Anzahl der Partitionsschlüssel und der dritte Parameter ist die Anzahl der Sortierschlüssel. Standardmäßig verwendet das Programm 1-10 für Partitions-Schlüsselwerte und 1-10 für Sortierschlüsselwerte für insgesamt 100 Elemente, die in die Tabelle geschrieben werden. Weitere Informationen finden Sie unter TryDaxHelper.java.
- Ändern Sie die Anzahl der `GetItem`-, `Query`- und `Scan`-Tests und ändern Sie deren Parameter.
- Kommentieren Sie die Zeilen, die `helper.createTable` und `helper.deleteTable` beinhalten (wenn Sie die Tabelle nicht bei jeder Nutzung des Programms erstellen und löschen möchten).

> **Note**
>
> Zur Ausführung dieses Programms können Sie Maven so einrichten, dass der Client für das DAX SDK for Java und AWS SDK für Java als Abhängigkeiten verwendet werden. Weitere Informationen finden Sie unter Verwenden eines Clients als Apache Maven-Abhängigkeit. Sie können den DAX-Java-Client und AWS SDK für Java auch herunterladen und in den Klassenpfad einschließen. Unter Java und DAX finden Sie ein Beispiel für die Einrichtung Ihrer `CLASSPATH`-Variablen.

```
public class TryDax {

    public static void main(String[] args) throws Exception {

        TryDaxHelper helper = new TryDaxHelper();
        TryDaxTests tests = new TryDaxTests();

        DynamoDB ddbClient = helper.getDynamoDBClient();
        DynamoDB daxClient = null;
        if (args.length >= 1) {
            daxClient = helper.getDaxClient(args[0]);
        }

        String tableName = "TryDaxTable";

        System.out.println("Creating table...");
        helper.createTable(tableName, ddbClient);
        System.out.println("Populating table...");
        helper.writeData(tableName, ddbClient, 10, 10);
```

```
        DynamoDB testClient = null;
        if (daxClient != null) {
            testClient = daxClient;
        } else {
            testClient = ddbClient;
        }

        System.out.println("Running GetItem, Scan, and Query tests...");
        System.out.println("First iteration of each test will result in cache misses");
        System.out.println("Next iterations are cache hits\n");

        // GetItem
        tests.getItemTest(tableName, testClient, 1, 10, 5);

        // Query
        tests.queryTest(tableName, testClient, 5, 2, 9, 5);

        // Scan
        tests.scanTest(tableName, testClient, 5);

        helper.deleteTable(tableName, ddbClient);
    }

}
```

TryDaxHelper.java

Die `TryDaxHelper.java`-Datei enthält Dienstprogrammmethoden.

Die `getDynamoDBClient`- und `getDaxClient`-Methoden stellen Amazon-DynamoDB- und DynamoDB-Accelerator-(DAX)-Clients bereit. Für Vorgänge der Steuerebene (`CreateTable`, `DeleteTable`) und Schreibvorgänge verwendet das Programm den DynamoDB-Client. Wenn Sie einen DAX-Cluster-Endpunkt angeben, erstellt das Hauptprogramm einen DAX-Client für das Durchführen von Schreibvorgänge (`GetItem`, `Query`, `Scan`).

Die anderen `TryDaxHelper`-Methoden (`createTable`, `writeData`, `deleteTable`) dienen dem Einrichten und Entfernen der DynamoDB-Tabelle und der enthaltenen Daten.

Sie können das Programm auf verschiedene Arten ändern.

- Verwenden Sie verschiedene bereitgestellte Durchsatzeinstellungen für die Tabelle.

- Ändern Sie die Größe jedes geschriebenen Elements (siehe `stringSize`-Variablen in der `writeData`-Methode).
- Ändern Sie die Anzahl der `GetItem`-, `Query`- und `Scan`-Tests und deren Parameter.
- Kommentieren Sie die Zeilen, die `helper.CreateTable` und `helper.DeleteTable` beinhalten (wenn Sie die Tabelle nicht bei jeder Nutzung des Programms erstellen und löschen möchten).

> **Note**
>
> Zur Ausführung dieses Programms können Sie Maven so einrichten, dass der Client für das DAX SDK for Java und AWS SDK für Java als Abhängigkeiten verwendet werden. Weitere Informationen finden Sie unter Verwenden eines Clients als Apache Maven-Abhängigkeit. Sie können den DAX-Java-Client und AWS SDK für Java auch herunterladen und in den Klassenpfad einschließen. Unter Java und DAX finden Sie ein Beispiel für die Einrichtung Ihrer `CLASSPATH`-Variablen.

```
import com.amazon.dax.client.dynamodbv2.AmazonDaxClientBuilder;
import com.amazonaws.services.dynamodbv2.AmazonDynamoDB;
import com.amazonaws.services.dynamodbv2.AmazonDynamoDBClientBuilder;
import com.amazonaws.services.dynamodbv2.document.DynamoDB;
import com.amazonaws.services.dynamodbv2.document.Item;
import com.amazonaws.services.dynamodbv2.document.Table;
import com.amazonaws.services.dynamodbv2.model.AttributeDefinition;
import com.amazonaws.services.dynamodbv2.model.KeySchemaElement;
import com.amazonaws.services.dynamodbv2.model.KeyType;
import com.amazonaws.services.dynamodbv2.model.ProvisionedThroughput;
import com.amazonaws.services.dynamodbv2.model.ScalarAttributeType;
import com.amazonaws.util.EC2MetadataUtils;

public class TryDaxHelper {

    private static final String region = EC2MetadataUtils.getEC2InstanceRegion();

    DynamoDB getDynamoDBClient() {
        System.out.println("Creating a DynamoDB client");
        AmazonDynamoDB client = AmazonDynamoDBClientBuilder.standard()
                .withRegion(region)
                .build();
```

```
        return new DynamoDB(client);
    }

    DynamoDB getDaxClient(String daxEndpoint) {
        System.out.println("Creating a DAX client with cluster endpoint " +
daxEndpoint);
        AmazonDaxClientBuilder daxClientBuilder = AmazonDaxClientBuilder.standard();
        daxClientBuilder.withRegion(region).withEndpointConfiguration(daxEndpoint);
        AmazonDynamoDB client = daxClientBuilder.build();
        return new DynamoDB(client);
    }

    void createTable(String tableName, DynamoDB client) {
        Table table = client.getTable(tableName);
        try {
            System.out.println("Attempting to create table; please wait...");

            table = client.createTable(tableName,
                    Arrays.asList(
                            new KeySchemaElement("pk", KeyType.HASH), // Partition key
                            new KeySchemaElement("sk", KeyType.RANGE)), // Sort key
                    Arrays.asList(
                            new AttributeDefinition("pk", ScalarAttributeType.N),
                            new AttributeDefinition("sk", ScalarAttributeType.N)),
                    new ProvisionedThroughput(10L, 10L));
            table.waitForActive();
            System.out.println("Successfully created table.  Table status: " +
                    table.getDescription().getTableStatus());

        } catch (Exception e) {
            System.err.println("Unable to create table: ");
            e.printStackTrace();
        }
    }

    void writeData(String tableName, DynamoDB client, int pkmax, int skmax) {
        Table table = client.getTable(tableName);
        System.out.println("Writing data to the table...");

        int stringSize = 1000;
        StringBuilder sb = new StringBuilder(stringSize);
        for (int i = 0; i < stringSize; i++) {
            sb.append('X');
        }
```

```
        String someData = sb.toString();

        try {
            for (Integer ipk = 1; ipk <= pkmax; ipk++) {
                System.out.println(("Writing " + skmax + " items for partition key: " +
 ipk));
                for (Integer isk = 1; isk <= skmax; isk++) {
                    table.putItem(new Item()
                            .withPrimaryKey("pk", ipk, "sk", isk)
                            .withString("someData", someData));
                }
            }
        } catch (Exception e) {
            System.err.println("Unable to write item:");
            e.printStackTrace();
        }
    }

    void deleteTable(String tableName, DynamoDB client) {
        Table table = client.getTable(tableName);
        try {
            System.out.println("\nAttempting to delete table; please wait...");
            table.delete();
            table.waitForDelete();
            System.out.println("Successfully deleted table.");

        } catch (Exception e) {
            System.err.println("Unable to delete table: ");
            e.printStackTrace();
        }
    }

}
```

TryDaxTests.java

Die `TryDaxTests.java`-Datei enthält Methoden, die Leseoperationen über eine Testtabelle in Amazon DynamoDB durchführen. Diese Methoden berücksichtigen nicht, wie sie auf die Daten zugreifen (entweder mithilfe des DynamoDB-Clients oder des DAX-Clients). Daher ist es nicht erforderlich, die Anwendungslogik zu ändern.

Sie können das Programm auf verschiedene Arten ändern.

- Ändern Sie die `queryTest`-Methode, damit sie einen anderen `KeyConditionExpression` verwendet.
- Fügen Sie der `scanTest`-Methode einen `ScanFilter` hinzu, damit nur einige der Elemente an Sie zurückgegeben werden.

> **Note**
>
> Zur Ausführung dieses Programms können Sie Maven so einrichten, dass der Client für das DAX SDK for Java und AWS SDK für Java als Abhängigkeiten verwendet werden. Weitere Informationen finden Sie unter Verwenden eines Clients als Apache Maven-Abhängigkeit. Sie können den DAX-Java-Client und AWS SDK für Java auch herunterladen und in den Klassenpfad einschließen. Unter Java und DAX finden Sie ein Beispiel für die Einrichtung Ihrer `CLASSPATH`-Variablen.

```
import java.util.Iterator;

import com.amazonaws.services.dynamodbv2.document.DynamoDB;
import com.amazonaws.services.dynamodbv2.document.Item;
import com.amazonaws.services.dynamodbv2.document.ItemCollection;
import com.amazonaws.services.dynamodbv2.document.QueryOutcome;
import com.amazonaws.services.dynamodbv2.document.ScanOutcome;
import com.amazonaws.services.dynamodbv2.document.Table;
import com.amazonaws.services.dynamodbv2.document.spec.QuerySpec;

public class TryDaxTests {

    void getItemTest(String tableName, DynamoDB client, int pk, int sk, int iterations)
 {
        long startTime, endTime;
        System.out.println("GetItem test - partition key " + pk + " and sort keys 1-" +
 sk);
        Table table = client.getTable(tableName);

        for (int i = 0; i < iterations; i++) {
            startTime = System.nanoTime();
            try {
                for (Integer ipk = 1; ipk <= pk; ipk++) {
                    for (Integer isk = 1; isk <= sk; isk++) {
                        table.getItem("pk", ipk, "sk", isk);
```

```
                    }
                }
            } catch (Exception e) {
                System.err.println("Unable to get item:");
                e.printStackTrace();
            }
            endTime = System.nanoTime();
            printTime(startTime, endTime, pk * sk);
        }
    }

    void queryTest(String tableName, DynamoDB client, int pk, int sk1, int sk2, int
iterations) {
        long startTime, endTime;
        System.out.println("Query test - partition key " + pk + " and sort keys between
" + sk1 + " and " + sk2);
        Table table = client.getTable(tableName);

        HashMap<String, Object> valueMap = new HashMap<String, Object>();
        valueMap.put(":pkval", pk);
        valueMap.put(":skval1", sk1);
        valueMap.put(":skval2", sk2);

        QuerySpec spec = new QuerySpec()
                .withKeyConditionExpression("pk = :pkval and sk between :skval1
and :skval2")
                .withValueMap(valueMap);

        for (int i = 0; i < iterations; i++) {
            startTime = System.nanoTime();
            ItemCollection<QueryOutcome> items = table.query(spec);

            try {
                Iterator<Item> iter = items.iterator();
                while (iter.hasNext()) {
                    iter.next();
                }
            } catch (Exception e) {
                System.err.println("Unable to query table:");
                e.printStackTrace();
            }
            endTime = System.nanoTime();
            printTime(startTime, endTime, iterations);
        }
```

```
    }

    void scanTest(String tableName, DynamoDB client, int iterations) {
        long startTime, endTime;
        System.out.println("Scan test - all items in the table");
        Table table = client.getTable(tableName);

        for (int i = 0; i < iterations; i++) {
            startTime = System.nanoTime();
            ItemCollection<ScanOutcome> items = table.scan();
            try {

                Iterator<Item> iter = items.iterator();
                while (iter.hasNext()) {
                    iter.next();
                }
            } catch (Exception e) {
                System.err.println("Unable to scan table:");
                e.printStackTrace();
            }
            endTime = System.nanoTime();
            printTime(startTime, endTime, iterations);
        }
    }

    public void printTime(long startTime, long endTime, int iterations) {
        System.out.format("\tTotal time: %.3f ms - ", (endTime - startTime) /
 (1000000.0));
        System.out.format("Avg time: %.3f ms\n", (endTime - startTime) / (iterations *
 1000000.0));
    }
}
```

Verwenden einer vorhandenen SDK for Java 1.x Anwendung zur Nutzung von DAX

Wenn Sie bereits über eine Java-Anwendung verfügen, die Amazon DynamoDB verwendet, müssen Sie sie so ändern, dass sie auf den DynamoDB-Accelerator-(DAX)-Cluster zugreifen kann. Sie müssen nicht die gesamte Anwendung umschreiben, da der DAX-Java-Client dem DynamoDB-Low-Level-Client ähnelt, der in AWS SDK für Java enthalten ist.

> **Note**
>
> Diese Anweisungen gelten für Anwendungen, die AWS SDK for Java 1.x verwenden. Für Anwendungen, die AWS SDK for Java 2.x verwenden, siehe Ändern einer vorhandenen Anwendung für die Verwendung von DAX.

Angenommen, Sie verfügen über eine DynamoDB-Tabelle mit dem Namen `Music`. Der Partitionsschlüssel für diese Tabelle lautet `Artist` und der Sortierschlüssel ist `SongTitle`. Das folgende Programm liest ein Element direkt aus der Tabelle `Music`.

```
import java.util.HashMap;

import com.amazonaws.services.dynamodbv2.AmazonDynamoDB;
import com.amazonaws.services.dynamodbv2.AmazonDynamoDBClientBuilder;
import com.amazonaws.services.dynamodbv2.model.AttributeValue;
import com.amazonaws.services.dynamodbv2.model.GetItemRequest;
import com.amazonaws.services.dynamodbv2.model.GetItemResult;

public class GetMusicItem {

    public static void main(String[] args) throws Exception {

        // Create a DynamoDB client
        AmazonDynamoDB client = AmazonDynamoDBClientBuilder.standard().build();

        HashMap<String, AttributeValue> key = new HashMap<String, AttributeValue>();
        key.put("Artist", new AttributeValue().withS("No One You Know"));
        key.put("SongTitle", new AttributeValue().withS("Scared of My Shadow"));

        GetItemRequest request = new GetItemRequest()
            .withTableName("Music").withKey(key);

        try {
            System.out.println("Attempting to read the item...");
            GetItemResult result = client.getItem(request);
            System.out.println("GetItem succeeded: " + result);

        } catch (Exception e) {
            System.err.println("Unable to read item");
            System.err.println(e.getMessage());
        }
```

```
    }
}
```

Zum Ändern des Programms ersetzen Sie den DynamoDB-Client durch einen DAX-Client.

```
import java.util.HashMap;

import com.amazonaws.services.dynamodbv2.AmazonDynamoDB;
import com.amazon.dax.client.dynamodbv2.AmazonDaxClientBuilder;
import com.amazonaws.services.dynamodbv2.model.AttributeValue;
import com.amazonaws.services.dynamodbv2.model.GetItemRequest;
import com.amazonaws.services.dynamodbv2.model.GetItemResult;

public class GetMusicItem {

    public static void main(String[] args) throws Exception {

    //Create a DAX client

    AmazonDaxClientBuilder daxClientBuilder = AmazonDaxClientBuilder.standard();
    daxClientBuilder.withRegion("us-
east-1").withEndpointConfiguration("mydaxcluster.2cmrwl.clustercfg.dax.use1.cache.amazonaws.com
    AmazonDynamoDB client = daxClientBuilder.build();


       /*
       ** ...
       ** Remaining code omitted (it is identical)
       ** ...
       */

    }
}
```

Verwenden der DynamoDB-Dokument-API

Das AWS SDK für Java stellt eine Dokumentschnittstelle für DynamoDB bereit. Die Dokument-API agiert als Wrapper um den Low-Level-Client von DynamoDB. Weitere Informationen finden Sie unter Dokumentschnittstellen.

Die Dokumentschnittstelle kann auch mit dem Low-Level-Client von DAX verwendet werden, wie im folgenden Beispiel gezeigt.

```
import com.amazonaws.services.dynamodbv2.AmazonDynamoDB;
import com.amazon.dax.client.dynamodbv2.AmazonDaxClientBuilder;
import com.amazonaws.services.dynamodbv2.document.DynamoDB;
import com.amazonaws.services.dynamodbv2.document.GetItemOutcome;
import com.amazonaws.services.dynamodbv2.document.Table;

public class GetMusicItemWithDocumentApi {

    public static void main(String[] args) throws Exception {

        //Create a DAX client

        AmazonDaxClientBuilder daxClientBuilder = AmazonDaxClientBuilder.standard();
        daxClientBuilder.withRegion("us-
east-1").withEndpointConfiguration("mydaxcluster.2cmrwl.clustercfg.dax.use1.cache.amazonaws.com
        AmazonDynamoDB client = daxClientBuilder.build();

        // Document client wrapper
        DynamoDB docClient = new DynamoDB(client);

        Table table = docClient.getTable("Music");

        try {
            System.out.println("Attempting to read the item...");
            GetItemOutcome outcome = table.tgetItemOutcome(
                "Artist", "No One You Know",
                "SongTitle", "Scared of My Shadow");
            System.out.println(outcome.getItem());
            System.out.println("GetItem succeeded: " + outcome);
        } catch (Exception e) {
            System.err.println("Unable to read item");
            System.err.println(e.getMessage());
        }

    }
}
```

DAX-Async-Client

Der `AmazonDaxClient` ist synchron. Lang andauernde DAX-API-Operationen wie z.B. ein `Scan` einer sehr großen Tabelle können die Ausführung des Programms blockieren, bis die Operation

abgeschlossen ist. Wenn Ihr Programm andere Aufgaben erledigen muss, während eine DAX-API-Operation ausgeführt wird, können Sie stattdessen `ClusterDaxAsyncClient` verwenden.

Das folgende Programm zeigt, wie Sie `ClusterDaxAsyncClient` mit Java `Future` verwenden, um eine Lösung zu implementieren, die das Programm nicht behindert.

```
import java.util.concurrent.ExecutionException;
import java.util.concurrent.Future;

import com.amazon.dax.client.dynamodbv2.ClientConfig;
import com.amazon.dax.client.dynamodbv2.ClusterDaxAsyncClient;
import com.amazonaws.auth.profile.ProfileCredentialsProvider;
import com.amazonaws.handlers.AsyncHandler;
import com.amazonaws.services.dynamodbv2.AmazonDynamoDBAsync;
import com.amazonaws.services.dynamodbv2.model.AttributeValue;
import com.amazonaws.services.dynamodbv2.model.GetItemRequest;
import com.amazonaws.services.dynamodbv2.model.GetItemResult;

public class DaxAsyncClientDemo {
 public static void main(String[] args) throws Exception {

  ClientConfig daxConfig = new ClientConfig().withCredentialsProvider(new
 ProfileCredentialsProvider())
    .withEndpoints("mydaxcluster.2cmrwl.clustercfg.dax.use1.cache.amazonaws.com:8111");

  AmazonDynamoDBAsync client = new ClusterDaxAsyncClient(daxConfig);

  HashMap<String, AttributeValue> key = new HashMap<String, AttributeValue>();
  key.put("Artist", new AttributeValue().withS("No One You Know"));
  key.put("SongTitle", new AttributeValue().withS("Scared of My Shadow"));

  GetItemRequest request = new GetItemRequest()
    .withTableName("Music").withKey(key);

  // Java Futures
  Future<GetItemResult> call = client.getItemAsync(request);
  while (!call.isDone()) {
   // Do other processing while you're waiting for the response
   System.out.println("Doing something else for a few seconds...");
   Thread.sleep(3000);
  }
  // The results should be ready by now
```

```
  try {
   call.get();

  } catch (ExecutionException ee) {
   // Futures always wrap errors as an ExecutionException.
   // The *real* exception is stored as the cause of the
   // ExecutionException
   Throwable exception = ee.getCause();
   System.out.println("Error getting item: " + exception.getMessage());
  }

  // Async callbacks
  call = client.getItemAsync(request, new AsyncHandler<GetItemRequest, GetItemResult>()
 {

   @Override
   public void onSuccess(GetItemRequest request, GetItemResult getItemResult) {
    System.out.println("Result: " + getItemResult);
   }

   @Override
   public void onError(Exception e) {
    System.out.println("Unable to read item");
    System.err.println(e.getMessage());
    // Callers can also test if exception is an instance of
    // AmazonServiceException or AmazonClientException and cast
    // it to get additional information
   }

  });
  call.get();

 }
}
```

Abfragen von globalen sekundären Indizes mit SDK for Java 1.x

Sie können Amazon DynamoDB Accelerator (DAX) verwenden, um globale sekundäre Indizes von DynamoDB-Programmierschnittstellen abzufragen.

Das folgende Beispiel zeigt, wie DAX verwendet wird, um den `CreateDateIndex` globalen sekundären Index abzufragen, der in Beispiel: Globale Sekundärindizes mithilfe der AWS SDK für Java Dokument-API erstellt wurde.

Die `DAXClient`-Klasse instanziiert die Client-Objekte, die für die Interaktion mit den DynamoDB-Programmierschnittstellen benötigt werden.

```
import com.amazon.dax.client.dynamodbv2.AmazonDaxClientBuilder;
import software.amazon.dynamodb.datamodeling.DynamoDBMapper;
import software.amazon.dynamodb.document.DynamoDB;
import com.amazonaws.util.EC2MetadataUtils;
import software.amazon.dynamodb.AmazonDynamoDB;

public class DaxClient {

 private static final String region = EC2MetadataUtils.getEC2InstanceRegion();

 DynamoDB getDaxDocClient(String daxEndpoint) {
  System.out.println("Creating a DAX client with cluster endpoint " + daxEndpoint);
  AmazonDaxClientBuilder daxClientBuilder = AmazonDaxClientBuilder.standard();

  daxClientBuilder.withRegion(region).withEndpointConfiguration(daxEndpoint);
  AmazonDynamoDB client = daxClientBuilder.build();

  return new DynamoDB(client);
 }

 DynamoDBMapper getDaxMapperClient(String daxEndpoint) {
  System.out.println("Creating a DAX client with cluster endpoint " + daxEndpoint);
  AmazonDaxClientBuilder daxClientBuilder = AmazonDaxClientBuilder.standard();

  daxClientBuilder.withRegion(region).withEndpointConfiguration(daxEndpoint);
  AmazonDynamoDB client = daxClientBuilder.build();

  return new DynamoDBMapper(client);
 }
}
```

Sie können globale sekundäre Indizes folgendermaßen abfragen:

- Verwenden Sie die Methode `queryIndex` für die im folgenden Beispiel definierte `QueryIndexDax`-Klasse. `QueryIndexDax` verwendet als Parameter das Client-Objekt, das von der Methode `getDaxDocClient` für die `DaxClient`-Klasse zurückgegeben wird.
- Wenn Sie die Objektpersistenzschnittstelle verwenden, verwenden Sie die Methode `queryIndexMapper` für die im folgenden Beispiel definierte `QueryIndexDax`-Klasse. `queryIndexMapper` verwendet als Parameter das Client-Objekt, das von der Methode `getDaxMapperClient` für die `DaxClient`-Klasse zurückgegeben wird.

```
import java.util.Iterator;
import software.amazon.dynamodb.datamodeling.DynamoDBMapper;
import java.util.List;
import software.amazon.dynamodb.datamodeling.DynamoDBQueryExpression;
import software.amazon.dynamodb.model.AttributeValue;
import java.util.HashMap;
import software.amazon.dynamodb.document.Item;
import software.amazon.dynamodb.document.utils.ValueMap;
import software.amazon.dynamodb.document.spec.QuerySpec;
import software.amazon.dynamodb.document.QueryOutcome;
import software.amazon.dynamodb.document.ItemCollection;
import software.amazon.dynamodb.document.Index;
import software.amazon.dynamodb.document.Table;
import software.amazon.dynamodb.document.DynamoDB;

public class QueryIndexDax {

 //This is used to query Index using the low-level interface.
 public static void queryIndex(DynamoDB client, String tableName, String indexName) {
  Table table = client.getTable(tableName);

  System.out.println("\n*************************************************************
\n");
  System.out.print("Querying index " + indexName + "...");

  Index index = table.getIndex(indexName);

  ItemCollection<QueryOutcome> items = null;

  QuerySpec querySpec = new QuerySpec();

  if (indexName == "CreateDateIndex") {
   System.out.println("Issues filed on 2013-11-01");
```

```
  querySpec.withKeyConditionExpression("CreateDate = :v_date and
begins_with(IssueId, :v_issue)")
    .withValueMap(new ValueMap().withString(":v_date",
"2013-11-01").withString(":v_issue", "A-"));
  items = index.query(querySpec);
 } else {
  System.out.println("\nNo valid index name provided");
  return;
 }

 Iterator<Item> iterator = items.iterator();

 System.out.println("Query: printing results...");

 while (iterator.hasNext()) {
  System.out.println(iterator.next().toJSONPretty());
 }

}

//This is used to query Index using the high-level mapper interface.
public static void queryIndexMapper(DynamoDBMapper mapper, String tableName, String
indexName) {
 HashMap<String, AttributeValue> eav = new HashMap<String, AttributeValue>();
 eav.put(":v_date", new AttributeValue().withS("2013-11-01"));
 eav.put(":v_issue", new AttributeValue().withS("A-"));
 DynamoDBQueryExpression<CreateDate> queryExpression = new
DynamoDBQueryExpression<CreateDate>()
   .withIndexName("CreateDateIndex").withConsistentRead(false)
   .withKeyConditionExpression("CreateDate = :v_date and
begins_with(IssueId, :v_issue)")
   .withExpressionAttributeValues(eav);

 List<CreateDate> items = mapper.query(CreateDate.class, queryExpression);
 Iterator<CreateDate> iterator = items.iterator();

 System.out.println("Query: printing results...");

 while (iterator.hasNext()) {
  CreateDate iterObj = iterator.next();
  System.out.println(iterObj.getCreateDate());
  System.out.println(iterObj.getIssueId());
 }
}
```

```
}
```

Die folgende Klassendefinition stellt die Tabelle der Probleme dar und wird in der Methode `queryIndexMapper` verwendet.

```
import software.amazon.dynamodb.datamodeling.DynamoDBTable;
import software.amazon.dynamodb.datamodeling.DynamoDBIndexHashKey;
import software.amazon.dynamodb.datamodeling.DynamoDBIndexRangeKey;
import software.amazon.dynamodb.datamodeling.DynamoDBHashKey;

@DynamoDBTable(tableName = "Issues")
public class CreateDate {
 private String createDate;
 @DynamoDBHashKey(attributeName = "IssueId")
 private String issueId;

 @DynamoDBIndexHashKey(globalSecondaryIndexName = "CreateDateIndex", attributeName =
 "CreateDate")
 public String getCreateDate() {
  return createDate;
 }

 public void setCreateDate(String createDate) {
  this.createDate = createDate;
 }

 @DynamoDBIndexRangeKey(globalSecondaryIndexName = "CreateDateIndex", attributeName =
 "IssueId")
 public String getIssueId() {
  return issueId;
 }

 public void setIssueId(String issueId) {
  this.issueId = issueId;
 }
}
```

AWS Beispiele für SDK for Go 1.x

Dieser Abschnitt enthält Beispielcode für DAX-Anwendungen, die Go 1.x verwenden.

Themen

- DAX SDK für Go

DAX SDK für Go

Gehen Sie wie folgt vor, um die Amazon-DynamoDB-Accelerator-(DAX)-SDK für Go auf Ihrer Amazon-EC2-Instance auszuführen.

So führen Sie das SDK-für-Go-Beispiel für DAX aus

1. Richten Sie das SDK für Go auf Ihrer Amazon-EC2-Instance ein:

   a. Installieren Sie die Go-Programmiersprache (`Golang`).

   ```
   sudo yum install -y golang
   ```

   b. Testen Sie, ob Golang installiert ist und ordnungsgemäß ausgeführt wird.

   ```
   go version
   ```

   Eine Nachricht wie diese sollte erscheinen.

   ```
   go version go1.15.5 linux/amd64
   ```

   Die restlichen Anweisungen beruhen auf der Modulunterstützung, die mit Go Version 1.13 zum Standard wurde.

2. Installieren Sie die Golang-Beispielanwendung.

   ```
   go get github.com/aws-samples/aws-dax-go-sample
   ```

3. Führen Sie die folgenden Golang-Programme aus. Das erste Programm erstellt eine DynamoDB-Tabelle mit dem Namen `TryDaxGoTable`. Das zweite Programm schreibt Daten in die Tabelle.

   ```
   go run ~/go/pkg/mod/github.com/aws-samples/aws-dax-go-sample@v1.0.2/try_dax.go -
   service dynamodb -command create-table
   ```

   ```
   go run ~/go/pkg/mod/github.com/aws-samples/aws-dax-go-sample@v1.0.2/try_dax.go -
   service dynamodb -command put-item
   ```

4. Führen Sie die folgenden Golang-Programme aus.

```
go run ~/go/pkg/mod/github.com/aws-samples/aws-dax-go-sample@v1.0.2/try_dax.go -
service dynamodb -command get-item
```

```
go run ~/go/pkg/mod/github.com/aws-samples/aws-dax-go-sample@v1.0.2/try_dax.go -
service dynamodb -command query
```

```
go run ~/go/pkg/mod/github.com/aws-samples/aws-dax-go-sample@v1.0.2/try_dax.go -
service dynamodb -command scan
```

Beachten Sie die Zeitinformationen – die Anzahl der benötigten Millisekunden für den `GetItem`-, `Query`- und `Scan`-Test.

5. Im vorherigen Schritt haben Sie die Programme für den DynamoDB-Endpunkt ausgeführt. Führen Sie die Programme jetzt erneut aus. Dieses Mal werden die `GetItem`-, `Query`- und `Scan`-Operationen aber vom DAX-Cluster verarbeitet.

   Um den Endpunkt für Ihren DAX-Cluster zu bestimmen, wählen Sie einen der folgenden Schritte aus:

   - Using the DynamoDB console (Verwenden der DynamoDB-Konsole) — Wählen Sie Ihren DAX-Cluster aus. Der Cluster-Endpunkt wird auf der Konsole angezeigt, wie im folgenden Beispiel gezeigt.

     ```
     dax://my-cluster.l6fzcv.dax-clusters.us-east-1.amazonaws.com
     ```

   - Verwenden von AWS CLI— Geben Sie den folgenden Befehl ein.

     ```
     aws dax describe-clusters --query "Clusters[*].ClusterDiscoveryEndpoint"
     ```

     Der Cluster-Endpunkt wird wie im folgenden Beispiel in der Ausgabe angezeigt.

     ```
     {
         "Address": "my-cluster.l6fzcv.dax-clusters.us-east-1.amazonaws.com",
         "Port": 8111,
         "URL": "dax://my-cluster.l6fzcv.dax-clusters.us-east-1.amazonaws.com"
     }
     ```

Führen Sie jetzt die Programme erneut aus. Geben Sie dieses Mal jedoch den Cluster-Endpunkt als Befehlszeilenparameter an.

```
go run ~/go/pkg/mod/github.com/aws-samples/aws-dax-go-sample@v1.0.2/try_dax.go
 -service dax -command get-item -endpoint my-cluster.l6fzcv.dax-clusters.us-
east-1.amazonaws.com:8111
```

```
go run ~/go/pkg/mod/github.com/aws-samples/aws-dax-go-sample@v1.0.2/try_dax.go
 -service dax -command query -endpoint my-cluster.l6fzcv.dax-clusters.us-
east-1.amazonaws.com:8111
```

```
go run ~/go/pkg/mod/github.com/aws-samples/aws-dax-go-sample@v1.0.2/try_dax.go
 -service dax -command scan -endpoint my-cluster.l6fzcv.dax-clusters.us-
east-1.amazonaws.com:8111
```

Sehen Sie sich den Rest der Ausgabe an und notieren Sie die Zeitinformationen. Die verstrichene Zeit sollte für `GetItem`, `Query` und `Scan` mit DAX deutlich kürzer sein als mit DynamoDB.

6. Führen Sie das folgende Golang-Programm aus, um `TryDaxGoTable` zu löschen:

```
go run ~/go/pkg/mod/github.com/aws-samples/aws-dax-go-sample@v1.0.2/try_dax.go -
service dynamodb -command delete-table
```

AWS Beispiele für SDK für Node.js 2.x

Dieser Abschnitt enthält Beispielcode für DAX-Anwendungen, die das AWS SDK für Node.js 2.x verwenden.

Themen

- Node.js und DAX

Node.js und DAX

Um die Node.js-Beispielanwendung auf Ihrer Amazon-EC2-Instance auszuführen, gehen Sie wie folgt vor.

So führen Sie das Node.js-Beispiel für DAX aus

1. Richten Sie Node.js auf Ihrer Amazon-EC2-Instance wie folgt ein:

   a. Installieren Sie den Node Version Manager (`nvm`).

   ```
   curl -o- https://raw.githubusercontent.com/nvm-sh/nvm/v0.35.3/install.sh | bash
   ```

   b. Installieren Sie Node.js mit dem nvm.

   ```
   nvm install 12.16.3
   ```

   c. Testen Sie, ob Node.js installiert ist und ordnungsgemäß ausgeführt wird.

   ```
   node -e "console.log('Running Node.js ' + process.version)"
   ```

   Die folgende Meldung sollte angezeigt werden.

   ```
   Running Node.js v12.16.3
   ```

2. Installieren Sie den DAX-Node.js-Client unter Verwendung des Knotenpaketmanagers (`npm`).

   ```
   npm install amazon-dax-client
   ```

3. Downloaden Sie den Quellcode des Beispielprogramms (`.zip`-Datei):

   ```
   wget http://docs.aws.amazon.com/amazondynamodb/latest/developerguide/samples/
   TryDax.zip
   ```

   Wenn der Download abgeschlossen ist, extrahieren Sie die Quelldateien.

   ```
   unzip TryDax.zip
   ```

4. Führen Sie die folgenden Node.js-Programme aus. Das erste Programm erstellt eine Amazon-DynamoDB-Tabelle mit dem Namen `TryDaxTable`. Das zweite Programm schreibt Daten in die Tabelle.

   ```
   node 01-create-table.js
   node 02-write-data.js
   ```

5. Führen Sie die folgenden Node.js-Programme aus

```
node 03-getitem-test.js
node 04-query-test.js
node 05-scan-test.js
```

Beachten Sie die Zeitinformationen – die Anzahl der benötigten Millisekunden für den `GetItem`-, `Query`- und `Scan`-Test.

6. Im vorherigen Schritt haben Sie die Programme für den DynamoDB-Endpunkt ausgeführt. Führen Sie die Programme jetzt erneut aus. Dieses Mal werden die `GetItem`-, `Query`- und `Scan`-Operationen aber vom DAX-Cluster verarbeitet.

   Um den Endpunkt für Ihren DAX-Cluster zu bestimmen, wählen Sie einen der folgenden Schritte aus.

   - Verwenden der DynamoDB-Konsole – Wählen Sie Ihren DAX-Cluster aus. Der Cluster-Endpunkt wird auf der Konsole angezeigt, wie im folgenden Beispiel gezeigt.

     ```
     dax://my-cluster.l6fzcv.dax-clusters.us-east-1.amazonaws.com
     ```

   - Verwenden Sie AWS CLI — Geben Sie den folgenden Befehl ein.

     ```
     aws dax describe-clusters --query "Clusters[*].ClusterDiscoveryEndpoint"
     ```

     Der Cluster-Endpunkt wird wie im folgenden Beispiel in der Ausgabe angezeigt.

     ```
     {
         "Address": "my-cluster.l6fzcv.dax-clusters.us-east-1.amazonaws.com",
         "Port": 8111,
         "URL": "dax://my-cluster.l6fzcv.dax-clusters.us-east-1.amazonaws.com"
     }
     ```

   Führen Sie jetzt die Programme erneut aus. Geben Sie dieses Mal jedoch den Cluster-Endpunkt als Befehlszeilenparameter an.

   ```
   node 03-getitem-test.js dax://my-cluster.l6fzcv.dax-clusters.us-
   east-1.amazonaws.com
   node 04-query-test.js dax://my-cluster.l6fzcv.dax-clusters.us-east-1.amazonaws.com
   node 05-scan-test.js dax://my-cluster.l6fzcv.dax-clusters.us-east-1.amazonaws.com
   ```

Sehen Sie sich den Rest der Ausgabe an und notieren Sie die Zeitinformationen. Die verstrichene Zeit sollte für `GetItem`, `Query` und `Scan` mit DAX deutlich kürzer sein als mit DynamoDB.

7. Führen Sie das folgende Node.js-Programm aus, um `TryDaxTable` zu löschen.

```
node 06-delete-table
```

Weitere Informationen zu diesen Programmen finden Sie in folgenden Abschnitten:

- 01-create-table.js
- 02-write-data.js
- 03-getitem-test.js
- 04-query-test.js
- 05-scan-test.js
- 06-delete-table.js

01-create-table.js

Das Programm `01-create-table.js` erstellt eine Tabelle (`TryDaxTable`). Die restlichen Node.js-Programme in diesem Abschnitt hängen von dieser Tabelle ab.

```
const AmazonDaxClient = require("amazon-dax-client");
var AWS = require("aws-sdk");

var region = "us-west-2";

AWS.config.update({
  region: region,
});

var dynamodb = new AWS.DynamoDB(); //low-level client

var tableName = "TryDaxTable";

var params = {
  TableName: tableName,
  KeySchema: [
```

```
    { AttributeName: "pk", KeyType: "HASH" }, //Partition key
    { AttributeName: "sk", KeyType: "RANGE" }, //Sort key
  ],
  AttributeDefinitions: [
    { AttributeName: "pk", AttributeType: "N" },
    { AttributeName: "sk", AttributeType: "N" },
  ],
  ProvisionedThroughput: {
    ReadCapacityUnits: 10,
    WriteCapacityUnits: 10,
  },
};

dynamodb.createTable(params, function (err, data) {
  if (err) {
    console.error(
      "Unable to create table. Error JSON:",
      JSON.stringify(err, null, 2)
    );
  } else {
    console.log(
      "Created table. Table description JSON:",
      JSON.stringify(data, null, 2)
    );
  }
});
```

02-write-data.js

Das Programm `02-write-data.js` schreibt Testdaten in `TryDaxTable`.

```
const AmazonDaxClient = require("amazon-dax-client");
var AWS = require("aws-sdk");

var region = "us-west-2";

AWS.config.update({
  region: region,
});

var ddbClient = new AWS.DynamoDB.DocumentClient();
```

```
var tableName = "TryDaxTable";

var someData = "X".repeat(1000);
var pkmax = 10;
var skmax = 10;

for (var ipk = 1; ipk <= pkmax; ipk++) {
  for (var isk = 1; isk <= skmax; isk++) {
    var params = {
      TableName: tableName,
      Item: {
        pk: ipk,
        sk: isk,
        someData: someData,
      },
    };

    //
    //put item

    ddbClient.put(params, function (err, data) {
      if (err) {
        console.error("Unable to write data: ", JSON.stringify(err, null, 2));
      } else {
        console.log("PutItem succeeded");
      }
    });
  }
}
```

03-getitem-test.js

Das Programm `03-getitem-test.js` führt `GetItem`-Operationen für `TryDaxTable` aus.

```
const AmazonDaxClient = require("amazon-dax-client");
var AWS = require("aws-sdk");

var region = "us-west-2";

AWS.config.update({
  region: region,
});
```

```
var ddbClient = new AWS.DynamoDB.DocumentClient();
var daxClient = null;

if (process.argv.length > 2) {
  var dax = new AmazonDaxClient({
    endpoints: [process.argv[2]],
    region: region,
  });
  daxClient = new AWS.DynamoDB.DocumentClient({ service: dax });
}

var client = daxClient != null ? daxClient : ddbClient;
var tableName = "TryDaxTable";

var pk = 1;
var sk = 10;
var iterations = 5;

for (var i = 0; i < iterations; i++) {
  var startTime = new Date().getTime();

  for (var ipk = 1; ipk <= pk; ipk++) {
    for (var isk = 1; isk <= sk; isk++) {
      var params = {
        TableName: tableName,
        Key: {
          pk: ipk,
          sk: isk,
        },
      };

      client.get(params, function (err, data) {
        if (err) {
          console.error(
            "Unable to read item. Error JSON:",
            JSON.stringify(err, null, 2)
          );
        } else {
          // GetItem succeeded
        }
      });
    }
  }
```

```
  var endTime = new Date().getTime();
  console.log(
    "\tTotal time: ",
    endTime - startTime,
    "ms - Avg time: ",
    (endTime - startTime) / iterations,
    "ms"
  );
}
```

04-query-test.js

Das Programm `04-query-test.js` führt `Query`-Operationen für `TryDaxTable` aus.

```
const AmazonDaxClient = require("amazon-dax-client");
var AWS = require("aws-sdk");

var region = "us-west-2";

AWS.config.update({
  region: region,
});

var ddbClient = new AWS.DynamoDB.DocumentClient();
var daxClient = null;

if (process.argv.length > 2) {
  var dax = new AmazonDaxClient({
    endpoints: [process.argv[2]],
    region: region,
  });
  daxClient = new AWS.DynamoDB.DocumentClient({ service: dax });
}

var client = daxClient != null ? daxClient : ddbClient;
var tableName = "TryDaxTable";

var pk = 5;
var sk1 = 2;
var sk2 = 9;
var iterations = 5;
```

```
var params = {
  TableName: tableName,
  KeyConditionExpression: "pk = :pkval and sk between :skval1 and :skval2",
  ExpressionAttributeValues: {
    ":pkval": pk,
    ":skval1": sk1,
    ":skval2": sk2,
  },
};

for (var i = 0; i < iterations; i++) {
  var startTime = new Date().getTime();

  client.query(params, function (err, data) {
    if (err) {
      console.error(
        "Unable to read item. Error JSON:",
        JSON.stringify(err, null, 2)
      );
    } else {
      // Query succeeded
    }
  });

  var endTime = new Date().getTime();
  console.log(
    "\tTotal time: ",
    endTime - startTime,
    "ms - Avg time: ",
    (endTime - startTime) / iterations,
    "ms"
  );
}
```

05-scan-test.js

Das Programm `05-scan-test.js` führt `Scan`-Operationen für `TryDaxTable` aus.

```
const AmazonDaxClient = require("amazon-dax-client");
var AWS = require("aws-sdk");
```

```
var region = "us-west-2";

AWS.config.update({
  region: region,
});

var ddbClient = new AWS.DynamoDB.DocumentClient();
var daxClient = null;

if (process.argv.length > 2) {
  var dax = new AmazonDaxClient({
    endpoints: [process.argv[2]],
    region: region,
  });
  daxClient = new AWS.DynamoDB.DocumentClient({ service: dax });
}

var client = daxClient != null ? daxClient : ddbClient;
var tableName = "TryDaxTable";

var iterations = 5;

var params = {
  TableName: tableName,
};
var startTime = new Date().getTime();
for (var i = 0; i < iterations; i++) {
  client.scan(params, function (err, data) {
    if (err) {
      console.error(
        "Unable to read item. Error JSON:",
        JSON.stringify(err, null, 2)
      );
    } else {
      // Scan succeeded
    }
  });
}

var endTime = new Date().getTime();
console.log(
  "\tTotal time: ",
  endTime - startTime,
  "ms - Avg time: ",
```

```
  (endTime - startTime) / iterations,
  "ms"
);
```

06-delete-table.js

Das `06-delete-table.js`-Programm löscht `TryDaxTable`. Führen Sie dieses Programm aus, sobald Sie mit dem Testen fertig sind.

```
const AmazonDaxClient = require("amazon-dax-client");
var AWS = require("aws-sdk");

var region = "us-west-2";

AWS.config.update({
  region: region,
});

var dynamodb = new AWS.DynamoDB(); //low-level client

var tableName = "TryDaxTable";

var params = {
  TableName: tableName,
};

dynamodb.deleteTable(params, function (err, data) {
  if (err) {
    console.error(
      "Unable to delete table. Error JSON:",
      JSON.stringify(err, null, 2)
    );
  } else {
    console.log(
      "Deleted table. Table description JSON:",
      JSON.stringify(data, null, 2)
    );
  }
});
```

Dokumentverlauf für DynamoDB

In der folgenden Tabelle sind die wichtigsten Änderungen in jeder Version des DynamoDB-Entwicklerhandbuchs ab dem 3. Juli 2018 beschrieben. Für Benachrichtigungen über Änderungen an dieser Dokumentation können Sie den RSS-Feed abonnieren (in der oberen linken Ecke auf dieser Seite).

Änderung	Beschreibung	Datum
DAX unterstützt jetzt AWS PrivateLink in kommerziellen und China Regionen	Es wurde eine Anleitung zur Verwendung AWS PrivateLink mit DynamoDB Accelerator (DAX) hinzugefügt, um Benutzern zu helfen, private, sichere Verbindungen zu den Endpunkten der DAX-Steuerebene innerhalb derselben AWS-Region herzustellen. Weitere Informationen finden Sie unter Generieren von Infrastrukturcode für Amazon DynamoDB mit. Console-to-Code	30. Oktober 2025
DynamoDB unterstützt jetzt Internet Protocol Version 6 () IPv6	DynamoDB unterstützt jetzt Internetprotokolladressen der Version 6 (IPv6) in Ihrer Amazon VPC, wenn Sie eine Verbindung zu DynamoDB-Tabellen, Streams und DAX herstellen, auch mit AWS PrivateLink Gateway- und Interface-Endpunkten. Sie können IPv6 damit Ihren Netzwerk-Stack vereinfachen und Compliance-Anforde	3. Oktober 2025

	rungen mit einem Netzwerk erfüllen, das sowohl als auch unterstützt. IPv4 IPv6	
Dokumentation für das neue Console-to-Code Feature für DynamoDB hinzugefügt	Console-to-Code wandelt manuelle Schritte zur DynamoDB-Tabellene rstellung in der AWS-Manag ementkonsole in automatis ierten, reproduzierbaren Infrastrukturcode in mehreren Formaten wie CDK und um. AWS CloudFormation Weitere Informationen finden Sie unter Generieren von Infrastrukturcode für Amazon DynamoDB mit. Console-to-Code	23. September 2025
Lokale Version 3.1.0 von DynamoDB veröffentlicht	Durch dieses Update wird die Leistung von PartiQL-Abfragen verbessert, einschließlich der Joda-Zeitabhängigkeit.	14. September 2025

DynamoDB hat seine Funktionen zur Drosselungsdiagnose um einen neuen CloudWatch Contributor Insights-Modus erweitert, Details zu Drosselungsausnahmen und gezielte Metriken hinzugefügt CloudWatch	DynamoDB bietet erweiterte Drosselungsdiagnosen mit einem neuen Modus `Throttled keys nur` in CloudWatch Contributor Insights, der speziell die am stärksten gedrosselten Partitionsschlüssel verfolgt, detailliertere Drosselungsausnahmen, die strukturierte `ThrottlingReason` Felder enthalten, spezifische Ursachencodes und Ressourcen bereitstellt, mit denen genau identifiziert werden kann, welche Ressource ARNs gedrosselt wird und warum, sowie 16 neue gezielte CloudWatch Metriken, die verschiedenen Drosselungsszenarien entsprechen, für eine präzise Überwachung bestimmter Drosselungsereignisse. Weitere Informationen finden Sie unter Beheben von Drosselungsereignissen in Amazon DynamoDB.	14. August 2025

Das Limit für das Umschalten zwischen Kapazitätsmodi wurde erhöht	Das Limit für den Wechsel einer Tabelle aus dem Bereitstellungs- in den On-Demand-Kapazitätsmodus wurde von einmal auf viermal je 24 Stunden erhöht. Diese Änderung bietet mehr Flexibilität für Workloads, bei denen große Datenmengen mehrmals täglich geladen werden oder die eine unerwartete Zunahme des Datenverkehrs bewältigen müssen. Weitere Informationen finden Sie unter Überlegungen beim Umstellen der Kapazitätsmodi in DynamoDB.	12. August 2025
DynamoDB Local Version 3.0 veröffentlicht, die vom AWS SDK for Java V1 auf V2 migriert	Dieses Update entfernt die V1-Abhängigkeit, verbessert die Leistung und entspricht der AWS SDK V2-Paketstruktur.	17. Juli 2025

Unterstützung für optimierte Shard-Discovery in DynamoDB Streams hinzugefügt	Der neue Shard-Erkennungsmechanismus reduziert die anfängliche Erkennungszeit, verbessert die Steady-State-Verarbeitungsleistung, bietet Einblicke in den Verarbeitungsstatus und sorgt für eine konsistente p100-Latenz bei umfangreichen Vorgängen. Weitere Informationen finden Sie unter AWS PrivateLink für Amazon DynamoDB Streams.	17. Juli 2025
Unterstützung für die Multi-Region Strong Consistency (MRSC) für globale DynamoDB-Tabellen hinzugefügt	DynamoDB unterstützt jetzt die Multi-Region Strong Consistency (MRSC) für globale DynamoDB-Tabellen mit optionaler Witness-Region. MRSC ermöglicht strikt konsistente Lesevorgänge in allen Regionen mit einem Recovery Point Objective (RPO) von Null bei regionalen Ausfällen. Globale MRSC-Tabellen müssen in genau drei Regionen bereitgestellt werden, entweder mit drei Replikaten oder einem Replikat und einem Witness. Weitere Informationen und verfügbare Regionen finden Sie unter Multi-Region Strong Consistency (MRSC).	30. Juni 2025

Adapter für Amazon DynamoDB Streams Kinesis Client Library (KCL) 2.x	DynamoDB unterstützt jetzt einen aktualisierten DynamoDB-Streams-Adapter, der mit KCL 2.x und AWS -SDK v2 kompatibel ist. Weitere Informationen finden Sie unter Migration von X. zu KCL1 KCL 3.x.	11. Juni 2025
Unterstützung für die lokale Ausführung von DynamoDB hinzugefügt in AWS CloudShell	Sie können DynamoDB jetzt lokal direkt in Ihrem Browser über die AWS-Management konsole ausführen, ohne Software herunterladen oder installieren zu müssen. Weitere Informationen finden Sie unter Bereitstellen von DynamoDB lokal auf Ihrem Computer.	19. Mai 2025
Unterstützung für r7i-Instance-Typen in DAX hinzugefügt	Die Optionen für Instance-Typen in DAX wurden erweitert und bieten nun mehr Flexibilität in Bezug auf Leistung und Kapazität. Weitere Informationen zu den neuen r7i-Instance-Typen und ihren Funktionen finden Sie unter DAX-Verschlüsselung im Ruhezustand.	29. April 2025

Die neue Richtlinie AmazonDynamoDBFullAccess_v2 mit eingeschränktem Geltungsbereich für vollen Zugriff ersetzt die Richtlinie AmazonDynamoDBFullAccess.	Mit Wirkung vom 25. April 2025 hat DynamoDB die Richtlinie AmazonDynamoDBFullAccess als veraltet eingestuft und durch eine Richtlinie mit eingeschränktem Geltungsbereich mit dem Namen AmazonDynamoDBFullAccess_v2 ersetzt. Weitere Informationen finden Sie unter AWS verwaltete Richtlinie: AmazonDynamoDBFull Access_v2.	28. April 2025
DynamoDB führt AWS PrivateLink Unterstützung für DynamoDB Streams ein	AWS AWS PrivateLink Unterstützung für DynamoDB Streams hinzugefügt, sodass Kunden DynamoDB Streams über private Netzwerkkonnektivität innerhalb APIs von Amazon Virtual Private Cloud (VPC) aufrufen können. Dieses Feature ermöglicht einen vereinfachten Zugriff über private Netzwerke, ohne das öffentliche Internet zu durchlaufen, und unterstützt so die Compliance- und Sicherheitsanforderungen für DynamoDB-Workloads. Weitere Informationen finden Sie unter AWS PrivateLink für Amazon DynamoDB Streams.	26. März 2025

DAX-SDK für JS 3.x und Go 2.x jetzt verfügbar	Das DynamoDB Accelerator (DAX) SDK für JS 3.x ist jetzt verfügbar und mit dem AWS SDK for Java 3.x kompatibel.	17. März 2025
On-Demand-Kapazitätsmodus in NoSQL Workbench 3.13.5 für Standard-Tabellene instellungen veröffentlicht	Wenn Sie eine Tabelle mit Standardeinstellungen erstellen, erstellt DynamoDB eine Tabelle, die anstelle des Modus bereitgestellter Kapazität den Modus „On-Demand-Kapazität“ verwendet .	24. Februar 2025
Neue bewährte Methode für die Konfiguration von DAX-Clients	Es wurde ein neues Thema mit bewährten DAX-Anlei tungen für die Konfiguration von DAX-Clients veröffentlicht. Weitere Informationen finden Sie unter Konfigurieren Ihres DAX-Clients.	17. Februar 2025
Neue bewährte Methoden für Massendatenoperationen	Es wurde ein neues Thema mit bewährten Methoden für die Arbeit mit komplexen Datenmodellen in DynamoDB veröffentlicht. Weitere Informationen finden Sie unter Bewährte Methoden für die Verwendung von Massendat enoperationen in DynamoDB.	9. Januar 2025

Amazon DynamoDB unterstützt jetzt konfigurierbare point-in-time Wiederherstellung (PITR)	DynamoDB unterstützt jetzt konfigurierbare Wiederherstellungszeiträume für PITR. Der PITR-Zeitraum kann jetzt für jede Tabelle auf einen Wert zwischen einem und 35 Tagen festgelegt werden. Weitere Informationen finden Sie unter Point-in-time Backups für DynamoDB.	7. Januar 2025
Neues Thema zur Integration von Amazon Managed Streaming für Apache Kafka in Amazon DynamoDB veröffentlicht	Erfahren Sie, wie Amazon Managed Streaming für Apache Kafka in Amazon DynamoDB integriert wird, indem Daten aus Apache Kafka-Themen gelesen und in DynamoDB gespeichert werden. Weitere Informationen finden Sie unter Integrieren von DynamoDB in Amazon Managed Streaming für Apache Kafka.	26. Dezember 2024
DynamoDB führt Unterstützung für eine Zero-ETL-Integration mit AI Lakehouse ein SageMaker	DynamoDB führt eine Null-ETL-Integration ein, die das Extrahieren und Laden von Daten aus Amazon DynamoDB in den Data Lake eines Kunden automatisiert. Weitere Informationen finden Sie unter DynamoDB Zero-ETL-Integration mit AI Lakehouse. SageMaker	03. Dezember 2024

Multi-Region Strong Consistency für globale DynamoDB-Tabellen	Dank der Multi-Region Strong Consistency können hochverfügbare, multiregionale Anwendungen mit einem Recovery Point Objective (RPO) von Null erstellt und so ein Höchstmaß an Ausfallsicherheit erreicht werden. Weitere Informationen finden Sie unter Globale Tabellen mit Multi-Region Strong Consistency.	03. Dezember 2024
Aktualisierung der von DynamoDB verwalteten Richtlinie	Es wurden neue Berechtigungen zur vom `AmazonDynamoDBReadOnlyAccess` verwalteten Richtline hinzugefügt: `dynamodb:GetAbacStatus` und `dynamodb:UpdateAbacStatus`. Mit diesen Berechtigungen können Sie den ABAC-Status anzeigen und ABAC für Sie in der aktuellen Region aktivieren. AWS-Konto Weitere Informationen finden Sie unter AWS Verwaltete Richtlinie:. AmazonDynamo DBRead OnlyAccess	18. November 2024

Unterstützung für attributbasierte Zugriffskontrolle (ABAC) in DynamoDB eingeführt)	ABAC ist eine Autorisierungsstrategie, mit der Sie Zugriffsberechtigungen auf der Grundlage von Tags definieren können, die Benutzern, Rollen und AWS Ressourcen zugeordnet sind. ABAC verwendet tagbasierte Bedingungen in Ihren AWS Identity and Access Management (IAM-) Richtlinien oder anderen Richtlinien, um bestimmte Aktionen an Ihren Tabellen oder Indizes zuzulassen oder zu verweigern, wenn die Tags der IAM-Prinzipale mit den Tags für die Tabellen übereinstimmen. Weitere Informationen finden Sie unter Verwenden der attributbasierten Zugriffskontrolle mit DynamoDB.	18. November 2024
DynamoDB führt Warmdurchsatz für On-Demand-Tabellen und bereitgestellte Tabellen ein	DynamoDB unterstützt jetzt den Warmdurchsatz. Der „Warmdurchsatz“ bietet einen Überblick über die Anzahl der Lese- und Schreibvorgänge, die Ihre DynamoDB-Tabelle sofort unterstützen kann, sowie die Möglichkeit der Vorwärmung von DynamoDB-Tabellen. Weitere Informationen finden Sie unter DynamoDB warm throughput.	13. November 2024

Möglichkeit, Amazon DynamoDB Streams-Datensätze mit Apache Flink zu verwenden	Sie können jetzt Amazon DynamoDB Streams-Datensätze mit Apache Flink nutzen und den Amazon Managed Service für Apache Flink nutzen, um Anwendungen zur Stream-Verarbeitung schnell zu erstellen und zu verwalten end-to-end. Weitere Informationen finden Sie unter DynamoDB-Streams and Apache Fink.	12. November 2024
Neue Themen zur Fakturierung für globale Tabellen und Backups veröffentlicht	Es wurden zwei neue Themen zur Fakturierung für globale Tabellen und Backups veröffentlicht. Weitere Informationen finden Sie unter Grundlegendes zur Amazon DynamoDB-Fakturierung für globale Tabellen und Grundlegendes zur Amazon DynamoDB-Fakturierung für Backups.	16. Oktober 2024
Null-ETL-Integration von DynamoDB in Amazon Redshift	Die Null-ETL-Integration von Amazon DynamoDB in Amazon Redshift bietet eine vollständig verwaltete ETL-Pipeline ohne Code mit Replikation von DynamoDB nach Amazon Redshift. Weitere Informationen finden Sie unter Null-ETL-Integration von Amazon DynamoDB in Amazon Redshift.	15. Oktober 2024

Dokumentation aktualisiert: Thema zur Verwendung generativer KI mit DynamoDB hinzugefügt	Es wurde ein neues Thema mit Informationen zur Verwendung generativer KI mit DynamoDB veröffentlicht, darunter Beispiele für Anwendungsfälle der generativen KI für DynamoDB. Weitere Informationen finden Sie unter Verwenden von generativer KI mit DynamoDB.	11. Oktober 2024
Das SDK unterstützt jetzt kontobasierte Endpunkte AWS	Es wurde Dokumentation für kontenbasierte Endpunkte und die `ACCOUNT_ID_ENDPOINT_MODE` -Einstellung für SDK-Clients hinzugefügt. Weitere Informationen finden Sie unter SDK-Unterstützung für kontobasierte Endpunkte. AWS	3. September 2024
Neugestaltung der ersten Schritte	Die Benutzeroberfläche für die ersten Schritte wurde neu gestaltet; Informationen wurden zusammengefasst, um den Einstieg zu beschleunigen. Weitere Informationen finden Sie unter Erste Schritte mit DynamoDB.	1. August 2024
DAX-Expansion in neue Regionen für Spanien und Schweden	DAX ist jetzt in den Regionen Spanien und Schweden verfügbar. Weitere Informationen finden Sie unter DAX cluster components.	30. Juli 2024

Dokumentation für DynamoDB-Backup und -Wiederherstellung umstrukturiert und konsolidiert	Die Themen für Backup und Wiederherstellung im Entwicklerhandbuch für DynamoDB wurden umstrukturiert. Weitere Informationen finden Sie unter Backup und Wiederherstellung für DynamoDB.	2. Juli 2024
Thema „Was ist Amazon DynamoDB?“ überarbeitet	Es wurde eine überarbeitete und aktualisierte Version des Themas „Was ist Amazon DynamoDB?“ veröffentlicht. Weitere Informationen finden Sie unter Was ist Amazon DynamoDB?.	21. Juni 2024
Integrieren Sie DynamoDB Streams mit EventBridge	Es wurde ein neues Thema zur Integration von DynamoDB Streams mit veröffentlicht. EventBridge Weitere Informationen finden Sie unter Integrieren mit. EventBridge	21. Juni 2024

Vorgeschriebene Anleitung für DAX	Es wurde ein neues Thema mit bewährten Methoden veröffentlicht, das umfassende Einblicke in die effiziente Nutzung von DynamoDB Accelerator bietet. Dieses Thema behandelt die Leistungsoptimierung, das Kostenmanagement und bewährte Methoden für die Ausführung. Weitere Informationen finden Sie unter Vorgeschriebene Anleitung für DAX.	3. Juni 2024
Migrieren einer DynamoDB-Tabelle zwischen zwei Konten	Es wurde ein neues Thema zur Migration von DynamoDB-Tabellen zwischen zwei Konten hinzugefügt. Weitere Informationen finden Sie unter Migrieren einer DynamoDB-Tabelle zwischen zwei Konten.	29. Mai 2024
Dokumentation für DynamoDB-Überwachung- und -Protokollierung umstrukturiert und konsolidiert	Die neue Struktur für die Überwachung und Protokollierung von DynamoDB umfasst drei kurze Kapitel mit Metriken, Protokollierungsvorgängen und Erkenntnissen von Beteiligten.	3. Mai 2024

Dokumentation für DynamoDB-Kapazitätsmodus umstrukturiert und konsolidiert	Der DynamoDB-Leitfaden enthält jetzt ein neues Kapitel mit allen Informationen zu den DynamoDB-Kapazitätsmodi On-Demand und Bereitgestellt. Mit diesem Update wurde das Thema Überlegungen bei der Änderung des read/write Kapazitätsmodus in das Kapitel Bewährte Methoden verschoben. Dieses Thema wurde jetzt in Überlegungen beim Umstellen der Kapazitätsmodi umbenannt und enthält ausführliche Informationen zu den bewährten Methoden beim Umschalten zwischen den Kapazitätsmodi. Darüber hinaus enthält der Leitfaden jetzt ein neues Kapitel mit allen Informationen über Lese- und Schreibvorgänge in DynamoDB sowie dem Verbrauch von Kapazitätseinheiten für Lese- und Schreibvorgänge. Weitere Informationen finden Sie unter DynamoDB throughput capacity, Considerations when switching capacity modes und DynamoDB reads and writes.	1. Mai 2024

Maximale Anzahl von On-Demand-Anfragen	Sie können jetzt die maximale Anzahl von On-Demand-Anfragen angeben, die von einer einzelnen Tabelle, einem Index oder beidem ausgeführt werden kann. Die Angabe des maximalen On-Demand-Durchsatzes begrenzt die Nutzung und die Kosten auf Tabellenebene und schützt vor dem unbeabsichtigten Anstieg verbrauchter Ressourcen. Weitere Informationen finden Sie unter Maximaler Durchsatz für On-Demand-Tabellen.	1. Mai 2024
Verbesserungen von NoSQL Workbench Operation Builder	NoSQL Workbench bietet jetzt native Unterstützung für den dunklen Modus. Die Tabellen- und Elementoperationen in Operation Builder wurden verbessert. Die Ergebnisse zu Elementen und Informationen zu Operation-Builder-Anfragen sind im JSON-Format verfügbar. Weitere Informationen finden Sie unter NoSQL Workbench Operation Builder.	24. April 2024

Ressourcenbasierte Richtlinien für Amazon DynamoDB-Ressourcen	DynamoDB unterstützt jetzt ressourcenbasierte Richtlinien für Tabellen, Indizes und Streams. Mit ressourcenbasierten Richtlinien können Sie Zugriffsberechtigungen definieren, indem Sie angeben, wer Zugriff auf jede Ressource hat und welche Aktionen für jede Ressource ausgeführt werden dürfen. Weitere Informationen finden Sie unter Verwenden von ressourcenbasierten Richtlinien für DynamoDB.	20. März 2024
Aktualisierung der von DynamoDB verwalteten Richtlinie	Es wurde eine neue Berechtigung `dynamodb:GetResourcePolicy` zur verwalteten Richtline `AmazonDynamoDBReadOnlyAccess` hinzugefügt. Diese Berechtigung ermöglicht den Zugriff auf Richtlinien, die auf Leseressourcen basieren und die DynamoDB-Ressourcen zugeordnet sind. Weitere Informationen finden Sie unter AWS Verwaltete Richtlinie: AmazonDynamo DBRead OnlyAccess.	20. März 2024

AWS PrivateLink für Amazon DynamoDB	Amazon DynamoDB unterstützt jetzt. AWS PrivateLink Mit AWS PrivateLink können Sie die private Netzwerkkonnektivität zwischen Virtual Private Clouds (VPCs), DynamoDB und Ihren lokalen Rechenzentren mithilfe von VPC-Schnittstellen-Endpunkten und privaten IP-Adressen vereinfachen. Weitere Informationen finden Sie unter AWS PrivateLink für DynamoDB.	19. März 2024
Programmierung mit JavaScript Anleitung	Amazon DynamoDB präsentiert einen Programmierleitfaden für. AWS SDK für JavaScript Erfahren Sie mehr über die AWS SDK für JavaScript, Abstraktionsebenen, die Konfiguration der Verbindung, den Umgang mit Fehlern, die Definition von Wiederholungsrichtlinien, die Verwaltung von Keep-Alive und mehr. Weitere Informationen finden Sie unter Programmieren mit. JavaScript	6. März 2024

Programmieren mit AWS SDK for Java 2.x Anleitung	Es wurde ein neues Programmierhandbuch erstellt, das sich eingehend mit High-Level-, Low-Level - und Dokumentenschnitts tellen, HTTP-Clients und deren Konfiguration sowie der Fehlerbehandlung befasst und in dem die häufigsten Konfigurationseinstellungen aufgeführt werden, die bei der Verwendung des SDK für Java 2.x berücksichtig werden sollten. Weitere Informationen finden Sie unter Programmi eren von Amazon DynamoDB mit. AWS SDK for Java 2.x	5. März 2024
Klonen von Tabellen mit NoSQL Workbench	Ermöglicht Entwicklern, NoSQL Workbench zum Kopieren oder Klonen von Tabellen zwischen Entwicklungsumgebungen und Regionen (lokale und Web-Version von DynamoDB) zu verwenden. Weitere Informati onen finden Sie unter Klonen von Tabellen mit NoSQL Workbench.	26. Februar 2024

Handbuch für die Programmierung mit Python	Es wurde ein neues Handbuch erstellt, das sich eingehend mit High- und Low-Level-Bibliotheken und die gängigsten Konfigurationseinstellungen behandelt, die bei der Verwendung des Python-SDK berücksichtigt werden sollten. Weitere Informationen finden Sie unter Programmierung mit Python.	5. Januar 2024
Überarbeitung des Themas „Time to Live (TTL)“	Der Abschnitt „TTL“ im Handbuch wurde vollständig überarbeitet. Das neue Handbuch hilft Ihnen bei den ersten Schritten mit TTL, indem es Ihnen ready-to-use Codefragmente zur Verfügung stellt. Die aktuell bereitgestellten Codefragmente basieren auf Python und Javascript. Weitere Informationen finden Sie unter TTL.	20. Dezember 2023
Bewährte Methoden zum Verständnis Ihrer AWS Abrechnungs- und Nutzungsberichte	Es wurde ein neuer Abschnitt hinzugefügt, in dem die verschiedenen Nutzungsarten und die Gebühren für diese Nutzungsarten in DynamoDB erläutert werden. Weitere Informationen finden Sie unter Fakturierungs- und Nutzungsberichte.	15. Dezember 2023

Amazon DynamoDB Zero-ETL-Integration mit Amazon Service OpenSearch	Amazon DynamoDB unterstützt jetzt die Zero-ETL-Integration mit Amazon OpenSearch Service, sodass Sie Ihre DynamoDB-Daten durchsuchen können, indem Sie sie automatisch replizieren und transformieren, ohne benutzerdefinierten Code oder Infrastruktur. Weitere Informationen finden Sie unter DynamoDB Zero-ETL-Integration mit Amazon Service. OpenSearch	28. November 2023
Migrieren aus einer relationalen Datenbank zu DynamoDB	Es wurde ein Migrationsleitfaden erstellt, der Benutzer bei der Migration von einer relationalen Datenbank zu DynamoDB unterstützt.	27. November 2023
Generieren von Beispieldaten mit NoSQL Workbench	NoSQL Workbench für Amazon DynamoDB unterstützt jetzt die Erstellung von Datenmodellen direkt aus Beispieldatenmodellvorlagen, um Sie beim Entwerfen von Datenschemata für Ihre Workloads zu unterstützen. Sie können diese Funktion verwenden, um sich mit den bewährten Verfahren für die NoSQL-Datenmodellierung vertraut zu machen, wenn Sie Ihre Anwendungen auf DynamoDB erstellen.	28. September 2023

Inkrementeller Export nach S3	Sie können jetzt Daten, die eingefügt, aktualisiert oder gelöscht wurden, in kleinen Schritten exportier en. Mit dem inkrementellen Export können Sie geänderte Daten im Bereich von einigen Megabyte bis hin zu Terabyte mit wenigen Klicks in der AWS Management Console, einem API-Aufruf oder der Befehlsze ilenschnittstelle exportieren. AWS	26. September 2023
Datenmodellierung für DynamoDB	Sie können jetzt mehr über Datenmodellierung anhand von DynamoDB-Beispiele n erfahren, die sich auf bestimmte Anwendungsfälle und deren Zugriffsmuster konzentrieren und step-by-s tep Anleitungen zur Realisier ung dieser Zugriffsmuster enthalten.	14. Juli 2023
Abschnitt Fehlerbehebung	Sie finden jetzt Inhalte zur Fehlerbehebung für Latenz- und Drosselungsprobleme, die in DynamoDB-Tabellen auftreten können.	13. März 2023

Löschschutz für Amazon DynamoDB	Der Löschschutz ist jetzt für Amazon-DynamoDB-Tabellen in allen AWS -Regionen verfügbar. DynamoDB ermöglicht jetzt den Schutz von Tabellen vor versehentlichem Löschen während regulärer Tabellenverwaltungsoperationen.	08. März 2023
CloudFormation Unterstützung für KDSD in globalen Tabellen	Amazon Kinesis Data Streams für DynamoDB unterstützt CloudFormation jetzt globale DynamoDB-Tabellen, was bedeutet, dass Sie das Streaming zu Amazon Kinesis Data Streams auf Ihren globalen DynamoDB-Tabellen mit Vorlagen aktivieren können. CloudFormation	15. Februar 2023
Die lokale Version von DynamoDB unterstützt 100 Aktionen pro Transaktion.	Sie können jetzt bis zu 100 Aktionen in einer einzigen Transaktion in der lokalen Version von DynamoDB ausführen.	9. Februar 2023
Verwendung von DynamoDB Well-Architected Lens zur Optimierung Ihres DynamoDB-Workloads	Sie können jetzt DynamoDB Well-Architected Lens verwenden, eine Sammlung von Gestaltungsprinzipien und Leitlinien für die Gestaltung gut strukturierter DynamoDB-Workloads.	3. Februar 2023

PartiQL Verfügbarkeit GovGloud	PartiQL — Eine SQL-kompatible Abfragesprache für Amazon DynamoDB wird jetzt in AWS GovCloud (US-Ost) und (US-West) unterstützt. AWS GovCloud	21. Dezember 2022
Einzelinstallationssuite für NoSQL Workbench und die lokale Version von DynamoDB	NoSQL Workbench für DynamoDB enthält jetzt Installationsanleitungen für die lokale Version von DynamoDB, um die Einrichtung Ihrer DynamoDB-local-Entwicklungsumgebung zu optimieren.	6. Dezember 2022
Massenimport aus S3	Amazon DynamoDB erleichtert jetzt die Migration und das Laden von Daten in neue DynamoDB-Tabellen, indem Massendatenimporte aus Amazon S3 unterstützt werden.	18. August 2022
Verbesserte Integration mit Service Quotas	-Servicekontingente ermöglichen Ihnen jetzt, Ihre Konto- und Tabellenkontingente proaktiv zu verwalten. Sie können aktuelle Werte anzeigen, Alarme festlegen, wenn Ihre Nutzung eines Kontingents einen konfigurierbaren Schwellenwert überschreitet, und vieles mehr.	15. Juni 2022

NoSQL Workbench fügt Tabellen- und GSI-Unterstützung hinzu	Sie können NoSQL Workbench jetzt für Operationen auf der Steuerungsebene von Tabellen und Global Secondary Index (GSI) wie CreateTable, UpdateTable und verwenden. DeleteTable	2. Juni 2022
Standard-Infrequent-Access-Tabellenklasse jetzt in China verfügbar	Die Standard-Infrequent-Access-Tabellenklasse von Amazon DynamoDB ist jetzt in den chinesischen Regionen verfügbar. Senken Sie Ihre DynamoDB-Kosten um bis zu 60 Prozent, indem Sie diese neue Tabellenklasse für Tabellen verwenden, die selten aufgerufene Daten speichern.	18. April 2022
Erhöhung der Standardservicekontingente sowie der Zahl der Tabellenverwaltungsvorgänge	In DynamoDB wurde das Standardkontingent für die Tabellenzahl pro Konto und Region von 256 auf 2 500 Tabellen erhöht. Zudem wurde die Zahl der gleichzeitigen Tabellenverwaltungsvorgänge von 50 auf 500 erhöht.	9. März 2022
Optionale Begrenzung von Elementen mit PartiQL für DynamoDB	DynamoDB kann die Anzahl der in PartiQL verarbeiteten Elemente für DynamoDB-Vorgänge über einen optionalen Parameter für jede Anforderung begrenzen.	8. März 2022

AWS Backup Integration in den Regionen China (Peking und Ningxia) verfügbar	AWS Backup ist jetzt in den Regionen China (Peking und Ningxia) in DynamoDB integriert. Durch erweitert e Backup-Funktionen wie konto- und regionsübergreifen de Backups können Sie die Anforderungen an AWS Backup Compliance und Geschäftskontinuität leichter erfüllen.	26. Januar 2022
Informationen zur Durchsatz kapazität über PartiQL-API-Aufrufe	DynamoDB kann die von PartiQL-API-Aufrufen die verbrauchte Durchsatz kapazität zurückgeben, um Sie bei der Optimierung Ihrer Abfragen und Durchsatzkosten zu unterstützen.	18. Januar 2022
AWS Backup Integration	Mit den erweiterten Backup-Funktionen in AWS Backup wie z. B. konto- und regionsüb ergreifenden Backups macht es DynamoDB jetzt leichter möglich, die Anforderungen in Bezug auf Compliance und Business Continuity zu erfüllen.	24. November 2021
NoSQL import/export Workbench-Datensätze im CSV-Format	Mit NoSQL Workbench für Amazon DynamoDB können Sie jetzt Beispieldaten importieren und automatis ch auffüllen, um leichter Ihre Datenmodelle zu erstellen und zu visualisieren.	11. Oktober 2021

Filtern und Abrufen von Amazon DynamoDB Streams-Datenebenenaktivitäten mit AWS CloudTrail	Amazon DynamoDB bietet Ihnen jetzt eine genauere Kontrolle der Prüfungs-Protokollierung, indem Sie die API-Aktivität auf der Datenebene von Streams in AWS CloudTrail filtern können.	22. September 2021
Aktualisierte Konsole	Die DynamoDB-Konsole ist jetzt Ihre Standardk onsole, über die Sie Daten leichter verwalten, allgemein e Aufgaben vereinfachen und schneller auf Ressourcen und Funktionen zugreifen können.	25. August 2021
DAX-SDK für Java 2.x jetzt verfügbar	Das DynamoDB Accelerat or (DAX) SDK for Java 2.x ist jetzt verfügbar und mit dem AWS SDK for Java 2.x kompatibel. Sie können die neuesten Funktionen nutzen, darunter auch nicht blockiere nde I/O.	29. Juli 2021
Aktualisierung von NoSQL-Workbench-Funktionen einschließlich Operationen auf Steuerebene	NoSQL Workbench für Amazon DynamoDB erleichte rt Ihnen jetzt das Ausführen häufiger Operationen, um auf Tabellendaten zuzugreifen und sie zu verändern.	28. Juli 2021

Globale DynamoDB-Tabellen jetzt in der Region Asien-Pazifik verfügbar	Globale DynamoDB-Tabellen sind jetzt in der Region Asien-Pazifik (Osaka) verfügbar. Ihre DynamoDB-Tabellen können automatisch in 22 AWS -Regionen Ihrer Wahl repliziert werden.	28. Juli 2021
DAX ist jetzt in China verfügbar	DynamoDB Accelerator (DAX) ist jetzt in der Region China (Peking) verfügbar, betrieben von Sinnet.	28. Juli 2021
DAX-Verschlüsselung während der Übertragung	DynamoDB Accelerator (DAX) unterstützt jetzt die Verschlüsselung bei der Übertragung von Daten zwischen Ihren Anwendungen und DAX-Clustern sowie zwischen den Knoten innerhalb eines DAX-Clusters.	24. Juli 2021
CloudFormation und Integration CloudTrail	Integration AWS CloudFormation und Verbesserung der Sicherheit durch Protokollierung auf CloudFormation Datenebene.	18. Juni 2021
CloudFormation wird jetzt für globale Tabellen unterstützt	Amazon DynamoDB unterstützt jetzt globale Tabellen AWS CloudFormation, was bedeutet, dass Sie globale Tabellen erstellen und deren Einstellungen mit CloudFormation Vorlagen verwalten können.	14. Mai 2021

Die lokale Version von Amazon DynamoDB unterstützt Java 2.x	Sie können jetzt das AWS -SDK für Java 2.x mit der lokalen Version von DynamoDB verwenden, der herunterladbaren Version von Amazon DynamoDB. Mit der lokalen Version von DynamoDB können Sie Anwendungen entwickeln und testen, indem Sie eine Version von DynamoDB verwenden, die in Ihrer lokalen Entwicklungsumgebung läuft, ohne dass zusätzliche Kosten anfallen.	3. Mai 2021
NoSQL Workbench unterstützt jetzt AWS CloudFormation	NoSQL Workbench für Amazon DynamoDB unterstützt jetzt AWS CloudFormation, sodass Sie DynamoDB-Datenmodelle mit Vorlagen verwalten und ändern können. CloudFormation Darüber hinaus können Sie jetzt Tabellenkapazitätseinstellungen in NoSQL Workbench konfigurieren.	22. April 2021
DynamoDB und AWS Amplify jetzt Funktionsintegration	AWS Amplify orchestriert jetzt mehrere DynamoDB-Aktualisierungen des globalen sekundären Index in einer einzigen Bereitstellung.	20. April 2021

AWS CloudTrail um die Datenebene von Amazon DynamoDB Streams zu protokollieren APIs	Sie können jetzt AWS CloudTrail verwenden, um die API-Aktivität auf der Datenebene von Amazon DynamoDB Streams zu protokollieren und Änderungen auf Elementebene in Ihren DynamoDB-Tabellen zu überwachen und zu untersuchen.	20. April 2021
Amazon Kinesis Data Streams für Amazon DynamoDB unterstützt jetzt AWS CloudFormation	Amazon Kinesis Data Streams für Amazon DynamoDB unterstützt jetzt AWS CloudFormation, was bedeutet, dass Sie das Streaming in einen Amazon Kinesis Kinesis-Datenstream in Ihren DynamoDB-Tabellen mit Vorlagen aktivieren können. CloudFormation Indem Sie Ihre DynamoDB-Datenänderungen in einen Kinesis-Datenstream streamen, können Sie erweiterte Streaming-Anwendungen mit AAmazon Kinesis-Diensten erstellen.	12. April 2021

Amazon Keyspaces bietet jetzt FIPS-140-2-konforme Endpunkte	Amazon Keyspaces (für Apache Cassandra) bietet jetzt FIPS-140-2-konforme Endpunkte, damit Sie streng regulierte Workloads einfacher verarbeiten können. FIPS (Federal Information Processing Standard) 140-2 ist ein US-amerikanischer und kanadischer Regierungsstandard, mit dem die Sicherheitsanforderungen für Verschlüsselungsmodule angegeben werden, die vertrauliche Informationen schützen.	8. April 2021
Amazon EC2-T3-Instances für DAX	DAX unterstützt jetzt Amazon EC2-T3-Instance-Typen, die ein Basisniveau der CPU-Leistung mit der Möglichkeit bieten, bei Bedarf über das Basisniveau hinauszugehen.	15. Februar 2021
NoSQL Workbench für Amazon DynamoDB – Unterstützung für PartiQL	Sie können jetzt mit NoSQL Workbench für DynamoDB zum Erstellen von PartiQL-Anweisungen für DynamoDB fortfahren.	4. Dezember 2020

PartiQL für DynamoDB	Sie können jetzt PartiQL für DynamoDB — eine SQL-kompatible Abfragesprache — verwenden, um mit DynamoDB-Tabellen zu interagieren und Ad-hoc-Abfragen mithilfe von, und DynamoDB for PartiQL auszuführen. AWS-Managementkonsole AWS Command Line Interface APIs	23. November 2020
Amazon Kinesis Data Streams für Amazon DynamoDB	Sie können jetzt Amazon Kinesis Data Streams für Amazon DynamoDB mit Ihren DynamoDB-Tabellen verwenden, um Änderungen auf Elementebene zu erfassen und in einen Kinesis Data Stream zu replizieren.	23. November 2020
Exportieren von DynamoDB-Tabellen	Sie können jetzt Ihre DynamoDB-Tabellen nach Amazon S3 exportieren, sodass Sie Analysen und komplexe Abfragen Ihrer Daten mit Services wie Athena und Lake Formation durchführen können. AWS Glue	9. November 2020

Unterstützung für leere Werte	DynamoDB unterstützt jetzt leere Werte für Zeichenketten- und Binärattribute ohne Schlüssel in DynamoDB-Tabellen. Die Unterstützung für leere Werte bietet Ihnen eine größere Flexibilität bei der Verwendung von Attributen für einen breiteren Satz von Anwendungsfällen, ohne solche Attribute transformieren zu müssen, bevor sie an DynamoDB gesendet werden. Die Datentypen List, Map und Set unterstützen auch leere String- und Binärwerte.	18. Mai 2020
NoSQL-Workbench für Amazon DynamoDB – Unterstützung für Linux	NoSQL-Workbench für Amazon DynamoDB wird unter Linux-Ubuntu, Fedora und Debian unterstützt.	4. Mai 2020
CloudWatch Einblicke von Mitwirkenden für DynamoDB — GA	CloudWatchContributor Insights for DynamoDB ist allgemein verfügbar. CloudWatch Contributor Insights for DynamoDB ist ein Diagnosetool, das einen at-a-glance Überblick über die Datenverkehrstrends Ihrer DynamoDB-Tabelle bietet und Ihnen hilft, die am häufigsten aufgerufenen Schlüssel Ihrer Tabelle zu identifizieren (auch als Hotkeys bezeichnet).	2. April 2020

Aktualisieren globaler Tabellen	Sie können jetzt Ihre globalen Tabellen von Version 2017.11.29 auf dieaktuelle Version der globalen Tabellen (2019.11.21), mit ein paar Klicks in der DynamoDB Konsole aktualisieren. Durch ein Upgrade der Version Ihrer globalen Tabellen können Sie die Verfügbarkeit Ihrer DynamoDB-Tabellen auf einfache Weise erhöhen, indem Sie Ihre vorhandenen Tabellen um zusätzliche AWS Regionen erweitern, ohne dass eine Neuerstellung der Tabellen erforderlich ist.	16. März 2020
NoSQL-Workbench für Amazon DynamoDB – GA	NoSQL-Workbench für Amazon DynamoDB ist allgemein verfügbar. Verwenden Sie NoSQL-Workbench zum Gestalten , Erstellen, Abfragen und Verwalten von DynamoDB-Tabellen.	2. März 2020
DAX-Cache-Cluster-Metriken	DAX-Unterstützung für neue CloudWatch Metriken, mit denen Sie die Leistung Ihres DAX-Clusters besser verstehen können.	6. Februar 2020

CloudWatch Contributor Insights für DynamoDB — Vorschau	CloudWatchContributor Insights for DynamoDB ist ein Diagnosetool, das einen at-a-glance Überblick über die Datenverkehrstrends Ihrer DynamoDB-Tabelle bietet und Ihnen hilft, die am häufigsten aufgerufenen Schlüssel Ihrer Tabelle zu identifizieren (auch als Hotkeys bezeichnet).	26. November 2019
Unterstützung der adaptiven Kapazität für unausgewogenen Workload	Die Anpassungskapazität von Amazon DynamoDB kann nun unausgewogene Workloads besser bewältigen, indem häufig aufgerufene Elemente automatisch isoliert werden. Wenn eine Anwendung unverhältnismäßig viel Datenverkehr an einzelne oder mehrere Elemente leitet, gleicht DynamoDB die Funktion für adaptive Kapazität die Partitionen so aus, dass häufig aufgerufene Elemente nicht in derselben Partition abgelegt werden.	26. November 2019
Unterstützung für kundenverwaltete Schlüssel	DynamoDB unterstützt jetzt kundenverwaltete Schlüssel. Dies bedeutet, dass Sie die volle Kontrolle darüber haben, wie Sie die Ihre DynamoDB-Daten verschlüsseln und die Sicherheit verwalten.	25. November 2019

NoSQL-Workbench-Support für die lokale Version von DynamoDB (herunterladbare Version)	NoSQL-Workbench unterstützt jetzt Verbindungen zur lokalen Version von DynamoDB (herunterladbare Version) für den Entwurf, die Erstellung, die Abfrage und die Verwaltung von DynamoDB-Tabellen.	8. November 2019
NoSQL Workbench - Vorversion	Dies ist die erste Version von NoSQL Workbench für DynamoDB. Verwenden Sie NoSQL Workbench für zum Gestalten, Erstellen, Abfragen und Verwalten von DynamoDB-Tabellen. Weitere Informationen finden Sie unter NoSQL-Workbench für Amazon DynamoDB (Vorversion).	16. September 2019
DAX fügt Unterstützung für Transaktionsvorgänge mit Python und .NET hinzu	DAX unterstützt das `TransactWriteItems` und `TransactGetItems` APIs für Anwendungen, die in Go, Java, .NET, Node.js und Python geschrieben wurden. Weitere Informationen finden Sie unter In-Memory-Beschleunigung mit DAX.	14. Februar 2019

Aktualisierungen der lokalen Version von Amazon DynamoDB (herunterladbare Version)	DynamoDB local (herunterladbare Version) unterstützt jetzt transaktionale APIs read/write On-Demand-Kapazität, Kapazitätsberichte für Lese- und Schreibvorgänge und 20 globale Sekundärindizes. Weitere Informationen finden Sie unter Unterschiede zwischen der lokalen Ausführung von DynamoDB und dem Amazon-DynamoDB-Webservice.	4. Februar 2019
Amazon DynamoDB On-Demand	DynamoDB-On-Demand-Option ist eine flexible Fakturierungsoption, mit der Tausende von Anforderungen pro Sekunde ohne Kapazitätsplanung bedient werden können. DynamoDB On-Demand bietet pay-per-request Preise für Lese- und Schreibanforderungen, sodass Sie nur für das bezahlen, was Sie tatsächlich nutzen. Weitere Informationen finden Sie unter DynamoDB throughput capacity.	28. November 2018

Amazon DynamoDB Transactions	DynamoDB-Transaktionen nehmen koordinierte all-or-nothing Änderungen an mehreren Elementen sowohl innerhalb als auch tabellenübergreifend vor und sorgen so für Atomizität, Konsistenz, Isolation und Haltbarkeit (ACID) in DynamoDB. Weitere Informationen finden Sie unter Amazon DynamoDB Transactions.	27. November 2018
Amazon DynamoDB verschlüsselt alle gespeicherten Kundendaten im Ruhestand	Die gespeicherte DynamoDB-Verschlüsselung bietet eine zusätzliche Datenschutzebene, indem Ihre Daten in der verschlüsselten Tabelle gesichert werden, einschließlich Primärschlüssel, lokaler und globaler sekundärer Indizes, Streams, globale Tabellen, Backups und DAX-Cluster, wenn die Daten auf dauerhaften Datenträgern gespeichert werden. Weitere Informationen finden Sie unter Amazon-DynamoDB-Verschlüsselung im Ruhezustand.	15. November 2018

Einfachere Verwendung der lokalen Version von Amazon DynamoDB mit dem neuen Docker-Image	Jetzt ist es einfacher, die lokale Version von DynamoDB, die herunterladbare Version von DynamoDB, zu verwenden, um Sie bei der Entwicklung und beim Testen Ihrer DynamoDB-Anwendungen mithilfe des neuen Docker-Images der lokalen Version von DynamoDB zu unterstützen. Weitere Informationen finden Sie unter DynamoDB (herunterladbare Version) und Docker.	22. August 2018
DynamoDB Accelerator (DAX) fügt Support für die Verschlüsselung im Ruhezustand hinzu	DynamoDB Accelerator (DAX) unterstützt jetzt die ruhende Verschlüsselung für neue DAX-Cluster, um Lesevorgänge aus Amazon-DynamoDB-Tabellen in sicherheitsrelevanten Anwendungen zu beschleunigen, die strengen Compliance- und behördlichen Anforderungen unterliegen. Weitere Informationen finden Sie unter DAX-Verschlüsselung im Ruhezustand.	9. August 2018

point-in-timeDynamoDB-Wiederherstellung (PITR) bietet Unterstützung für die Wiederherstellung gelöschter Tabellen	Wenn Sie eine Tabelle mit aktivierter point-in-time Wiederherstellung löschen, wird automatisch eine Systemsicherung erstellt, die 35 Tage lang aufbewahrt wird (ohne zusätzliche Kosten). Weitere Informationen finden Sie unter Bevor Sie die zeitpunktbezogene Wiederherstellung verwenden.	7. August 2018
Aktualisierungen jetzt über RSS verfügbar	Sie können jetzt den RSS-Feed abonnieren (in der linken oberen Ecke dieser Seite), um Benachrichtigungen über Updates im Amazon DynamoDB-Entwicklerleitfaden zu erhalten.	3. Juli 2018

Frühere Aktualisierungen

In der folgenden Tabelle werden die wichtigen Änderungen am DynamoDB-Entwicklerhandbuch bevor dem 3. Juli 2018 beschrieben.

Änderungen	Beschreibung	Änderungsdatum
Unterstützung für DAX	Jetzt können Sie die Leseleistung von Mikrosekunden für Amazon-DynamoDB-Tabellen in Ihren Anwendungen aktivieren, die in der Programmiersprache Go geschrieben wurden, indem Sie das neue DynamoDB Accelerator (DAX) SDK for Go	26. Juni 2018

Änderungen	Beschreibung	Änderungsdatum
	verwenden. Weitere Informationen finden Sie unter DAX SDK für Go.	
DynamoDB kündigt SLA an	DynamoDB hat eine SLA für öffentliche Verfügbarkeit freigegeben. Weitere Informationen finden Sie unter Amazon DynamoDB Service Level Agreement.	19. Juni 2018
Kontinuierliche DynamoDB-Backups und Point-In-Time - Wiederherstellung (PITR)	Point-in-time Recovery schützt Ihre Amazon DynamoDB-Tabellen vor versehentlichen Schreib- oder Löschvorgängen. Mit der zeitpunktbezogenen Wiederherstellung müssen Sie sich keine Gedanken über das Erstellen, Warten oder Planen von On-Demand-Backups machen. Angenommen, Sie schreiben einen Skript-Test versehentlich in eine aktive DynamoDB-Tabelle. Mit point-in-time Recovery können Sie diese Tabelle zu einem beliebigen Zeitpunkt der letzten 35 Tage wiederherstellen. DynamoDB verwaltet inkrementelle Backups Ihrer Tabelle. Weitere Informationen finden Sie unter Zeitpunktbezogene Backups für DynamoDB.	25. April 2018

Änderungen	Beschreibung	Änderungsdatum
Verschlüsselung im Ruhezustand für DynamoDB	DynamoDB-Verschlüsselung im Ruhezustand, die für neue DynamoDB-Tabellen verfügbar ist, hilft Ihnen, Ihre Anwendungsdaten in Amazon-DynamoDB-Tabellen mithilfe von AWS-verwalteten Verschlüsselungsschlüsseln in AWS Key Management Service zu sichern. Weitere Informationen finden Sie unter Ruhende DynamoDB-Verschlüsselung.	8. Februar 2018
Backup und Wiederherstellen von DynamoDB	Mit On-Demand-Backup können Sie vollständige Backups Ihrer DynamoDB-Tabellendaten für die Datenarchivierung erstellen , sodass Sie Ihre Corporate und behördliche Anforderungen erfüllen können. Sie können Tabellen von wenigen Megabyte bis zu Hunderten von Terabyte an Daten sichern, ohne Auswirkungen auf die Leistung und Verfügbarkeit Ihrer Produktionsanwendungen zu haben. Weitere Informationen finden Sie unter Backup und Wiederherstellung für DynamoDB.	29. November 2017

Änderungen	Beschreibung	Änderungsdatum
Globale DynamoDB-Tabellen	Global Tables baut auf dem globalen Footprint von DynamoDB auf, um Ihnen eine vollständig verwaltete, multiregionale und multiaktive Datenbank zur Verfügung zu stellen, die schnelle, lokale Lese- und Schreibleistung für massiv skalierte, globale Anwendungen bietet. Global Tables repliziert Ihre Amazon DynamoDB-Tabellen automatisch in den Regionen Ihrer Wahl. AWS Weitere Informationen finden Sie unter Globale Tabellen: multiaktive, regionsübergreifende Replikation.	29. November 2017
Node.js-Support für DAX	Node.js-Entwickler können unter Verwendung des DAX-Clients für Node.js den Amazon DynamoDB Accelerator (DAX) nutzen. Weitere Informationen finden Sie unter In-Memory-Beschleunigung mit DynamoDB Accelerator (DAX).	5. Oktober 2017

Änderungen	Beschreibung	Änderungsdatum
VPC-Endpunkte für DynamoDB	DynamoDB-Endpunkte erlauben Amazon-EC2-Instanc es in Ihrer Amazon VPC, auf DynamoDB zuzugreif en, ohne dass es im öffentlic hen Internet erreichbar sein muss. Netzwerkverkehr zwischen Ihrer VPC und DynamoDB verlässt das Amazon-Netzwerk nicht. Weitere Informationen finden Sie unter Verwenden von Amazon-VPC-Endpunkten für den Zugriff auf DynamoDB.	16. August 2017

Änderungen	Beschreibung	Änderungsdatum
Auto Scaling für DynamoDB	Das Auto Scaling von DynamoDB lässt die Notwendigkeit einer manuellen Definition oder Anpassung bereitgestellter Durchsatz einstellungen wegfallen. Stattdessen passt das Auto Scaling von DynamoDB die Lese- und Schreibka pazität abhängig von aktuellen Datenverkehrsmustern an. Auf diese Weise kann eine Tabelle oder ein globaler sekundärer Index die bereitgestellte Lese- und Schreibkapazität zum Verarbeiten eines plötzlich en Datenverkehrsanstiegs ohne Drosselung erhöhen. Wenn der Workload nachlässt , verringert DynamoDB Auto Scaling die verfügbare Kapazität. Weitere Informati onen finden Sie unter Automatische Verwaltung der Durchsatzkapazität mit DynamoDB-Auto-Scaling.	14. Juni 2017

Änderungen	Beschreibung	Änderungsdatum
DynamoDB Accelerator (DAX)	DynamoDB Accelerator (DAX) ist ein vollständig verwalteter, hochverfügbarer In-Memory -Cache für DynamoDB, der eine bis zu 10-fache Leistungs steigerung bereitstellt – von Millisekunde bis Mikroseku nde – sogar bei Millionen von Anfragen pro Sekunde. Weitere Informationen finden Sie unter In-Memory-Beschleu nigung mit DynamoDB Accelerator (DAX).	19. April 2017
DynamoDB unterstützt jetzt den automatischen Elementab lauf mit Time-to-Live (TTL)	Amazon DynamoDB-Time-to-Live (TTL) ermöglicht Ihnen abgelaufene Elemente aus Ihren Tabellen automatisch, ohne zusätzliche Kosten, zu löschen. Weitere Informati onen finden Sie unter Verwenden von Time to Live (TTL) in DynamoDB.	27. Februar 2017
DynamoDB unterstützt jetzt Kostenzuordnungs-Tags	Sie können nun Ihren Amazon-DynamoDB-Tabellen Tags für eine verbesserte Nutzungskategorisierung und präzisere Kostenber ichte hinzufügen. Weitere Informationen finden Sie unter Hinzufügen von Tags und Labels zu Ressourcen in DynamoDB.	19. Januar 2017

Änderungen	Beschreibung	Änderungsdatum
Neue DynamoDB `DescribeLimits` -API	Die `DescribeLimits` API gibt die aktuell bereitgestellten Kapazitätsgrenzen für Ihr AWS Konto in einer Region zurück, sowohl für die Region als Ganzes als auch für jede einzelne DynamoDB-Tabelle, die Sie dort erstellen. Sie lässt Sie bestimmen, welche Ihre derzeitigen Beschränkungen auf Kontoebene sind, damit Sie sie mit der bereitgestellten Kapazität, die Sie derzeit verwenden, vergleichen können und dadurch genügend Zeit haben, eine Erhöhung anzufordern, bevor Sie einen Grenzwert erreichen. Weitere Informationen finden Sie unter Kontingente in Amazon DynamoDB und DescribeLimitsin der Amazon DynamoDB DynamoDB-API-Referenz.	1. März 2016

Änderungen	Beschreibung	Änderungsdatum
DynamoDB-Konsolenaktualisierung und neue Terminologie für Primärschlüsselattribute	Die DynamoDB-Managementkonsole wurde intuitiver und benutzerfreundlicher gestaltet. Im Rahmen dieser Aktualisierung führen wir eine neue Terminologie für primäre Schlüsselattribute ein: • Partition Key – auch als Hash-Attribut bekannt. • Sort Key – auch als Bereichsattribut bekannt. Nur die Namen haben sich geändert; die Funktionalität bleibt unverändert. Wenn Sie eine Tabelle oder einen sekundären Index erstellen, können Sie entweder einen einfachen Primärschlüssel (nur Partitionsschlüssel) oder einen zusammengesetzten Primärschlüssel (Partitionsschlüssel und Sortierschlüssel) auswählen. Die DynamoDB-Dokumentation wurde aktualisiert, um diese Änderungen anzuzeigen.	12. November 2015

Änderungen	Beschreibung	Änderungsdatum
Amazon-DynamoDB-Speicher-Backend für Titan	Das DynamoDB-Speicher-Backend für Titan ist ein Storage-Backend für die in Amazon DynamoDB implementierte Titan-Diagrammdatenbank. Wenn Sie das DynamoDB-Speicher-Backend für Titan nutzen, profitieren Ihre Daten von dem DynamoDB-Schutz, der in den hochverfügbaren Rechenzentren von Amazon ausgeführt wird. Das Plugin ist für die Titan-Version 0.4.4 (in erster Linie für die Kompatibilität mit vorhandenen Anwendungen) und Titan-Version 0.5.4 (empfohlen für neue Anwendungen) verfügbar. Wie bei anderen Storage-Backends for Titan unterstützt dieses Plugin Tinkerpop-Stack (Versionen 2.4 und 2.5), einschließlich der Blueprints-API und der Gremlin-Shell.	20. August 2015

Änderungen	Beschreibung	Änderungsdatum
DynamoDB Streams, regionsübergreifende Replikation und Scan mit Strongly-Consistent-Lesevorgängen	DynamoDB Streams erfasst eine zeitlich geordnete Abfolge von Änderungen auf Elementebene in jeder beliebigen DynamoDB-Tabelle und speichert diese Informationen bis zu 24 Stunden in einem Protokoll. Anwendungen können auf dieses Protokoll zugreifen und die Datenelemente vor und nach der Änderung nahezu in Echtzeit aufrufen. Weitere Informationen finden Sie unter Ändern Sie die Datenerfassung für DynamoDB Streams und der DynamoDB-Streams-API-Referenz. Die regionsübergreifende DynamoDB-Replikation ist eine clientseitige Lösung zur Verwaltung identischer Kopien von DynamoDB-Tabellen in verschiedenen AWS Regionen nahezu in Echtzeit. Sie können die regionsübergreifende Replikation verwenden, um DynamoDB-Tabellen zu sichern oder einen Zugriff auf Daten mit geringer Latenz bereitzustellen, in denen Benutzer geografisch verteilt werden.	16. Juli 2015

Änderungen	Beschreibung	Änderungsdatum
	Die DynamoDB `Scan`-Operation verwendet standardmäßig Eventually-Consistent-Lesevorgänge. Sie können stattdessen Strongly-Consistent-Lesevorgänge verwenden, indem Sie den `ConsistentRead` -Parameter auf True setzen. Weitere Informationen finden Sie in Lesekonsistenz für Scan und Scan in der Amazon-DynamoDB-API-Referenz.	
AWS CloudTrail Unterstützung für Amazon DynamoDB	DynamoDB ist jetzt in integriert. CloudTrail CloudTrail erfasst API-Aufrufe, die über die DynamoDB-Konsole oder über die DynamoDB-API getätigt wurden, und verfolgt sie in Protokolldateien. Weitere Informationen finden Sie im Protokollieren von DynamoDB-Vorgängen mithilfe von AWS CloudTrail und dem AWS CloudTrail Benutzerhandbuch.	28. Mai 2015

Änderungen	Beschreibung	Änderungsdatum
Verbesserte Unterstützung für Abfrageausdrücke	Diese Version fügt einen neuen `KeyConditionExpression` -Parameter der `Query`-API hinzu. Eine `Query` liest Elemente aus einer Tabelle oder einem Index mithilfe von Primärschlüsselwerten. Der `KeyConditionExpression` -Parameter ist eine Zeichenfolge, die Primärschlüsselnamen und Bedingungen identifiziert, die auf den Schlüsselwert angewendet werden sollen. Die `Query` ruft ausschließlich die Elemente ab, die den Ausdruck erfüllen. Die Syntax von `KeyConditionExpression` ist ähnlich wie bei anderen Ausdrucksparametern in DynamoDB und erlaubt es Ihnen, Substitutionsvariablen für die Namen und Werte innerhalb des Ausdrucks zu definieren. Weitere Informationen finden Sie unter Abfragen von Tabellen in DynamoDB.	27. April 2015

Änderungen	Beschreibung	Änderungsdatum
Neue Vergleichsfunktionen für bedingte Schreibvorgänge	In DynamoDB bestimmt der `ConditionExpression` -Parameter, ob ein `PutItem`, `UpdateItem` , oder `DeleteItem` gelingt: Das Element wird nur geschrieben, wenn die Bedingung True ergibt. Diese Version umfasst zwei neue Funktionen, `attribute_type` und `size` für die Verwendung mit `ConditionExpression` . Mit diesen Funktionen können Sie einen bedingten Schreibvorgang basierend auf dem Datentyp oder der Größe eines Attributs in einer Tabelle durchführen. Weitere Informationen finden Sie unter CLI-Beispiel für DynamoDB-Bedingungsausdrücke.	27. April 2015

Änderungen	Beschreibung	Änderungsdatum
Durchsuchen der API nach sekundären Indizes	In DynamoDB liest eine `Scan`-Operation alle Elemente in einer Tabelle, wendet benutzerdefinierte Filterkriterien an und gibt die ausgewählten Datenelemente an die Anwendung zurück. Diese Funktion ist jetzt auch für sekundäre Indizes verfügbar. Zum Durchsuchen eines lokalen sekundären Indexes oder eines globalen sekundären Indexes, geben Sie den Indexnamen und den Namen seiner übergeordneten Tabelle an. Standardmäßig gibt ein `Scan`-Index alle Daten in den Index zurück. Sie können einen Filterausdruck zum Eingrenzen der Ergebnisse, die an die Anwendung zurückgegeben werden, nutzen. Weitere Informationen finden Sie unter Scannen von Tabellen in DynamoDB.	10. Februar 2015

Änderungen	Beschreibung	Änderungsdatum
Onlineoperationen für globale sekundäre Indizes	Mit Online-Indizierung können Sie globale sekundäre Indizes für vorhandene Tabellen hinzufügen oder entfernen. Mithilfe der Online-Indizierung müssen Sie nicht alle Indizes einer Tabelle definieren, wenn Sie eine Tabelle erstellen . Stattdessen können Sie jederzeit einen neuen Index hinzufügen. Wenn Sie einen Index dementsprechend nicht länger benötigen, können Sie ihn jederzeit entfernen. Online-Indizierungsvorgänge sind blockierungsfrei, sodass die Tabelle weiterhin für Lese- und Schreibaktivitäten verfügbar ist, während Indizes hinzugefügt oder entfernt werden. Weitere Informationen finden Sie unter Verwalten globaler sekundärer Indizes in DynamoDB.	27. Januar 2015

Änderungen	Beschreibung	Änderungsdatum
Dokumentmodell-Unterstützung mit JSON	DynamoDB ermöglicht Ihnen das Speichern und Abrufen von Dokumenten mit vollständigem Support für Dokumentmodelle. Neue Datentypen sind vollständig kompatibel mit dem JSON-Standard und ermöglichen Ihnen, Dokumentelemente ineinander zu verschachteln. Sie können Dokumentpfad-Dereferenzierungsoperatoren nutzen, um einzelne Elemente zu lesen und zu schreiben, ohne dass Sie das gesamte Dokument abrufen müssen. Diese Version führt, beim Lesen oder Schreiben von Datenelementen, auch neue Ausdrucksparameter für die Angabe von Prognosen, Bedingungen und Aktualisierungsaktionen ein. Weitere Informationen zur Dokumentmodell-Unterstützung mit JSON finden Sie unter Datentypen und Verwenden von Ausdrücken in DynamoDB.	7. Oktober 2014

Änderungen	Beschreibung	Änderungsdatum
Flexible Skalierung	Für Tabellen und globale sekundäre Indizes können Sie bereitgestellte Lese- und Schreibdurchsatzkapazität in beliebigem Umfang erhöhen, vorausgesetzt Sie bleiben pro Tabelle und Konto innerhalb Ihres Grenzwerts. Weitere Informationen finden Sie unter Kontingente in Amazon DynamoDB.	7. Oktober 2014
Größere Elementgrößen	Die maximale Elementgröße in DynamoDB ist von 64 KB auf 400 KB gestiegen. Weitere Informationen finden Sie unter Kontingente in Amazon DynamoDB.	7. Oktober 2014

Änderungen	Beschreibung	Änderungsdatum
Verbesserte bedingte Ausdrücke	DynamoDB erweitert die verfügbaren Operatoren für bedingte Ausdrücke. Dadurch erhalten Sie noch mehr Flexibilität für bedingte Ablege-, Aktualisierungs- und Löschvorgänge. Mit den neuen verfügbaren Operatoren können Sie prüfen, ob ein Attribut vorhanden ist oder nicht, größer oder kleiner ist als ein bestimmter Wert, zwischen zwei Werten ist, mit bestimmten Zeichen beginnt und vieles mehr. DynamoDB bietet auch einen optionalen OR-Operator für die Bewertung von mehrere Bedingungen. Standardmäßig werden mehrere Bedingungen in einem Ausdruck zusammengesetzt (UND), sodass der Ausdruck nur dann True ist, wenn alle seine Bedingungen True sind. Wenn Sie stattdessen ODER angeben, ist der Ausdruck True, wenn eine oder mehrere Bedingungen True sind. Weitere Informationen finden Sie unter Arbeiten mit Elementen und Attributen in DynamoDB.	24. April 2014

Änderungen	Beschreibung	Änderungsdatum
Abfragefilter	Die DynamoDB`Query`-API unterstützt eine neue `QueryFilter` -Option. Standardmäßig findet `Query` Elemente, die einem bestimmten Partitions-Schlüsselwert und einer optionalen Sortierschlüsselbedingung entsprechen. Ein `Query`-Filter wendet bedingte Ausdrücke auf andere Nicht-Schlüsselattribute an. Wenn ein `Query`-Filter vorhanden ist, werden Elemente, die nicht den Filterbedingungen entsprechen, verworfen, bevor die `Query`-Ergebnisse an die Anwendung zurückgegeben werden. Weitere Informationen finden Sie unter Abfragen von Tabellen in DynamoDB.	24. April 2014

Änderungen	Beschreibung	Änderungsdatum
Datenexport und -import mit dem AWS-Managementkonsole	Die DynamoDB-Konsole wurde erweitert, um Exporte und Importe von Daten in DynamoDB-Tabellen zu vereinfachen. Mit nur wenigen Klicks können Sie einen AWS Data Pipeline zur Orchestrierung des Workflows und einen Amazon MapReduce Elastic-Cluster zum Kopieren von Daten aus DynamoDB-Tabellen in einen Amazon S3 S3-Bucket einrichten oder umgekehrt. Sie können nur einmalig einen Export oder Import durchführen oder einen täglichen Exportauftrag einrichten. Sie können sogar regionsübergreifende Exporte und Importe durchführen und DynamoDB-Daten aus einer Tabelle in einer Region in eine Tabelle in einer anderen AWS Region kopieren. AWS	6. März 2014

Änderungen	Beschreibung	Änderungsdatum
Umstrukturierte Higher-Level-API-Dokumentation	Informationen zu folgenden Themen APIs sind jetzt einfacher zu finden: • Java: Dynamo DBMappper • .NET: Dokument-Modell und Objekt-Persistenz-Modell Diese höheren Ebenen APIs sind jetzt hier dokumentiert:. Higher-Level-Programmierschnittstellen für DynamoDB	20. Januar 2014

Änderungen	Beschreibung	Änderungsdatum
Globale sekundäre Indizes	DynamoDB fügt Support für globale sekundäre Indizes hinzu. Wie bei einem lokalen sekundären Index definieren Sie einen globalen sekundäre n Index, indem Sie einen alternativen Schlüssel aus einer Tabelle nutzen und dann eine Abfrageanforderung an den Index ausgeben. Im Gegensatz zu einem lokalen sekundären Index, muss der Partitionsschlüssel für den globalen sekundären Index nicht derselbe sein wie der aus der Tabelle. Er kann ein beliebiges skalares Attribut aus der Tabelle sein. Der Sortierschlüssel ist optional und kann auch ein beliebiges skalares Attribut der Tabelle sein. Ein globaler sekundärer Index verfügt auch über eigene bereitgestellte Durchsätze, die unabhängig von denjenigen der übergeordneten Tabelle sind. Weitere Informati onen erhalten Sie unter Verbessern des Datenzugr iffs mit sekundären Indizes in DynamoDB und Verwenden globaler sekundärer Indizes in DynamoDB.	12. Dezember 2013

Änderungen	Beschreibung	Änderungsdatum
Differenzierte Zugriffskontrolle	DynamoDB fügt Support für eine differenzierte Zugriffsk ontrolle hinzu. Diese Funktion ermöglicht es Kunden anzugeben, welche Prinzipal s (Benutzer, Gruppen oder Rollen) auf einzelne Elemente und Attribute in einer DynamoDB-Tabelle oder einem sekundären Index zugreifen können. Anwendung en können auch einen Web-Identitätsverbund nutzen, um die Aufgabe der Benutzera uthentifizierung an einen Drittidentitätsanbieter wie Facebook, Google oder Login with Amazon auszulagern. Auf diese Weise, können Anwendungen (einschli eßlich mobiler Anwendung en) eine sehr große Zahl von Benutzern verarbeiten, wobei gleichzeitig sichergestellt wird, dass niemand Zugriff auf DynamoDB-Datenelemente hat, es sei denn, derjenige n ist autorisiert. Weitere Informationen finden Sie unter Verwenden von IAM-Richt linienbedingungen für die differenzierte Zugriffskontrolle.	29. Oktober 2013

Änderungen	Beschreibung	Änderungsdatum
4 KB Lesekapazitätseinheitsgröße	Die Schreibkapazitätseinheitsgröße für Lesevorgänge ist von 1 KB auf 4 KB gestiegen. Diese Erweiterung kann die Anzahl der, von vielen Anwendungen benötigten, bereitgestellten Lesekapazitätseinheiten verringern. Zum Beispiel verbrauchte das Lesen eines 10 KB großen Elements vor dieser Version noch 10 Lesekapazitätseinheiten. Jetzt verbraucht das Lesen dieser 10 KB lediglich 3 Einheiten (10 KB/4 KB, aufgerundet auf den nächsten 4-KB-Grenzwert). Weitere Informationen finden Sie unter DynamoDB-Durchsatzkapazität.	14. Mai 2013

Änderungen	Beschreibung	Änderungsdatum
Parallele Scans	DynamoDB fügt Support für parallele Scan-Operationen hinzu. Anwendungen können nun eine Tabelle in logische Segmente unterteilen und alle Segmente gleichzeitig scannen. Diese Funktion reduziert die erforderliche Zeit, die ein Scan benötigt, um abzuschließen und nutzt die bereitgestellte Lesekapazität einer Tabelle vollständig. Weitere Informationen finden Sie unter Scannen von Tabellen in DynamoDB.	14. Mai 2013
Lokale sekundäre Indizes	DynamoDB fügt Unterstützung für lokale sekundäre Indizes hinzu. Sie können Sortierschlüsselindizes für Nicht-Schlüsselattribute definieren und diese dann in Abfrageanforderungen nutzen. Mit lokalen sekundären Indexen können Anwendungen Datenelemente über mehrere Dimensionen effizient abrufen. Weitere Informationen finden Sie unter Lokale sekundäre Indizes.	18. April 2013

Änderungen	Beschreibung	Änderungsdatum
Neue API-Version	In dieser Version stellt DynamoDB eine neue API-Version (2012-08-10) bereit. Die vorherige API-Version (2011-12-05) wird weiterhin für die Rückwärtskompatibilität mit vorhandenen Anwendungen unterstützt. Neue Anwendungen sollten die neue API-Version 2012-08-10 verwenden. Wir empfehlen, dass Sie Ihre vorhandenen Anwendungen auf die API-Version 2012-08-10 migrieren, da neue DynamoDB-Funktionen (z. B. lokale sekundäre Indizes) nicht auf die vorherige API-Version nachgezogen werden. Weitere Informationen zur API-Version 2012-08-10 finden Sie unter Amazon-DynamoDB-API-Referenz.	18. April 2013

Änderungen	Beschreibung	Änderungsdatum
Unterstützung für IAM-Richtlinienvariablen	Die IAM-Zugriffsrichtliniensprache unterstützt jetzt Variablen. Wenn eine Richtlinie ausgewertet wird, werden alle Richtlinienvariablen durch Werte ersetzt, die von Kontext-basierten Informationen aus der Sitzung des authentifizierten Benutzers bereitgestellt werden. Sie können Richtlinienvariablen zum Definieren von allgemeinen Richtlinien nutzen, ohne explizit alle Komponenten der Richtlinien aufzulisten. Weitere Informationen zu Richtlinienvariablen finden Sie unter Richtlinienvariablen im AWS Identity and Access Management Verwenden-von-IAM-Handbuch. Beispiele für Richtlinienvariablen in DynamoDB finden Sie unter Identity and Access Management für Amazon DynamoDB.	4. April 2013

Änderungen	Beschreibung	Änderungsdatum
PHP-Codebeispiele wurden für AWS SDK für PHP Version 2 aktualisiert	Version 2 von AWS SDK für PHP ist jetzt verfügbar . Die PHP-Codebeispiele im Amazon-DynamoDB-Entwicklerhandbuch wurden aktualisiert, um dieses neue SDK zu nutzen. Weitere Informationen zur Version 2 des SDK finden Sie unter AWS SDK für PHP.	23. Januar 2013
Neuer Endpunkt	DynamoDB expandiert in die Region AWS GovCloud (US-West). Eine aktuelle Liste von Service-Endpunkten und Protokollen finden Sie unter Regionen und Endpunkte.	3. Dezember 2012
Neuer Endpunkt	DynamoDB expandiert in die Region Südamerika (São Paulo). Eine aktuelle Liste von unterstützten Endpunkten finden Sie unter Regionen und Endpunkte.	3. Dezember 2012
Neuer Endpunkt	DynamoDB expandiert in die Region Asien-Pazifik (Sydney). Eine aktuelle Liste von unterstützten Endpunkten finden Sie unter Regionen und Endpunkte.	13. November 2012

Änderungen	Beschreibung	Änderungsdatum
DynamoDB implementiert Unterstützung für CRC32 Prüfsummen, unterstützt stark konsistente Batch-Abrufe und entfernt Einschränkungen für gleichzeitige Tabellena ktualisierungen.	• DynamoDB berechnet eine CRC32 Prüfsumme der HTTP-Nutzlast und gibt diese Prüfsumme in einem neuen Header zurück,. `x-amz-crc32` Weitere Informationen finden Sie unter DynamoDB Low-Level -API. • Leseoperationen, die von der `BatchGetItem` -API durchgeführt werden, sind standardmäßig Eventuall y Consistent. Ein neuer `ConsistentRead` - Parameter in `BatchGetI tem` ermöglicht Ihnen stattdessen eine starke Lesekonsistenz für alle Tabellen in der Anforderu ng auszuwählen. Weitere Informationen finden Sie unter Beschreibung. • Diese Version entfernt einige Einschränkungen beim gleichzeitigen Aktualisi eren vieler Tabellen. Die Gesamtanzahl der Tabellen, die gleichzeitig aktualisi ert werden können, liegt immer noch bei 10. Diese Tabellen können nun jedoch eine beliebige Kombination des `CREATING`, `UPDATING`	2. November 2012

Änderungen	Beschreibung	Änderungsdatum
	oder `DELETING`-Status sein. Außerdem gibt es für eine Tabelle keinen Mindestbetrag mehr für das Erhöhen oder Verkleinern von oder. ReadCapacityUnitsWriteCapacityUnits Weitere Informationen finden Sie unter Kontingente in Amazon DynamoDB.	
Dokumentation bewährte Methoden	Das Amazon-DynamoDB-Entwicklerhandbuch identifiziert bewährte Methoden für die Arbeit mit Tabellen und Elementen zusammen mit Empfehlungen für Abfrage- und Scan-Operationen.	28. September 2012

Änderungen	Beschreibung	Änderungsdatum
Unterstützung für binären Datentyp	Zusätzlich zu den Zahlen- und Zeichenfolgetypen unterstützt DynamoDB jetzt Binärdatentypen. Für die Speicherung von Binärdaten vor Erscheinen dieser Version, haben Sie Ihre binären Daten in das Zeichenfolgeformat konvertiert und diese in DynamoDB gespeichert. Zusätzlich zu der erforderlichen clientseitigen Konvertierungsarbeit, erhöhte die Konvertierung oft die Größe des Datenelements, was mehr Speicherplatz und potenziell zusätzlich bereitgestellte Durchsatzkapazität erforderte. Mit dem Binärtypattribut können Sie jetzt alle binären Daten speichern, z. B. komprimierte Daten, verschlüsselte Daten und Bilder. Weitere Informationen finden Sie unter Datentypen. Praktische Beispiele für den Umgang mit binären Daten mithilfe von finden Sie in den folgenden Abschnitten: AWS SDKs	21. August 2012

Änderungen	Beschreibung	Änderungsdatum
	• Beispiel: Umgang mit binären Typattributen mithilfe der AWS SDK für Java Dokument-API • Beispiel: Umgang mit binären Attributen mithilfe der AWS SDK für .NET Low-Level-API Für die in der AWS SDKs hinzugefügte Unterstützung für binäre Datentypen müssen Sie die neueste Version herunterl aden SDKs und möglicher weise auch alle vorhandenen Anwendungen aktualisieren. Weitere Informationen zum Herunterladen von AWS SDKs finden Sie unter .NET-Code beispiele.	
DynamoDB-Tabellenelemente können mit der DynamoDB-Konsole aktualisiert und kopiert werden	DynamoDB-Benutzer können jetzt mit der DynamoDB-Konsole Tabellenelemente zusätzlich zu dem Hinzufüge n und Löschen von Elementen aktualisieren und kopieren. Diese neue Funktionalität vereinfacht die Änderungen an einzelnen Elementen mit der Konsole.	14. August 2012

Änderungen	Beschreibung	Änderungsdatum
DynamoDB senkt die minimalen Durchsatzanforderungen der Tabelle	DynamoDB unterstützt jetzt niedrigere minimale Durchsatzanforderungen der Tabelle, insbesondere 1 Schreib- und 1 Lesekapazitätseinheit. Weitere Informationen finden Sie unter dem Kontingente in Amazon DynamoDB Thema im Amazon-DynamoDB-Entwicklerhandbuch.	9. August 2012
Unterstützung für Signatur-Version 4	DynamoDB unterstützt jetzt Signatur-Version 4 zum Authentifizieren von Anforderungen.	5. Juli 2012
Support für Tabellen-Explorer in der DynamoDB-Konsole	Die DynamoDB-Konsole unterstützt jetzt einen Tabellen-Explorer, mit dem Sie die Daten in Ihren Tabellen durchsuchen und abfragen können. Sie können auch neue Elemente einfügen oder vorhandene Elemente löschen. Die `SampleData` und Verwenden der Konsole-Abschnitte wurden für diese Funktionen aktualisiert.	22. Mai 2012

Änderungen	Beschreibung	Änderungsdatum
Neue Endpunkte	Die Verfügbarkeit von DynamoDB wird um neue Endpunkte in der Region USA West (Nordkalifornien), USA West (Oregon) und Asien-Pazifik (Singapur) erweitert. Eine aktuelle Liste von unterstützten Endpunkten finden Sie unter Regionen und Endpunkte.	24. April 2012
BatchWriteItem API-Unterstützung	DynamoDB unterstützt jetzt eine Batch-Write-API, mit der Sie mehrere Elemente aus einer oder mehreren Tabellen mit einem einzelnen API-Aufruf ablegen oder löschen können. Weitere Informationen zu der DynamoDB-Batch-Write-API finden Sie unter BatchWriteItem. Hinweise zur Arbeit mit Elementen und zur Verwendung der Batch-Schreibfunktion finden Sie unter Arbeiten mit Elementen und Attributen in DynamoDB und.NET-Codebeispiele. AWS SDKs	19. April 2012
Weitere Fehlercodes dokumentiert	Weitere Informationen finden Sie unter Fehlerbehandlung mit DynamoDB.	5. April 2012

Änderungen	Beschreibung	Änderungsdatum
Neuer Endpunkt	DynamoDB expandiert in die Region Asien-Pazifik (Tokio). Eine aktuelle Liste von unterstützten Endpunkten finden Sie unter Regionen und Endpunkte.	29. Februar 2012
`ReturnedItemCount` - Metrik hinzugefügt	Eine neue Metrik,`ReturnedItemCount` , gibt die Anzahl der Elemente an, die als Antwort auf einen Abfrage- oder Scanvorgang zurückgegeben wurden, damit DynamoDB überwacht werden kann. CloudWatch	24. Februar 2012
Beispiele für inkrementelle Werte hinzugefügt	DynamoDB unterstützt das Erhöhen und Verringern vorhandener numerischer Werte. Beispiele zeigen das Hinzufügen auf vorhanden e Werte in den Abschnitten „Aktualisieren eines Elements“ bei: Arbeiten mit Elementen: Java. Arbeiten mit Elementen: .NET.	25. Januar 2012
Erste Produktversion	DynamoDB wird als neuer Service in der Beta-Version eingeführt.	18. Januar 2012

Ältere Features von DynamoDB

Bei den folgenden Themen handelt es sich um ältere Features, die DynamoDB weiterhin unterstützt. Diese werden nicht mehr aktiv weiterentwickelt.

Themen

- Version 2017.11.29 (Legacy) der globalen Tabellen
- Vorherige Version der Low-Level-API für DynamoDB (2011-12-05)
- Bedingte Legacy-Parameter für DynamoDB

Version 2017.11.29 (Legacy) der globalen Tabellen

> **Important**
>
> Diese Dokumentation bezieht sich auf globale Tabellen der Version 2017.11.29 (veraltet), die für neue globale Tabellen vermieden werden sollte. Kunden sollten nach Möglichkeit die Version 2019.11.21 (Aktuell) für globale Tabellen verwenden, da sie mehr Flexibilität und höhere Effizienz bietet und weniger Schreibkapazität verbraucht als Version 2017.11.29 (Legacy).
> Informationen dazu, welche Version Sie verwenden, finden Sie unter Ermitteln der Version einer globalen Tabelle. Informationen zur Aktualisierung globaler Tabellen von Version 2017.11.29 (veraltet) auf Version 2019.11.21 (aktuell) finden Sie unter Versionen von globalen DynamoDB-Tabellen.

Themen

- Globale Tabellen: Funktionsweise
- Bewährte Methoden und Anforderungen für die Verwaltung globaler Tabellen
- Eine globale Tabelle erstellen (Version 2017.11.29)
- Überwachen globaler Tabellen
- Verwenden von IAM mit globalen Tabellen

Globale Tabellen: Funktionsweise

> **Important**
>
> Diese Dokumentation bezieht sich auf globale Tabellen der Version 2017.11.29 (veraltet), die für neue globale Tabellen vermieden werden sollte. Kunden sollten nach Möglichkeit die Version 2019.11.21 (Aktuell) für globale Tabellen verwenden, da sie mehr Flexibilität und höhere Effizienz bietet und weniger Schreibkapazität verbraucht als Version 2017.11.29 (Legacy).
> Informationen dazu, welche Version Sie verwenden, finden Sie unter Ermitteln der Version einer globalen Tabelle. Informationen zur Aktualisierung globaler Tabellen von Version 2017.11.29 (veraltet) auf Version 2019.11.21 (aktuell) finden Sie unter Versionen von globalen DynamoDB-Tabellen.

In den folgenden Abschnitten erfahren Sie mehr über die Konzepte und das Verhalten globaler Tabellen in Amazon DynamoDB.

Globale Tabellenkonzepte für Version 2017.11.29 (veraltet)

Eine globale Tabelle ist eine Sammlung von einer oder mehreren Replikattabellen, die alle einem einzigen AWS Konto gehören.

Eine Replikattabelle (oder kurz ein Replikat) ist eine einzelne DynamoDB-Tabelle, die als Teil einer globalen Tabelle fungiert. Jedes Replikat speichert die gleichen Datenelemente. Jede bestimmte globale Tabelle kann nur über eine Replikattabelle pro AWS -Region verfügen.

Im Folgenden finden Sie einen konzeptionellen Überblick darüber, wie eine globale Tabelle erstellt wird.

1. Erstellen Sie eine normale DynamoDB-Tabelle mit aktivierten DynamoDB Streams in einer Region. AWS
2. Wiederholen Sie Schritt 1 für alle anderen Regionen, in denen Sie Ihre Daten replizieren möchten.
3. Definieren Sie eine globale Tabelle in DynamoDB basierend auf den Tabellen, die Sie erstellt haben.

Das AWS-Managementkonsole automatisiert diese Aufgaben, sodass Sie schneller und einfacher eine globale Tabelle erstellen können. Weitere Informationen finden Sie unter Eine globale Tabelle erstellen (Version 2017.11.29).

Die resultierende globale DynamoDB-Tabelle besteht aus mehreren Replikattabellen, eine pro Region, die DynamoDB als eine Einheit behandelt. Jedes Replikat hat den gleichen Tabellennamen und das gleiche Primärschlüsselschema. Wenn eine Anwendung Daten in eine Replikattabelle in einer Region schreibt, verteilt DynamoDB den Schreibvorgang automatisch auf die anderen Replikattabellen in den übrigen AWS -Regionen.

Important

Um Ihre Tabellendaten synchron zu halten, erstellen globale Tabellen automatisch die folgenden Attribute für jedes Element:

- `aws:rep:deleting`
- `aws:rep:updatetime`
- `aws:rep:updateregion`

Ändern Sie diese Attribute nicht oder erstellen Sie Attribute mit demselben Namen.

Sie können der globalen Tabelle Replikattabellen hinzufügen, sodass sie in weiteren Regionen verfügbar ist. (Dazu muss die globale Tabelle leer sein. Anders ausgedrückt, in keiner Replikattabelle dürfen Daten enthalten sein.)

Sie können eine Replikattabelle auch aus einer globalen Tabelle entfernen. Wenn Sie dies tun, wird die Tabelle vollständig von der globalen Tabelle getrennt. Diese nun unabhängige Tabelle interagiert nicht mehr mit der globalen Tabelle und es werden keine Daten mehr in die und aus der globalen Tabelle übertragen.

Warning

Beachten Sie, dass das Entfernen eines Replikats kein unteilbarer Prozess ist. Um ein konsistentes Verhalten und einen bekannten Zustand zu gewährleisten, sollten Sie erwägen, den Schreibdatenverkehr Ihrer Anwendung vorher von dem Replikat wegzuleiten, das entfernt werden soll. Warten Sie nach dem Entfernen, bis alle Endpunkte der Replikatregion

das Replikat als getrennt anzeigen, bevor Sie weitere Schreibvorgänge als eigene isolierte Tabelle der Regionen vornehmen.

Allgemeine Aufgaben

Allgemeine Aufgaben für globale Tabellen funktionieren wie folgt.

Sie können die Replikattabelle einer globalen Tabelle genauso löschen wie eine reguläre Tabelle. Dadurch wird die Replikation in diese Region beendet und die in dieser Region gespeicherte Tabellenkopie gelöscht. Sie können die Replikation nicht trennen und Kopien der Tabelle als unabhängige Entitäten existieren lassen.

> **Note**
>
> Sie können eine Quelltabelle erst 24 Stunden, nachdem sie zum Initiieren einer neuen Region verwendet wurde, löschen. Wenn Sie versuchen, sie zu früh zu löschen, erhalten Sie eine Fehlermeldung.

Konflikte entstehen, wenn Anwendungen dasselbe Element in verschiedenen Regionen fast zum gleichen Zeitpunkt aktualisieren. Zur Sicherstellung der letztendliche Konsistenz verwenden globale DynamoDB-Tabellen bei gleichzeitigen Updates einen Mechanismus, bei dem der letzte Schreibvorgang gültig ist. Alle Replikate verständigen sich auf das neueste Update und nähern sich einem Zustand an, in dem sie alle über identische Daten verfügen.

> **Note**
>
> Es gibt mehrere Möglichkeiten, Konflikte zu vermeiden, unter anderem:
>
> - Verwenden einer IAM-Richtlinie, um nur Schreibvorgänge in die Tabelle in einer Region zuzulassen.
> - Verwenden einer IAM-Richtlinie, um Benutzer nur an eine Region weiterzuleiten und die andere als Standby-Region zu belassen, oder abwechselnd ungerade Benutzer an eine Region und gerade Benutzer an eine andere Region weiterzuleiten.
> - Vermeiden von nicht idempotenten Updates wie Bookmark = Bookmark + 1 und Bevorzugen statischer Updates wie Bookmark=25

Überwachen globaler Tabellen

Sie können es verwenden CloudWatch , um die Metrik `ReplicationLatency` zu beobachten. Diese Metrik misst die verstrichene Zeit zwischen dem Erscheinen eines aktualisierten Elements im DynamoDB-Stream für eine Replikattabelle und dem Erscheinen dieses Elements in einem anderen Replikat in der globalen Tabelle. `ReplicationLatency` wird in Millisekunden ausgedrückt und für jedes Quell- und Zielregionspaar ausgegeben. Dies ist die einzige CloudWatch Metrik, die von Global Tables v2 bereitgestellt wird.

Die Latenzen, die Sie beobachten werden, hängen von der Entfernung zwischen den ausgewählten Regionen sowie von anderen Variablen ab. Latenzen im Bereich von 0,5 bis 2,5 Sekunden für Regionen kommen innerhalb desselben geografischen Gebiets häufig vor.

Time to Live (TTL)

Sie können Time to Live (TTL) verwenden, um einen Attributnamen anzugeben, dessen Wert die Ablaufzeit für das Element angibt. Dieser Wert wird als Zahl in Sekunden seit Beginn der Unix-Epoche angegeben.

Bei der älteren Version von globalen Tabellen werden die TTL-Löschungen nicht automatisch auf andere Replikate repliziert. Wenn ein Element mithilfe einer TTL-Regel gelöscht wird, wird diese Aufgabe ausgeführt, ohne dass Schreibeinheiten verbraucht werden.

Hinweis: Wenn die Quell- und Zieltabellen eine sehr geringe bereitgestellte Schreibkapazität haben, kann dies zu einer Drosselung führen, da die TTL-Löschungen Schreibkapazität erfordern.

Streams und Transaktionen mit globalen Tabellen

Jede globale Tabelle erzeugt einen unabhängigen Stream, der auf all ihren Schreibvorgängen basiert, unabhängig vom Ausgangspunkt dieser Schreibvorgänge. Sie können wählen, ob Sie diesen DynamoDB-Stream in einer Region oder in allen Regionen unabhängig voneinander nutzen möchten.

Wenn Sie verarbeitete lokale Schreibvorgänge, aber keine replizierten Schreibvorgänge möchten, können Sie jedem Element Ihr eigenes Regionsattribut hinzufügen. Dann können Sie einen Lambda-Ereignisfilter verwenden, um nur Lambda für Schreibvorgänge in der lokalen Region aufzurufen.

Transaktionale Operationen umfassen ACID (Atomizität, Konsistenz, Isolierung und Zuverlässigkeit) NUR in der Region, in der der Schreibvorgang ursprünglich durchgeführt wurde. Regionsübergreifende Transaktionen werden in globalen Tabellen nicht unterstützt.

Wenn Sie beispielsweise über eine globale Tabelle mit Replikaten in den Regionen USA Ost (Ohio) und USA West (Oregon) verfügen und einen TransactWriteItems Vorgang in der Region USA Ost (Ohio) ausführen, können Sie beobachten, wie teilweise abgeschlossene Transaktionen in der Region USA West (Oregon) repliziert werden. Die Änderungen werden erst in die anderen Regionen repliziert, nachdem sie in der Quellregion in die Datenbank eingetragen wurden.

> **Note**
>
> - Globale Tabellen „umgehen“ DynamoDB Accelerator (DAX), indem sie DynamoDB direkt aktualisieren. Infolgedessen ist DAX nicht bekannt, dass er veraltete Daten speichert. Der DAX-Cache wird erst aktualisiert, wenn die TTL des Caches abläuft.
> - Tags in globalen Tabellen werden nicht automatisch weitergegeben.

Lese- und Schreibdurchsatz

Globale Tabellen verwalten den Lese- und Schreibdurchsatz wie folgt.

- Die Schreibkapazität muss auf allen Tabellen-Instances in sämtlichen Regionen gleich sein.
- Bei Version 2019.11.21 (Aktuell) wird die Schreibkapazität automatisch synchronisiert, wenn die Tabelle so eingestellt ist, dass sie Auto Scaling unterstützt oder sich im On-Demand-Modus befindet. Die derzeit in jeder Region bereitgestellte Schreibkapazität steigt und fällt innerhalb dieser synchronisierten Auto-Scaling-Einstellungen unabhängig voneinander. Wenn die Tabelle in den On-Demand-Modus versetzt wird, wird dieser Modus mit den anderen Replikaten synchronisiert.
- Die Lesekapazität kann je nach Region unterschiedlich sein, da die Lesevorgänge möglicherweise nicht identisch sind. Beim Hinzufügen eines globalen Replikats zu einer Tabelle wird die Kapazität der Quellregion übertragen. Nach der Erstellung können Sie die Lesekapazität für ein Replikat anpassen und diese neue Einstellung wird nicht auf die andere Seite übertragen.

Konsistenz und Konfliktlösung

Alle Änderungen, die an einem Element in einer Replikattabelle vorgenommen werden, werden auf alle anderen Replikate innerhalb derselben globalen Tabelle repliziert. In einer globalen Tabelle wird ein neu geschriebenes Element in der Regel innerhalb von wenigen Sekunden auf alle Replikattabellen verteilt.

Mit einer globalen Tabelle speichert jede Replikattabelle die gleichen Datenelemente. DynamoDB unterstützt keine Teilreplikation nur einiger Elemente.

Eine Anwendung hat Lese- und Schreibzugriff auf alle Replikattabellen. DynamoDB unterstützt letztendlich konsistente Lesevorgänge regionsübergreifend, jedoch keine strikt konsistenten Lesevorgänge in allen Regionen. Wenn Ihre Anwendung nur eventuell konsistente Lesevorgänge verwendet und Lesevorgänge nur für eine AWS Region ausgibt, funktioniert sie ohne Änderungen. Wenn Ihre Anwendung jedoch strikt konsistente Lesevorgänge erfordert, muss sie alle strikt konsistenten Lese- und Schreibvorgänge in derselben Region ausführen. Wenn Sie in eine Region schreiben und aus einer anderen Region lesen, kann die Leseantwort veraltete Daten enthalten, die nicht die Ergebnisse kürzlich abgeschlossener Schreibvorgänge in der anderen Region widerspiegeln.

Konflikte entstehen, wenn Anwendungen dasselbe Element in verschiedenen Regionen fast zum gleichen Zeitpunkt aktualisieren. Um die letztendliche Konsistenz sicherzustellen, verwenden globale DynamoDB-Tabellen den letzter-Verfasser-gewinnt-Abgleich zwischen gleichzeitigen Updates, bei dem DynamoDB sich nach besten Kräften bemüht, den letzten Verfasser zu ermitteln. Mit diesem Konfliktlösungsmechanismus verständigen sich alle Replikate auf das neueste Update und nähern sich einem Zustand an, in dem sie alle über identische Daten verfügen.

Verfügbarkeit und Beständigkeit

Wenn eine einzelne AWS Region isoliert oder beeinträchtigt wird, kann Ihre Anwendung in eine andere Region umleiten und Lese- und Schreibvorgänge für eine andere Replikattabelle ausführen. Sie können benutzerdefinierte Geschäftslogik anwenden, um zu ermitteln, wann Anforderungen an andere Regionen weitergeleitet werden sollen.

Wenn eine Region isoliert oder heruntergestuft wird, verfolgt DynamoDB alle Schreibvorgänge, die zwar durchgeführt, aber noch nicht auf alle Replikattabellen verteilt wurden. Wenn die Region wieder online ist, setzt DynamoDB die Weitergabe aller ausstehenden Schreibvorgänge aus dieser Region an die Replikattabellen in anderen Regionen fort. Außerdem wird das Übertragen von Schreibvorgängen aus anderen Replikattabellen in die Region fortgesetzt, die jetzt wieder online ist. Alle zuvor erfolgreichen Schreibvorgänge werden letztendlich weitergegeben, unabhängig davon, wie lange die Region isoliert ist.

Bewährte Methoden und Anforderungen für die Verwaltung globaler Tabellen

> **Important**
>
> Diese Dokumentation bezieht sich auf globale Tabellen der Version 2017.11.29 (veraltet), die für neue globale Tabellen vermieden werden sollte. Kunden sollten nach Möglichkeit die Version 2019.11.21 (Aktuell) für globale Tabellen verwenden, da sie mehr Flexibilität und höhere Effizienz bietet und weniger Schreibkapazität verbraucht als Version 2017.11.29 (Legacy).
> Informationen dazu, welche Version Sie verwenden, finden Sie unter Ermitteln der Version einer globalen Tabelle. Informationen zur Aktualisierung globaler Tabellen von Version 2017.11.29 (veraltet) auf Version 2019.11.21 (aktuell) finden Sie unter Versionen von globalen DynamoDB-Tabellen.

Mithilfe globaler Amazon DynamoDB-Tabellen können Sie Ihre Tabellendaten regionsübergreifend replizieren. AWS Es ist wichtig, dass die Replikattabellen und sekundären Indexen in Ihrer globalen Tabelle über identische Schreibkapazitätseinstellungen verfügen, um eine ordnungsgemäße Replikation der Daten sicherzustellen.

Themen

- Version der globalen Tabellen
- Anforderungen für das Hinzufügen einer neuen Replikattabelle
- Bewährte Methoden und Anfoderungen zum Verwalten der Kapazität

Version der globalen Tabellen

Es gibt zwei Versionen von globalen DynamoDB-Tabellen: Version 2019.11.21 (Aktuell) und Version 2017.11.29 (Legacy) der globalen Tabellen. Kunden sollten nach Möglichkeit die Version 2019.11.21 (Aktuell) für globale Tabellen verwenden, da sie mehr Flexibilität und höhere Effizienz bietet und weniger Schreibkapazität verbraucht als Version 2017.11.29 (Legacy).

Informationen dazu, welche Version Sie verwenden, finden Sie unter Ermitteln der Version einer globalen Tabelle. Informationen zur Aktualisierung globaler Tabellen von Version 2017.11.29 (veraltet) auf Version 2019.11.21 (aktuell) finden Sie unter Aktualisieren globaler Tabellen.

Anforderungen für das Hinzufügen einer neuen Replikattabelle

Wenn Sie einer globalen Tabelle eine neue Replikattabelle hinzufügen möchten, müssen die folgenden Bedingungen erfüllt sein:

- Die Tabelle muss über den gleichen Partitionsschlüssel wie alle anderen Replikate verfügen.
- Die Tabelle muss über die gleichen Schreibkapazitätsverwaltungseinstellungen verfügen.
- Die Tabelle muss den gleichen Namen wie alle anderen Replikate aufweisen.
- Für die Tabelle muss DynamoDB Streams aktiviert sein und der Stream muss sowohl die neuen als auch die alten Abbilder des Elements enthalten.
- Keine der neuen oder vorhandenen Replikattabellen in der globalen Tabelle darf Daten enthalten.

Wenn globale sekundäre Indxe angegeben werden, müssen auch die folgenden Bedingungen erfüllt sein:

- Die globalen sekundären Indexe müssen denselben Namen haben.
- Die globalen sekundären Indexe müssen über denselben Partitionsschlüssel und denselben Sortierschlüssel verfügen (sofern vorhanden).

> **Important**
>
> Die Einstellungen für die Schreibkapazität sollten konsistent für alle Replikattabellen Ihrer globalen Tabellen und übereinstimmenden sekundären Indexen festgelegt werden. Um die Einstellungen für die Schreibkapazität für Ihre globale Tabelle zu aktualisieren, wird dringend empfohlen, die DynamoDB-Konsole oder die `UpdateGlobalTableSettings`-API-Operation. `UpdateGlobalTableSettings` wendet Änderungen an, um Kapazitätseinstellungen für alle Replikattabellen und übereinstimmende sekundäre Indizes in einer globalen Tabelle automatisch zu schreiben. Wenn Sie die `UpdateTable`, `RegisterScalableTarget`, oder `PutScalingPolicy`-Operationen verwenden, sollten Sie die Änderung auf jede Replikattabelle und den entsprechenden sekundären Index einzeln anwenden. Weitere Informationen finden Sie UpdateGlobalTableSettingsin der Amazon DynamoDB DynamoDB-API-Referenz.
>
> Wir empfehlen Ihnen, Auto Scaling zu aktivieren, um bereitgestellte Schreibkapazitätseinstellungen zu verwalten. Wenn Sie Schreibkapazitätseinstellungen lieber manuell verwalten möchten, sollten Sie allen Replikattabellen gleich replizierte Schreibkapazitätseinheiten bereitstellen. Stellen Sie auch gleiche replizierte

Schreibkapazitätseinheiten für übereinstimmende sekundäre Indizes in Ihrer globalen Tabelle bereit.
Sie müssen außerdem über die entsprechenden AWS Identity and Access Management (IAM-) Berechtigungen verfügen. Weitere Informationen finden Sie unter Verwenden von IAM mit globalen Tabellen.

Bewährte Methoden und Anfoderungen zum Verwalten der Kapazität

Beachten Sie Folgendes, wenn Sie Kapazitätseinstellungen für Replikattabellen in DynamoDB verwalten.

Auto Scaling von DynamoDB

Die Verwendung der automatischen Skalierung von DynamoDB ist die empfohlene Methode zum Verwalten der Durchsatzkapazität für Replikattabellen, die den bereitgestellten Modus verwenden. DynamoDB Auto Scaling passt automatisch die Lesekapazitätseinheiten (RCUs) und Schreibkapazitätseinheiten (WCUs) für jede Replikattabelle an, basierend auf Ihrer tatsächlichen Anwendungsauslastung. Weitere Informationen finden Sie unter Automatische Verwaltung der Durchsatzkapazität mit DynamoDB-Auto-Scaling.

Wenn Sie Ihre Replikattabellen mit dem erstellen AWS-Managementkonsole, ist Auto Scaling standardmäßig für jede Replikattabelle aktiviert, mit standardmäßigen Auto-Scaling-Einstellungen für die Verwaltung von Lesekapazitätseinheiten und Schreibkapazitätseinheiten.

Änderungen an den Einstellungen für die automatische Skalierung für eine Replikattabelle oder einen sekundären Index, die über die DynamoDB Konsole oder mithilfe des `UpdateGlobalTableSettings`-Aufrufs werden automatisch auf alle Replikattabellen und übereinstimmenden sekundären Indexen in der globalen Tabelle angewendet. Diese Änderungen überschreiben alle vorhandenen Einstellungen für die automatische Skalierung. Dadurch wird sichergestellt, dass die bereitgestellten Schreibkapazitätseinstellungen über die Replikattabellen und sekundären Indexe in der globalen Tabelle hinweg konsistent sind. Wenn Sie die Aufrufe `UpdateTable`, `RegisterScalableTarget` oder `PutScalingPolicy` verwenden, sollten Sie die Änderung auf jede Replikattabelle und den entsprechenden sekundären Index einzeln anwenden.

> **Note**
>
> Wenn die automatische Skalierung die Kapazitätsänderungen Ihrer Anwendung nicht erfüllt (unvorhersehbare Workload) oder wenn Sie die Einstellungen (Zieleinstellungen für Minimum,

Maximum oder Auslastungsschwellenwert) nicht konfigurieren möchten, können Sie den Bedarfsmodus verwenden, um die Kapazität für Ihre globalen Tabellen zu verwalten. Weitere Informationen finden Sie unter On-Demand-Modus.
Wenn Sie den On-Demand-Modus für eine globale Tabelle aktivieren, entspricht Ihr Verbrauch an replizierten Schreibanforderungseinheiten (rWCUs) der Art und Weise, wie R WCUs bereitgestellt wird. Wenn Sie beispielsweise 10 Schreibvorgänge in eine lokale Tabelle ausführen, die in zwei zusätzlichen Regionen repliziert wird, verbrauchen Sie 60 Schreibanforderungseinheiten (10 + 10 + 10 = 30; 30 x 2 = 60). Die verbrauchten 60 Schreibanforderungseinheiten enthalten den zusätzlichen Schreibvorgang, der von globalen Tabellen der Version 2017.11.29 (veraltet) verwendet wird, um die Attribute `aws:rep:deleting`, `aws:rep:updatetime` und `aws:rep:updateregion` zu aktualisieren.

Manuelle Verwaltung der Kapazität

Wenn Sie DynamoDB-Auto-Scaling nicht verwenden möchten, müssen Sie die Lese- und Schreibkapazitätseinstellungen für jede Replikattabelle manuell festlegen.

Die bereitgestellten replizierten Schreibkapazitätseinheiten (rWCUs) in jeder Replikattabelle sollten auf die Gesamtzahl von r, die für Anwendungsschreibvorgänge in allen Regionen WCUs benötigt werden, multipliziert mit zwei festgelegt werden. Auf diese Weise werden Anwendungsschreibvorgänge in der lokalen Region sowie replizierte Anwendungsschreibvorgänge aus anderen Regionen berücksichtigt. Angenommen, Sie erwarten 5 Schreibvorgänge pro Sekunde in die Replikattabelle in Ohio und 5 Schreibvorgänge pro Sekunde in die Replikattabelle in N. Virginia. In diesem Fall sollten Sie 20 r WCUs für jede Replikattabelle bereitstellen (5 + 5 = 10; 10 x 2 = 20).

Um die Einstellungen für die Schreibkapazität für Ihre globale Tabelle zu aktualisieren, wird dringend empfohlen, die DynamoDB-Konsole oder die `UpdateGlobalTableSettings`-API-Operation. `UpdateGlobalTableSettings` wendet Änderungen an, um Kapazitätseinstellungen für alle Replikattabellen und übereinstimmende sekundäre Indizes in einer globalen Tabelle automatisch zu schreiben. Wenn Sie die Aufrufe `UpdateTable`, `RegisterScalableTarget` oder `PutScalingPolicy` verwenden, sollten Sie die Änderung auf jede Replikattabelle und den entsprechenden sekundären Index einzeln anwenden. Weitere Informationen finden Sie in der Amazon DynamoDB-API-Referenz.

> **Note**
>
> Zum Aktualisieren der Einstellungen (`UpdateGlobalTableSettings`) für eine globale Tabelle in DynamoDB benötigen Sie die Berechtigungen `dynamodb:UpdateGlobalTable`, `dynamodb:DescribeLimits`, `application-autoscaling:DeleteScalingPolicy` und `application-autoscaling:DeregisterScalableTarget`. Weitere Informationen finden Sie unter Verwenden von IAM mit globalen Tabellen.

Eine globale Tabelle erstellen (Version 2017.11.29)

> **Important**
>
> Diese Dokumentation bezieht sich auf globale Tabellen der Version 2017.11.29 (veraltet), die für neue globale Tabellen vermieden werden sollte. Kunden sollten nach Möglichkeit die Version 2019.11.21 (Aktuell) für globale Tabellen verwenden, da sie mehr Flexibilität und höhere Effizienz bietet und weniger Schreibkapazität verbraucht als Version 2017.11.29 (Legacy).
> Informationen dazu, welche Version Sie verwenden, finden Sie unter Ermitteln der Version einer globalen Tabelle. Informationen zur Aktualisierung globaler Tabellen von Version 2017.11.29 (veraltet) auf Version 2019.11.21 (aktuell) finden Sie unter Versionen von globalen DynamoDB-Tabellen.

In diesem Abschnitt wird beschrieben, wie Sie mit der Amazon DynamoDB DynamoDB-Konsole oder der AWS Command Line Interface ()AWS CLI eine globale Tabelle erstellen.

Themen

- Erstellen einer globalen Tabelle (Konsole)
- Erstellen einer globalen Tabelle (AWS CLI)

Erstellen einer globalen Tabelle (Konsole)

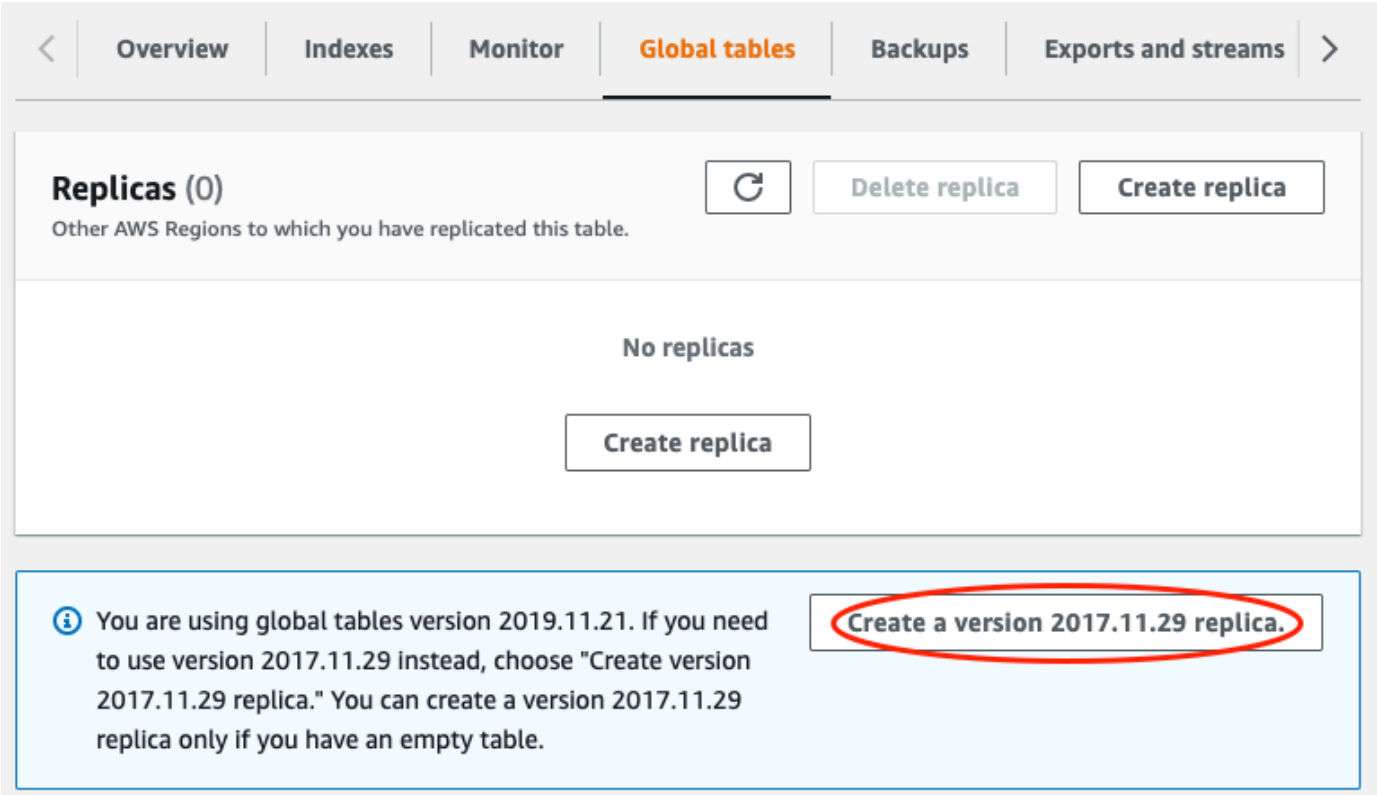
Gehen Sie wie folgt vor, um eine globale Tabelle mit der Konsole zu erstellen. Das folgende Beispiel erstellt eine globale Tabelle mit Replikattabellen in den USA und Europa.

1. Öffnen Sie die DynamoDB-Konsole zu Hausehttps://console.aws.amazon.com/dynamodb/.
   Wählen Sie für dieses Beispiel die Region us-east-2 (USA Ost (Ohio)) aus.
2. Klicken Sie im Navigationsbereich auf der linken Seite der Konsole auf Tables (Tabellen).
3. Wählen Sie Create Table (Tabelle erstellen) aus.

   Geben Sie für Table name (Tabellenname) **`Music`** ein.

   Geben Sie für Primary key (Primärschlüssel) **`Artist`** ein. Wählen Sie Add sort key (Sortierschlüssel hinzufügen) aus und geben Sie **`SongTitle`** ein. (**`Artist`** und **`SongTitle`** sollten jeweils Zeichenfolgen sein.)

   Klicken Sie auf Create (Erstellen), um die Tabelle zu erstellen. Diese Tabelle dient als erste Replikattabelle in einer neuen globalen Tabelle. Sie stellt den Prototyp für andere Replikattabellen dar, die Sie später hinzufügen.
4. Wählen Sie die Registerkarte Globale Tabellen und anschließend Ein Versionsreplikat 2017.11.29 (veraltet) erstellen aus.



5. Wählen Sie im Dropdown Available replication Regions (Verfügbare Replikationsregionen) USA West (Oregon) aus.

Die Konsole stellt anhand einer Prüfung sicher, dass keine Tabelle mit demselben Namen in der ausgewählten Region vorhanden ist. Wenn eine Tabelle mit demselben Namen vorhanden ist, müssen Sie die vorhandene Tabelle löschen, bevor Sie eine neue Replikattabelle in der betreffenden Region erstellen können.

6. Wählen Sie Create Replica (Replikat erstellen) aus. Dies startet den Prozess der Erstellung von Tabellen in USA West (Oregon).

   Die Registerkarte Globale Tabelle für die ausgewählte Tabelle (und für alle anderen Replikattabellen) zeigt, dass die Tabelle in mehreren Regionen repliziert wurde.

7. Fügen Sie im nächsten Schritt eine andere Region hinzu, sodass Ihre globale Tabelle in den USA und Europa repliziert und synchronisiert wird. Wiederholen Sie dazu Schritt 5 und geben Sie EU (Frankfurt) anstelle von USA West (Oregon) an.
8. Sie sollten die weiterhin AWS-Managementkonsole in der Region USA Ost (Ohio) verwenden. Wählen Sie Items (Elemente) im linken Navigationsmenü, wählen Sie die Tabelle Music (Musik) und anschließend Create Item (Element erstellen) aus.

   a. Machen Sie für Artist die Eingabe **`item_1`**.

   b. Geben Sie unter SongTitle den Wert **`Song Value 1`** ein.

   c. Um das Element zu schreiben, wählen SieCreate item (Element erstellen) aus.

9. Nach kurzer Zeit wird das Element in allen drei Regionen Ihrer globalen Tabelle repliziert. Um dies zu verifizieren, klicken Sie in der Konsole auf die Regionsauswahl in der Ecke rechts oben und wählen Sie Europa (Frankfurt) aus. Die Tabelle `Music` in der Region Europa (Frankfurt) enthält ein neues Element.
10. Wiederholen Sie Schritt 9 und wählen Sie USA West (Oregon) aus, um die Replikation in dieser Region zu überprüfen.

Erstellen einer globalen Tabelle (AWS CLI)

Gehen Sie wie folgt vor, um eine globale Tabelle mit dem Namen `Music` mit der AWS CLI zu erstellen. Das folgende Beispiel erstellt eine globale Tabelle mit Replikattabellen in den USA und Europa.

1. Erstellen Sie eine neue Tabelle (`Music`) in USA Ost (Ohio) mit aktivierter DynamoDB Streams (`NEW_AND_OLD_IMAGES`) enthalten.

```
aws dynamodb create-table \
    --table-name Music \
    --attribute-definitions \
        AttributeName=Artist,AttributeType=S \
        AttributeName=SongTitle,AttributeType=S \
    --key-schema \
        AttributeName=Artist,KeyType=HASH \
        AttributeName=SongTitle,KeyType=RANGE \
    --provisioned-throughput \
        ReadCapacityUnits=10,WriteCapacityUnits=5 \
    --stream-specification StreamEnabled=true,StreamViewType=NEW_AND_OLD_IMAGES \
    --region us-east-2
```

2. Erstellen einer identischen `Music`-Tabelle in USA Ost (Nord-Virginia).

```
aws dynamodb create-table \
    --table-name Music \
    --attribute-definitions \
        AttributeName=Artist,AttributeType=S \
        AttributeName=SongTitle,AttributeType=S \
    --key-schema \
        AttributeName=Artist,KeyType=HASH \
        AttributeName=SongTitle,KeyType=RANGE \
    --provisioned-throughput \
        ReadCapacityUnits=10,WriteCapacityUnits=5 \
    --stream-specification StreamEnabled=true,StreamViewType=NEW_AND_OLD_IMAGES \
    --region us-east-1
```

3. Erstellen Sie eine globale Tabelle (`Music`) mit Replikattabellen in den Regionen `us-east-2` und `us-east-1`.

```
aws dynamodb create-global-table \
    --global-table-name Music \
    --replication-group RegionName=us-east-2 RegionName=us-east-1 \
    --region us-east-2
```

> **Note**
>
> Der Name der globalen Tabelle (`Music`) muss mit dem Namen der einzelnen Replikattabellen (`Music`) übereinstimmen. Weitere Informationen finden Sie unter Bewährte Methoden für globale Tabellen.

4. Erstellen Sie eine andere Tabelle in Europa (Irland) mit den gleichen Einstellungen wie in Schritt 1 und Schritt 2:

```
aws dynamodb create-table \
    --table-name Music \
    --attribute-definitions \
        AttributeName=Artist,AttributeType=S \
        AttributeName=SongTitle,AttributeType=S \
    --key-schema \
        AttributeName=Artist,KeyType=HASH \
        AttributeName=SongTitle,KeyType=RANGE \
    --provisioned-throughput \
        ReadCapacityUnits=10,WriteCapacityUnits=5 \
    --stream-specification StreamEnabled=true,StreamViewType=NEW_AND_OLD_IMAGES \
    --region eu-west-1
```

   Fügen Sie nach diesem Schritt die neue Tabelle zur `Music` globalen Tabelle.

```
aws dynamodb update-global-table \
    --global-table-name Music \
    --replica-updates 'Create={RegionName=eu-west-1}' \
    --region us-east-2
```

5. Fügen Sie der Tabelle `Music` in der Region USA Ost (Ohio) ein neues Element hinzu, um zu überprüfen, ob die Replikation funktioniert.

```
aws dynamodb put-item \
    --table-name Music \
    --item '{"Artist": {"S":"item_1"},"SongTitle": {"S":"Song Value 1"}}' \
    --region us-east-2
```

6. Warten Sie einige Sekunden und überprüfen Sie, ob das Element erfolgreich in den Regionen USA Ost (Nord-Virginia) und Europa (Irland) repliziert wurde.

```
aws dynamodb get-item \
    --table-name Music \
    --key '{"Artist": {"S":"item_1"},"SongTitle": {"S":"Song Value 1"}}' \
    --region us-east-1
```

```
aws dynamodb get-item \
    --table-name Music \
    --key '{"Artist": {"S":"item_1"},"SongTitle": {"S":"Song Value 1"}}' \
    --region eu-west-1
```

Überwachen globaler Tabellen

> **Important**
>
> Diese Dokumentation bezieht sich auf globale Tabellen der Version 2017.11.29 (veraltet), die für neue globale Tabellen vermieden werden sollte. Kunden sollten nach Möglichkeit die Version 2019.11.21 (Aktuell) für globale Tabellen verwenden, da sie mehr Flexibilität und höhere Effizienz bietet und weniger Schreibkapazität verbraucht als Version 2017.11.29 (Legacy).
> Informationen dazu, welche Version Sie verwenden, finden Sie unter Ermitteln der Version einer globalen Tabelle. Informationen zur Aktualisierung globaler Tabellen von Version 2017.11.29 (veraltet) auf Version 2019.11.21 (aktuell) finden Sie unter Versionen von globalen DynamoDB-Tabellen.

Sie können Amazon CloudWatch verwenden, um das Verhalten und die Leistung einer globalen Tabelle zu überwachen. Amazon DynamoDB veröffentlicht `ReplicationLatency` und `PendingReplicationCount`-Metriken für jedes Replikat in der globalen Tabelle.

- **`ReplicationLatency`** – Die verstrichene Zeit zwischen dem Erscheinen eines aktualisierten Elements im DynamoDB-Stream für eine Replikattabelle und dem Erscheinen dieses Elements in einem anderen Replikat in der globalen Tabelle. `ReplicationLatency` wird in Millisekunden ausgedrückt und für jedes Quell- und Zielregionspaar ausgegeben.

  Im Normalbetrieb sollte `ReplicationLatency` relativ konstant sein. Ein erhöhter Wert für `ReplicationLatency` könnte darauf hinweisen, dass Updates von einem Replikat nicht in einem

angemessenen Zeitraum an andere Replikattabellen verteilt werden. Dies kann im Lauf der Zeit dazu führen, dass andere Replikattabellen zurückfallen, da sie Updates nicht mehr konsistent erhalten. In diesem Fall müssen Sie überprüfen, dass die Lese- und Schreibkapazitätseinheiten für jede Replikattabelle identisch sind. Außerdem müssen Sie bei Auswahl der Einstellungen für die Schreibkapazitätseinheiten (WCU) die Empfehlungen in Version der globalen Tabellen beachten.

`ReplicationLatency` kann steigen, wenn eine AWS-Region veraltet ist und Sie über eine Replikattabelle in dieser Region verfügen. In diesem Fall können Sie die Lese- und Schreibaktivitäten Ihrer Anwendung vorübergehend an eine andere AWS-Region weiterleiten.

- **`PendingReplicationCount`** – Die Anzahl der Elementaktualisierungen, die in eine Replikattabelle geschrieben werden, aber noch nicht in ein anderes Replikat in der globalen Tabelle geschrieben wurden. `PendingReplicationCount` wird in der Anzahl der Elemente ausgedrückt und für jedes Quell- und Zielregionspaar ausgegeben.

  Im Normalbetrieb sollte `PendingReplicationCount` sehr niedrig sein. Wenn `PendingReplicationCount` für einen längeren Zeitraum steigt, sollten Sie prüfen, ob die Einstellungen für die bereitgestellte Schreibkapazität Ihrer Replikattabellen für Ihre aktuelle Workload ausreichen.

  `PendingReplicationCount` kann steigen, wenn eine AWS-Region veraltet ist und Sie über eine Replikattabelle in dieser Region verfügen. In diesem Fall können Sie die Lese- und Schreibaktivitäten Ihrer Anwendung vorübergehend an eine andere AWS-Region weiterleiten.

Weitere Informationen finden Sie unter DynamoDB-Metriken und -Dimensionen.

Verwenden von IAM mit globalen Tabellen

> **Important**
>
> Diese Dokumentation bezieht sich auf globale Tabellen der Version 2017.11.29 (veraltet), die für neue globale Tabellen vermieden werden sollte. Kunden sollten nach Möglichkeit die Version 2019.11.21 (Aktuell) für globale Tabellen verwenden, da sie mehr Flexibilität und höhere Effizienz bietet und weniger Schreibkapazität verbraucht als Version 2017.11.29 (Legacy).
> Informationen dazu, welche Version Sie verwenden, finden Sie unter Ermitteln der Version einer globalen Tabelle. Informationen zur Aktualisierung globaler Tabellen von

Version 2017.11.29 (veraltet) auf Version 2019.11.21 (aktuell) finden Sie unter Versionen von globalen DynamoDB-Tabellen.

Beim ersten Erstellen einer globalen Tabelle generiert Amazon DynamoDB automatisch eine mit dem AWS Identity and Access Management -(IAM)-Service verknüpfte Rolle für Sie. Diese Rolle trägt den Namen und ermöglicht DynamoDB, die regionsübergreifende Replikation für globale Tabellen in Ihrem Namen zu verwalten. Löschen Sie diese serviceverknüpfte Rolle nicht. Andernfalls funktionieren alle globalen Tabellen nicht mehr.

Weitere Informationen zu serviceverknüpften Rollen finden Sie unter Verwenden serviceverknüpfter Rollen im IAM-Benutzerhandbuch.

Zum Erstellen und Verwalten globaler Tabellen in DynamoDB müssen Sie über die `dynamodb:CreateGlobalTable`-Berechtigung für den Zugriff auf die folgenden Komponenten verfügen:

- Die Replikattabelle, die Sie hinzufügen möchten.
- Jedes vorhandene Replikat, das bereits Teil der globalen Tabelle ist.
- Die globale Tabelle selbst.

Zum Aktualisieren der Einstellungen (`UpdateGlobalTableSettings`) für eine globale Tabelle in DynamoDB benötigen Sie die Berechtigungen `dynamodb:UpdateGlobalTable`, `dynamodb:DescribeLimits`, `application-autoscaling:DeleteScalingPolicy` und `application-autoscaling:DeregisterScalableTarget`.

Die `application-autoscaling:DeleteScalingPolicy` und `application-autoscaling:DeregisterScalableTarget`-Berechtigungen sind erforderlich, wenn eine vorhandene Skalierungsrichtlinie aktualisiert wird. Dadurch kann der globale Tabellendienst die alte Skalierungsrichtlinie entfernen, bevor die neue Richtlinie an die Tabelle oder den sekundären Index anfügt.

Wenn Sie eine IAM-Richtlinie verwenden, um den Zugriff auf eine Replikattabelle zu verwalten, sollten Sie eine identische Richtlinie auf alle anderen Replikate innerhalb dieser globalen Tabelle anwenden. Auf diese Weise wird ein einheitliches Berechtigungsmodell für alle Replikattabellen aufrechterhalten.

Durch die Verwendung identischer IAM-Richtlinien für alle Replikate in einer globalen Tabelle können Sie auch vermeiden, unbeabsichtigten Lese- und Schreibzugriff auf Ihre globalen Tabellendaten zu gewähren. Nehmen wir als Beispiel einen Benutzer, der nur Zugriff auf ein Replikat in einer globalen Tabelle hat. Wenn dieser Benutzer in dieses Replikat schreiben kann, übertragt DynamoDB den Schreibvorgang auf alle anderen Replikattabellen. Der Benutzer kann so (indirekt) in alle anderen Replikate der globalen Tabelle schreiben. Dieses Szenario kann mit konsistenten IAM-Richtlinien für alle Replikattabellen vermieden werden.

Beispiel: CreateGlobalTable Aktion zulassen

Bevor Sie einer globalen Tabelle ein Replikat hinzufügen können, müssen Sie über die `dynamodb:CreateGlobalTable`-Berechtigung für die globale Tabelle und für alle zugehörigen Replikattabellen verfügen.

Mit der folgenden IAM-Richtlinie werden die Berechtigungen zum Zulassen der Aktion `CreateGlobalTable` für alle Tabellen erteilt.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": ["dynamodb:CreateGlobalTable"],
            "Resource": "*"
        }
    ]
}
```

Beispiel: Erlauben Sie die UpdateGlobalTable Aktionen, DescribeLimits, application-autoscaling: DeleteScalingPolicy und application-autoscaling: DeregisterScalableTarget

Zum Aktualisieren der Einstellungen (`UpdateGlobalTableSettings`) für eine globale Tabelle in DynamoDB benötigen Sie die Berechtigungen `dynamodb:UpdateGlobalTable`, `dynamodb:DescribeLimits`, `application-autoscaling:DeleteScalingPolicy` und `application-autoscaling:DeregisterScalableTarget`.

Mit der folgenden IAM-Richtlinie werden die Berechtigungen zum Zulassen der Aktion `UpdateGlobalTableSettings` für alle Tabellen erteilt.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "dynamodb:UpdateGlobalTable",
                "dynamodb:DescribeLimits",
                "application-autoscaling:DeleteScalingPolicy",
                "application-autoscaling:DeregisterScalableTarget"
            ],
            "Resource": "*"
        }
    ]
}
```

Beispiel: Erlauben Sie die CreateGlobalTable Aktion für einen bestimmten globalen Tabellennamen, wobei Replikate nur in bestimmten Regionen zulässig sind

Die folgende IAM-Richtlinie erteilt Berechtigungen, damit die `CreateGlobalTable`-Aktion eine globale Tabelle namens `Customers` mit Replikaten in zwei Regionen erstellen kann.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "dynamodb:CreateGlobalTable",
            "Resource": [
                "arn:aws:dynamodb::123456789012:global-table/Customers",
                "arn:aws:dynamodb:us-east-1:123456789012:table/Customers",
                "arn:aws:dynamodb:us-west-1:123456789012:table/Customers"
            ]
```

```
        }
    ]
}
```

Vorherige Version der Low-Level-API für DynamoDB (2011-12-05)

In diesem Abschnitt werden die Operationen dokumentiert, die in der vorherigen Version der DynamoDB-Low-Level-API (2011-12-05) verfügbar sind. Diese Version der Low-Level-API wird für die Abwärtskompatibilität mit vorhandenen Anwendungen beibehalten.

Neue Anwendungen sollten die aktuelle API-Version (2012-08-10) verwenden. Weitere Informationen finden Sie unter Referenz zur DynamoDB-API.

> **Note**
>
> Wir empfehlen, dass Sie Ihre vorhandenen Anwendungen auf die aktuelle API-Version (2012-08-10) migrieren, da neue DynamoDB-Funktionen nicht auf die vorherige API-Version rückportiert werden.

Themen

- BatchGetItem
- BatchWriteItem
- CreateTable
- DeleteItem
- DeleteTable
- DescribeTables
- GetItem
- ListTables
- PutItem
- Abfrage
- Scan
- UpdateItem
- UpdateTable

BatchGetItem

> **Important**
>
> ***Dieser Abschnitt bezieht sich auf die veraltete API-Version 2011-12-05, die nicht für neue Anwendungen verwendet werden sollte.***
> Eine Dokumentation zur aktuellen Low-Level-API finden Sie in der Amazon DynamoDB-API-Referenz.

Beschreibung

Die `BatchGetItem`-Operation gibt die Attribute für mehrere Elemente aus verschiedenen Tabellen unter Verwendung ihrer Primärschlüssel zurück. Die maximale Anzahl von Elementen, die für eine einzelne Operation abgerufen werden können, ist 100. Außerdem wird die Anzahl der abgerufenen Elemente durch eine Größenbeschränkung von 1 MB begrenzt. Wenn die Größenbeschränkung der Antwort überschritten wird oder ein Teilergebnis aufgrund einer Überschreitung des bereitgestellten Durchsatzes der Tabelle oder aufgrund eines internen Verarbeitungsfehlers zurückgegeben wird, gibt DynamoDB einen `UnprocessedKeys`-Wert zurück, sodass Sie die Operation mit den nächsten abzurufenden Element wiederholen können. DynamoDB passt die Anzahl von Elementen, die pro Seite zurückgegeben werden, automatisch an, um diese Größenbeschränkung zu erzwingen. Beispiel: Wenn Sie 100 Elemente abrufen möchten, jedes Element aber 50 KB umfasst, gibt das System nur 20 Elemente und einen entsprechenden `UnprocessedKeys`-Wert zurück, damit Sie die nächste Ergebnisseite abrufen können. Wenn Sie möchten, kann Ihre Anwendung eine eigene Logik für die Zusammenstellung der Ergebnisseiten in einem Satz einbeziehen.

Wenn aufgrund eines unzureichenden, bereitgestellten Durchsatzes für die von der Anforderung betroffenen Tabellen keine Elemente verarbeitet werden konnten, gibt DynamoDB einen `ProvisionedThroughputExceededException`-Fehler zurück.

> **Note**
>
> Standardmäßig führt `BatchGetItem` „Eventually Consistent“-Lesevorgänge für jede von der Anforderung betroffenen Tabelle aus. Sie können den Parameter `ConsistentRead` für einzelne Tabellen auf `true` einstellen, wenn stattdessen Consistent-Lesevorgänge ausgeführt werden sollen.
> `BatchGetItem` ruft Elemente parallel ab, um Antwortlatenzen zu minimieren.

Bedenken Sie bei der Entwicklung Ihrer Anwendung, dass DynamoDB nicht sicherstellt, in welcher Reihenfolge die Attribute in der Antwort zurückgegeben werden. Schließen Sie die Primärschlüsselwerte in `AttributesToGet` für die Elemente in Ihrer Anforderung ein, um die Antwort nach Element zu analysieren.
Wenn die angeforderten Elemente nicht vorhanden sind, wird in der Antwort für diese Elemente nichts zurückgegeben. Anforderungen für nicht vorhandene Elemente verbrauchen je nach Typ des Lesevorgangs die Mindestlesekapazitätseinheiten. Weitere Informationen finden Sie unter Elementgrößen und -formate in DynamoDB.

Anforderungen

Syntax

```
// This header is abbreviated. For a sample of a complete header, see DynamoDB Low-Level-API.
POST / HTTP/1.1
x-amz-target: DynamoDB_20111205.BatchGetItem
content-type: application/x-amz-json-1.0

{"RequestItems":
    {"Table1":
        {"Keys":
            [{"HashKeyElement": {"S":"KeyValue1"}, "RangeKeyElement":
{"N":"KeyValue2"}},
            {"HashKeyElement": {"S":"KeyValue3"}, "RangeKeyElement":{"N":"KeyValue4"}},
            {"HashKeyElement": {"S":"KeyValue5"}, "RangeKeyElement":
{"N":"KeyValue6"}}],
        "AttributesToGet":["AttributeName1", "AttributeName2", "AttributeName3"]},
    "Table2":
        {"Keys":
            [{"HashKeyElement": {"S":"KeyValue4"}},
            {"HashKeyElement": {"S":"KeyValue5"}}],
        "AttributesToGet": ["AttributeName4", "AttributeName5", "AttributeName6"]
        }
    }
}
```

Name	Beschreibung	Erforderlich
`RequestItems`	Ein Container des Tabellennamens und die entsprechenden Elemente, die nach Primärschlüssel abgerufen werden sollen. Beim Anfordern von Elementen kann jeder Tabellenname nur einmal pro Operation aufgerufen werden. Typ: Zeichenfolge Standard: keiner	Ja
`Table`	Der Name der Tabelle, die die abzurufenden Elemente enthält. Der Eintrag ist einfach eine Zeichenfolge, die eine vorhandene Tabelle ohne Bezeichnung angibt. Typ: Zeichenfolge Standard: keiner	Ja
`Table:Keys`	Die Primärschlüsselwerte, die die Elemente in der angegebenen Tabelle definieren. Weitere Informationen zu Primärschlüsseln finden Sie unter Primärschlüssel. Typ: Schlüssel	Ja
`Table:AttributesToGet`	Array von Attributnamen in der angegebenen Tabelle. Wenn Attributnamen nicht	Nein

Name	Beschreibung	Erforderlich
	angegeben sind, dann werden alle Attribute zurückgegeben. Wenn einige Attribute nicht gefunden werden, sind sie nicht im Abfrageergebnis enthalten. Typ: Array	
`Table:ConsistentRead`	Wenn auf `true` festgeleg t, dann wird ein Consistent-Lesevorgang ausgegeben, andernfalls wird Eventually Consistent verwendet. Typ: Boolesch	Nein

Antworten

Syntax

```
HTTP/1.1 200
x-amzn-RequestId: 8966d095-71e9-11e0-a498-71d736f27375
content-type: application/x-amz-json-1.0
content-length: 855

{"Responses":
    {"Table1":
        {"Items":
        [{"AttributeName1": {"S":"AttributeValue"},
        "AttributeName2": {"N":"AttributeValue"},
        "AttributeName3": {"SS":["AttributeValue", "AttributeValue", "AttributeValue"]}
        },
        {"AttributeName1": {"S": "AttributeValue"},
        "AttributeName2": {"S": "AttributeValue"},
        "AttributeName3": {"NS": ["AttributeValue", "AttributeValue",
 "AttributeValue"]}
        }],
```

```
    "ConsumedCapacityUnits":1},
    "Table2":
        {"Items":
        [{"AttributeName1": {"S":"AttributeValue"},
        "AttributeName2": {"N":"AttributeValue"},
        "AttributeName3": {"SS":["AttributeValue", "AttributeValue", "AttributeValue"]}
        },
        {"AttributeName1": {"S": "AttributeValue"},
        "AttributeName2": {"S": "AttributeValue"},
        "AttributeName3": {"NS": ["AttributeValue", "AttributeValue","AttributeValue"]}
        }],
    "ConsumedCapacityUnits":1}
    },
    "UnprocessedKeys":
        {"Table3":
        {"Keys":
            [{"HashKeyElement": {"S":"KeyValue1"}, "RangeKeyElement":
{"N":"KeyValue2"}},
            {"HashKeyElement": {"S":"KeyValue3"}, "RangeKeyElement":{"N":"KeyValue4"}},
            {"HashKeyElement": {"S":"KeyValue5"}, "RangeKeyElement":
{"N":"KeyValue6"}}],
        "AttributesToGet":["AttributeName1", "AttributeName2", "AttributeName3"]}
        }
}
```

Name	Beschreibung
`Responses`	Tabellennamen und die entsprechenden Elementattribute aus den Tabellen. Typ: Zuordnung
`Table`	Der Name der Tabelle, die die Elemente enthält. Der Eintrag ist einfach eine Zeichenfolge, die die Tabelle ohne Bezeichnung angibt. Typ: Zeichenfolge
`Items`	Container für die Attributnamen und Werte, die mit den Operationsparametern übereinstimmen.

Name	Beschreibung
	Typ: Zuordnung der Attributnamen und ihrer Datentypen und Werte.
`ConsumedCapacityUnits`	Die Anzahl der Lesekapazitätseinheiten, die für jede Tabelle verbraucht werden. Dieser Wert zeigt die Anzahl, die für Ihren bereitgestellten Durchsatz gültig ist. Anforderungen für nicht vorhandene Elemente verbrauchen je nach Typ des Lesevorgangs die Mindestlesekapazitätseinheiten. Weitere Informationen finden Sie unter DynamoDB – Modus mit bereitgestellter Kapazität. Typ: Zahl
`UnprocessedKeys`	Enthält ein Array von Tabellen mit den entsprechenden Schlüsseln, die mit der aktuellen Antwort nicht verarbeitet wurden, da das Limit der Antwortgröße möglicherweise erreicht wurde. Der `UnprocessedKeys` -Wert hat das gleiche Format wie ein `RequestItems` -Parameter (sodass der Wert einer folgenden `BatchGetItem` -Operation direkt bereitgestellt werden kann). Weitere Informationen finden Sie unter dem obigen `RequestItems` -Parameter. Typ: Array
`UnprocessedKeys` : `Table`: `Keys`	Die Attributwerte des Primärschlüssels, die die Elemente und die den Elementen zugeordneten Attributen definieren. Weitere Informationen zu Primärschlüsseln finden Sie unter Primärschlüssel. Typ: Array von Attribut-Namen-Wert-Paare.

Name	Beschreibung
`UnprocessedKeys` : `Table`: `AttributesToGet`	Attributnamen in der angegebenen Tabelle. Wenn Attributnamen nicht angegeben sind, dann werden alle Attribute zurückgegeben. Wenn einige Attribute nicht gefunden werden, sind sie nicht im Abfrageergebnis enthalten. Typ: Array von Attributnamen.
`UnprocessedKeys` : `Table`: `ConsistentRead`	Wenn dieser Wert auf `true` festgelegt ist, wird ein Consistent-Lesevorgang für die angegebene Tabelle verwendet, andernfalls ein Eventually Consistent-Lesevorgang. Typ: boolescher Wert.

Spezielle Fehler

Fehler	Beschreibung
`ProvisionedThroughputExceededException`	Der maximal zulässige bereitgestellte Durchsatz wurde überschritten.

Beispiele

Die folgenden Beispiele zeigen eine HTTP-POST-Anforderung und Antwort mithilfe der BatchGetItem-Operation. Beispiele für die Verwendung des AWS SDK finden Sie unter Arbeiten mit Elementen und Attributen in DynamoDB.

Beispielanforderung

Mit dem folgenden Beispiel werden Attribute aus zwei verschiedenen Tabellen angefordert.

```
// This header is abbreviated.
// For a sample of a complete header, see DynamoDB Low-Level-API.
POST / HTTP/1.1
```

```
x-amz-target: DynamoDB_20111205.BatchGetItem
content-type: application/x-amz-json-1.0
content-length: 409

{"RequestItems":
    {"comp1":
        {"Keys":
            [{"HashKeyElement":{"S":"Casey"},"RangeKeyElement":{"N":"1319509152"}},
            {"HashKeyElement":{"S":"Dave"},"RangeKeyElement":{"N":"1319509155"}},
            {"HashKeyElement":{"S":"Riley"},"RangeKeyElement":{"N":"1319509158"}}],
        "AttributesToGet":["user","status"]},
    "comp2":
        {"Keys":
            [{"HashKeyElement":{"S":"Julie"}},{"HashKeyElement":{"S":"Mingus"}}],
        "AttributesToGet":["user","friends"]}
    }
}
```

Beispielantwort

Das folgende Beispiel zeigt die Antwort.

```
HTTP/1.1 200 OK
x-amzn-RequestId: GTPQVRM4VJS792J1UFJTKUBVV4KQNSO5AEMVJF66Q9ASUAAJG
content-type: application/x-amz-json-1.0
content-length: 373
Date: Fri, 02 Sep 2011 23:07:39 GMT

{"Responses":
    {"comp1":
        {"Items":
            [{"status":{"S":"online"},"user":{"S":"Casey"}},
            {"status":{"S":"working"},"user":{"S":"Riley"}},
            {"status":{"S":"running"},"user":{"S":"Dave"}}],
        "ConsumedCapacityUnits":1.5},
    "comp2":
        {"Items":
            [{"friends":{"SS":["Elisabeth", "Peter"]},"user":{"S":"Mingus"}},
            {"friends":{"SS":["Dave", "Peter"]},"user":{"S":"Julie"}}],
        "ConsumedCapacityUnits":1}
    },
    "UnprocessedKeys":{}
}
```

BatchWriteItem

> **Important**
>
> ***Dieser Abschnitt bezieht sich auf die veraltete API-Version 2011-12-05, die nicht für neue Anwendungen verwendet werden sollte.***
> Eine Dokumentation zur aktuellen Low-Level-API finden Sie in der Amazon DynamoDB-API-Referenz.

Beschreibung

Diese Operation ermöglicht Ihnen, anhand eines einzigen Aufrufs, das Ablegen und Löschen mehrerer Elemente in mehreren Tabellen.

Für den Upload eines Elements können Sie `PutItem` und für das Löschen eines Elements `DeleteItem` verwenden. Wenn Sie allerdings große Datenmengen hochladen oder löschen möchten, wie z. B. das Hochladen großer Datenmengen von Amazon EMR (Amazon EMR) oder das Migrieren von Daten aus einer anderen Datenbank zu DynamoDB, bietet `BatchWriteItem` eine effiziente Alternative.

Wenn Sie Sprachen wie Java verwenden, können Sie Threads nutzen, um Elemente parallel hochzuladen. Dadurch wird Ihrer Anwendung Komplexität zum Behandeln von Threads hinzugefügt. Andere Sprachen unterstützen Threading nicht. Wenn Sie beispielsweise PHP verwenden, müssen Sie Elemente nacheinander hochladen oder löschen. In beiden Fällen bietet `BatchWriteItem` eine Alternative, in der die angegebenen Ablege- und Löschoperationen parallel verarbeitet werden. Somit erhalten Sie die Leistung des Thread-Pool-Ansatzes, ohne der Anwendung Komplexität hinzufügen zu müssen.

Beachten Sie, dass jeder einzelne Ablege- und Löschvorgang, der in einer `BatchWriteItem`-Operation angegeben wird, dasselbe in Bezug auf verbrauchte Kapazitätseinheiten kostet. Da `BatchWriteItem` die angegebenen Operationen jedoch parallel durchführt, erhalten Sie eine niedrigere Latenz. Delete-Operationen für nicht vorhandene Elemente verbrauchen eine Schreibkapazitätseinheit. Weitere Informationen zu verbrauchten Kapazitätseinheiten finden Sie unter Arbeiten mit Tabellen und Daten in DynamoDB.

Wenn Sie `BatchWriteItem` verwenden, beachten Sie die folgenden Einschränkungen:

- Maximale Operationen in einer einzigen Anforderung – Sie können insgesamt bis zu 25 Ablege- oder Löschoperationen angeben. Jedoch darf die Gesamtgröße der Anforderung 1 MB (die HTTP-Nutzlast) nicht überschreiten.
- Sie können die `BatchWriteItem`-Operation ausschließlich zum Ablegen und Löschen von Elemente nutzen. Sie können sie nicht verwenden, um vorhandene Elemente zu aktualisieren.
- Nicht-atomare Operation – Einzeloperationen, die in einem `BatchWriteItem` angegeben werden, sind atomar. Jedoch ist `BatchWriteItem` als Ganzes eine Operation "nach besten Kräften" und keine atomare Operation. Das bedeutet, dass einige Operationen in einer `BatchWriteItem` Anforderung möglicherweise erfolgreich sind und andere fehlschlagen könnten. Die fehlgeschlagenen Operationen werden in einem `UnprocessedItems`-Feld in der Antwort zurückgegeben. Einige dieser Ausfälle sind möglicherweise durch eine Überschreitung des für die Tabelle konfigurierten bereitgestellten Durchsatzes oder durch einen vorübergehenden Fehler, wie beispielsweise einen Netzwerkfehler, aufgetreten. Sie können die Anforderungen prüfen und optional erneut senden. In der Regel rufen Sie `BatchWriteItem` in einer Schleife auf. Dann prüfen Sie auf unverarbeitete Elemente in jeder Iteration und übermitteln eine neue `BatchWriteItem`-Anforderung mit diesen unverarbeiteten Elementen.
- Gibt keine Elemente zurück – Das `BatchWriteItem` ist für das effiziente Hochladen großer Datenmengen entwickelt worden. Es liefert einige Raffinessen nicht, die von `PutItem` und `DeleteItem` angeboten werden. Beispielsweise unterstützt `DeleteItem` das `ReturnValues`-Feld in Ihrem Anforderungstext, um das gelöschte Element in der Antwort anzufordern. Die `BatchWriteItem`-Operation gibt keine Elemente in der Antwort zurück.
- Im Gegensatz zu `PutItem` und `DeleteItem`, erlaubt `BatchWriteItem` Ihnen nicht, Bedingungen für einzelne Schreibanforderungen in der Operation anzugeben.
- Attributwerte dürfen nicht Null sein; Zeichenfolge- und Binärtypattribute müssen Längen haben, die größer als Null sind und festgelegte Typenattribute dürfen nicht leer sein. Anforderungen mit leeren Werten werden mit einem `ValidationException` abgelehnt.

DynamoDB lehnt den gesamten Batchschreibvorgang ab, wenn eine der folgenden Bedingungen erfüllt ist:

- Wenn eine oder mehrere Tabellen, die in der `BatchWriteItem`-Anforderung angegeben wurden, nicht vorhanden sind.
- Wenn Primärschlüsselattribute, die für ein Element in der Anforderung angegeben wurden, nicht mit dem entsprechenden Schlüsselschema der Tabelle übereinstimmen.

- Wenn Sie versuchen, mehrere Operationen für das gleiche Element in der gleichen `BatchWriteItem`-Anforderung durchzuführen. Sie können beispielsweise das gleiche Element in derselben `BatchWriteItem`-Anforderung nicht ablegen oder löschen.
- Wenn die Gesamtgröße der Anforderung die 1 MB Anforderungsgrößenbeschränkung (die HTTP-Nutzlast) überschreitet.
- Wenn ein einzelnes Element in einem Stapel die 64 KB Elementgrößenbeschränkung überschreitet.

Anforderungen

Syntax

```
// This header is abbreviated. For a sample of a complete header, see DynamoDB Low-Level-API.
POST / HTTP/1.1
x-amz-target: DynamoDB_20111205.BatchGetItem
content-type: application/x-amz-json-1.0

{
  "RequestItems" :  RequestItems
}

RequestItems
{
     "TableName1" :  [ Request, Request, ... ],
     "TableName2" :  [ Request, Request, ... ],
     ...
}

Request ::=
  PutRequest | DeleteRequest

PutRequest ::=
{
  "PutRequest" : {
     "Item" : {
        "Attribute-Name1" : Attribute-Value,
        "Attribute-Name2" : Attribute-Value,
         ...
     }
  }
```

```
}

DeleteRequest ::=
{
   "DeleteRequest" : {
      "Key" : PrimaryKey-Value
   }
}


PrimaryKey-Value ::= HashTypePK | HashAndRangeTypePK

HashTypePK ::=
{
   "HashKeyElement" : Attribute-Value
}

HashAndRangeTypePK
{
   "HashKeyElement" : Attribute-Value,
   "RangeKeyElement" : Attribute-Value,
}

Attribute-Value ::= String | Numeric| Binary | StringSet | NumericSet | BinarySet

Numeric ::=
{
   "N": "Number"
}

String ::=
{
   "S": "String"
}

Binary ::=
{
    "B": "Base64 encoded binary data"
}

StringSet ::=
{
   "SS": [ "String1", "String2", ... ]
}
```

```
NumberSet ::=
{
   "NS": [ "Number1", "Number2", ... ]
}

BinarySet ::=
{
   "BS": [ "Binary1", "Binary2", ... ]
}
```

Im Anforderungstext beschreibt das JSON-Objekt `RequestItems` die Operationen, die Sie durchführen möchten. Die Operationen werden nach Tabellen gruppiert. Sie können mehrere Elemente aus mehreren Tabellen mit `BatchWriteItem` aktualisieren oder löschen. Für jede bestimmte Schreibanforderung müssen Sie den Anforderungstyp (`PutItem`, `DeleteItem`), gefolgt von Detailinformationen über die Operation identifizieren.

- Für eine `PutRequest` stellen Sie das Element bereit, d. h. eine Liste der Attribute und deren Werte.
- Legen Sie für eine `DeleteRequest` den Primärschlüsselnamen und Wert fest.

Antworten

Syntax

Folgendes ist die Syntax des JSON-Anforderungstextes, die in der Antwort zurückgegeben wurde.

```
{
  "Responses" :         ConsumedCapacityUnitsByTable
  "UnprocessedItems" :  RequestItems
}

ConsumedCapacityUnitsByTable
{
    "TableName1" : { "ConsumedCapacityUnits", : NumericValue },
    "TableName2" : { "ConsumedCapacityUnits", : NumericValue },
     ...
}

RequestItems
This syntax is identical to the one described in the JSON syntax in the request.
```

Spezielle Fehler

Keine nur für diese Operation spezifischen Fehler.

Beispiele

Das folgende Beispiel zeigt eine HTTP-POST-Anforderung und die Antwort einer `BatchWriteItem`-Operation. Die Anforderung gibt die folgenden Operationen in der Reply- und der Thread-Tabelle an:

- Einfügen und Löschen eines Elements aus der Reply-Tabelle
- Ablegen eines Element in die Thread-Tabelle

Beispiele für die Verwendung des AWS SDK finden Sie unter Arbeiten mit Elementen und Attributen in DynamoDB.

Beispielanforderung

```
// This header is abbreviated. For a sample of a complete header, see DynamoDB Low-Level-API.
POST / HTTP/1.1
x-amz-target: DynamoDB_20111205.BatchGetItem
content-type: application/x-amz-json-1.0

{
  "RequestItems":{
    "Reply":[
      {
        "PutRequest":{
          "Item":{
            "ReplyDateTime":{
              "S":"2012-04-03T11:04:47.034Z"
            },
            "Id":{
              "S":"DynamoDB#DynamoDB Thread 5"
            }
          }
        }
      },
      {
        "DeleteRequest":{
          "Key":{
            "HashKeyElement":{
```

```
              "S":"DynamoDB#DynamoDB Thread 4"
            },
            "RangeKeyElement":{
              "S":"oops - accidental row"
            }
          }
        }
      }
    ],
    "Thread":[
      {
        "PutRequest":{
          "Item":{
            "ForumName":{
              "S":"DynamoDB"
            },
            "Subject":{
              "S":"DynamoDB Thread 5"
            }
          }
        }
      }
    ]
  }
}
```

Beispielantwort

Das folgende Antwortbeispiel zeigt, dass eine Ablegeoperation für die Thread- und Reply-Tabelle erfolgreich war und eine Delete-Operation für die Reply-Tabelle fehlgeschlagen ist (aus Gründen wie einer Drosselung, die verursacht wird, wenn Sie den bereitgestellten Durchsatz für die Tabelle überschreiten). In der JSON-Antwort ist Folgendes zu beachten:

- Das `Responses`-Objekt zeigt, dass eine Kapazitätseinheit sowohl für die `Thread`- als auch für die `Reply`-Tabelle als Ergebnis der erfolgreichen Ablegeoperation für jede dieser Tabellen verbraucht wurde.
- Das `UnprocessedItems`-Objekt zeigt die erfolglose Delete-Operation für die `Reply`-Tabelle. Sie können dann einen neuen `BatchWriteItem`-Aufruf ausführen, um diese unverarbeiteten Anforderungen anzugehen.

```
HTTP/1.1 200 OK
```

```
x-amzn-RequestId: G8M9ANLOE5QA26AEUHJKJE0ASBVV4KQNSO5AEMVJF66Q9ASUAAJG
Content-Type: application/x-amz-json-1.0
Content-Length: 536
Date: Thu, 05 Apr 2012 18:22:09 GMT

{
   "Responses":{
      "Thread":{
         "ConsumedCapacityUnits":1.0
      },
      "Reply":{
         "ConsumedCapacityUnits":1.0
      }
   },
   "UnprocessedItems":{
      "Reply":[
         {
            "DeleteRequest":{
               "Key":{
                  "HashKeyElement":{
                     "S":"DynamoDB#DynamoDB Thread 4"
                  },
                  "RangeKeyElement":{
                     "S":"oops - accidental row"
                  }
               }
            }
         }
      ]
   }
}
```

CreateTable

> **Important**
>
> ***Dieser Abschnitt bezieht sich auf die veraltete API-Version 2011-12-05, die nicht für neue Anwendungen verwendet werden sollte.***
> Eine Dokumentation zur aktuellen Low-Level-API finden Sie in der Amazon DynamoDB-API-Referenz.

Beschreibung

Die `CreateTable`-Operation fügt dem Konto eine neue Tabelle hinzu.

Tabellennamen müssen innerhalb derer eindeutig sein, die dem AWS-Konto zugehören, das die Anforderung ausführt und der AWS-Region, welche die Anforderung empfängt (z.B. dynamodb.us-west-2.amazonaws.com). Jeder DynamoDB-Endpunkt ist vollständig unabhängig. Wenn Sie beispielsweise zwei Tabellen namens „MyTable“ haben, eine in dynamodb.us-west-2.amazonaws.com und eine in dynamodb.us-west-1.amazonaws.com, sind sie völlig unabhängig und teilen sich keine Daten.

Die `CreateTable`-Operation löst eine asynchrone Workload aus, um mit der Erstellung der Tabelle zu beginnen. DynamoDB gibt unmittelbar den Status der Tabelle (`CREATING`) zurück, bis sich die Tabelle im Status `ACTIVE` befindet. Sobald sich die Tabelle im Status `ACTIVE` befindet, können Sie Datenebenenoperationen durchführen.

Verwenden Sie die DescribeTables-Operation, um den Status der Tabelle zu überprüfen.

Anforderungen

Syntax

```
// This header is abbreviated.
// For a sample of a complete header, see DynamoDB Low-Level-API.
POST / HTTP/1.1
x-amz-target: DynamoDB_20111205.CreateTable
content-type: application/x-amz-json-1.0

{"TableName":"Table1",
    "KeySchema":
        {"HashKeyElement":{"AttributeName":"AttributeName1","AttributeType":"S"},
        "RangeKeyElement":{"AttributeName":"AttributeName2","AttributeType":"N"}},
    "ProvisionedThroughput":{"ReadCapacityUnits":5,"WriteCapacityUnits":10}
}
```

Name	Beschreibung	Erforderlich
`TableName`	Der Name der zu erstellenden Tabelle.	Ja

Name	Beschreibung	Erforderlich
	Zulässige Zeichen sind a-z, A-Z, 0-9, „_“ (Unterstrich), „-“ (Bindestrich) und „.“ (Punkt). Namen können zwischen 3 und 255 Zeichen lang sein. Typ: Zeichenfolge	

Name	Beschreibung	Erforderlich
`KeySchema`	Die Struktur (einfach oder zusammengesetzt) des Primärschlüssels für die Tabelle. Ein Name-Wert-Paar ist für das `HashKeyElement` und optional für das `RangeKeyElement` erforderlich (nur für zusammengesetzte Primärschlüssel erforderlich). Weitere Informationen zu Primärschlüsseln finden Sie unter Primärschlüssel. Elementnamen der Primärschlüssel können zwischen 1 und 255 Zeichen lang sein und unterliegen keinen Zeichenei nschränkungen. Mögliche Werte für den AttributeType sind "S" (Zeichenfolge), "N" (numerisch) oder "B" (binär). Typ: Zuordnung von `HashKeyElement` oder `HashKeyElement` und `RangeKeyElement` für einen zusammengesetzten Primärschlüssel.	Ja

Name	Beschreibung	Erforderlich
`ProvisionedThroughput`	Neuer Durchsatz für die angegebene Tabelle, bestehend aus Werten für `ReadCapacityUnits` und `WriteCapacityUnits`. Details hierzu finden Sie unter DynamoDB – Modus mit bereitgestellter Kapazität. **Note** Aktuelle Höchst- und Mindestwerte finden Sie unter Kontingente in Amazon DynamoDB. Typ: Array	Ja

Name	Beschreibung	Erforderlich
`ProvisionedThroughput : ReadCapacityUnits`	Legt die Mindestanzahl von konsistenten `ReadCapacityUnits` fest, die pro Sekunde für die angegebene Tabelle verbraucht wird, bevor DynamoDB die Last mit anderen Operationen ausgleicht. Eventually Consistent-Leseoperationen erfordern weniger Aufwand als Consistent-Leseoperationen. Daher stellt die Festlegung von 50 konsistenten `ReadCapacityUnits` pro Sekunde 100 Eventually Consistent-`ReadCapacityUnits` pro Sekunde bereit. Typ: Zahl	Ja
`ProvisionedThroughput : WriteCapacityUnits`	Legt die Mindestanzahl von `WriteCapacityUnits` fest, die pro Sekunde für die angegebene Tabelle verbraucht wird, bevor DynamoDB die Last mit anderen Operationen ausgleicht. Typ: Zahl	Ja

Antworten

Syntax

```
HTTP/1.1 200 OK
x-amzn-RequestId: CSOC7TJPLR0OOKIRLGOHVAICUFVV4KQNSO5AEMVJF66Q9ASUAAJG
content-type: application/x-amz-json-1.0
content-length: 311
Date: Tue, 12 Jul 2011 21:31:03 GMT

{"TableDescription":
    {"CreationDateTime":1.310506263362E9,
    "KeySchema":
        {"HashKeyElement":{"AttributeName":"AttributeName1","AttributeType":"S"},
        "RangeKeyElement":{"AttributeName":"AttributeName2","AttributeType":"N"}},
    "ProvisionedThroughput":{"ReadCapacityUnits":5,"WriteCapacityUnits":10},
    "TableName":"Table1",
    "TableStatus":"CREATING"
    }
}
```

Name	Beschreibung
`TableDescription`	Ein Container für die Eigenschaften der Tabelle.
`CreationDateTime`	Datum als die Tabelle in UNIX epoch time erstellt wurde. Typ: Zahl
`KeySchema`	Die Struktur (einfach oder zusammengesetzt) des Primärschlüssels für die Tabelle. Ein Name-Wert-Paar ist für das `HashKeyElement` und optional für das `RangeKeyElement` erforderlich (nur für zusammengesetzte Primärschlüssel erforderlich). Weitere Informationen zu Primärschlüsseln finden Sie unter Primärschlüssel.

Name	Beschreibung
	Typ: Zuordnung von `HashKeyElement` oder `HashKeyElement` und `RangeKeyElement` für einen zusammengesetzten Primärschlüssel.
`ProvisionedThroughput`	Der Durchsatz für eine angegebene Tabelle, bestehend aus Werten für `ReadCapacityUnits` und `WriteCapacityUnits`. Siehe DynamoDB – Modus mit bereitgestellter Kapazität. Typ: Array
`ProvisionedThroughput:ReadCapacityUnits`	Die minimale Anzahl von `ReadCapacityUnits`, die pro Sekunde verbraucht werden, bevor DynamoDB die Last mit anderen Operationen ausgleicht Typ: Zahl
`ProvisionedThroughput:WriteCapacityUnits`	Die minimale Anzahl von `ReadCapacityUnits`, die pro Sekunde verbraucht werden, bevor `WriteCapacityUnits` die Last mit anderen Operationen ausgleicht Typ: Zahl
`TableName`	Der Name der erstellten Tabelle. Typ: Zeichenfolge

Name	Beschreibung
`TableStatus`	Der aktuelle Status der Tabelle (`CREATING`). Sobald sich die Tabelle im `ACTIVE`-Status befindet, können Sie in dieser Daten ablegen. Verwenden Sie die DescribeTables-API, um den Status der Tabelle zu überprüfen. Typ: Zeichenfolge

Spezielle Fehler

Fehler	Beschreibung
`ResourceInUseException`	Versuch, eine bereits vorhandene Tabelle neu zu erstellen.
`LimitExceededException`	Die Anzahl von gleichzeitigen Tabellena nforderungen (kumulative Anzahl von Tabellen im Status `CREATING`, `DELETING` oder `UPDATING`) überschreitet die maximal zulässige. **Note** Aktuelle Höchst- und Mindestwerte finden Sie unter Kontingente in Amazon DynamoDB. .

Beispiele

Das folgende Beispiel erstellt eine Tabelle mit einem zusammengesetzten Primärschlüssel, der eine Zeichenfolge und eine Zahl enthält. Beispiele für die Verwendung des AWS SDK finden Sie unter Arbeiten mit Tabellen und Daten in DynamoDB.

Beispielanforderung

```
// This header is abbreviated.
// For a sample of a complete header, see DynamoDB Low-Level-API.
POST / HTTP/1.1
x-amz-target: DynamoDB_20111205.CreateTable
content-type: application/x-amz-json-1.0


{"TableName":"comp-table",
    "KeySchema":
        {"HashKeyElement":{"AttributeName":"user","AttributeType":"S"},
        "RangeKeyElement":{"AttributeName":"time","AttributeType":"N"}},
    "ProvisionedThroughput":{"ReadCapacityUnits":5,"WriteCapacityUnits":10}
}
```

Beispielantwort

```
HTTP/1.1 200 OK
x-amzn-RequestId: CSOC7TJPLR0OOKIRLGOHVAICUFVV4KQNSO5AEMVJF66Q9ASUAAJG
content-type: application/x-amz-json-1.0
content-length: 311
Date: Tue, 12 Jul 2011 21:31:03 GMT

{"TableDescription":
    {"CreationDateTime":1.310506263362E9,
    "KeySchema":
        {"HashKeyElement":{"AttributeName":"user","AttributeType":"S"},
        "RangeKeyElement":{"AttributeName":"time","AttributeType":"N"}},
    "ProvisionedThroughput":{"ReadCapacityUnits":5,"WriteCapacityUnits":10},
    "TableName":"comp-table",
    "TableStatus":"CREATING"
    }
}
```

Zugehörige Aktionen

- DescribeTables
- DeleteTable

DeleteItem

> **Important**
>
> ***Dieser Abschnitt bezieht sich auf die veraltete API-Version 2011-12-05, die nicht für neue Anwendungen verwendet werden sollte.***
> Eine Dokumentation zur aktuellen Low-Level-API finden Sie in der Amazon DynamoDB-API-Referenz.

Beschreibung

Löscht ein einzelnes Element in einer Tabelle nach Primärschlüssel Sie können eine bedingte Löschoperation durchführen, die das Element löscht, wenn es vorhanden ist oder wenn es über einen erwarteten Attributwert verfügt.

> **Note**
>
> Wenn Sie `DeleteItem` ohne Attribute oder Werte angeben, werden alle Attribute für das Element gelöscht.
> Sofern Sie keine Bedingungen angeben, ist `DeleteItem` eine idempotente Operation.
> Das mehrmalige Ausführen für das gleiche Element oder Attribut resultiert nicht in einer Fehlermeldung.
> Bedingte Löschungen eignen sich nur für das Löschen von Elementen und Attributen, wenn bestimmte Bedingungen erfüllt sind. Wenn die Bedingungen erfüllt sind, führt DynamoDB die Löschung durch. Andernfalls wird das Element nicht gelöscht.
> Sie können die erwartete bedingte Prüfung für ein Attribut pro Operation durchführen.

Anforderungen

Syntax

```
// This header is abbreviated.
// For a sample of a complete header, see DynamoDB Low-Level-API.
POST / HTTP/1.1
x-amz-target: DynamoDB_20111205.DeleteItem
content-type: application/x-amz-json-1.0

{"TableName":"Table1",
    "Key":
        {"HashKeyElement":{"S":"AttributeValue1"},"RangeKeyElement":
{"N":"AttributeValue2"}},
    "Expected":{"AttributeName3":{"Value":{"S":"AttributeValue3"}}},
    "ReturnValues":"ALL_OLD"}
}
```

Name	Beschreibung	Erforderlich
`TableName`	Der Name der Tabelle, die das zu löschende Element enthält. Typ: Zeichenfolge	Ja
`Key`	Der Primärschlüssel, der das Element definiert. Weitere Informationen zu Primärsch lüsseln finden Sie unter Primärschlüssel. Typ: Zuordnung von `HashKeyElement` zu seinem Wert und von `RangeKeyElement` zu seinem Wert.	Ja
`Expected`	Bestimmt ein Attribut für eine bedingte Löschung. Der Parameter `Expected`	Nein

Name	Beschreibung	Erforderlich
	ermöglicht es Ihnen, einen Attributnamen anzugeben und zu entscheiden, ob DynamoDB prüfen soll oder nicht, ob das Attribut über einen bestimmten Wert verfügt, bevor es gelöscht wird. Typ: Zuordnung von Attributn amen.	
`Expected:AttributeName`	Der Name des Attributs für die bedingte Put-Operation. Typ: Zeichenfolge	Nein

Name	Beschreibung	Erforderlich

`Expected:AttributeName: ExpectedAttributeValue`	Verwenden Sie diesen Parameter, um anzugeben, ob ein Wert für das Attributname-Wert-Paar bereits vorhanden ist oder nicht.
	
	Die folgende JSON-Notation löscht das Element, wenn das Attribut "Farbe" für dieses Element nicht vorhanden ist:
	
	`"Expected" : {"Color":{"Exists":false}}`
	
	Die folgende JSON-Notation prüft, ob das Attribut mit dem Namen "Farbe" über einen vorhandenen Wert für "Gelb" verfügt, bevor das Element gelöscht wird:
	
	`"Expected" : {"Color":{"Exists":true},{"Value":{"S":"Yellow"}}}`
	
	Wenn Sie den Parameter `Expected` verwenden und einen `Value` angeben, geht DynamoDB davon aus, dass das Attribut vorhanden ist und einen zu ersetzend en aktuellen Wert hat. Sie müssen `{"Exists":true}` demnach nicht angeben, weil
		Nein

Name	Beschreibung	Erforderlich
	er enthalten ist. Sie können die Anforderung verkürzen, um: `"Expected" : {"Color":{"Value":{"S":"Yellow"}}}` **Note** Wenn Sie `{"Exists":true}` ohne einen zu prüfenden Attributwert angeben, gibt DynamoDB einen Fehler zurück.	
`ReturnValues`	Verwenden Sie diesen Parameter, wenn Sie die Attribut-Namen-Wert-Paare erhalten möchten, bevor sie gelöscht wurden. Mögliche Parameterwerte sind `NONE` (Standard) oder `ALL_OLD`. Wenn `ALL_OLD` angegeben wird, werden die Inhalte des alten Elements zurückgegeben. Wenn dieser Parameter nicht angegeben wird oder `NONE` ist, wird nichts zurückgegeben. Typ: Zeichenfolge	Nein

Antworten

Syntax

```
HTTP/1.1 200 OK
x-amzn-RequestId: CSOC7TJPLR0OOKIRLGOHVAICUFVV4KQNSO5AEMVJF66Q9ASUAAJG
content-type: application/x-amz-json-1.0
content-length: 353
Date: Tue, 12 Jul 2011 21:31:03 GMT

{"Attributes":
    {"AttributeName3":{"SS":["AttributeValue3","AttributeValue4","AttributeValue5"]},
    "AttributeName2":{"S":"AttributeValue2"},
    "AttributeName1":{"N":"AttributeValue1"}
    },
"ConsumedCapacityUnits":1
}
```

Name	Beschreibung
`Attributes`	Wenn der Parameter `ReturnValues` als `ALL_OLD` in der Anforderung angegeben wird, gibt DynamoDB ein Array von Attribut-Namen-Wert-Paare zurück (im Wesentlichen, das gelöschte Element). Andernfalls enthält die Antwort einen leeren Satz. Typ: Array von Attribut-Namen-Wert-Paare.
`ConsumedCapacityUnits`	Die Anzahl der Schreibkapazitätseinheiten, die von dem Vorgang verbraucht werden. Dieser Wert zeigt die Anzahl, die für Ihren bereitges tellten Durchsatz gültig ist. Löschoperationen für nicht vorhandene Elemente verbrauchen 1 Schreibkapazitätseinheit. Weitere Informati onen finden Sie unter DynamoDB – Modus mit bereitgestellter Kapazität. Typ: Zahl

Spezielle Fehler

Fehler	Beschreibung
`ConditionalCheckFailedException`	Bedingte Prüfung fehlgeschlagen. Ein erwartete r Attributwert wurde nicht gefunden.

Beispiele

Beispielanforderung

```
// This header is abbreviated.
// For a sample of a complete header, see DynamoDB Low-Level-API.
POST / HTTP/1.1
x-amz-target: DynamoDB_20111205.DeleteItem
content-type: application/x-amz-json-1.0

{"TableName":"comp-table",
    "Key":
        {"HashKeyElement":{"S":"Mingus"},"RangeKeyElement":{"N":"200"}},
    "Expected":
        {"status":{"Value":{"S":"shopping"}}},
    "ReturnValues":"ALL_OLD"
}
```

Beispielantwort

```
HTTP/1.1 200 OK
x-amzn-RequestId: U9809LI6BBFJA5N2R0TB0P017JVV4KQNSO5AEMVJF66Q9ASUAAJG
content-type: application/x-amz-json-1.0
content-length: 353
Date: Tue, 12 Jul 2011 22:31:23 GMT

{"Attributes":
    {"friends":{"SS":["Dooley","Ben","Daisy"]},
    "status":{"S":"shopping"},
    "time":{"N":"200"},
    "user":{"S":"Mingus"}
    },
"ConsumedCapacityUnits":1
```

```
}
```

Zugehörige Aktionen

- PutItem

DeleteTable

> **Important**
>
> ***Dieser Abschnitt bezieht sich auf die veraltete API-Version 2011-12-05, die nicht für neue Anwendungen verwendet werden sollte.***
> Eine Dokumentation zur aktuellen Low-Level-API finden Sie in der Amazon DynamoDB-API-Referenz.

Beschreibung

Die `DeleteTable`-Operation löscht eine Tabelle und alle ihre Elemente. Nach einer `DeleteTable`-Anforderung, befindet sich die angegebene Tabelle im Status `DELETING`, bis DynamoDB die Löschung abschließt. Wenn sich die Tabelle im Status `ACTIVE` befindet, können Sie sie löschen. Wenn eine Tabelle sich im Status `CREATING` oder `UPDATING` befindet, gibt DynamoDB den Fehler `ResourceInUseException` zurück. Wenn die angegebene Tabelle nicht vorhanden ist, gibt DynamoDB einen `ResourceNotFoundException` zurück. Wenn die Tabelle sich bereits im Status `DELETING` befindet, wird kein Fehler zurückgegeben.

> **Note**
>
> DynamoDB kann möglicherweise Operationsanforderungen auf Datenebene, z. B. `GetItem` und `PutItem`, in einer Tabelle mit dem Status `DELETING` akzeptieren, bis die Löschung der Tabelle abgeschlossen ist.

Alle Tabellen, die mit dem AWS-Konto verknüpft sind, das die Anforderung ausgibt, und die AWS-Region, welche die Anforderung empfängt, sind eindeutig (z.B. dynamodb.us-west-1.amazonaws.com). Jeder DynamoDB-Endpunkt ist vollständig unabhängig. Wenn Sie beispielsweise zwei Tabellen namens „MyTable“ haben, eine in dynamodb.us-

west-2.amazonaws.com und eine in dynamodb.us-west-1.amazonaws.com, sind sie völlig unabhängig und teilen sich keine Daten; das Löschen der einen löscht die andere nicht.

Verwenden Sie die DescribeTables-Operation, um den Status der Tabelle zu überprüfen.

Anforderungen

Syntax

```
// This header is abbreviated.
// For a sample of a complete header, see DynamoDB Low-Level-API.
POST / HTTP/1.1
x-amz-target: DynamoDB_20111205.DeleteTable
content-type: application/x-amz-json-1.0

{"TableName":"Table1"}
```

Name	Beschreibung	Erforderlich
`TableName`	Der Name der zu löschenden Tabelle. Typ: Zeichenfolge	Ja

Antworten

Syntax

```
HTTP/1.1 200 OK
x-amzn-RequestId: 4HONCKIVH1BFUDQ1U68CTG3N27VV4KQNSO5AEMVJF66Q9ASUAAJG
content-type: application/x-amz-json-1.0
content-length: 311
Date: Sun, 14 Aug 2011 22:56:22 GMT

{"TableDescription":
    {"CreationDateTime":1.313362508446E9,
    "KeySchema":
        {"HashKeyElement":{"AttributeName":"user","AttributeType":"S"},
        "RangeKeyElement":{"AttributeName":"time","AttributeType":"N"}},
    "ProvisionedThroughput":{"ReadCapacityUnits":10,"WriteCapacityUnits":10},
```

```
    "TableName":"Table1",
    "TableStatus":"DELETING"
    }
}
```

Name	Beschreibung
`TableDescription`	Ein Container für die Eigenschaften der Tabelle.
`CreationDateTime`	Das Datum, an dem die Tabelle erstellt wurde. Typ: Zahl
`KeySchema`	Die Struktur (einfach oder zusammengesetzt) des Primärschlüssels für die Tabelle. Ein Name-Wert-Paar ist für das `HashKeyElement` und optional für das `RangeKeyElement` erforderlich (nur für zusammengesetzte Primärschlüssel erforderlich). Weitere Informationen zu Primärschlüsseln finden Sie unter Primärschlüssel. Typ: Zuordnung von `HashKeyElement` oder `HashKeyElement` und `RangeKeyElement` für einen zusammengesetzten Primärschlüssel.
`ProvisionedThroughput`	Der Durchsatz für eine angegebene Tabelle, bestehend aus Werten für `ReadCapacityUnits` und `WriteCapacityUnits`. Siehe DynamoDB – Modus mit bereitgestellter Kapazität.
`ProvisionedThroughput`: `ReadCapacityUnits`	Die Mindestanzahl von `ReadCapacityUnits`, die pro Sekunde für die angegebene Tabelle verbraucht werden, bevor DynamoDB die Last mit anderen Operationen ausgleicht.

Name	Beschreibung
	Typ: Zahl
`ProvisionedThroughput:WriteCapacityUnits`	Die Mindestanzahl von `WriteCapacityUnits`, die pro Sekunde für die angegebene Tabelle verbraucht werden, bevor DynamoDB die Last mit anderen Operationen ausgleicht. Typ: Zahl
`TableName`	Der Name der gelöschten Tabelle. Typ: Zeichenfolge
`TableStatus`	Der aktuelle Status der Tabelle (`DELETING`). Sobald die Tabelle gelöscht wird, werden nachfolgende Anforderungen für die Tabelle zurückgegeben `resource not found`. Verwenden Sie die DescribeTables-Operation, um den Status der Tabelle zu überprüfen. Typ: Zeichenfolge

Spezielle Fehler

Fehler	Beschreibung
`ResourceInUseException`	Die Tabelle befindet sich im Status `CREATING` oder `UPDATING` und kann nicht gelöscht werden.

Beispiele

Beispielanforderung

```
// This header is abbreviated. For a sample of a complete header, see DynamoDB Low-Level-API.
POST / HTTP/1.1
x-amz-target: DynamoDB_20111205.DeleteTable
content-type: application/x-amz-json-1.0
content-length: 40

{"TableName":"favorite-movies-table"}
```

Beispielantwort

```
HTTP/1.1 200 OK
x-amzn-RequestId: 4HONCKIVH1BFUDQ1U68CTG3N27VV4KQNSO5AEMVJF66Q9ASUAAJG
content-type: application/x-amz-json-1.0
content-length: 160
Date: Sun, 14 Aug 2011 17:20:03 GMT

{"TableDescription":
    {"CreationDateTime":1.313362508446E9,
    "KeySchema":
        {"HashKeyElement":{"AttributeName":"name","AttributeType":"S"}},
    "TableName":"favorite-movies-table",
    "TableStatus":"DELETING"
}
```

Zugehörige Aktionen

- CreateTable
- DescribeTables

DescribeTables

> **Important**
>
> ***Dieser Abschnitt bezieht sich auf die veraltete API-Version 2011-12-05, die nicht für neue Anwendungen verwendet werden sollte.***

Eine Dokumentation zur aktuellen Low-Level-API finden Sie in der Amazon DynamoDB-API-Referenz.

Beschreibung

Gibt Informationen über die Tabelle, einschließlich ihrem aktuellen Status, das Primärschlüsselschema und den Zeitpunkt der Tabellenerstellung zurück. DescribeTables-Ergebnisse sind Eventually Consistent. Wenn Sie DescribeTables bei der Erstellung einer Tabelle zu früh verwenden, gibt DynamoDB eine `ResourceNotFoundException` zurück. Wenn Sie DescribeTables bei der Aktualisierung einer Tabelle zu früh verwenden, werden die neuen Werte möglicherweise nicht sofort verfügbar sind.

Anforderungen

Syntax

```
// This header is abbreviated.
// For a sample of a complete header, see DynamoDB Low-Level-API.
POST / HTTP/1.1
x-amz-target: DynamoDB_20111205.DescribeTable
content-type: application/x-amz-json-1.0

{"TableName":"Table1"}
```

Name	Beschreibung	Erforderlich
`TableName`	Der Name der zu beschreibenden Tabelle. Typ: Zeichenfolge	Ja

Antworten

Syntax

```
HTTP/1.1 200
x-amzn-RequestId: 8966d095-71e9-11e0-a498-71d736f27375
content-type: application/x-amz-json-1.0
```

```
Content-Length: 543


{"Table":
    {"CreationDateTime":1.309988345372E9,
    ItemCount:1,
    "KeySchema":
        {"HashKeyElement":{"AttributeName":"AttributeName1","AttributeType":"S"},
        "RangeKeyElement":{"AttributeName":"AttributeName2","AttributeType":"N"}},
    "ProvisionedThroughput":{"LastIncreaseDateTime": Date, "LastDecreaseDateTime":
 Date, "ReadCapacityUnits":10,"WriteCapacityUnits":10},
    "TableName":"Table1",
    "TableSizeBytes":1,
    "TableStatus":"ACTIVE"
    }
}
```

Name	Beschreibung
`Table`	Container für die Tabelle, die beschrieben wird Typ: Zeichenfolge
`CreationDateTime`	Datum als die Tabelle in UNIX epoch time erstellt wurde.
`ItemCount`	Anzahl der Elemente in der angegebenen Tabelle DynamoDB aktualisiert diesen Wert ca. alle sechs Stunden. Neueste Änderungen werden in diesem Wert möglicherweise nicht wiedergegeben. Typ: Zahl
`KeySchema`	Die Struktur (einfach oder zusammengesetzt) des Primärschlüssels für die Tabelle. Ein Name-Wert-Paar ist für das `HashKeyElement` und optional für das `RangeKeyElement` erforderlich (nur für zusammengesetzte Primärschlüssel erforderlich). Die

Name	Beschreibung
	maximale Hash-Schlüsselgröße ist 2048 Byte. Die maximale Range-Schlüsselgröße ist 1024 Byte. Beide Grenzen werden separat durchgesetzt (d. h. Sie können einen kombinierten Hash + Range 2048 + 1024-Schlüssel haben). Weitere Informationen zu Primärschlüsseln finden Sie unter Primärschlüssel.
`ProvisionedThroughput`	Der Durchsatz für die angegebene Tabelle, bestehend aus Werten für `LastIncreaseDateTime` (falls zutreffend), `LastDecreaseDateTime` (falls zutreffend), `ReadCapacityUnits` und `WriteCapacityUnits`. Wenn der Durchsatz für die Tabelle noch nie erhöht oder verringert wurde, gibt DynamoDB keine Werte für diese Elemente zurück. Siehe DynamoDB – Modus mit bereitgestellter Kapazität. Typ: Array
`TableName`	Der Name der angeforderten Tabelle. Typ: Zeichenfolge
`TableSizeBytes`	Die Gesamtgröße der angegebenen Tabelle in Bytes. DynamoDB aktualisiert diesen Wert ca. alle sechs Stunden. Neueste Änderungen werden in diesem Wert möglicherweise nicht wiedergegeben. Typ: Zahl
`TableStatus`	Der aktuelle Status der Tabelle (`CREATING`, `ACTIVE`, `DELETING` oder `UPDATING`). Sobald sich die Tabelle in dem `ACTIVE`-Status befindet, können Sie Daten hinzufügen.

Spezielle Fehler

Keine Fehler sind für diese Operation spezifisch.

Beispiele

Die folgenden Beispiele zeigen eine HTTP-POST-Anforderung und Antwort mithilfe der "DescribeTable"-Operation für eine Tabelle mit dem Namen "comp-table". Die Tabelle verfügt über einen zusammengesetzten Primärschlüssel.

Beispielanforderung

```
// This header is abbreviated.
// For a sample of a complete header, see DynamoDB Low-Level-API.
POST / HTTP/1.1
x-amz-target: DynamoDB_20111205.DescribeTable
content-type: application/x-amz-json-1.0

{"TableName":"users"}
```

Beispielantwort

```
HTTP/1.1 200
x-amzn-RequestId: 8966d095-71e9-11e0-a498-71d736f27375
content-type: application/x-amz-json-1.0
content-length: 543

{"Table":
    {"CreationDateTime":1.309988345372E9,
    "ItemCount":23,
    "KeySchema":
        {"HashKeyElement":{"AttributeName":"user","AttributeType":"S"},
        "RangeKeyElement":{"AttributeName":"time","AttributeType":"N"}},
    "ProvisionedThroughput":{"LastIncreaseDateTime": 1.309988345384E9,
 "ReadCapacityUnits":10,"WriteCapacityUnits":10},
    "TableName":"users",
    "TableSizeBytes":949,
    "TableStatus":"ACTIVE"
    }
}
```

Zugehörige Aktionen

- CreateTable
- DeleteTable
- ListTables

GetItem

> **Important**
>
> ***Dieser Abschnitt bezieht sich auf die veraltete API-Version 2011-12-05, die nicht für neue Anwendungen verwendet werden sollte.***
> Eine Dokumentation zur aktuellen Low-Level-API finden Sie in der Amazon DynamoDB-API-Referenz.

Beschreibung

Die `GetItem`-Operation gibt einen `Attributes`-Satz für ein Element zurück, das mit dem Primärschlüssel übereinstimmt. Wenn kein passendes Element vorhanden ist, gibt `GetItem` keine Daten zurück.

Die `GetItem`-Operation stellt standardmäßig einen Eventually Consistent-Lesevorgang bereit. Wenn Eventually Consistent-Lesevorgänge für Ihre Anwendung nicht akzeptabel sind, verwenden Sie `ConsistentRead`. Obwohl diese Operation länger dauern könnte als ein Standardlesevorgang, gibt sie immer den letzten aktualisierten Wert zurück. Weitere Informationen finden Sie unter DynamoDB-Lesekonsistenz.

Anforderungen

Syntax

```
// This header is abbreviated.
// For a sample of a complete header, see DynamoDB Low-Level-API.
POST / HTTP/1.1
x-amz-target: DynamoDB_20111205.GetItem
content-type: application/x-amz-json-1.0
```

```
{"TableName":"Table1",
  "Key":
  {"HashKeyElement": {"S":"AttributeValue1"},
  "RangeKeyElement": {"N":"AttributeValue2"}
 },
 "AttributesToGet":["AttributeName3","AttributeName4"],
 "ConsistentRead":Boolean
}
```

Name	Beschreibung	Erforderlich
`TableName`	Der Name der Tabelle, die das angeforderte Element enthält. Typ: Zeichenfolge	Ja
`Key`	Die Primärschlüsselwerte, die das Element definiere n. Weitere Informationen zu Primärschlüsseln finden Sie unter Primärschlüssel. Typ: Zuordnung von `HashKeyElement` zu seinem Wert und von `RangeKeyElement` zu seinem Wert.	Ja
`AttributesToGet`	Array von Attributnamen. Wenn Attributnamen nicht angegeben sind, dann werden alle Attribute zurückgegeben. Wenn einige Attribute nicht gefunden werden, sind sie nicht im Abfrageergebnis enthalten. Typ: Array	Nein

Name	Beschreibung	Erforderlich
`ConsistentRead`	Wenn auf `true` festgeleg t, dann wird ein Consistent-Lesevorgang ausgegeben, andernfalls wird Eventually Consistent verwendet. Typ: Boolesch	Nein

Antworten

Syntax

```
HTTP/1.1 200
x-amzn-RequestId: 8966d095-71e9-11e0-a498-71d736f27375
content-type: application/x-amz-json-1.0
content-length: 144

{"Item":{
 "AttributeName3":{"S":"AttributeValue3"},
 "AttributeName4":{"N":"AttributeValue4"},
 "AttributeName5":{"B":"dmFsdWU="}
 },
"ConsumedCapacityUnits": 0.5
}
```

Name	Beschreibung
`Item`	Enthält die angeforderten Attribute. Typ: Zuordnung von Attribut-Namen-Wert-Paare.
`ConsumedCapacityUnits`	Die Anzahl der Lesekapazitätseinheiten, die von der Operation verbraucht werden. Dieser Wert zeigt die Anzahl, die für Ihren bereitges tellten Durchsatz gültig ist. Anforderungen für nicht vorhandene Elemente verbrauchen je

Name	Beschreibung
	nach Typ des Lesevorgangs die Mindestle sekapazitätseinheiten. Weitere Informationen finden Sie unter DynamoDB – Modus mit bereitgestellter Kapazität. Typ: Zahl

Spezielle Fehler

Keine nur für diese Operation spezifischen Fehler.

Beispiele

Beispiele für die Verwendung des AWS SDK finden Sie unter Arbeiten mit Elementen und Attributen in DynamoDB.

Beispielanforderung

```
// This header is abbreviated.
// For a sample of a complete header, see DynamoDB Low-Level-API.
POST / HTTP/1.1
x-amz-target: DynamoDB_20111205.GetItem
content-type: application/x-amz-json-1.0

{"TableName":"comptable",
 "Key":
  {"HashKeyElement":{"S":"Julie"},
  "RangeKeyElement":{"N":"1307654345"}},
 "AttributesToGet":["status","friends"],
 "ConsistentRead":true
}
```

Beispielantwort

Beachten Sie, dass der Wert für die verbrauchten Kapazitätseinheiten 1 ist, weil der optionale Parameter `ConsistentRead` auf `true` festgelegt ist. Wenn `ConsistentRead` für die gleiche Anforderung auf `false` festgelegt ist (oder nicht angegeben ist), ist die Antwort Eventually Consistent und der Wert für die verbrauchten Kapazitätseinheiten wäre 0,5.

```
HTTP/1.1 200
x-amzn-RequestId: 8966d095-71e9-11e0-a498-71d736f27375
content-type: application/x-amz-json-1.0
content-length: 72

{"Item":
 {"friends":{"SS":["Lynda, Aaron"]},
 "status":{"S":"online"}
 },
"ConsumedCapacityUnits": 1
}
```

ListTables

> **Important**
>
> ***Dieser Abschnitt bezieht sich auf die veraltete API-Version 2011-12-05, die nicht für neue Anwendungen verwendet werden sollte.***
> Eine Dokumentation zur aktuellen Low-Level-API finden Sie in der Amazon DynamoDB-API-Referenz.

Beschreibung

Gibt ein Array aller Tabellen zurück, die dem aktuellen Konto und Endpunkt zugeordnet sind.

Jeder DynamoDB-Endpunkt ist vollständig unabhängig. Wenn Sie beispielsweise zwei Tabellen namens „MyTable“ haben, eine in dynamodb.us-west-2.amazonaws.com und eine in dynamodb.us-east-1.amazonaws.com, sind sie völlig unabhängig und teilen sich keine Daten. Die ListTables-Operation gibt alle Tabellennamen, die mit dem Konto verknüpft sind, von dem die Anforderung stammt, für den Endpunkt, der die Anforderung empfängt, zurück.

Anforderungen

Syntax

```
// This header is abbreviated.
// For a sample of a complete header, see DynamoDB Low-Level-API.
POST / HTTP/1.1
x-amz-target: DynamoDB_20111205.ListTables
```

```
content-type: application/x-amz-json-1.0

{"ExclusiveStartTableName":"Table1","Limit":3}
```

Die ListTables-Operation fordert standardmäßig alle Tabellennamen, die mit dem Konto verknüpft sind, von dem die Anforderung stammt, für den Endpunkt, der die Anforderung empfängt, an.

Name	Beschreibung	Erforderlich
`Limit`	Eine Reihe von zurückzug ebenden maximalen Tabellenn amen. Typ: Ganzzahl	Nein
`ExclusiveStartTabl eName`	Der Name der Tabelle, mit der die Liste beginnt. Wenn Sie bereits eine ListTables-Operati on ausgeführt und einen `LastEvaluatedTable Name` -Wert in der Antwort erhalten haben, verwenden Sie diesen Wert, um die Liste fortzuführen. Typ: Zeichenfolge	Nein

Antworten

Syntax

```
HTTP/1.1 200 OK
x-amzn-RequestId: S1LEK2DPQP8OJNHVHL8OU2M7KRVV4KQNSO5AEMVJF66Q9ASUAAJG
content-type: application/x-amz-json-1.0
content-length: 81
Date: Fri, 21 Oct 2011 20:35:38 GMT

{"TableNames":["Table1","Table2","Table3"], "LastEvaluatedTableName":"Table3"}
```

Name	Beschreibung
`TableNames`	Die Namen der Tabellen, die dem aktuellen Konto am aktuellen Endpunkt zugeordnet sind. Typ: Array
`LastEvaluatedTableName`	Der Name der letzten Tabelle in der aktuellen Liste, sofern einige Tabellen für das Konto und den Endpunkt nicht zurückgegeben wurden. Dieser Wert ist in einer Antwort nicht vorhanden , wenn alle Tabellennamen bereits zurückgeg eben wurden. Verwenden Sie diesen Wert als `ExclusiveStartTableName` in einer neuen Anforderung, um die Liste fortzufüh ren, bis alle Tabellennamen zurückgegeben werden. Typ: Zeichenfolge

Spezielle Fehler

Keine Fehler sind für diese Operation spezifisch.

Beispiele

Die folgenden Beispiele zeigen eine HTTP-POST-Anforderung und Antwort mithilfe der ListTables-Operation.

Beispielanforderung

```
// This header is abbreviated.
// For a sample of a complete header, see DynamoDB Low-Level-API.
POST / HTTP/1.1
x-amz-target: DynamoDB_20111205.ListTables
content-type: application/x-amz-json-1.0

{"ExclusiveStartTableName":"comp2","Limit":3}
```

Beispielantwort

```
HTTP/1.1 200 OK
x-amzn-RequestId: S1LEK2DPQP8OJNHVHL8OU2M7KRVV4KQNSO5AEMVJF66Q9ASUAAJG
content-type: application/x-amz-json-1.0
content-length: 81
Date: Fri, 21 Oct 2011 20:35:38 GMT

{"LastEvaluatedTableName":"comp5","TableNames":["comp3","comp4","comp5"]}
```

Zugehörige Aktionen

- DescribeTables
- CreateTable
- DeleteTable

PutItem

> **Important**
>
> ***Dieser Abschnitt bezieht sich auf die veraltete API-Version 2011-12-05, die nicht für neue Anwendungen verwendet werden sollte.***
> Eine Dokumentation zur aktuellen Low-Level-API finden Sie in der Amazon DynamoDB-API-Referenz.

Beschreibung

Erstellt ein neues Element oder ersetzt ein altes durch ein neues Element (einschließlich aller Attribute). Wenn ein Element in der angegebenen Tabelle mit demselben Primärschlüssel bereits vorhanden ist, wird das vorhandene vollständig durch das neue Element ersetzt. Sie können eine bedingte PUT-Transaktion ausführen (ein neues Element einfügen, wenn keins mit dem angegebenen Primärschlüssel vorhanden ist) oder ein vorhandenes Element ersetzen, wenn es bestimmte Attributwerte besitzt.

Attributwerte dürfen nicht Null sein, Zeichenketten- und Binärtypattribute müssen Längen haben, die größer als Null sind, und festgelegte Typattribute dürfen nicht leer sein. Anfragen mit leeren Werten werden mit einer `ValidationException` abgelehnt.

> **Note**
>
> Um sicherzustellen, dass ein vorhandenes nicht durch ein neues Element ersetzt wird, verwenden Sie eine bedingte PUT-Operation, wobei `Exists` für das bzw. die Primärschlüsselattribute auf `false` festgelegt wird.

Weitere Informationen zur Verwendung von `PutItem` finden Sie unter Arbeiten mit Elementen und Attributen in DynamoDB.

Anforderungen

Syntax

```
// This header is abbreviated.
// For a sample of a complete header, see DynamoDB Low-Level-API.
POST / HTTP/1.1
x-amz-target: DynamoDB_20111205.PutItem
content-type: application/x-amz-json-1.0

{"TableName":"Table1",
    "Item":{
        "AttributeName1":{"S":"AttributeValue1"},
        "AttributeName2":{"N":"AttributeValue2"},
        "AttributeName5":{"B":"dmFsdWU="}
    },
    "Expected":{"AttributeName3":{"Value": {"S":"AttributeValue"}, "Exists":Boolean}},
    "ReturnValues":"ReturnValuesConstant"}
```

Name	Beschreibung	Erforderlich
`TableName`	Der Name der Tabelle, die das Element enthält. Typ: Zeichenfolge	Ja
`Item`	Eine Übersicht über die Attribute für das Element, die die Primärschlüsselwerte zur Definition des Elements	Ja

Name	Beschreibung	Erforderlich
	enthalten muss. Andere Attribut-Namen-Wert-Paare können für das Element bereitgestellt werden. Weitere Informationen zu Primärsch lüsseln finden Sie unter Primärschlüssel. Typ: Zuweisung von Attributn amen und zu Attributwerten.	
`Expected`	Gibt ein Attribut für eine bedingte PUT-Operation an. Der Parameter `Expected` erlaubt es Ihnen, einen Attributnamen anzugeben und festzulegen, ob DynamoDB überprüfen soll, ob der Attributwert bereits vorhanden ist bzw. ob der Attributwert vorhanden ist und über einen bestimmten Wert verfügt, bevor er geändert wird. Typ: Zuweisung eines Attributnamens zu einem Attributwert und ob er vorhanden ist.	Nein
`Expected:AttributeName`	Der Name des Attributs für die bedingte Put-Operation. Typ: Zeichenfolge	Nein

Name	Beschreibung	Erforderlich

`Expected:AttributeName: ExpectedAttributeValue`	Verwenden Sie diesen Parameter, um anzugeben, ob ein Wert für das Attributname-Wert-Paar bereits vorhanden ist oder nicht.
	
	Die folgende JSON-Notation ersetzt das Element, wenn das Attribut "Farbe" für dieses Element noch nicht vorhanden ist:
	
	`"Expected" : {"Color":{"Exists":false}}`
	
	Die folgende JSON-Notation prüft, ob das Attribut mit dem Namen "Farbe" über einen vorhandenen "Gelb"-Wert verfügt, bevor das Element ersetzt wird:
	
	`"Expected" : {"Color":{"Exists":true,{"Value":{"S":"Yellow"}}}}`
	
	Wenn Sie den Parameter `Expected` verwenden und einen `Value` angeben, geht DynamoDB standardmäßig davon aus, dass das Attribut vorhanden ist und einen zu ersetzenden aktuellen Wert hat. Sie müssen `{"Exists"`
		Nein

Name	Beschreibung	Erforderlich
	`:true}` demnach nicht angeben, weil er enthalten ist. Sie können die Anforderung verkürzen, um: `"Expected" : {"Color":{"Value": {"S":"Yellow"}}}` **Note** Wenn Sie `{"Exists" :true}` ohne einen zu prüfenden Attributwert angeben, gibt DynamoDB einen Fehler zurück.	

Name	Beschreibung	Erforderlich
`ReturnValues`	Verwenden Sie diesen Parameter, wenn Sie die Attribut-Namen-Wert-Paare erhalten möchten, bevor sie mit der `PutItem`-Anforderung aktualisiert wurden. Mögliche Parameterwerte sind `NONE` (Standard) oder `ALL_OLD`. Wenn `ALL_OLD` angegeben ist und ein Attribut-Namen-Wert-Paar durch `PutItem` überschrieben wurde, wird der Inhalt des alten Elements zurückgegeben. Wenn dieser Parameter nicht angegeben wird oder `NONE` ist, wird nichts zurückgegeben. Typ: Zeichenfolge	Nein

Antworten

Syntax

In dem folgenden Syntax-Beispiel wird davon ausgegangen, dass die Anforderung für den Parameter `ReturnValues` `ALL_OLD` angegeben hat; andernfalls beinhaltet die Antwort nur das `ConsumedCapacityUnits`-Element.

```
HTTP/1.1 200
x-amzn-RequestId: 8966d095-71e9-11e0-a498-71d736f27375
content-type: application/x-amz-json-1.0
content-length: 85

{"Attributes":
 {"AttributeName3":{"S":"AttributeValue3"},
 "AttributeName2":{"SS":"AttributeValue2"},
```

```
 "AttributeName1":{"SS":"AttributeValue1"},
 },
"ConsumedCapacityUnits":1
}
```

Name	Beschreibung
`Attributes`	Attributwerte vor der PUT-Operation, aber nur wenn der Parameter `ReturnValues` in der Anforderung mit `ALL_OLD` angegeben wird. Typ: Zuordnung von Attribut-Namen-Wert-Paare.
`ConsumedCapacityUnits`	Die Anzahl der Schreibkapazitätseinheiten, die von dem Vorgang verbraucht werden. Dieser Wert zeigt die Anzahl, die für Ihren bereitges tellten Durchsatz gültig ist. Weitere Informati onen finden Sie unter DynamoDB – Modus mit bereitgestellter Kapazität. Typ: Zahl

Spezielle Fehler

Fehler	Beschreibung
`ConditionalCheckFailedException`	Bedingte Prüfung fehlgeschlagen. Ein erwartete r Attributwert wurde nicht gefunden.
`ResourceNotFoundException`	Das angegebene Element oder Attribut wurde nicht gefunden.

Beispiele

Beispiele für die Verwendung des AWS SDK finden Sie unter Arbeiten mit Elementen und Attributen in DynamoDB.

Beispielanforderung

```
// This header is abbreviated. For a sample of a complete header, see DynamoDB Low-Level-API.
POST / HTTP/1.1
x-amz-target: DynamoDB_20111205.PutItem
content-type: application/x-amz-json-1.0

{"TableName":"comp5",
 "Item":
  {"time":{"N":"300"},
  "feeling":{"S":"not surprised"},
  "user":{"S":"Riley"}
  },
 "Expected":
  {"feeling":{"Value":{"S":"surprised"},"Exists":true}}
 "ReturnValues":"ALL_OLD"
}
```

Beispielantwort

```
HTTP/1.1 200
x-amzn-RequestId: 8952fa74-71e9-11e0-a498-71d736f27375
content-type: application/x-amz-json-1.0
content-length: 84

{"Attributes":
 {"feeling":{"S":"surprised"},
 "time":{"N":"300"},
 "user":{"S":"Riley"}},
"ConsumedCapacityUnits":1
}
```

Zugehörige Aktionen

- UpdateItem
- DeleteItem
- GetItem
- BatchGetItem

Abfrage

> **Important**
>
> ***Dieser Abschnitt bezieht sich auf die veraltete API-Version 2011-12-05, die nicht für neue Anwendungen verwendet werden sollte.***
> Eine Dokumentation zur aktuellen Low-Level-API finden Sie in der Amazon DynamoDB-API-Referenz.

Beschreibung

Eine `Query`-Operation ruft die Werte eines oder mehrerer Elemente mit den zugehörigen Attributen nach Primärschlüssel ab (`Query` ist nur für Hash- und Bereichsprimärschlüssel-Tabellen verfügbar). Sie müssen einen spezifischen `HashKeyValue` angeben und können den Umfang der Abfrage einschränken, indem Sie Vergleichsoperatoren auf den `RangeKeyValue` des Primärschlüssels anwenden. Verwenden Sie den Parameter `ScanIndexForward`, um Ergebnisse nach Bereichsschlüssel in vorwärts gerichteter oder umgekehrter Reihenfolge zu erhalten.

Abfragen, die keine Ergebnisse zurückgeben, verbrauchen die minimalen Lesekapazitätseinheiten entsprechend dem Lesetyp.

> **Note**
>
> Wenn die Gesamtanzahl der Elemente, die den Abfrageparametern entsprechen, das Limit von 1 MB überschreitet, wird die Abfrage beendet und die Ergebnisse werden an den Benutzer mit einem `LastEvaluatedKey` zurückgegeben. Die Abfrage wird in einer nachfolgenden Operation fortgesetzt. Im Gegensatz zu einer Scan-Operation gibt eine Abfrage-Operation niemals einen leeren Ergebnissatz und einen `LastEvaluatedKey` zurück. Der `LastEvaluatedKey` wird nur bereitgestellt, wenn die Ergebnisse 1 MB überschreiten oder wenn Sie den Parameter `Limit` verwendet haben.
> Das Ergebnis kann für einen Consistent-Lesevorgang mit dem Parameter `ConsistentRead` festgelegt werden.

Anforderungen

Syntax

```
// This header is abbreviated.
// For a sample of a complete header, see DynamoDB Low-Level-API.
POST / HTTP/1.1
x-amz-target: DynamoDB_20111205.Query
content-type: application/x-amz-json-1.0

{"TableName":"Table1",
 "Limit":2,
 "ConsistentRead":true,
 "HashKeyValue":{"S":"AttributeValue1":},
 "RangeKeyCondition": {"AttributeValueList":
[{"N":"AttributeValue2"}],"ComparisonOperator":"GT"}
 "ScanIndexForward":true,
 "ExclusiveStartKey":{
  "HashKeyElement":{"S":"AttributeName1"},
  "RangeKeyElement":{"N":"AttributeName2"}
 },
    "AttributesToGet":["AttributeName1", "AttributeName2", "AttributeName3"]},
}
```

Name	Beschreibung	Erforderlich
`TableName`	Der Name der Tabelle, die die angeforderten Elemente enthält. Typ: Zeichenfolge	Ja
`AttributesToGet`	Array von Attributnamen. Wenn Attributnamen nicht angegeben sind, dann werden alle Attribute zurückgegeben. Wenn einige Attribute nicht gefunden werden, sind sie nicht im Abfrageergebnis enthalten.	Nein

Name	Beschreibung	Erforderlich
	Typ: Array	
`Limit`	Die maximale Anzahl der zurückzugebenden Elemente (nicht notwendigerweise die Anzahl der übereinst immenden Elemente). Wenn DynamoDB bei der Abfrage der Tabelle die Anzahl der Elemente bis zum Limit verarbeitet, wird die Abfrage beendet und es werden die bis zu diesem Punkt übereinst immenden Werte zurückgeg eben. Außerdem wird ein `LastEvaluatedKey` ausgegeben, der bei einer nachfolgenden Operation angewendet wird, um die Abfrage fortzusetzen. Wenn der Ergebnissatz 1 MB überschreitet, bevor DynamoDB dieses Limit erreicht, wird die Abfrage beendet und es werden die übereinstimmenden Werte zurückgegeben. Außerdem wird der `LastEvalu atedKey` ausgegeben, der bei einer nachfolgenden Operation angewendet wird, um mit der Abfrage fortzufah ren. Typ: Zahl	Nein

Name	Beschreibung	Erforderlich
`ConsistentRead`	Wenn auf `true` festgeleg t, dann wird ein Consistent-Lesevorgang ausgegeben, andernfalls wird Eventually Consistent verwendet. Typ: Boolesch	Nein
`Count`	Wenn auf `true` festgeleg t, gibt DynamoDB die Gesamtzahl der Elemente zurück, die mit den Abfragepa rametern übereinstimmen, anstatt eine Liste der übereinst immenden Elemente und der zugehörigen Attribute. Sie können den Parameter `Limit` auf Abfragen anwenden, die nur zählen. Setzen Sie `Count` nicht auf `true`, wenn Sie eine Liste von `AttributesToGet` bereitstellen, sonst gibt DynamoDB einen Validieru ngsfehler zurück. Weitere Informationen finden Sie unter Zählen der Elemente in den Ergebnissen. Typ: Boolesch	Nein

Name	Beschreibung	Erforderlich
`HashKeyValue`	Attributwert der Hash-Komponente des zusammengesetzten Primärschlüssels. Typ: Zeichenfolge, Zahl oder Binärzahl	Ja
`RangeKeyCondition`	Ein Container für die Attributwerte und Vergleichsoperatoren für die Abfrage. Eine Abfrageanforderung erfordert keine `RangeKeyCondition`. Wenn Sie nur den `HashKeyValue` angeben, gibt DynamoDB alle Elemente mit dem angegebenen Hash-Schlüsselelementwert zurück. Typ: Zuordnung	Nein
`RangeKeyCondition : AttributeValueList`	Die Attributwerte zur Bewertung der Abfrageparameter. `AttributeValueList` enthält einen Attributwert, es sei denn, es wurde ein `BETWEEN`-Vergleich angegeben. Für den `BETWEEN`-Vergleich enthält `AttributeValueList` zwei Attributwerte. Typ: Zuordnung von `AttributeValue` zu einer `ComparisonOperator`.	Nein

Name	Beschreibung	Erforderlich
`RangeKeyCondition : ComparisonOperator`	Die Kriterien für die Bewertung der bereitgestellten Attribute , z. B. gleich, größer als usw. Die folgenden Operatoren sind gültige Vergleichsoperatoren für eine Abfrage-Operation. **Note** Vergleiche von Zeichenfolgenwerte n für größer als, gleich oder kleiner als basieren auf ASCII-Zeichensatzwerten. Beispiel: `a` ist größer als `A` und `aa` ist größer als `B`. Eine Liste von Codewerten finden Sie unter http://e n.wikipedia.org/w iki/ASCII#ASCII_p rintable_characters. Beim Binärtyp behandelt DynamoDB jedes Byte der Binärdaten als ohne Vorzeichen, wenn binäre Werte vergliche n werden (z. B. bei der Bewertung von Abfrageausdrücken).	Nein

Name	Beschreibung	Erforderlich
	Typ: Zeichenfolge oder Binärzahl	
	`EQ` : gleich. Bei `EQ` kann `AttributeValueList` nur einen `AttributeValue` vom Typ Zeichenfolge, Zahl oder Binärzahl (keinen Satz) enthalten. Wenn ein Element einen `AttributeValue` eines anderen Typs enthält, als in der Anforderung angegeben, stimmt der Wert nicht überein. Beispiel: `{"S":"6"}` ist nicht gleich `{"N":"6"}` . Auch, `{"N":"6"}` ist nicht gleich `{"NS":["6", "2", "1"]}`.	

Name	Beschreibung	Erforderlich
	`LE` : kleiner als oder gleich. Bei `LE` kann `AttributeValueList` nur einen `AttributeValue` vom Typ Zeichenfolge, Zahl oder Binärzahl (keinen Satz) enthalten. Wenn ein Element einen `AttributeValue` eines anderen Typs enthält, als in der Anforderung angegeben, stimmt der Wert nicht überein. Beispiel: `{"S":"6"}` ist nicht gleich `{"N":"6"}` . Auch, `{"N":"6"}` entsprich t nicht `{"NS":["6", "2", "1"]}`.	

Name	Beschreibung	Erforderlich
	`LT` : kleiner als. Bei `LT` kann `AttributeValueList` nur einen `AttributeValue` vom Typ Zeichenfolge, Zahl oder Binärzahl (keinen Satz) enthalten. Wenn ein Element einen `AttributeValue` eines anderen Typs enthält, als in der Anforderung angegeben, stimmt der Wert nicht überein. Beispiel: `{"S":"6"}` ist nicht gleich `{"N":"6"}` . Auch, `{"N":"6"}` entsprich t nicht `{"NS":["6", "2", "1"]}`.	

Name	Beschreibung	Erforderlich
	`GE` : größer als oder gleich. Bei `GE` kann `AttributeValueList` nur einen `AttributeValue` vom Typ Zeichenfolge, Zahl oder Binärzahl (keinen Satz) enthalten. Wenn ein Element einen `AttributeValue` eines anderen Typs enthält, als in der Anforderung angegeben, stimmt der Wert nicht überein. Beispiel: `{"S":"6"}` ist nicht gleich `{"N":"6"}` . Auch entspricht `{"N":"6"}` nicht `{"NS":["6", "2", "1"]}` .	

Name	Beschreibung	Erforderlich
	`GT` : größer als. Bei `GT` kann `AttributeValueList` nur einen `AttributeValue` vom Typ Zeichenfolge, Zahl oder Binärzahl (keinen Satz) enthalten. Wenn ein Element einen `AttributeValue` eines anderen Typs enthält, als in der Anforderung angegeben, stimmt der Wert nicht überein. Beispiel: `{"S":"6"}` ist nicht gleich `{"N":"6"}` . Auch, `{"N":"6"}` entsprich t nicht `{"NS":["6", "2", "1"]}`.	
	`BEGINS_WITH` : Prüft auf ein Präfix. Bei `BEGINS_WITH` kann `AttributeValueList` nur einen `AttributeValue` vom Typ Zeichenfolge oder Binärzahl (keine Zahl bzw. keinen Satz) enthalten. Das Zielattribut des Vergleichs muss vom Typ Zeichenfolge oder Binärzahl sein (keine Zahl bzw. kein Satz).	

Name	Beschreibung	Erforderlich
	`BETWEEN`: Größer als oder gleich dem ersten Wert und kleiner als oder gleich dem zweiten Wert. Für `BETWEEN` muss `AttributeValueList` zwei `AttributeValue` -Elemente desselben Typs enthalten, und zwar Zeichenfolge, Zahl oder Binärzahl (keinen Satztyp). Es kommt zu einer Übereinstimmung mit dem Zielattribut, wenn der Zielwert größer als oder gleich dem ersten Element und kleiner als oder gleich dem zweiten Element ist. Wenn ein Element einen `AttributeValue` eines anderen Typs enthält, als in der Anforderung angegeben, stimmt der Wert nicht überein. Beispiel: `{"S":"6"}` stimmt nicht mit `{"N":"6"}` überein. Auch entspricht `{"N":"6"}` nicht `{"NS":["6", "2", "1"]}`.	

Name	Beschreibung	Erforderlich
`ScanIndexForward`	Gibt die auf- oder absteigende Traversierung des Index an. DynamoDB gibt Ergebnisse zurück, die der angeforderten Reihenfolge entsprechen, die vom Bereichsschlüssel bestimmt wird: Wenn der Datentyp „Zahl“ lautet, werden die Ergebnisse in numerischer Reihenfolge zurückgegeben, andernfalls basiert der Traversal auf ASCII-Zeichencodewerten. Typ: Boolesch Der Standardwert ist `true` (aufsteigend).	Nein

Name	Beschreibung	Erforderlich
`ExclusiveStartKey`	Der Primärschlüssel des Elements, von dem eine frühere Abfrage fortgesetzt wird. Eine frühere Abfrage kann diesen Wert als `LastEvaluatedKey` bereitstellen, wenn diese Abfrage-Operation vor Abschluss unterbrochen wurde – entweder aufgrund der Größe des Ergebnissatzes oder aufgrund des `Limit`-Parameters. Der `LastEvaluatedKey` kann in einer neuen Abfrageanforderung zurückgegeben werden, um die Operation von diesem Punkt fortzusetzen. Typ: `HashKeyElement` oder `HashKeyElement` und `RangeKeyElement` für einen zusammengesetzten Primärschlüssel.	Nein

Antworten

Syntax

```
HTTP/1.1 200
x-amzn-RequestId: 8966d095-71e9-11e0-a498-71d736f27375
content-type: application/x-amz-json-1.0
content-length: 308

{"Count":2,"Items":[{
    "AttributeName1":{"S":"AttributeValue1"},
```

```
    "AttributeName2":{"N":"AttributeValue2"},
    "AttributeName3":{"S":"AttributeValue3"}
    },{
    "AttributeName1":{"S":"AttributeValue3"},
    "AttributeName2":{"N":"AttributeValue4"},
    "AttributeName3":{"S":"AttributeValue3"},
    "AttributeName5":{"B":"dmFsdWU="}
}],
    "LastEvaluatedKey":{"HashKeyElement":{"AttributeValue3":"S"},
                        "RangeKeyElement":{"AttributeValue4":"N"}
     },
     "ConsumedCapacityUnits":1
}
```

Name	Beschreibung
`Items`	Elementattribute, die den Abfrageparametern entsprechen. Typ: Zuordnung der Attributnamen und ihrer Datentypen und Werte.
`Count`	Anzahl der Elemente in der Antwort. Weitere Informationen finden Sie unter Zählen der Elemente in den Ergebnissen. Typ: Zahl
`LastEvaluatedKey`	Primärschlüssel des Elements, an dem die Abfrage-Operation beendet wurde, einschließlich des vorherigen Ergebnissatzes. Verwenden Sie diesen Wert, um eine neue Operation ohne diesen Wert in der neuen Anforderung zu starten. Der `LastEvaluatedKey` ist `null`, wenn der Ergebnissatz der Abfrage vollständig ist (das heißt, wenn die "letzte Seite" von der Operation verarbeitet wurde).

Name	Beschreibung
	Typ: `HashKeyElement` oder `HashKeyElement` und `RangeKeyElement` für einen zusammengesetzten Primärschlüssel.
`ConsumedCapacityUnits`	Die Anzahl der Lesekapazitätseinheiten, die von der Operation verbraucht werden. Dieser Wert zeigt die Anzahl, die für Ihren bereitgestellten Durchsatz gültig ist. Weitere Informationen finden Sie unter DynamoDB – Modus mit bereitgestellter Kapazität. Typ: Zahl

Spezielle Fehler

Fehler	Beschreibung
`ResourceNotFoundException`	Die angegebene Tabelle wurde nicht gefunden.

Beispiele

Beispiele für die Verwendung des AWS SDK finden Sie unter Abfragen von Tabellen in DynamoDB.

Beispielanforderung

```
// This header is abbreviated. For a sample of a complete header, see DynamoDB Low-Level-API.
POST / HTTP/1.1
x-amz-target: DynamoDB_20111205.Query
content-type: application/x-amz-json-1.0

{"TableName":"1-hash-rangetable",
 "Limit":2,
 "HashKeyValue":{"S":"John"},
 "ScanIndexForward":false,
 "ExclusiveStartKey":{
```

```
  "HashKeyElement":{"S":"John"},
  "RangeKeyElement":{"S":"The Matrix"}
 }
}
```

Beispielantwort

```
HTTP/1.1 200
x-amzn-RequestId: 3647e778-71eb-11e0-a498-71d736f27375
content-type: application/x-amz-json-1.0
content-length: 308

{"Count":2,"Items":[{
 "fans":{"SS":["Jody","Jake"]},
 "name":{"S":"John"},
 "rating":{"S":"***"},
 "title":{"S":"The End"}
 },{
 "fans":{"SS":["Jody","Jake"]},
 "name":{"S":"John"},
 "rating":{"S":"***"},
 "title":{"S":"The Beatles"}
 }],
 "LastEvaluatedKey":{"HashKeyElement":{"S":"John"},"RangeKeyElement":{"S":"The
 Beatles"}},
"ConsumedCapacityUnits":1
}
```

Beispielanforderung

```
// This header is abbreviated. For a sample of a complete header, see DynamoDB Low-
Level-API.
POST / HTTP/1.1
x-amz-target: DynamoDB_20111205.Query
content-type: application/x-amz-json-1.0

{"TableName":"1-hash-rangetable",
 "Limit":2,
 "HashKeyValue":{"S":"Airplane"},
 "RangeKeyCondition":{"AttributeValueList":[{"N":"1980"}],"ComparisonOperator":"EQ"},
 "ScanIndexForward":false}
```

Beispielantwort

```
HTTP/1.1 200
x-amzn-RequestId: 8b9ee1ad-774c-11e0-9172-d954e38f553a
content-type: application/x-amz-json-1.0
content-length: 119

{"Count":1,"Items":[{
 "fans":{"SS":["Dave","Aaron"]},
 "name":{"S":"Airplane"},
 "rating":{"S":"***"},
 "year":{"N":"1980"}
 }],
"ConsumedCapacityUnits":1
}
```

Zugehörige Aktionen

- Scan

Scan

> **Important**
>
> ***Dieser Abschnitt bezieht sich auf die veraltete API-Version 2011-12-05, die nicht für neue Anwendungen verwendet werden sollte.***
> Eine Dokumentation zur aktuellen Low-Level-API finden Sie in der Amazon DynamoDB-API-Referenz.

Beschreibung

Die `Scan`-Operation gibt ein oder mehrere Elemente mit zugehörigen Attributen zurück, indem sie einen vollständigen Scan einer Tabelle durchführt. Geben Sie einen `ScanFilter` an, um spezifischere Ergebnisse zu erhalten.

> **Note**
>
> Wenn die Gesamtanzahl der gescannten Elemente den Grenzwert von 1 MB überschreitet, wird der Scan beendet und die Ergebnisse werden an den Benutzer mit einem

`LastEvaluatedKey` zurückgegeben, um den Scan in einer nachfolgenden Operation fortzusetzen. Die Ergebnisse enthalten auch die Anzahl der Elemente, die den Grenzwert überschreiten. Ein Scan kann dazu führen, dass keine der Tabellendaten die Filterkriterien erfüllen.
Der Ergebnissatz ist letztlich konsistent.

Anforderungen

Syntax

```
// This header is abbreviated.
// For a sample of a complete header, see DynamoDB Low-Level-API.
POST / HTTP/1.1
x-amz-target: DynamoDB_20111205.Scan
content-type: application/x-amz-json-1.0

{"TableName":"Table1",
    "Limit": 2,
    "ScanFilter":{
        "AttributeName1":{"AttributeValueList":
[{"S":"AttributeValue"}],"ComparisonOperator":"EQ"}
    },
    "ExclusiveStartKey":{
        "HashKeyElement":{"S":"AttributeName1"},
        "RangeKeyElement":{"N":"AttributeName2"}
    },
    "AttributesToGet":["AttributeName1", "AttributeName2", "AttributeName3"]},
}
```

Name	Beschreibung	Erforderlich
`TableName`	Der Name der Tabelle, die die angeforderten Elemente enthält. Typ: Zeichenfolge	Ja
`AttributesToGet`	Array von Attributnamen. Wenn Attributnamen nicht	Nein

Name	Beschreibung	Erforderlich
	angegeben sind, dann werden alle Attribute zurückgegeben. Wenn einige Attribute nicht gefunden werden, sind sie nicht im Abfrageergebnis enthalten. Typ: Array	

Name	Beschreibung	Erforderlich
`Limit`	Die maximale Anzahl der auszuwertenden Elemente (nicht notwendigerweise die Anzahl der übereinst immenden Elemente). Wenn DynamoDB bei der Ergebnisv erarbeitung die Anzahl der Elemente bis zum Grenzwert verarbeitet, wird der Vorgang beendet und es werden die bis zu diesem Punkt übereinst immenden Werte zurückgeg eben. Außerdem wird ein `LastEvaluatedKey` ausgegeben, der bei einer nachfolgenden Operation angewendet wird, um weitere Elemente abzurufen. Wenn die gescannten Daten 1 MB überschreiten, bevor DynamoDB diesen Grenzwert erreicht, wird der Scan beendet und es werden die bis zu diesem Grenzwert übereinstimmenden Werte zurückgegeben. Außerdem wird ein`LastEvaluatedKey` ausgegeben, der bei einer nachfolgenden Operation angewendet wird, um mit dem Scan fortzufahren. Typ: Zahl	Nein

Name	Beschreibung	Erforderlich
`Count`	Wenn die Option auf `true` gesetzt ist, gibt DynamoDB für die Scan-Operation alle Elemente zurück, auch wenn die Operation keine übereinst immenden Elemente für den zugeordneten Filter ergibt. Sie können den Limit-Parameter auf Scans anwenden, die nur zählen. Setzen Sie `Count` nicht auf `true`, wenn Sie eine Liste von `AttributesToGet` bereitstellen, sonst gibt DynamoDB einen Validieru ngsfehler zurück. Weitere Informationen finden Sie unter Zählen der Elemente in den Ergebnissen. Typ: Boolesch	Nein

Name	Beschreibung	Erforderlich
`ScanFilter`	Wertet die Scanergeb nisse aus und gibt nur die gewünschten Werte zurück. Mehrere Bedingung en werden als "AND"-Ope rationen behandelt, das heißt, alle Bedingungen müssen erfüllt sein, damit sie in den Ergebnissen berücksichtigt werden. Typ: Zuordnung von Attributn amen zu Werten mit Vergleich soperatoren.	Nein
`ScanFilter :AttributeValueList`	Die Werte und Bedingungen für die Auswertung der Scan-Ergebnisse für den Filter. Typ: Zuordnung von `AttributeValue` zu einer `Condition` .	Nein

Name	Beschreibung	Erforderlich
`ScanFilter : ComparisonOperator`	Die Kriterien für die Bewertung der bereitgestellten Attribute , z. B. gleich, größer als usw. Die folgenden Operatoren sind gültige Vergleichsoperatoren für eine Scan-Operation. **Note** Vergleiche von Zeichenfolgenwerten für größer als, gleich oder kleiner als basieren auf ASCII-Zeichensatzwerten. Beispiel: `a` ist größer als `A` und `aa` ist größer als `B`. Eine Liste von Codewerten finden Sie unter http://en.wikipedia.org/wiki/ASCII#ASCII_printable_characters. Beim Binärtyp behandelt DynamoDB jedes Byte der Binärdaten als ohne Vorzeichen, wenn binäre Werte verglichen werden (z. B. bei der Bewertung von Abfrageausdrücken).	Nein

Name	Beschreibung	Erforderlich
	Typ: Zeichenfolge oder Binärzahl	
	`EQ` : gleich. Bei `EQ` kann `AttributeValueList` nur einen `AttributeValue` vom Typ Zeichenfolge, Zahl oder Binärzahl (keinen Satz) enthalten. Wenn ein Element einen `AttributeValue` eines anderen Typs enthält, als in der Anforderung angegeben, stimmt der Wert nicht überein. Beispiel: `{"S":"6"}` ist nicht gleich `{"N":"6"}` . Auch, `{"N":"6"}` ist nicht gleich `{"NS":["6", "2", "1"]}`.	

Name	Beschreibung	Erforderlich
	`NE`: nicht gleich. Bei `NE` kann `AttributeValueList` nur einen `AttributeValue` vom Typ Zeichenfolge, Zahl oder Binärzahl (keinen Satz) enthalten. Wenn ein Element einen `AttributeValue` eines anderen Typs enthält, als in der Anforderung angegeben, stimmt der Wert nicht überein. Beispiel: `{"S":"6"}` ist nicht gleich `{"N":"6"}` . Auch, `{"N":"6"}` ist nicht gleich `{"NS":["6", "2", "1"]}`.	

Name	Beschreibung	Erforderlich
	`LE` : kleiner als oder gleich. Bei `LE` kann `AttributeValueList` nur einen `AttributeValue` vom Typ Zeichenfolge, Zahl oder Binärzahl (keinen Satz) enthalten. Wenn ein Element einen `AttributeValue` eines anderen Typs enthält, als in der Anforderung angegeben, stimmt der Wert nicht überein. Beispiel: `{"S":"6"}` ist nicht gleich `{"N":"6"}` . Auch, `{"N":"6"}` entsprich t nicht `{"NS":["6", "2", "1"]}`.	

Name	Beschreibung	Erforderlich
	`LT` : kleiner als. Bei `LT` kann `AttributeValueList` nur einen `AttributeValue` vom Typ Zeichenfolge, Zahl oder Binärzahl (keinen Satz) enthalten. Wenn ein Element einen `AttributeValue` eines anderen Typs enthält, als in der Anforderung angegeben, stimmt der Wert nicht überein. Beispiel: `{"S":"6"}` ist nicht gleich `{"N":"6"}` . Auch, `{"N":"6"}` entsprich t nicht `{"NS":["6", "2", "1"]}`.	

Name	Beschreibung	Erforderlich
	`GE` : größer als oder gleich. Bei `GE` kann `AttributeValueList` nur einen `AttributeValue` vom Typ Zeichenfolge, Zahl oder Binärzahl (keinen Satz) enthalten. Wenn ein Element einen `AttributeValue` eines anderen Typs enthält, als in der Anforderung angegeben, stimmt der Wert nicht überein. Beispiel: `{"S":"6"}` ist nicht gleich `{"N":"6"}` . Auch entspricht `{"N":"6"}` nicht `{"NS":["6", "2", "1"]}` .	
	`GT` : größer als. Bei `GT` kann `AttributeValueList` nur einen `AttributeValue` vom Typ Zeichenfolge, Zahl oder Binärzahl (keinen Satz) enthalten. Wenn ein Element einen `AttributeValue` eines anderen Typs enthält, als in der Anforderung angegeben, stimmt der Wert nicht überein. Beispiel: `{"S":"6"}` ist nicht gleich `{"N":"6"}` . Auch entspricht `{"N":"6"}` nicht `{"NS":["6", "2", "1"]}` .	

Name	Beschreibung	Erforderlich
	`NOT_NULL`: Attribut ist vorhanden.	
	`NULL`: Attribut ist nicht vorhanden.	
	`CONTAINS`: Prüft auf eine Teilsequenz oder einen Wert in einem Satz. Bei `CONTAINS` kann `AttributeValueList` nur einen `AttributeValue` vom Typ Zeichenfolge, Zahl oder Binärzahl (keinen Satz) enthalten. Wenn das Zielattribut des Vergleichs eine Zeichenfolge ist, dann prüft die Operation, ob eine Teilzeichenfolge übereinstimmt. Wenn das Zielattribut des Vergleichs vom Typ Binärzahl ist, sucht die Operation nach einer Teilsequenz des Ziels, die mit der Eingabe übereinstimmt. Wenn das Zielattribut des Vergleichs ein Satz ist ("SS", "NS" oder "BS"), prüft die Operation auf ein Mitglied des Satzes (nicht als Teilzeichenfolge).	

Name	Beschreibung	Erforderlich
	`NOT_CONTAINS` : Prüft auf fehlende Teilsequenz oder fehlenden Wert in einem Satz. Bei `NOT_CONTAINS` kann `AttributeValueList` nur einen `AttributeValue` vom Typ Zeichenfolge, Zahl oder Binärzahl (keinen Satz) enthalten. Wenn das Zielattribut des Vergleichs eine Zeichenfolge ist, dann prüft die Operation, ob die Übereinstimmung einer Teilzeichenfolge fehlt. Wenn das Zielattribut des Vergleichs vom Typ Binärzahl ist, prüft die Operation, ob eine Teilsequenz des Ziels, die mit der Eingabe übereinstimmt, fehlt. Wenn das Zielattribut des Vergleichs ein Satz ist ("SS", "NS" oder "BS"), prüft die Operation, ob ein Mitglied des Satzes fehlt (nicht als Teilzeichenfolge).	

Name	Beschreibung	Erforderlich
	`BEGINS_WITH` : Prüft auf ein Präfix. Bei `BEGINS_WITH` kann `AttributeValueList` nur einen `AttributeValue` vom Typ Zeichenfolge oder Binärzahl (keine Zahl bzw. keinen Satz) enthalten. Das Zielattribut des Vergleichs muss vom Typ Zeichenfolge oder Binärzahl sein (keine Zahl bzw. kein Satz).	
	`IN`: Prüft auf genaue Übereinst immungen. Bei `IN` kann `Attribute ValueList` mehr als einen `AttributeValue` vom Typ Zeichenfolge, Zahl oder Binärzahl (keinen Satz) enthalten. Das Zielattribut des Vergleichs muss vom selben Typ sein und mit dem genauen Wert übereinst immen. Eine Zeichenfolge stimmt niemals mit einem Zeichenfolgensatz überein.	

Name	Beschreibung	Erforderlich
	`BETWEEN`: Größer als oder gleich dem ersten Wert und kleiner als oder gleich dem zweiten Wert. Für `BETWEEN` muss `AttributeValueList` zwei `AttributeValue` -Elemente desselben Typs enthalten, und zwar Zeichenfolge, Zahl oder Binärzahl (keinen Satztyp). Es kommt zu einer Übereinstimmung mit dem Zielattribut, wenn der Zielwert größer als oder gleich dem ersten Element und kleiner als oder gleich dem zweiten Element ist. Wenn ein Element einen `AttributeValue` eines anderen Typs enthält, als in der Anforderung angegeben, stimmt der Wert nicht überein. Beispiel: `{"S":"6"}` stimmt nicht mit `{"N":"6"}` überein. Auch entspricht `{"N":"6"}` nicht `{"NS":["6", "2", "1"]}`.	

Name	Beschreibung	Erforderlich
`ExclusiveStartKey`	Der Primärschlüssel des Elements, von dem ein früherer Scan fortgesetzt wird. Ein früherer Scan kann diesen Wert bereitstellen, wenn diese Scan-Operation unterbrochen wurde, bevor die ganze Tabelle gescannt wurde – entweder aufgrund der Größe des Ergebnissatzes oder aufgrund des `Limit`-Parameters. Der `LastEvaluatedKey` kann in einer neuen Scan-Anforderung zurückgegeben werden, um die Operation von diesem Punkt fortzusetzen. Typ: `HashKeyElement` oder `HashKeyElement` und `RangeKeyElement` für einen zusammengesetzten Primärschlüssel.	Nein

Antworten

Syntax

```
HTTP/1.1 200
x-amzn-RequestId: 8966d095-71e9-11e0-a498-71d736f27375
content-type: application/x-amz-json-1.0
content-length: 229

{"Count":2,"Items":[{
    "AttributeName1":{"S":"AttributeValue1"},
    "AttributeName2":{"S":"AttributeValue2"},
```

```
    "AttributeName3":{"S":"AttributeValue3"}
    },{
    "AttributeName1":{"S":"AttributeValue4"},
    "AttributeName2":{"S":"AttributeValue5"},
    "AttributeName3":{"S":"AttributeValue6"},
    "AttributeName5":{"B":"dmFsdWU="}
    }],
    "LastEvaluatedKey":
        {"HashKeyElement":{"S":"AttributeName1"},
        "RangeKeyElement":{"N":"AttributeName2"},
    "ConsumedCapacityUnits":1,
    "ScannedCount":2}
}
```

Name	Beschreibung
`Items`	Container für die Attribute, die mit den Operationsparametern übereinstimmen. Typ: Zuordnung der Attributnamen und ihrer Datentypen und Werte.
`Count`	Anzahl der Elemente in der Antwort. Weitere Informationen finden Sie unter Zählen der Elemente in den Ergebnissen. Typ: Zahl
`ScannedCount`	Anzahl der Elemente im vollständigen Scan, bevor Filter angewendet werden. Ein hoher `ScannedCount` -Wert mit wenigen oder gar keinen `Count`-Ergebnissen weist auf eine ineffiziente Scan-Operation hin. Weitere Informationen finden Sie unter Zählen der Elemente in den Ergebnissen. Typ: Zahl
`LastEvaluatedKey`	Der Primärschlüssel des Elements, an dem die Scan-Operation beendet wurde. Stellen

Name	Beschreibung
	Sie diesen Wert in einer nachfolgenden Scan-Operation bereit, um die Operation von diesem Punkt fortzusetzen. Der `LastEvaluatedKey` ist `null`, wenn der Ergebnissatz der Scan-Operation vollständig ist (das heißt, wenn die "letzte Seite" von der Operation verarbeitet wurde).
`ConsumedCapacityUnits`	Die Anzahl der Lesekapazitätseinheiten, die von der Operation verbraucht werden. Dieser Wert zeigt die Anzahl, die für Ihren bereitges tellten Durchsatz gültig ist. Weitere Informati onen finden Sie unter DynamoDB – Modus mit bereitgestellter Kapazität. Typ: Zahl

Spezielle Fehler

Fehler	Beschreibung
`ResourceNotFoundException`	Die angegebene Tabelle wurde nicht gefunden.

Beispiele

Beispiele für die Verwendung des AWS SDK finden Sie unter Scannen von Tabellen in DynamoDB.

Beispielanforderung

```
// This header is abbreviated. For a sample of a complete header, see DynamoDB Low-Level-API.
POST / HTTP/1.1
x-amz-target: DynamoDB_20111205.Scan
content-type: application/x-amz-json-1.0
```

```
{"TableName":"1-hash-rangetable","ScanFilter":{}}
```

Beispielantwort

```
HTTP/1.1 200
x-amzn-RequestId: 4e8a5fa9-71e7-11e0-a498-71d736f27375
content-type: application/x-amz-json-1.0
content-length: 465

{"Count":4,"Items":[{
 "date":{"S":"1980"},
 "fans":{"SS":["Dave","Aaron"]},
 "name":{"S":"Airplane"},
 "rating":{"S":"***"}
 },{
 "date":{"S":"1999"},
 "fans":{"SS":["Ziggy","Laura","Dean"]},
 "name":{"S":"Matrix"},
 "rating":{"S":"*****"}
 },{
 "date":{"S":"1976"},
 "fans":{"SS":["Riley"]},"
 name":{"S":"The Shaggy D.A."},
 "rating":{"S":"**"}
 },{
 "date":{"S":"1985"},
 "fans":{"SS":["Fox","Lloyd"]},
 "name":{"S":"Back To The Future"},
 "rating":{"S":"****"}
 }],
    "ConsumedCapacityUnits":0.5
 "ScannedCount":4}
```

Beispielanforderung

```
// This header is abbreviated. For a sample of a complete header, see DynamoDB Low-
Level-API.
POST / HTTP/1.1
x-amz-target: DynamoDB_20111205.Scan
content-type: application/x-amz-json-1.0
content-length: 125

{"TableName":"comp5",
```

```
 "ScanFilter":
  {"time":
   {"AttributeValueList":[{"N":"400"}],
   "ComparisonOperator":"GT"}
 }
}
```

Beispielantwort

```
HTTP/1.1 200 OK
x-amzn-RequestId: PD1CQK9QCTERLTJP20VALJ60TRVV4KQNSO5AEMVJF66Q9ASUAAJG
content-type: application/x-amz-json-1.0
content-length: 262
Date: Mon, 15 Aug 2011 16:52:02 GMT

{"Count":2,
 "Items":[
  {"friends":{"SS":["Dave","Ziggy","Barrie"]},
  "status":{"S":"chatting"},
  "time":{"N":"2000"},
  "user":{"S":"Casey"}},
  {"friends":{"SS":["Dave","Ziggy","Barrie"]},
  "status":{"S":"chatting"},
  "time":{"N":"2000"},
  "user":{"S":"Fredy"}
  }],
"ConsumedCapacityUnits":0.5
"ScannedCount":4
}
```

Beispielanforderung

```
// This header is abbreviated. For a sample of a complete header, see DynamoDB Low-
Level-API.
POST / HTTP/1.1
x-amz-target: DynamoDB_20111205.Scan
content-type: application/x-amz-json-1.0

{"TableName":"comp5",
 "Limit":2,
 "ScanFilter":
  {"time":
   {"AttributeValueList":[{"N":"400"}],
```

```
   "ComparisonOperator":"GT"}
 },
 "ExclusiveStartKey":
  {"HashKeyElement":{"S":"Fredy"},"RangeKeyElement":{"N":"2000"}}
}
```

Beispielantwort

```
HTTP/1.1 200 OK
x-amzn-RequestId: PD1CQK9QCTERLTJP20VALJ60TRVV4KQNSO5AEMVJF66Q9ASUAAJG
content-type: application/x-amz-json-1.0
content-length: 232
Date: Mon, 15 Aug 2011 16:52:02 GMT

{"Count":1,
 "Items":[
  {"friends":{"SS":["Jane","James","John"]},
  "status":{"S":"exercising"},
  "time":{"N":"2200"},
  "user":{"S":"Roger"}}
 ],
 "LastEvaluatedKey":{"HashKeyElement":{"S":"Riley"},"RangeKeyElement":{"N":"250"}},
"ConsumedCapacityUnits":0.5
"ScannedCount":2
}
```

Zugehörige Aktionen

- Abfrage
- BatchGetItem

UpdateItem

> **Important**
>
> ***Dieser Abschnitt bezieht sich auf die veraltete API-Version 2011-12-05, die nicht für neue Anwendungen verwendet werden sollte.***
> Eine Dokumentation zur aktuellen Low-Level-API finden Sie in der Amazon DynamoDB-API-Referenz.

Beschreibung

Bearbeitet Attribute eines existierenden Elements. Sie können eine bedingte Aktualisierung durchführen (fügen Sie ein neues Attributnamen-Wert-Paar ein, wenn es noch nicht vorhanden ist, oder ersetzen Sie ein vorhandenes Namen-Wert-Paar, wenn es über bestimmte zu erwartende Attributwerte verfügt).

> **Note**
>
> Sie können die Primärschlüsselattribute mit UpdateItem nicht aktualisieren. Löschen Sie stattdessen das Element und verwenden Sie PutItem zum Erstellen eines neuen Elements mit neuen Attributen.

Die UpdateItem-Operation enthält den Parameter `Action`, der festlegt, wie die Aktualisierung durchgeführt wird. Sie können Attributwerte hinzufügen, löschen oder setzen.

Attributwerte dürfen nicht Null sein, Zeichenketten- und Binärtypattribute müssen Längen haben, die größer als Null sind, und festgelegte Typattribute dürfen nicht leer sein. Anfragen mit leeren Werten werden mit einer `ValidationException` abgelehnt.

Wenn ein vorhandenes Element über den angegebenen Primärschlüssel verfügt:

- PUT – Fügt das angegebene Attribut hinzu. Wenn das Attribut vorhanden ist, wird es durch den neuen Wert ersetzt.
- DELETE – Ist kein Wert angegeben, wird dadurch das Attribut und sein Wert entfernt. Wenn eine Menge von Werten angegeben wird, werden die Werte in dem angegebenen Satz aus dem alten Satz entfernt. Wenn also der Attributwert [a,b,c] und die Löschaktion [a,c] enthält, dann ist der tatsächliche Attributwert [b]. Der Typ des angegebenen Werts muss dem vorhandenen Werttyp entsprechen. Einen leeren Satz anzugeben ist nicht gültig.
- ADD – Verwendung der Add-Aktion nur für Zahlen oder wenn das Zielattribut ein Satz ist (einschließlich Zeichenkettensätze). ADD funktioniert nicht, falls das Zielattribut ein einzelner Zeichenkettenwert oder ein skalarer Binärwert ist. Der angegebene Wert wird einem numerischen Wert hinzugefügt (Erhöhen oder Verringern des vorhandenen numerischen Werts) oder als zusätzlichen Wert in einem Zeichenkettensatz hinzugefügt. Wenn ein Menge von Werten angegeben wird, werden die Werte dem vorhandenen Satz hinzugefügt. Zum Beispiel, wenn der ursprüngliche Satz [1,2] ist und der bereitgestellte Wert [3] ist, dann ist der Satz nach dem Add-Vorgang [1,2,3] und nicht [4,5]. Ein Fehler tritt auf, wenn eine Add-Aktion für ein festgelegtes

Attribut angegeben wird und der angegebene Attributtyp nicht mit dem vorhandenen festgelegten Typ übereinstimmt.

Wenn Sie ADD für ein Attribut verwenden, das nicht existiert, wird das Attribut und seine Werte dem Element hinzugefügt.

Wenn kein Element mit dem angegebenen Primärschlüssel übereinstimmt:

- PUT – Erstellt ein neues Element mit dem angegebenen Primärschlüssel. Fügt dann das angegebene Attribut hinzu.
- DELETE – Nichts passiert.
- ADD – Erstellt ein Element mit einem bereitgestellten Primärschlüssel und einer Zahl (oder eine Menge von Zahlen) für den Attributwert. Nicht gültig für einen Zeichenkette- oder Binärtyp.

> **Note**
>
> Wenn Sie `ADD` einsetzen, um einen Zahlenwert für ein Element, das vor der Aktualisierung nicht vorhanden ist, zu erhöhen oder zu verringern, verwendet DynamoDB `0` als Anfangswert. Wenn Sie außerdem ein Element aktualisieren, indem Sie `ADD` verwenden, um einen Zahlenwert für ein Attribut zu erhöhen oder zu verringern, das vor der Aktualisierung nicht vorhanden ist (das Element ist jedoch vorhanden), verwendet DynamoDB die `0` als Anfangswert. Beispiel: Sie verwenden `ADD`, um `+3` einem Attribut hinzuzufügen, das vor der Aktualisierung nicht vorhanden war. DynamoDB verwendet `0` für den ersten Wert und der Wert nach der Aktualisierung ist `3`.

Weitere Informationen zur Verwendung dieser Operation finden Sie unter Arbeiten mit Elementen und Attributen in DynamoDB.

Anforderungen

Syntax

```
// This header is abbreviated.
// For a sample of a complete header, see DynamoDB Low-Level-API.
POST / HTTP/1.1
x-amz-target: DynamoDB_20111205.UpdateItem
content-type: application/x-amz-json-1.0
```

```
{"TableName":"Table1",
    "Key":
        {"HashKeyElement":{"S":"AttributeValue1"},
        "RangeKeyElement":{"N":"AttributeValue2"}},
    "AttributeUpdates":{"AttributeName3":{"Value":
{"S":"AttributeValue3_New"},"Action":"PUT"}},
    "Expected":{"AttributeName3":{"Value":{"S":"AttributeValue3_Current"}}},
    "ReturnValues":"ReturnValuesConstant"
}
```

Name	Beschreibung	Erforderlich
`TableName`	Der Name der Tabelle, die das zu aktualisierende Element enthält. Typ: Zeichenfolge	Ja
`Key`	Der Primärschlüssel, der das Element definiert. Weitere Informationen zu Primärsch lüsseln finden Sie unter Primärschlüssel. Typ: Zuordnung von `HashKeyElement` zu seinem Wert und von `RangeKeyElement` zu seinem Wert.	Ja
`AttributeUpdates`	Zuordnung des Attributn amens zu dem neuen Wert und der neuen Aktion für die Aktualisierung. Die Attributn amen legen die zu ändernden Attribute fest und können keine Primärschlüsselattribute enthalten.	

Name	Beschreibung	Erforderlich
	Typ: Zuordnung eines Attributnamens, eines Werts und einer Aktion für die Aktualisierung des Attributs.	
`AttributeUpdates:Action`	Gibt an, wie die Aktualisierung durchzuführen ist. Mögliche Werte: `PUT` (Standard) `ADD` oder `DELETE`. Die Semantiken finden Sie in der UpdateItem-Beschreibung. Typ: Zeichenfolge Standard: `PUT`	Nein
`Expected`	Gibt ein Attribut für eine bedingte Aktualisierung an. Der Parameter `Expected` erlaubt es Ihnen, einen Attributnamen anzugeben und festzulegen, ob DynamoDB überprüfen soll, ob der Attributwert bereits vorhanden ist bzw. ob der Attributwert vorhanden ist und über einen bestimmten Wert verfügt, bevor er geändert wird. Typ: Zuordnung von Attributnamen.	Nein
`Expected:AttributeName`	Der Name des Attributs für die bedingte Put-Operation. Typ: Zeichenfolge	Nein

Name	Beschreibung	Erforderlich

`Expected:AttributeName: ExpectedAttributeValue`	Verwenden Sie diesen Parameter, um anzugeben, ob ein Wert für das Attributname-Wert-Paar bereits vorhanden ist oder nicht.
	
	Die folgende JSON-Notation aktualisiert das Element, wenn das Attribut "Farbe" für dieses Element noch nicht vorhanden ist:
	
	`"Expected" : {"Color":{"Exists":false}}`
	
	Die folgende JSON-Notation prüft, ob das Attribut mit dem Namen "Farbe" über einen vorhandenen Wert für "Gelb" verfügt, bevor sie das Element aktualisiert:
	
	`"Expected" : {"Color":{"Exists":true},{"Value":{"S":"Yellow"}}}`
	
	Wenn Sie den Parameter `Expected` verwenden und einen `Value` angeben, geht DynamoDB standardmäßig davon aus, dass das Attribut vorhanden ist und einen zu ersetzenden aktuellen Wert hat. Sie müssen `{"Exists"`
		Nein

Name	Beschreibung	Erforderlich
	`:true}` demnach nicht angeben, weil er enthalten ist. Sie können die Anforderung verkürzen, um: `"Expected" : {"Color":{"Value": {"S":"Yellow"}}}` **Note** Wenn Sie `{"Exists" :true}` ohne einen zu prüfenden Attributwert angeben, gibt DynamoDB einen Fehler zurück.	

Name	Beschreibung	Erforderlich
`ReturnValues`	Verwenden Sie diesen Parameter, wenn Sie die Attribut-Namen-Wert-Paare erhalten möchten, bevor sie mit der `UpdateItem` -Anforderung aktualisiert wurden. Mögliche Parameterwerte sind `NONE` (Standard) oder `ALL_OLD`, `UPDATED_OLD` , `ALL_NEW` oder `UPDATED_NEW` . Wenn `ALL_OLD` angegeben ist und ein Attribut-Namen-Wert-Paar durch `UpdateItem` überschrieben wurde, wird der Inhalt des alten Elements zurückgegeben. Wenn dieser Parameter nicht angegeben wird oder `NONE` ist, wird nichts zurückgegeben. Wenn `ALL_NEW` angegeben wird, werden alle Attribute der neuen Version des Elements zurückgegeben. Wenn `UPDATED_NEW` angegeben wird, werden ausschließlich die neuen Versionen der aktualisierten Attribute zurückgegeben. Typ: Zeichenfolge	Nein

Antworten

Syntax

In dem folgenden Syntax-Beispiel wird davon ausgegangen, dass die Anforderung für den Parameter `ReturnValues` `ALL_OLD` angegeben hat; andernfalls beinhaltet die Antwort nur das `ConsumedCapacityUnits`-Element.

```
HTTP/1.1 200
x-amzn-RequestId: 8966d095-71e9-11e0-a498-71d736f27375
content-type: application/x-amz-json-1.0
content-length: 140

{"Attributes":{
 "AttributeName1":{"S":"AttributeValue1"},
 "AttributeName2":{"S":"AttributeValue2"},
 "AttributeName3":{"S":"AttributeValue3"},
 "AttributeName5":{"B":"dmFsdWU="}
 },
"ConsumedCapacityUnits":1
}
```

Name	Beschreibung
`Attributes`	Eine Zuordnung von Attribut-Namen-Wert-Paaren, aber nur, wenn der Parameter `ReturnValues` in der Anforderung anders angegeben wird als `NONE`. Typ: Zuordnung von Attribut-Namen-Wert-Paare.
`ConsumedCapacityUnits`	Die Anzahl der Schreibkapazitätseinheiten, die von dem Vorgang verbraucht werden. Dieser Wert zeigt die Anzahl, die für Ihren bereitgestellten Durchsatz gültig ist. Weitere Informationen finden Sie unter DynamoDB – Modus mit bereitgestellter Kapazität. Typ: Zahl

Spezielle Fehler

Fehler	Beschreibung
`ConditionalCheckFailedException`	Bedingte Prüfung fehlgeschlagen. Der Attributwert ("+ name +") ist ("+ value +"), aber ("+ expValue +") wurde erwartet
`ResourceNotFoundExceptions`	Das angegebene Element oder Attribut wurde nicht gefunden.

Beispiele

Beispiele für die Verwendung des AWS SDK finden Sie unter Arbeiten mit Elementen und Attributen in DynamoDB.

Beispielanforderung

```
// This header is abbreviated. For a sample of a complete header, see DynamoDB Low-Level-API.
POST / HTTP/1.1
x-amz-target: DynamoDB_20111205.UpdateItem
content-type: application/x-amz-json-1.0

{"TableName":"comp5",
    "Key":
        {"HashKeyElement":{"S":"Julie"},"RangeKeyElement":{"N":"1307654350"}},
    "AttributeUpdates":
        {"status":{"Value":{"S":"online"},
        "Action":"PUT"}},
    "Expected":{"status":{"Value":{"S":"offline"}}},
    "ReturnValues":"ALL_NEW"
}
```

Beispielantwort

```
HTTP/1.1 200 OK
x-amzn-RequestId: 5IMHO7F01Q9P7Q6QMKMMI3R3QRVV4KQNSO5AEMVJF66Q9ASUAAJG
content-type: application/x-amz-json-1.0
content-length: 121
```

```
Date: Fri, 26 Aug 2011 21:05:00 GMT

{"Attributes":
    {"friends":{"SS":["Lynda, Aaron"]},
    "status":{"S":"online"},
    "time":{"N":"1307654350"},
    "user":{"S":"Julie"}},
"ConsumedCapacityUnits":1
}
```

Zugehörige Aktionen

- PutItem
- DeleteItem

UpdateTable

> **Important**
>
> ***Dieser Abschnitt bezieht sich auf die veraltete API-Version 2011-12-05, die nicht für neue Anwendungen verwendet werden sollte.***
> Eine Dokumentation zur aktuellen Low-Level-API finden Sie in der Amazon DynamoDB-API-Referenz.

Beschreibung

Aktualisiert den bereitgestellten Durchsatz für die jeweilige Tabelle. Das Festlegen des Durchsatzes für eine Tabelle unterstützt Sie beim Verwalten der Leistung und ist Teil der Funktion des bereitgestellten Durchsatzes von DynamoDB. Weitere Informationen finden Sie unter DynamoDB – Modus mit bereitgestellter Kapazität.

Die bereitgestellten Durchsatzwerte können basierend auf den Maximal- und Mindestwerten, aufgeführt in Kontingente in Amazon DynamoDB, aktualisiert oder herabgestuft werden.

Die Tabelle muss sich im Status `ACTIVE` befinden, damit diese Operation erfolgreich ist. "UpdateTable" ist eine asynchrone Operation; während der Ausführung der Operation befindet sich die Tabelle im Status `UPDATING`. Während sich die Tabelle im Status `UPDATING` befindet, verfügt sie weiterhin über den bereitgestellten Durchsatz von vor dem Aufruf. Die neue Einstellung des

bereitgestellten Durchsatzes gilt nur, wenn die Tabelle nach der "UpdateTable"-Operation wieder in den Status `ACTIVE` zurückkehrt.

Anforderungen

Syntax

```
// This header is abbreviated.
// For a sample of a complete header, see DynamoDB Low-Level-API.
POST / HTTP/1.1
x-amz-target: DynamoDB_20111205.UpdateTable
content-type: application/x-amz-json-1.0

{"TableName":"Table1",
    "ProvisionedThroughput":{"ReadCapacityUnits":5,"WriteCapacityUnits":15}
}
```

Name	Beschreibung	Erforderlich
`TableName`	Der Name der zu erstellenden Tabelle. Typ: Zeichenfolge	Ja
`ProvisionedThroughput`	Neuer Durchsatz für die angegebene Tabelle, bestehend aus Werten für `ReadCapacityUnits` und `WriteCapacityUnits`. Siehe DynamoDB – Modus mit bereitgestellter Kapazität. Typ: Array	Ja
`ProvisionedThroughput:ReadCapacityUnits`	Legt die Mindestanzahl von konsistenten `ReadCapacityUnits` fest, die pro Sekunde für die angegebene Tabelle verbraucht wird,	Ja

Name	Beschreibung	Erforderlich
	bevor DynamoDB die Last mit anderen Operationen ausgleicht. Eventually Consistent-Leseoperationen erfordern weniger Aufwand als Consistent-Leseoperationen. Daher stellt die Festlegung von 50 konsistenten `ReadCapacityUnits` pro Sekunde 100 Eventually Consistent-`ReadCapacityUnits` pro Sekunde bereit. Typ: Zahl	
`ProvisionedThroughput` :`WriteCapacityUnits`	Legt die Mindestanzahl von `WriteCapacityUnits` fest, die pro Sekunde für die angegebene Tabelle verbraucht wird, bevor DynamoDB die Last mit anderen Operationen ausgleicht. Typ: Zahl	Ja

Antworten

Syntax

```
HTTP/1.1 200 OK
x-amzn-RequestId: CSOC7TJPLR0OOKIRLGOHVAICUFVV4KQNSO5AEMVJF66Q9ASUAAJG
Content-Type: application/json
```

```
Content-Length: 311
Date: Tue, 12 Jul 2011 21:31:03 GMT

{"TableDescription":
    {"CreationDateTime":1.321657838135E9,
    "KeySchema":
        {"HashKeyElement":{"AttributeName":"AttributeValue1","AttributeType":"S"},
        "RangeKeyElement":{"AttributeName":"AttributeValue2","AttributeType":"N"}},
    "ProvisionedThroughput":
        {"LastDecreaseDateTime":1.321661704489E9,
        "LastIncreaseDateTime":1.321663607695E9,
        "ReadCapacityUnits":5,
        "WriteCapacityUnits":10},
    "TableName":"Table1",
    "TableStatus":"UPDATING"}}
```

Name	Beschreibung
`CreationDateTime`	Das Datum, an dem die Tabelle erstellt wurde. Typ: Zahl
`KeySchema`	Die Struktur (einfach oder zusammengesetzt) des Primärschlüssels für die Tabelle. Ein Name-Wert-Paar ist für das `HashKeyElement` und optional für das `RangeKeyElement` erforderlich (nur für zusammengesetzte Primärschlüssel erforderlich). Die maximale Hash-Schlüsselgröße ist 2048 Byte. Die maximale Range-Schlüsselgröße ist 1024 Byte. Beide Grenzen werden separat durchgesetzt (d. h. Sie können einen kombinierten Hash + Range 2048 + 1024-Schlüssel haben). Weitere Informationen zu Primärschlüsseln finden Sie unter Primärschlüssel. Typ: Zuordnung von `HashKeyElement` oder `HashKeyElement` und `RangeKeyElement` für einen zusammengesetzten Primärschlüssel.

Name	Beschreibung
`ProvisionedThroughput`	Aktuelle Durchsatzeinstellungen für die angegebene Tabelle, einschließlich Werte für `LastIncreaseDateTime` (falls zutreffend), `LastDecreaseDateTime` (falls zutreffend), Typ: Array
`TableName`	Der Name der aktualisierten Tabelle. Typ: Zeichenfolge
`TableStatus`	Der aktuelle Status der Tabelle (`CREATING`, `ACTIVE`, `DELETING` oder `UPDATING`), der `UPDATING` sein sollte. Verwenden Sie die DescribeTables-Operation, um den Status der Tabelle zu überprüfen. Typ: Zeichenfolge

Spezielle Fehler

Fehler	Beschreibung
`ResourceNotFoundException`	Die angegebene Tabelle wurde nicht gefunden.
`ResourceInUseException`	Die Tabelle befindet sich nicht im Status `ACTIVE`.

Beispiele

Beispielanforderung

```
// This header is abbreviated.
// For a sample of a complete header, see DynamoDB Low-Level-API.
POST / HTTP/1.1
```

```
x-amz-target: DynamoDB_20111205.UpdateTable
content-type: application/x-amz-json-1.0

{"TableName":"comp1",
    "ProvisionedThroughput":{"ReadCapacityUnits":5,"WriteCapacityUnits":15}
}
```

Beispielantwort

```
HTTP/1.1 200 OK
content-type: application/x-amz-json-1.0
content-length: 390
Date: Sat, 19 Nov 2011 00:46:47 GMT

{"TableDescription":
    {"CreationDateTime":1.321657838135E9,
    "KeySchema":
        {"HashKeyElement":{"AttributeName":"user","AttributeType":"S"},
        "RangeKeyElement":{"AttributeName":"time","AttributeType":"N"}},
    "ProvisionedThroughput":
        {"LastDecreaseDateTime":1.321661704489E9,
        "LastIncreaseDateTime":1.321663607695E9,
        "ReadCapacityUnits":5,
        "WriteCapacityUnits":10},
    "TableName":"comp1",
    "TableStatus":"UPDATING"}
}
```

Zugehörige Aktionen

- CreateTable
- DescribeTables
- DeleteTable

Bedingte Legacy-Parameter für DynamoDB

Dieses Dokument bietet einen Überblick über bedingte Legacy-Parameter in DynamoDB und empfiehlt, stattdessen die neuen Ausdrucksparameter zu verwenden. Es behandelt Details zu Parametern wie AttributesToGet,, AttributeUpdates ConditionalOperator, Expected KeyConditions,

QueryFilter ScanFilter, und und bietet Beispiele dafür, wie die neuen Ausdrucksparameter als Ersatz verwendet werden können.

> **Important**
>
> Es wird empfohlen, nach Möglichkeit die neuen Ausdrucksparameter anstelle der alten Parameter zu verwenden. Weitere Informationen finden Sie unter Verwenden von Ausdrücken in DynamoDB.
> DynamoDB lässt keine Mischung aus bedingten Legacy-Parametern und Ausdrucksparametern in einem einzigen Aufruf zu. Das Aufrufen der `Query`-Operation mit `AttributesToGet` und `ConditionExpression` löst beispielsweise einen Fehler aus.

Die folgende Tabelle zeigt die DynamoDB-APIs, die die älteren Parameter nach wie vor unterstützen, und enthält Angaben dazu, welche Ausdrucksparameter stattdessen zu verwenden sind. Diese Tabelle kann hilfreich sein, wenn Sie Ihre so Anwendungen aktualisieren möchten, dass sie Ausdrucksparameter verwenden.

Wenn Sie diesen API-Vorgang verwenden…	Mit diesen Legacy-Parametern ...	Nutzen Sie diesen Ausdrucksparameter als Alternative
`BatchGetItem`	`AttributesToGet`	`ProjectionExpression`
`DeleteItem`	`Expected`	`ConditionExpression`
`GetItem`	`AttributesToGet`	`ProjectionExpression`
`PutItem`	`Expected`	`ConditionExpression`
`Query`	`AttributesToGet`	`ProjectionExpression`
	`KeyConditions`	`KeyConditionExpression`
	`QueryFilter`	`FilterExpression`
`Scan`	`AttributesToGet`	`ProjectionExpression`
	`ScanFilter`	`FilterExpression`

Wenn Sie diesen API-Vorgang verwenden…	Mit diesen Legacy-Parametern ...	Nutzen Sie diesen Ausdrucksparameter als Alternative
`UpdateItem`	`AttributeUpdates`	`UpdateExpression`
	`Expected`	`ConditionExpression`

In den folgenden Abschnitten finden Sie weitere Informationen zu bedingten Legacy-Parametern.

Themen

- AttributesToGet (altmodisch)
- AttributeUpdates (Vermächtnis)
- ConditionalOperator (Vermächtnis)
- Expected (veraltet)
- KeyConditions (Vermächtnis)
- QueryFilter (Vermächtnis)
- ScanFilter (Vermächtnis)
- Schreiben von Bedingungen mit Legacy-Parametern

AttributesToGet (altmodisch)

> **Note**
>
> Es wird empfohlen, nach Möglichkeit die neuen Ausdrucksparameter anstelle der alten Parameter zu verwenden. Weitere Informationen finden Sie unter Verwenden von Ausdrücken in DynamoDB. Spezifische Informationen zu dem neuen Parameter, der diesen ersetzt, finden Sie unter Stattdessen ProjectionExpression verwenden.

Der ältere Bedingungsparameter `AttributesToGet` ist ein Array von einem oder mehreren Attributen zum Abrufen von Daten aus DynamoDB. Wenn keine Attributnamen angegeben sind, werden alle Attribute zurückgegeben. Wenn eines der angeforderten Attribute nicht gefunden wird, ist es nicht im Abfrageergebnis enthalten.

Mit `AttributesToGet` können Sie Attribute vom Typ Liste oder Mapping abrufen. Es können jedoch keine einzelnen Elemente in einer Liste oder einem Mapping abgerufen werden.

Beachten Sie, dass `AttributesToGet` keine Auswirkung auf den Verbrauch des bereitgestellten Durchsatzes hat. DynamoDB ermittelt die verbrauchten Kapazitätseinheiten basierend auf der Elementgröße, nicht anhand der Menge der Daten, die an eine Anwendung zurückgegeben werden.

ProjectionExpressionStattdessen verwenden — Beispiel

Angenommen, Sie möchten ein Element aus der Tabelle Music abrufen, wobei aber nur einige der Attribute zurückgegeben werden sollen. Sie könnten eine `GetItem` Anfrage mit einem `AttributesToGet` Parameter verwenden, wie in diesem AWS CLI Beispiel:

```
aws dynamodb get-item \
    --table-name Music \
    --attributes-to-get '["Artist", "Genre"]' \
    --key '{
        "Artist": {"S":"No One You Know"},
        "SongTitle": {"S":"Call Me Today"}
    }'
```

Sie können stattdessen `ProjectionExpression` verwenden.

```
aws dynamodb get-item \
    --table-name Music \
    --projection-expression "Artist, Genre" \
    --key '{
        "Artist": {"S":"No One You Know"},
        "SongTitle": {"S":"Call Me Today"}
    }'
```

AttributeUpdates (Vermächtnis)

> **Note**
>
> Es wird empfohlen, nach Möglichkeit die neuen Ausdrucksparameter anstelle der alten Parameter zu verwenden. Weitere Informationen finden Sie unter Verwenden von Ausdrücken in DynamoDB. Spezifische Informationen zu dem neuen Parameter, der diesen ersetzt, finden Sie unter UpdateExpressionstattdessen verwenden..

In einer `UpdateItem`-Operation enthält der ältere Bedingungsparameter `AttributeUpdates` die Namen der zu ändernden Attribute, die für jedes Attribut auszuführende Aktion und den jeweils neuen Wert. Wenn Sie ein Attribut aktualisieren, bei dem es sich um ein Indexschlüsselattribut für Indexe dieser Tabelle handelt, muss der Attributtyp mit dem in der `AttributesDefinition` der Tabellenbeschreibung definierten Indexschlüsseltyp übereinstimmen. Mit `UpdateItem` können Sie Nicht-Schlüsselattribute aktualisieren.

Attributwerte dürfen nicht Null sein. Attribute vom Typ Zeichenfolge und Binärwert müssen Längen haben, die größer als Null sind. Attribute vom Typ Satz dürfen nicht leer sein. Anfragen mit leeren Werten werden mit einer `ValidationException`-Ausnahme abgelehnt.

Jedes `AttributeUpdates`-Element besteht aus einem zu ändernden Attributnamen zusammen mit Folgendem:

- `Value` – Der neue Wert, falls zutreffend, für dieses Attribut.
- `Action` – Ein Wert, der angibt, wie die Aktualisierung durchzuführen ist. Diese Aktion ist nur für ein vorhandenes Attribut gültig, dessen Datentyp Zahl ist, oder das ein Satz ist. Verwenden Sie `ADD` nicht für andere Datentypen.

  Wenn ein Element mit dem angegebenen Primärschlüssel in der Tabelle gefunden wird, führen die folgenden Werte die nachstehenden Aktionen aus:

  - `PUT` – Fügt dem Element das angegebene Attribut hinzu. Wenn das Attribut bereits vorhanden ist, wird es durch den neuen Wert ersetzt.
  - `DELETE` – Entfernt das Attribut und den Wert, wenn für `DELETE` kein Wert angegeben ist. Der Datentyp des angegebenen Werts muss dem Datentyp des vorhandenen Werts entsprechen.

    Wenn ein Satz von Werten angegeben wird, müssen diese Werte vom alten Satz abgezogen werden. Beispiel: Wenn der Attributwert der Satz `[a,b,c]` war und die `DELETE`-Aktion `[a,c]` angibt, ist der endgültige Attributwert `[b]`. Einen leeren Satz anzugeben, führt zu einem Fehler.
  - `ADD` – Fügt dem Element den angegebenen Wert hinzu, wenn das Attribut nicht bereits vorhanden ist. Wenn das Attribut nicht vorhanden ist, hängt das Verhalten von `ADD` vom Datentyp des Attributs ab:
    - Wenn das vorhandene Attribut und `Value` jeweils eine Zahl ist, dann wird `Value` mathematisch zum vorhandenen Attribut addiert. Wenn `Value` eine negative Zahl ist, wird sie von dem vorhandenen Attribut abgezogen.

> **Note**
>
> Wenn Sie `ADD` einsetzen, um einen Zahlenwert für ein Element, das vor der Aktualisierung nicht vorhanden ist, zu erhöhen oder zu verringern, verwendet DynamoDB `0` als Anfangswert.
> Wenn Sie `ADD` für ein vorhandenes Element nutzen, um einen Attributwert, der vor der Aktualisierung nicht vorhanden ist, zu erhöhen oder zu verringern, verwendet DynamoDB `0` als Anfangswert. Angenommen, das Element, das Sie aktualisieren möchten, hat kein Attribut mit dem Namen itemcount, aber Sie möchten trotzdem die Zahl `ADD` mit `3` zu diesem Attribut addieren. DynamoDB erstellt das itemcount-Attribut, legt den Anfangswert auf `0` fest und addiert `3`. Das Ergebnis ist ein neues itemcount-Attribut mit dem Wert `3`.

- Wenn der vorhandene Datentyp ein Satz ist und `Value` ebenfalls ein Satz ist, dann wird `Value` dem vorhandenen Satz angefügt. Beispiel: Wenn der Attributwert der Satz `[1,2]` ist und die `ADD`-Aktion `[3]` angibt, ist der endgültige Attributwert `[1,2,3]`. Ein Fehler tritt auf, wenn eine `ADD`-Aktion für ein festgelegtes Attribut angegeben wird und der angegebene Attributtyp nicht mit dem vorhandenen Satztyp übereinstimmt.

  Beide Sätze müssen denselben primitiven Datentyp besitzen. Wenn es sich bei dem vorhandenen Datentyp beispielsweise um einen Satz von Zeichenfolgen handelt, muss `Value` ebenfalls ein Zeichenfolgensatz sein.

Wenn kein Element mit dem angegebenen Primärschlüssel in der Tabelle gefunden wird, führen die folgenden Werte die nachstehenden Aktionen aus:

- `PUT` – Veranlasst DynamoDB, ein neues Element mit dem angegebenen Primärschlüssel zu erstellen, und fügt dann das Attribut hinzu.
- `DELETE` – Nichts passiert, da Attribute nicht aus einem nicht vorhandenen Element gelöscht werden können. Die Operation ist erfolgreich, aber DynamoDB erstellt kein neues Element.
- `ADD` – Veranlasst DynamoDB, ein Element mit dem bereitgestellten Primärschlüssel und einer Zahl (oder Zahlensätzen) für den Attributwert zu erstellen. Die einzigen zulässigen Datentypen sind Zahl und Zahlensatz.

Wenn Sie Attribute, die Teil eines Indexschlüssels sind, bereitstellen, müssen die Datentypen dieser Attribute mit den Typen des Schemas in der Attributdefinition der Tabelle übereinstimmen.

UpdateExpressionStattdessen verwenden — Beispiel

Angenommen, Sie möchten ein Element in der Tabelle Music ändern. Sie könnten eine `UpdateItem` Anfrage mit einem `AttributeUpdates` Parameter verwenden, wie in diesem AWS CLI Beispiel:

```
aws dynamodb update-item \
    --table-name Music \
    --key '{
        "SongTitle": {"S":"Call Me Today"},
        "Artist": {"S":"No One You Know"}
    }' \
    --attribute-updates '{
        "Genre": {
            "Action": "PUT",
            "Value": {"S":"Rock"}
        }
    }'
```

Sie können stattdessen `UpdateExpression` verwenden.

```
aws dynamodb update-item \
    --table-name Music \
    --key '{
        "SongTitle": {"S":"Call Me Today"},
        "Artist": {"S":"No One You Know"}
    }' \
    --update-expression 'SET Genre = :g' \
    --expression-attribute-values '{
        ":g": {"S":"Rock"}
    }'
```

ConditionalOperator (Vermächtnis)

> **Note**
>
> Es wird empfohlen, nach Möglichkeit die neuen Ausdrucksparameter anstelle der alten Parameter zu verwenden. Weitere Informationen finden Sie unter Verwenden von Ausdrücken in DynamoDB.

Der alte Bedingungsparameter `ConditionalOperator` ist ein logischer Operator, der auf die Bedingungen in einer `Expected`-, `QueryFilter`- oder `ScanFilter`-Zuordnung anzuwenden ist:

- AND – Wenn alle Bedingungen mit True ausgewertet werden, wird die gesamte Zuordnung mit True ausgewertet.
- OR – Wenn mindestens eine Bedingung mit True ausgewertet wird, wird die gesamte Zuordnung mit True ausgewertet.

Wenn Sie `ConditionalOperator` nicht angeben, ist AND die Standardeinstellung.

Die Operation wird nur dann erfolgreich ausgeführt, wenn die gesamte Zuordnung mit True ausgewertet wird.

> **Note**
>
> Dieser Parameter unterstützt keine Attribute vom Typ Liste oder Zuordnung.

Expected (veraltet)

> **Note**
>
> Es wird empfohlen, nach Möglichkeit die neuen Ausdrucksparameter anstelle der alten Parameter zu verwenden. Weitere Informationen finden Sie unter Verwenden von Ausdrücken in DynamoDB. Spezifische Informationen zu dem neuen Parameter, der diesen ersetzt, finden Sie unter ConditionExpressionstattdessen verwenden..

Der ältere bedingte Parameter `Expected` ist ein bedingter Block für eine `UpdateItem` Operation. `Expected`dist eine Karte von attribute/condition Paaren. Jedes Element des Mappings besteht aus einem Attributnamen, einem Vergleichsoperator und mindestens einem Wert. DynamoDB vergleicht das Attribut anhand des Vergleichsoperators mit dem bzw. den angegebenen Werten. Für jedes `Expected`-Element lautet das Ergebnis der Auswertung entweder True oder False.

Wenn Sie mehr als ein Element im `Expected`-Mapping angeben, müssen alle Bedingungen standardmäßig mit True ausgewertet werden. Mit anderen Worten: Die Bedingungen werden mit dem AND-Operator kombiniert. (Sie können das`ConditionalOperator`-Parameter auf ODER

zu den Bedingungen verwenden. In diesem Fall müssen nicht alle, sondern mindestens eine der Bedingungen mit True ausgewertet werden.)

Wenn die `Expected`-Zuordnung mit True ausgewertet wird, ist die bedingte Operation erfolgreich; andernfalls schlägt sie fehl.

`Expected` enthält Folgendes:

- `AttributeValueList` – Ein oder mehrere Werte, die anhand des angegebenen Attributs ausgewertet werden sollen. Die Anzahl der Werte in der Liste hängt vom verwendeten `ComparisonOperator` ab.

  Beim Zahlentyp sind Wertevergleiche numerisch.

  Vergleiche von Zeichenfolgenwerten für größer als, gleich oder kleiner als basieren auf Unicode mit UTF-8-Binärkodierung. Beispiel: `a` ist größer als `A` und `a` ist größer als `B`.

  Beim Binärtyp betrachtet DynamoDB jedes Byte der Binärdaten beim Vergleichen der binären Werte ohne Vorzeichen.
- `ComparisonOperator` – Ein Vergleichsoperator zum Auswerten der Attribute in der `AttributeValueList`. Für den Vergleich verwendet DynamoDB Strongly Consistent-Lesevorgänge.

  Die folgenden Vergleichsoperatoren sind verfügbar:

  ```
  EQ | NE | LE | LT | GE | GT | NOT_NULL | NULL | CONTAINS | NOT_CONTAINS |
  BEGINS_WITH | IN | BETWEEN
  ```

  Es folgen Beschreibungen der einzelnen Vergleichsoperatoren.

  - `EQ`: Gleich. `EQ` wird für alle Datentypen, einschließlich Listen und Zuordnungen, unterstützt.

    `AttributeValueList` kann nur ein `AttributeValue`-Element vom Typ Zeichenfolge, Zahl, Binärwert, Zeichenfolgensatz, Zahlensatz und Binärwertesatz sein. Wenn ein Element ein `AttributeValue`-Element eines anderen Typs enthält, als in der Anforderung angegeben, stimmt der Wert nicht überein. Zum Beispiel, `{"S":"6"}` ist nicht gleich `{"N":"6"}`. Auch, `{"N":"6"}` ist nicht gleich `{"NS":["6", "2", "1"]}`.
  - `NE`: Nicht gleich. `NE` wird für alle Datentypen, einschließlich Listen und Zuordnungen, unterstützt.

    `AttributeValueList` kann nur ein `AttributeValue` vom Typ Zeichenfolge, Zahl, Binärwert, Zeichenfolgensatz, Zahlensatz und Binärwertesatz sein. Wenn ein Element einen

`AttributeValue` eines anderen Typs enthält, als in der Anforderung angegeben, stimmt der Wert nicht überein. Beispiel: `{"S":"6"}` ist nicht gleich `{"N":"6"}`. Auch, `{"N":"6"}` ist nicht gleich `{"NS":["6", "2", "1"]}`.

- `LE` : kleiner als oder gleich.

  `AttributeValueList` kann nur ein `AttributeValue`-Element vom Typ Zeichenfolge, Zahl oder Binärwert (kein Satztyp) sein. Wenn ein Element ein `AttributeValue`-Element eines anderen Typs enthält, als in der Anforderung angegeben, stimmt der Wert nicht überein. Beispiel: `{"S":"6"}` ist nicht gleich `{"N":"6"}`. Auch, `{"N":"6"}` entspricht nicht `{"NS":["6", "2", "1"]}`.

- `LT` : kleiner als.

  `AttributeValueList` kann nur ein `AttributeValue` vom Typ Zeichenfolge, Zahl oder Binärwert (kein Satztyp) sein. Wenn ein Element ein `AttributeValue`-Element eines anderen Typs enthält, als in der Anforderung angegeben, stimmt der Wert nicht überein. Beispiel: `{"S":"6"}` ist nicht gleich `{"N":"6"}`. Auch, `{"N":"6"}` entspricht nicht `{"NS":["6", "2", "1"]}`.

- `GE` : größer als oder gleich.

  `AttributeValueList` kann nur ein `AttributeValue`-Element vom Typ Zeichenfolge, Zahl oder Binärwert (kein Satztyp) sein. Wenn ein Element ein `AttributeValue`-Element eines anderen Typs enthält, als in der Anforderung angegeben, stimmt der Wert nicht überein. Beispiel: `{"S":"6"}` ist nicht gleich `{"N":"6"}`. Auch entspricht `{"N":"6"}` nicht `{"NS":["6", "2", "1"]}`.

- `GT` : größer als.

  `AttributeValueList` kann nur ein `AttributeValue`-Element vom Typ Zeichenfolge, Zahl oder Binärwert (kein Satztyp) sein. Wenn ein Element ein `AttributeValue`-Element eines anderen Typs enthält, als in der Anforderung angegeben, stimmt der Wert nicht überein. Beispiel: `{"S":"6"}` ist nicht gleich `{"N":"6"}`. Auch entspricht `{"N":"6"}` nicht `{"NS":["6", "2", "1"]}`.

- `NOT_NULL`: Das Attribut ist vorhanden. `NOT_NULL` wird für alle Datentypen, einschließlich Listen und Zuordnungen, unterstützt.

> **Note**
>
> Dieser Operator prüft das Vorhandensein eines Attributs, nicht seines Datentyps. Wenn der Datentyp des Attributs „a“ Null ist und Sie ihn mit `NOT_NULL` auswerten, ist das Ergebnis ein Boolescher Wert `true`. Dieses Ergebnis ist darauf zurückzuführen, dass das Attribut „a“ vorhanden ist. Sein Datentyp ist für den Vergleichsoperator `NOT_NULL` nicht relevant.

- `NULL`: Das Attribut ist nicht vorhanden. `NULL` wird für alle Datentypen, einschließlich Listen und Zuordnungen, unterstützt.

  > **Note**
  >
  > Dieser Operator prüft das Nichtvorhandensein eines Attributs, nicht seines Datentyps. Wenn der Datentyp des Attributs „a“ Null ist und Sie ihn mit `NULL` auswerten, ist das Ergebnis ein Boolescher Wert `false`. Dies ist darauf zurückzuführen, dass das Attribut „a“ vorhanden ist. Sein Datentyp ist für den Vergleichsoperator `NULL` nicht relevant.

- `CONTAINS`: Prüft auf eine Teilsequenz oder einen Wert in einem Satz.

  `AttributeValueList` kann nur ein `AttributeValue`-Element vom Typ Zeichenfolge, Zahl oder Binärwert (kein Satztyp) sein. Wenn das Zielattribut des Vergleichs vom Typ Zeichenfolge ist, dann prüft der Operator, ob eine Teilzeichenfolge übereinstimmt. Wenn das Zielattribut des Vergleichs vom Typ Binärwert ist, sucht der Operator nach einer Teilsequenz des Ziels, die mit der Eingabe übereinstimmt. Wenn das Zielattribut des Vergleichs ein Satz ist („`SS`“, „`NS`“ oder „`BS`“), dann wertet der Operator die Prüfung mit True aus, wenn er eine genaue Übereinstimmung mit einem beliebigen Mitglied des Satzes findet.

  CONTAINS wird für Listen unterstützt: Beim Auswerten von „a `CONTAINS` b“, kann „a“ eine Liste sein. Dagegen kann „b“ kein Satz, keine Zuordnung und keine Liste sein.

- `NOT_CONTAINS`: Prüft ein Element auf eine fehlende Teilsequenz oder einen fehlenden Wert in einem Satz.

  `AttributeValueList` kann nur ein `AttributeValue`-Element vom Typ Zeichenfolge, Zahl oder Binärwert (kein Satztyp) sein. Wenn das Zielattribut des Vergleichs eine Zeichenfolge ist, dann prüft der Operator auf eine fehlende Übereinstimmung einer Teilzeichenfolge. Wenn das Zielattribut des Vergleichs vom Typ Binärwert ist, prüft der Operator, ob eine Teilsequenz des

Ziels, die mit der Eingabe übereinstimmt, fehlt. Wenn das Zielattribut des Vergleichs ein Satz ist („SS“, „NS“ oder „BS“), dann wertet der Operator mit True aus, wenn er `does not` keine genaue Übereinstimmung mit einem beliebigen Mitglied des Satzes findet.

NOT_CONTAINS wird für Listen unterstützt: Beim Auswerten von „a `NOT CONTAINS` b“ kann „a“ eine Liste sein. Dagegen kann „b“ kein Satz, keine Zuordnung und keine Liste sein.

- `BEGINS_WITH` : Prüft auf ein Präfix.

  `AttributeValueList` kann nur ein `AttributeValue` vom Typ Zeichenfolge oder Binärwert (keine Zahl oder Satztyp) sein. Das Zielattribut des Vergleichs muss vom Typ Zeichenfolge oder Binärwert sein (nicht Zahl oder Satz).

- `IN` : Überprüft, ob übereinstimmende Elemente in zwei Sätzen vorhanden sind.

  `AttributeValueList` kann nur ein oder mehrere `AttributeValue`-Elemente vom Typ Zeichenfolge, Zahl oder Binärwert (kein Satztyp) sein. Diese Attribute werden mit einem vorhandenen Satztypattribut eines Elements verglichen. Wenn ein beliebiges Element des Eingabesatzes im Elementattribut vorhanden ist, wird der Ausdruck mit True ausgewertet.

- `BETWEEN` : Größer als oder gleich dem ersten Wert und kleiner als oder gleich dem zweiten Wert.

  `AttributeValueList` muss zwei `AttributeValue`-Elemente desselben Typs enthalten, und zwar Zeichenfolge, Zahl oder Binärwert (kein Satztyp). Es kommt zu einer Übereinstimmung mit dem Zielattribut, wenn der Zielwert größer als oder gleich dem ersten Element und kleiner als oder gleich dem zweiten Element ist. Wenn ein Element ein `AttributeValue`-Element eines anderen Typs enthält, als in der Anforderung angegeben, stimmt der Wert nicht überein. Beispiel: `{"S":"6"}` stimmt nicht mit `{"N":"6"}` überein. Auch, `{"N":"6"}` entspricht nicht `{"NS": ["6", "2", "1"]}`.

Die folgenden Parameter können statt `AttributeValueList` und `ComparisonOperator` verwendet werden:

- `Value` – Ein Wert für DynamoDB zum Vergleichen mit einem Attribut.
- `Exists` – Ein Boolescher Wert, der DynamoDB veranlasst, den Wert auszuwerten, bevor die bedingte Operation ausgeführt wird:
  - Wenn `Exists` `true` ist, prüft DynamoDB, ob der Attributwert in der Tabelle bereits vorhanden ist. Falls gefunden, wird die Bedingung mit True ausgewertet, andernfalls mit False.

- Wenn `Exists` `false` ist, geht DynamoDB davon aus, dass der Attributwert `not` in der Tabelle vorhanden ist. Wenn der Wert tatsächlich nicht vorhanden ist, ist die Annahme gültig und die Bedingung wird mit True ausgewertet. Wenn der Wert trotz der Annahme, dass er nicht vorhanden ist, gefunden wird, wird die Bedingung mit True ausgewertet.

Der Standardwert für `Exists` ist `true`.

Die Parameter `Value` und `Exists` sind mit `AttributeValueList` und `ComparisonOperator` nicht kompatibel. Hinweis: Wenn Sie beide Parametersätze auf einmal verwenden, gibt DynamoDB eine `ValidationException`-Ausnahme zurück.

> **Note**
>
> Dieser Parameter unterstützt keine Attribute vom Typ Liste oder Zuordnung.

Verwenden Sie ConditionExpressionstattdessen — Beispiel

Angenommen, Sie möchten ein Element in der Tabelle Music ändern, jedoch nur, wenn eine bestimmte Bedingung True ist. Sie könnten eine `UpdateItem` Anfrage mit einem `Expected` Parameter verwenden, wie in diesem AWS CLI Beispiel:

```
aws dynamodb update-item \
    --table-name Music \
    --key '{
        "Artist": {"S":"No One You Know"},
        "SongTitle": {"S":"Call Me Today"}
    }' \
    --attribute-updates '{
        "Price": {
            "Action": "PUT",
            "Value": {"N":"1.98"}
        }
    }' \
    --expected '{
        "Price": {
            "ComparisonOperator": "LE",
            "AttributeValueList": [ {"N":"2.00"} ]
        }
    }'
```

Sie können stattdessen `ConditionExpression` verwenden.

```
aws dynamodb update-item \
    --table-name Music \
    --key '{
        "Artist": {"S":"No One You Know"},
        "SongTitle": {"S":"Call Me Today"}
    }' \
    --update-expression 'SET Price = :p1' \
    --condition-expression 'Price <= :p2' \
    --expression-attribute-values '{
        ":p1": {"N":"1.98"},
        ":p2": {"N":"2.00"}
    }'
```

KeyConditions (Vermächtnis)

> **Note**
>
> Es wird empfohlen, nach Möglichkeit die neuen Ausdrucksparameter anstelle der alten Parameter zu verwenden. Weitere Informationen finden Sie unter Verwenden von Ausdrücken in DynamoDB. Spezifische Informationen zu dem neuen Parameter, der diesen ersetzt, finden Sie unter KeyConditionExpressionstattdessen verwenden..

Der alte Bedingungsparameter `KeyConditions` enthält Auswahlkriterien für eine `Query`-Operation. Für eine Abfrage in einer Tabelle sind nur Bedingungen für die Primärschlüsselattribute der Tabelle möglich. Sie müssen den Namen und Wert des Partitionsschlüssels als `EQ`-Bedingung angeben. Sie können optional eine zweite Bedingung für den Sortierschlüssel angeben.

> **Note**
>
> Wenn Sie keine Sortierschlüsselbedingung festlegen, werden alle Elemente, die mit dem Partitionsschlüssel übereinstimmen, abgerufen. Wenn ein `FilterExpression` oder `QueryFilter` vorhanden ist, wird er angewendet, nachdem die Elemente abgerufen wurden.

Für eine Abfrage in einem Index sind nur Bedingungen für die Indexschlüsselattribute möglich. Sie müssen den Namen und Wert des Indexpartitionsschlüssels als EQ-Bedingung angeben. Sie können optional eine zweite Bedingung für den Indexsortierschlüssel angeben.

Jedes KeyConditions-Element besteht aus einem zu vergleichenden Attributnamen zusammen mit Folgendem:

- AttributeValueList – Ein oder mehrere Werte, die anhand des angegebenen Attributs ausgewertet werden sollen. Die Anzahl der Werte in der Liste hängt vom verwendeten ComparisonOperator ab.

  Beim Zahlentyp sind Wertevergleiche numerisch.

  Vergleiche von Zeichenfolgenwerten für größer als, gleich oder kleiner als basieren auf Unicode mit UTF-8-Binärkodierung. Beispiel: a ist größer als A und a ist größer als B.

  Beim Binärtyp betrachtet DynamoDB beim Vergleichen der binären Werte jedes Byte der Binärdaten ohne Vorzeichen.
- ComparisonOperator – Ein Vergleichsoperator zum Auswerten der Attribute. Zum Beispiel: ist gleich, größer als, kleiner als.

  Für KeyConditions werden nur die folgenden Vergleichsoperatoren unterstützt:

  ```
  EQ | LE | LT | GE | GT | BEGINS_WITH | BETWEEN
  ```

  Es folgen Beschreibungen dieser Vergleichsoperatoren.

  - EQ : gleich.

    AttributeValueList kann nur ein AttributeValue vom Typ Zeichenfolge, Zahl oder Binärwert (kein Satztyp) sein. Wenn ein Element ein AttributeValue-Element eines anderen Typs enthält, als in der Anforderung angegeben, stimmt der Wert nicht überein. Beispiel: {"S":"6"} ist nicht gleich {"N":"6"}. Auch, {"N":"6"} ist nicht gleich {"NS":["6", "2", "1"]}.
  - LE : kleiner als oder gleich.

    AttributeValueList kann nur ein AttributeValue-Element vom Typ Zeichenfolge, Zahl oder Binärwert (kein Satztyp) sein. Wenn ein Element ein AttributeValue-Element eines anderen Typs enthält, als in der Anforderung angegeben, stimmt der Wert nicht überein. Beispiel:

{"S":"6"} ist nicht gleich {"N":"6"}. Auch, {"N":"6"} entspricht nicht {"NS":["6", "2", "1"]}.

- LT : kleiner als.

  AttributeValueList kann nur ein AttributeValue vom Typ Zeichenfolge, Zahl oder Binärwert (kein Satztyp) sein. Wenn ein Element ein AttributeValue-Element eines anderen Typs enthält, als in der Anforderung angegeben, stimmt der Wert nicht überein. Beispiel: {"S":"6"} ist nicht gleich {"N":"6"}. Auch, {"N":"6"} entspricht nicht {"NS":["6", "2", "1"]}.

- GE : größer als oder gleich.

  AttributeValueList kann nur ein AttributeValue-Element vom Typ Zeichenfolge, Zahl oder Binärwert (kein Satztyp) sein. Wenn ein Element ein AttributeValue-Element eines anderen Typs enthält, als in der Anforderung angegeben, stimmt der Wert nicht überein. Beispiel: {"S":"6"} ist nicht gleich {"N":"6"}. Auch entspricht {"N":"6"} nicht {"NS":["6", "2", "1"]}.

- GT : größer als.

  AttributeValueList kann nur ein AttributeValue-Element vom Typ Zeichenfolge, Zahl oder Binärwert (kein Satztyp) sein. Wenn ein Element ein AttributeValue-Element eines anderen Typs enthält, als in der Anforderung angegeben, stimmt der Wert nicht überein. Beispiel: {"S":"6"} ist nicht gleich {"N":"6"}. Auch, {"N":"6"} entspricht nicht {"NS":["6", "2", "1"]}.

- BEGINS_WITH : Prüft auf ein Präfix.

  AttributeValueList kann nur ein AttributeValue vom Typ Zeichenfolge oder Binärwert (keine Zahl oder Satztyp) sein. Das Zielattribut des Vergleichs muss vom Typ Zeichenfolge oder Binärwert sein (nicht Zahl oder Satz).

- BETWEEN : Größer als oder gleich dem ersten Wert und kleiner als oder gleich dem zweiten Wert.

  AttributeValueList muss zwei AttributeValue-Elemente desselben Typs enthalten, und zwar Zeichenfolge, Zahl oder Binärwert (kein Satztyp). Es kommt zu einer Übereinstimmung mit dem Zielattribut, wenn der Zielwert größer als oder gleich dem ersten Element und kleiner als oder gleich dem zweiten Element ist. Wenn ein Element ein AttributeValue-Element eines anderen Typs enthält, als in der Anforderung angegeben, stimmt der Wert nicht überein. Beispiel:

`{"S":"6"}` stimmt nicht mit `{"N":"6"}` überein. Auch entspricht `{"N":"6"}` nicht `{"NS": ["6", "2", "1"]}`.

KeyConditionExpressionStattdessen verwenden — Beispiel

Angenommen, Sie möchten mehrere Elemente mit demselben Partitionsschlüssel aus der Tabelle Music abrufen. Sie könnten eine `Query` Anfrage mit einem `KeyConditions` Parameter verwenden, wie in diesem AWS CLI Beispiel:

```
aws dynamodb query \
    --table-name Music \
    --key-conditions '{
        "Artist":{
            "ComparisonOperator":"EQ",
            "AttributeValueList": [ {"S": "No One You Know"} ]
        },
        "SongTitle":{
            "ComparisonOperator":"BETWEEN",
            "AttributeValueList": [ {"S": "A"}, {"S": "M"} ]
        }
    }'
```

Sie können stattdessen `KeyConditionExpression` verwenden.

```
aws dynamodb query \
    --table-name Music \
    --key-condition-expression 'Artist = :a AND SongTitle BETWEEN :t1 AND :t2' \
    --expression-attribute-values '{
        ":a": {"S": "No One You Know"},
        ":t1": {"S": "A"},
        ":t2": {"S": "M"}
    }'
```

QueryFilter (Vermächtnis)

> **Note**
>
> Es wird empfohlen, nach Möglichkeit die neuen Ausdrucksparameter anstelle der alten Parameter zu verwenden. Weitere Informationen finden Sie unter Verwenden von

Ausdrücken in DynamoDB. Spezifische Informationen zu dem neuen Parameter, der diesen ersetzt, finden Sie unter FilterExpressionstattdessen verwenden..

In einer `Query`-Operation handelt es sich bei dem alten Bedingungsparameter `QueryFilter` um eine Bedingung, die die Abfrageergebnisse auswertet, nachdem die Elemente gelesen wurden, und die nur die gewünschten Werte zurückgibt.

Dieser Parameter unterstützt keine Attribute vom Typ Liste oder Zuordnung.

> **Note**
>
> Ein `QueryFilter` wird angewendet, nachdem die Elemente gelesen wurden. Der Filterprozess belegt keine zusätzlichen Lesekapazitätseinheiten.

Wenn Sie mehr als eine Bedingung im `QueryFilter`-Mapping angeben, müssen alle Bedingungen standardmäßig mit True ausgewertet werden. Mit anderen Worten: Die Bedingungen werden mit dem AND-Operator kombiniert. (Sie können dasConditionalOperator (Vermächtnis)-Parameter auf ODER zu den Bedingungen verwenden. In diesem Fall müssen nicht alle, sondern mindestens eine der Bedingungen mit True ausgewertet werden.)

Hinweis: `QueryFilter` lässt keine Schlüsselattribute zu. Sie können keine Filterbedingung für einen Partitionsschlüssel oder Sortierschlüssel definieren.

Jedes `QueryFilter`-Element besteht aus einem zu vergleichenden Attributnamen zusammen mit Folgendem:

- `AttributeValueList` – Ein oder mehrere Werte, die anhand des angegebenen Attributs ausgewertet werden sollen. Die Anzahl der Werte in der Liste hängt von dem in `ComparisonOperator` angegebenen Operator ab.

  Beim Zahlentyp sind Wertevergleiche numerisch.

  Vergleiche von Zeichenfolgenwerten für größer als, gleich oder kleiner als basieren auf UTF-8-Binärkodierung. Beispiel: a ist größer als A und a ist größer als B.

  Beim Binärtyp betrachtet DynamoDB jedes Byte der Binärdaten beim Vergleichen der binären Werte ohne Vorzeichen.

Informationen zum Angeben von Datentypen in JSON finden Sie unter DynamoDB Low-Level-API.

- `ComparisonOperator` – Ein Vergleichsoperator zum Auswerten der Attribute. Zum Beispiel: ist gleich, größer als, kleiner als.

  Die folgenden Vergleichsoperatoren sind verfügbar:

  `EQ | NE | LE | LT | GE | GT | NOT_NULL | NULL | CONTAINS | NOT_CONTAINS | BEGINS_WITH | IN | BETWEEN`

FilterExpressionStattdessen verwenden — Beispiel

Angenommen, Sie möchten die Tabelle Music abfragen und wenden eine Bedingung auf die übereinstimmenden Elemente an. Sie können eine `Query`-Anforderung mit dem Parameter `QueryFilter` wie in diesem AWS CLI -Beispiel verwenden:

```
aws dynamodb query \
    --table-name Music \
    --key-conditions '{
        "Artist": {
            "ComparisonOperator": "EQ",
            "AttributeValueList": [ {"S": "No One You Know"} ]
        }
    }' \
    --query-filter '{
        "Price": {
            "ComparisonOperator": "GT",
            "AttributeValueList": [ {"N": "1.00"} ]
        }
    }'
```

Sie können stattdessen `FilterExpression` verwenden.

```
aws dynamodb query \
    --table-name Music \
    --key-condition-expression 'Artist = :a' \
    --filter-expression 'Price > :p' \
    --expression-attribute-values '{
        ":p": {"N":"1.00"},
        ":a": {"S":"No One You Know"}
    }'
```

ScanFilter (Vermächtnis)

> **Note**
>
> Es wird empfohlen, nach Möglichkeit die neuen Ausdrucksparameter anstelle der alten Parameter zu verwenden. Weitere Informationen finden Sie unter Verwenden von Ausdrücken in DynamoDB. Spezifische Informationen zu dem neuen Parameter, der diesen ersetzt, finden Sie unter FilterExpressionstattdessen verwenden..

In einer `Scan`-Operation handelt es sich bei dem alten Bedingungsparameter `ScanFilter` um eine Bedingung, die die Abfrageergebnisse auswertet und die nur die gewünschten Werte zurückgibt.

> **Note**
>
> Dieser Parameter unterstützt keine Attribute vom Typ Liste oder Zuordnung.

Wenn Sie mehr als eine Bedingung im `ScanFilter`-Mapping angeben, müssen alle Bedingungen standardmäßig mit True ausgewertet werden. Mit anderen Worten, die Bedingungen stimmen ANDed überein. (Sie können das ConditionalOperator (Vermächtnis)-Parameter auf ODER zu den Bedingungen verwenden. In diesem Fall müssen nicht alle, sondern mindestens eine der Bedingungen mit True ausgewertet werden.)

Jedes `ScanFilter`-Element besteht aus einem zu vergleichenden Attributnamen zusammen mit Folgendem:

- `AttributeValueList` – Ein oder mehrere Werte, die anhand des angegebenen Attributs ausgewertet werden sollen. Die Anzahl der Werte in der Liste hängt von dem in `ComparisonOperator` angegebenen Operator ab.

  Beim Zahlentyp sind Wertevergleiche numerisch.

  Vergleiche von Zeichenfolgenwerten für größer als, gleich oder kleiner als basieren auf UTF-8-Binärkodierung. Beispiel: `a` ist größer als `A` und `a` ist größer als `B`.

  Beim Binärtyp betrachtet DynamoDB beim Vergleichen der binären Werte jedes Byte der Binärdaten ohne Vorzeichen.

Informationen zum Angeben von Datentypen in JSON finden Sie unter DynamoDB Low-Level-API.

- `ComparisonOperator` – Ein Vergleichsoperator zum Auswerten der Attribute. Zum Beispiel: ist gleich, größer als, kleiner als.

  Die folgenden Vergleichsoperatoren sind verfügbar:

  `EQ | NE | LE | LT | GE | GT | NOT_NULL | NULL | CONTAINS | NOT_CONTAINS | BEGINS_WITH | IN | BETWEEN`

Verwenden Sie FilterExpressionstattdessen — Beispiel

Angenommen, Sie möchten die Tabelle Music scannen und wenden eine Bedingung auf die übereinstimmenden Elemente an. Sie können eine `Scan`-Anforderung mit dem Parameter `ScanFilter` wie in diesem AWS CLI -Beispiel verwenden:

```
aws dynamodb scan \
    --table-name Music \
    --scan-filter '{
        "Genre":{
            "AttributeValueList":[ {"S":"Rock"} ],
            "ComparisonOperator": "EQ"
        }
    }'
```

Sie können stattdessen `FilterExpression` verwenden.

```
aws dynamodb scan \
    --table-name Music \
    --filter-expression 'Genre = :g' \
    --expression-attribute-values '{
        ":g": {"S":"Rock"}
    }'
```

Schreiben von Bedingungen mit Legacy-Parametern

> **Note**
>
> Es wird empfohlen, nach Möglichkeit die neuen Ausdrucksparameter anstelle der alten Parameter zu verwenden. Weitere Informationen finden Sie unter Verwenden von Ausdrücken in DynamoDB.

Im folgenden Abschnitt wird beschrieben, wie Sie Bedingungen für die Verwendung mit Legacy-Parametern, wie `Expected`, `QueryFilter` und `ScanFilter`, schreiben.

> **Note**
>
> Neue Anwendungen sollten stattdessen Ausdrucksparameter verwenden. Weitere Informationen finden Sie unter Verwenden von Ausdrücken in DynamoDB.

Einfache Bedingungen

Mit Attributwerten können Sie Bedingungen für Vergleiche mit Tabellenattributen erstellen. Eine Bedingung wird immer mit True oder False ausgewertet und besteht aus Folgendem:

- `ComparisonOperator` – größer als, kleiner als, gleich usw.
- `AttributeValueList` (optional) – Attributwerte für den Vergleich. Abhängig vom verwendeten `ComparisonOperator` enthält `AttributeValueList` möglicherweise einen, zwei oder mehr Werte oder ist nicht vorhanden.

In den folgenden Abschnitten werden die verschiedenen Vergleichsoperatoren beschrieben. Außerdem finden Sie einige Beispiele, wie Sie sie in Bedingungen verwenden können.

Vergleichsoperatoren ohne Attributwerte

- `NOT_NULL` – True, wenn ein Attribut vorhanden ist.
- `NULL` – True, wenn das Attribut nicht vorhanden ist.

Verwenden Sie diese Operatoren, um zu überprüfen, ob ein Attribut vorhanden oder nicht vorhanden ist. Da es keinen Wert gibt, mit dem verglichen werden soll, geben Sie `AttributeValueList` nicht an.

Beispiel

Der folgende Ausdruck wird mit True ausgewertet, wenn das Attribut Dimensions vorhanden ist.

```
...
    "Dimensions": {
         ComparisonOperator: "NOT_NULL"
    }
...
```

Vergleichsoperatoren mit einem Attributwert

- `EQ` – True, wenn ein Attribut gleich einem Wert ist.

  `AttributeValueList` kann nur einen Wert vom Typ Zeichenfolge, Zahl, Binärwert, Zeichenfolgensatz, Zahlensatz und Binärwertesatz enthalten. Wenn ein Element einen Wert eines anderen Typs enthält, als in der Anforderung angegeben, stimmt der Wert nicht überein. Beispielsweise ist die Zeichenfolge `"3"` nicht gleich der Zahl `3`. Die Zahl `3` ist auch nicht gleich dem Zahlensatz `[3, 2, 1]`.
- `NE` – True, wenn ein Attribut nicht gleich einem Wert ist.

  `AttributeValueList` kann nur einen Wert vom Typ Zeichenfolge, Zahl, Binärwert, Zeichenfolgensatz, Zahlensatz und Binärwertesatz enthalten. Wenn ein Element einen Wert eines anderen Typs enthält, als in der Anforderung angegeben, stimmt der Wert nicht überein.
- `LE` – True, wenn ein Attribut kleiner als oder gleich einem Wert ist.

  `AttributeValueList` kann nur einen Wert vom Typ Zeichenfolge, Zahl oder Binärwert (kein Satztyp) enthalten. Wenn ein Element einen `AttributeValue` eines anderen Typs enthält, als in der Anforderung angegeben, stimmt der Wert nicht überein.
- `LT` – True, wenn ein Attribut kleiner als ein Wert ist.

  `AttributeValueList` kann nur einen Wert vom Typ Zeichenfolge, Zahl oder Binärwert (kein Satztyp) enthalten. Wenn ein Element einen Wert eines anderen Typs enthält, als in der Anforderung angegeben, stimmt der Wert nicht überein.
- `GE` – True, wenn ein Attribut größer als oder gleich einem Wert ist.

`AttributeValueList` kann nur einen Wert vom Typ Zeichenfolge, Zahl oder Binärwert (kein Satztyp) enthalten. Wenn ein Element einen Wert eines anderen Typs enthält, als in der Anforderung angegeben, stimmt der Wert nicht überein.

- `GT` – True, wenn ein Attribut größer als ein Wert ist.

  `AttributeValueList` kann nur einen Wert vom Typ Zeichenfolge, Zahl oder Binärwert (kein Satztyp) enthalten. Wenn ein Element einen Wert eines anderen Typs enthält, als in der Anforderung angegeben, stimmt der Wert nicht überein.

- `CONTAINS` – True, wenn ein Wert in einem Satz vorhanden ist oder wenn ein Wert einen anderen enthält.

  `AttributeValueList` kann nur einen Wert vom Typ Zeichenfolge, Zahl oder Binärwert (kein Satztyp) enthalten. Wenn das Zielattribut des Vergleichs eine Zeichenfolge ist, dann prüft der Operator, ob eine Teilzeichenfolge übereinstimmt. Wenn das Zielattribut des Vergleichs vom Typ Binär ist, sucht der Operator nach einer Teilsequenz des Ziels, die mit der Eingabe übereinstimmt. Wenn das Zielattribut des Vergleichs ein Satz ist, dann wertet der Operator mit True aus, wenn er eine genaue Übereinstimmung mit einem beliebigen Mitglied des Satzes findet.

- `NOT_CONTAINS` – True, wenn ein Wert in einem Satz nicht vorhanden ist oder wenn ein Wert keinen anderen enthält.

  `AttributeValueList` kann nur einen Wert vom Typ Zeichenfolge, Zahl oder Binärwert (kein Satztyp) enthalten. Wenn das Zielattribut des Vergleichs eine Zeichenfolge ist, dann prüft der Operator auf eine fehlende Übereinstimmung einer Teilzeichenfolge. Wenn das Zielattribut des Vergleichs vom Typ Binärwert ist, prüft der Operator, ob eine Teilsequenz des Ziels, die mit der Eingabe übereinstimmt, fehlt. Wenn das Zielattribut des Vergleichs ein Satz ist, dann wertet der Operator mit True aus, wenn er keine genaue Übereinstimmung mit einem beliebigen Mitglied des Satzes findet.

- `BEGINS_WITH` – True, wenn die ersten Zeichen eines Attributs mit dem angegebenen Wert übereinstimmen. Verwenden Sie diesen Operator nicht für den Vergleich von Zahlen.

  `AttributeValueList` kann nur einen Wert vom Typ Zeichenfolge oder Binärwert (keine Zahl oder Satz) enthalten. Das Zielattribut des Vergleichs muss vom Typ Zeichenfolge oder Binärzahl sein (keine Zahl bzw. kein Satz).

Verwenden Sie diese Operatoren, um ein Attribut mit einem Wert zu vergleichen. Sie müssen eine `AttributeValueList` angeben, die aus einem einzigen Wert besteht. Für die meisten Operatoren muss dieser Wert ein Skalarwert sein. Die Operatoren `EQ` und `NE` unterstützen jedoch auch Sätze.

Beispiele

Die folgenden Ausdrücke werden mit True ausgewertet, wenn:

- der Produktpreis größer als 100 ist.

```
...
    "Price": {
        ComparisonOperator: "GT",
        AttributeValueList: [ {"N":"100"} ]
    }
...
```

- eine Produktkategorie mit „Bo“ beginnt.

```
...
    "ProductCategory": {
        ComparisonOperator: "BEGINS_WITH",
        AttributeValueList: [ {"S":"Bo"} ]
    }
...
```

- ein Produkt entweder in Rot, Grün oder Schwarz erhältlich ist:

```
...
    "Color": {
        ComparisonOperator: "EQ",
        AttributeValueList: [
            [ {"S":"Black"}, {"S":"Red"}, {"S":"Green"} ]
        ]
    }
...
```

> **Note**
>
> Beim Vergleichen von Satzdatentypen ist die Reihenfolge der Elemente unerheblich. DynamoDB sendet nur die Elemente mit demselben Wertesatz, unabhängig von der Reihenfolge, in der Sie sie in Ihrer Anforderung angeben.

Vergleichsoperatoren mit zwei Attributwerten

- `BETWEEN` – True, wenn ein Wert zwischen einer unteren und einer oberen Grenze liegt, Endpunkte eingeschlossen.

  `AttributeValueList` muss zwei Elemente desselben Typs enthalten, und zwar Zeichenfolge, Zahl oder Binärwert (kein Satztyp). Es kommt zu einer Übereinstimmung mit dem Zielattribut, wenn der Zielwert größer als oder gleich dem ersten Element und kleiner als oder gleich dem zweiten Element ist. Wenn ein Element einen Wert eines anderen Typs enthält, als in der Anforderung angegeben, stimmt der Wert nicht überein.

Verwenden Sie diesen Operator, um zu bestimmen, ob ein Attributwert innerhalb eines Bereichs liegt. Die `AttributeValueList` muss zwei skalare Elemente desselben Typs enthalten, und zwar Zeichenfolge, Zahl oder Binärwert.

Beispiel

Der folgende Ausdruck wird mit True ausgewertet, wenn ein Produktpreis zwischen 100 und 200 liegt.

```
...
    "Price": {
        ComparisonOperator: "BETWEEN",
        AttributeValueList: [ {"N":"100"}, {"N":"200"} ]
    }
...
```

Vergleichsoperatoren mit n Attributwerten

- `IN` – True, wenn ein Wert gleich einem der Werte in einer Aufzählung ist. Es werden nur skalare Werte und keine Sätze in der Liste unterstützt. Das Zielattribut muss vom selben Typ sein und mit dem genauen Wert übereinstimmen.

`AttributeValueList` kann nur ein oder mehrere Elemente vom Typ Zeichenfolge, Zahl oder Binärwert (kein Satztyp) enthalten. Diese Attribute werden mit einem vorhandenen Nicht-Satztypattribut eines Elements verglichen. Wenn ein beliebiges Element des Eingabesatzes im Elementattribut vorhanden ist, wird der Ausdruck mit True ausgewertet.

`AttributeValueList` kann nur ein oder mehrere Werte vom Typ Zeichenfolge, Zahl oder Binärwert (kein Satztyp) enthalten. Das Zielattribut des Vergleichs muss vom selben Typ sein und mit dem genauen Wert übereinstimmen. Eine Zeichenfolge stimmt niemals mit einem Zeichenfolgensatz überein.

Verwenden Sie diesen Operator, um zu ermitteln, ob sich der angegebene Wert innerhalb einer Aufzählung befindet. Sie können eine beliebige Anzahl von Skalarwerten in `AttributeValueList` angeben, diese müssen allerdings denselben Datentyp aufweisen.

Beispiel

Der folgende Ausdruck wird mit True ausgewertet, wenn der Wert für ID 201, 203 oder 205 ist.

```
...
    "Id": {
        ComparisonOperator: "IN",
        AttributeValueList: [ {"N":"201"}, {"N":"203"}, {"N":"205"} ]
    }
...
```

Verwenden mehrerer Bedingungen

Mit DynamoDB können Sie mehrere Bedingungen zu komplexen Ausdrücken kombinieren. Dazu geben Sie mindestens zwei Ausdrücke mit einem optionalen ConditionalOperator (Vermächtnis) an.

Wenn Sie mehr als eine Bedingung angeben, müssen alle Bedingungen mit True ausgewertet werden, damit der gesamte Ausdruck mit True ausgewertet wird. Mit anderen Worten, es wird eine implizite AND-Operation ausgeführt.

Beispiel

Der folgende Ausdruck wird mit True ausgewertet, wenn ein Produkt ein Buch mit mindestens 600 Seiten ist. Beide Bedingungen müssen mit True ausgewertet werden, da sie implizit durch AND verknüpft sind.

```
...
    "ProductCategory": {
        ComparisonOperator: "EQ",
        AttributeValueList: [ {"S":"Book"} ]
    },
    "PageCount": {
        ComparisonOperator: "GE",
        AttributeValueList: [ {"N":600"} ]
    }
...
```

Mithilfe von ConditionalOperator (Vermächtnis) können Sie verdeutlichen, dass eine AND-Operation ausgeführt wird. Das folgende Beispiel verhält sich ähnlich wie das vorherige.

```
...
    "ConditionalOperator" : "AND",
    "ProductCategory": {
        "ComparisonOperator": "EQ",
        "AttributeValueList": [ {"N":"Book"} ]
    },
    "PageCount": {
        "ComparisonOperator": "GE",
        "AttributeValueList": [ {"N":600"} ]
    }
...
```

Sie können `ConditionalOperator` auch auf OR festlegen, was bedeutet, dass mindestens eine der Bedingungen mit True ausgewertet werden muss.

Beispiel

Der folgende Ausdruck wird mit True ausgewertet, wenn ein Produkt eine Mountainbike ist, wenn es eine bestimmte Marke ist oder wenn der Preis größer als 100 ist.

```
...
    ConditionalOperator : "OR",
    "BicycleType": {
        "ComparisonOperator": "EQ",
        "AttributeValueList": [ {"S":"Mountain" ]
    },
    "Brand": {
        "ComparisonOperator": "EQ",
```

```
        "AttributeValueList": [ {"S":"Brand-Company A" ]
    },
    "Price": {
        "ComparisonOperator": "GT",
        "AttributeValueList": [ {"N":"100"} ]
    }
...
```

> **Note**
>
> In einem komplexen Ausdruck werden die Bedingungen der Reihe nach verarbeitet, angefangen von der ersten bis zur letzten Bedingung.
> Sie können nicht sowohl AND, als auch OR in einem einzigen Ausdruck verwenden.

Andere Bedingungsoperatoren

In früheren Versionen von DynamoDB verhielt sich der Parameter `Expected` für bedingte Schreibvorgänge anders. Jedes Element im `Expected`-Mapping stellte einen Attributnamen dar, den DynamoDB zusammen mit Folgendem prüfen sollte:

- `Value` – ein Wert für den Vergleich mit dem Attribut.
- `Exists` – bestimmt, ob der Wert vorhanden ist, bevor die Operation ausgeführt wird.

Die Optionen `Value` und `Exists` werden weiterhin in DynamoDB unterstützt. Sie ermöglichen allerdings nur eine Prüfung auf eine Gleichheitsbedingung oder darauf, ob ein Attribut vorhanden ist. Es wird empfohlen, stattdessen `ComparisonOperator` und `AttributeValueList` zu verwenden, da Sie mit diesen Optionen einen umfassenderen Bereich von Bedingungen erstellen können.

Example

Ein `DeleteItem` kann prüfen, ob ein Buch nicht mehr veröffentlicht wird, und es nur löschen, wenn die Bedingung erfüllt ist. Hier finden Sie ein AWS CLI -Beispiel mit einer Legacy-Bedingung:

```
aws dynamodb delete-item \
    --table-name ProductCatalog \
    --key '{
        "Id": {"N":"600"}
```

```
    }' \
    --expected '{
        "InPublication": {
            "Exists": true,
            "Value": {"BOOL":false}
        }
    }'
```

Im folgenden Beispiel werden dieselben Aktionen ausgeführt, allerdings ohne Legacy-Bedingung:

```
aws dynamodb delete-item \
    --table-name ProductCatalog \
    --key '{
        "Id": {"N":"600"}
    }' \
    --expected '{
        "InPublication": {
            "ComparisonOperator": "EQ",
            "AttributeValueList": [ {"BOOL":false} ]
        }
    }'
```

Example

Eine `PutItem`-Operation kann ein vorhandenes Element mit denselben Primärschlüsselattributen vor dem Überschreiben schützen. Hier finden Sie ein -Beispiel mit einer Legacy-Bedingung:

```
aws dynamodb put-item \
    --table-name ProductCatalog \
    --item '{
      "Id": {"N":"500"},
        "Title": {"S":"Book 500 Title"}
    }' \
    --expected '{
        "Id": { "Exists": false }
    }'
```

Im folgenden Beispiel werden dieselben Aktionen ausgeführt, allerdings ohne Legacy-Bedingung:

```
aws dynamodb put-item \
    --table-name ProductCatalog \
```

```
    --item '{
      "Id": {"N":"500"},
        "Title": {"S":"Book 500 Title"}
    }' \
    --expected '{
        "Id": { "ComparisonOperator": "NULL" }
    }'
```

> **Note**
>
> Verwenden Sie für Bedingungen im `Expected`-Mapping die Legacy-Optionen `Value` und `Exists` nicht gemeinsam mit `ComparisonOperator` und `AttributeValueList`. Andernfalls tritt beim bedingten Schreibvorgang ein Fehler auf.

Die vorliegende Übersetzung wurde maschinell erstellt. Im Falle eines Konflikts oder eines Widerspruchs zwischen dieser übersetzten Fassung und der englischen Fassung (einschließlich infolge von Verzögerungen bei der Übersetzung) ist die englische Fassung maßgeblich.