aws

Panduan Developer

Amazon DynamoDB

Versi API 2012-08-10

Copyright © 2025 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

Amazon DynamoDB: Panduan Developer

Copyright © 2025 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

Merek dagang dan tampilan dagang Amazon tidak boleh digunakan sehubungan dengan produk atau layanan apa pun yang bukan milik Amazon, dengan cara apa pun yang dapat menyebabkan kebingungan antara para pelanggan, atau dengan cara apa pun yang menghina atau mendiskreditkan Amazon. Semua merek dagang lain yang tidak dimiliki oleh Amazon adalah milik dari pemiliknya masing-masing, yang mungkin berafiliasi atau tidak berafiliasi dengan, terkait, atau disponsori oleh Amazon.

Table of Contents

Apa itu Amazon DynamoDB? 1
Karakteristik 2
Nirserver 2
NoSQL 2
Dikelola sepenuhnya 2
Kinerja milidetik satu digit pada skala apa pun 3
Kasus penggunaan 3
Kemampuan 4
Replikasi multi-aktif dengan tabel global 4
Transaksi ACID 4
Tangkapan data perubahan 5
Indeks sekunder 5
Integrasi layanan 5
Integrasi tanpa server 5
Mengimpor dan mengekspor data ke Amazon S3 6
Integrasi nol-ETL 6
Pembuatan cache 6
Keamanan 6
Ketahanan 7
Tabel global 8
Pencadangan dan pemulihan berkelanjutan point-in-time 8
Pencadangan dan pemulihan sesuai permintaan 8
Mengakses DynamoDB 8
Harga 9
Memulai 9
Mulai menggunakan DynamoDB 11
Sumber daya pengguna pertama kali 12
Amazon DynamoDB praktik terbaik tambahan untuk pengguna pertama kali 13
AWS CLI sumber daya 13
Sumber daya pemrograman 14
Mengakses DynamoDB 14
Menggunakan konsol 15
Menggunakan AWS CLI 15
Menggunakan API 18

Menggunakan NoSQL Workbench 18
Rentang alamat IP 19
Prasyarat 20
Menyiapkan DynamoDB 20
Menyiapkan DynamoDB (layanan web) 21
Menyiapkan DynamoDB lokal (versi yang dapat diunduh) 24
Langkah 1: Buat tabel 48
Langkah 2: Tulis data 100
Langkah 3: Baca data 131
Langkah 4: Perbarui data 157
Langkah 5: Cari data 189
Langkah 6: (Opsional) membersihkan 225
Langkah selanjutnya 243
Cara kerjanya 244
Kisi-kisi 244
Pengaturan awal 244
SDK atau CLI 245
Tindakan dasar 245
Peraturan penamaan 246
Dasar-dasar kuota layanan 247
Informasi lain 249
Komponen inti 249
Tabel, item, dan atribut 250
Kunci primer 253
Indeks sekunder 255
DynamoDB Streams 258
DynamoDB API 259
Bidang kontrol 260
Bidang data 260
DynamoDB Streams 262
Transaksi 263
Jenis data yang didukung dan aturan penamaan 263
Peraturan penamaan 264
Jenis Data 265
Deskriptor jenis daya 270
Kelas tabel DynamoDB 270

Partisi dan distribusi data di DynamoDB ... 271
Distribusi data: Kunci partisi ... 272
Distribusi data: Kunci partisi dan kunci urutan ... 273
Pelajari cara beralih dari SQL ke NoSQL ... 276
Relasional atau NoSQL? ... 277
Mengakses dan otentikasi ... 280
Membuat tabel ... 283
Mendapatkan informasi tentang tabel ... 285
Menulis data ke tabel ... 287
Membaca data dari tabel ... 292
Mengelola indeks ... 300
Memodifikasi data dalam tabel ... 306
Menghapus data dari tabel ... 309
Menghapus tabel ... 312
Sumber daya dan alat pembelajaran Amazon DynamoDB ... 313
Alat untuk pengodean dan visualisasi ... 313
Panduan Preskriptif ... 313
Pusat Pengetahuan ... 315
Posting blog, repositori, dan panduan ... 316
Pemodelan data dan pola desain ... 316
Kursus pelatihan ... 317
Membaca dan menulis ... 318
DynamoDB membaca konsistensi ... 318
Akhirnya konsisten membaca ... 318
Bacaan yang sangat konsisten ... 319
Tabel global membaca konsistensi ... 319
Membaca dan menulis operasi ... 319
Baca konsumsi operasi ... 319
Tulis konsumsi operasi ... 321
Kapasitas throughput DynamoDB ... 323
Mode sesuai permintaan ... 323
Mode yang disediakan ... 324
Mode kapasitas sesuai permintaan DynamoDB ... 324
Unit permintaan baca dan unit permintaan tulis ... 326
Throughput awal dan properti penskalaan ... 326
DynamoDB throughput maksimum untuk tabel sesuai permintaan ... 327

Tabel kapasitas yang disediakan ... 329
Membaca dan menulis unit kapasitas ... 331
Memilih pengaturan throughput awal ... 331
Penskalaan otomatis DynamoDB ... 333
Mengelola kapasitas throughput dengan penskalaan otomatis ... 334
Kapasitas terpesan ... 365
Throughput hangat ... 366
Periksa throughput hangat meja Anda ... 367
Tingkatkan throughput hangat meja Anda ... 370
Buat tabel dengan throughput hangat yang lebih tinggi ... 377
Skenario throughput hangat ... 388
Kapasitas burst dan adaptif ... 390
Kapasitas lonjakan ... 390
Kapasitas adaptif ... 390
Mode kapasitas pengalihan ... 392
Mode yang disediakan ke mode sesuai permintaan ... 392
Mode sesuai permintaan ke mode yang disediakan ... 394
Pemrograman dengan DynamoDB ... 396
Ikhtisar dukungan AWS SDK untuk DynamoDB ... 396
Dukungan SDK dari endpoint berbasis AWS akun ... 398
Antarmuka terprogram yang bekerja dengan DynamoDB ... 399
Antarmuka pemrograman tingkat tinggi ... 406
Menjalankan contoh kode ... 468
API tingkat rendah ... 475
Pemrograman dengan Python ... 481
Tentang Boto ... 482
Dokumentasi Boto ... 482
Lapisan klien dan sumber daya ... 483
Menggunakan batch_writer ... 486
Contoh kode tambahan ... 487
Sesi dan keamanan utas ... 487
Config ... 488
Penanganan kesalahan ... 493
Pencatatan log ... 495
Kait acara ... 497
Pagination dan Paginator ... 498

Pelayan 500
Pemrograman dengan JavaScript 500
Tentang AWS SDK untuk JavaScript 501
AWS SDK untuk JavaScript V3 501
JavaScript dokumentasi 502
Lapisan abstraksi 502
Fungsi utilitas Marshall 504
Membaca item 505
Penulisan bersyarat 507
Paginasi 508
Config 510
Pelayan 513
Penanganan kesalahan 514
Pencatatan log 516
Pertimbangan 517
Pemrograman dengan AWS SDK for Java 2.x 518
Tentang AWS SDK for Java 2.x 518
Memulai 519
Dokumentasi SDK for Java 2.x 529
Antarmuka yang didukung 529
Contoh kode tambahan 544
Sinkronisasi dan pemrograman async 544
Klien HTTP 545
Config 547
Penanganan kesalahan 554
AWS ID permintaan 555
Pencatatan log 555
Paginasi 558
Anotasi kelas data 559
Penanganan kesalahan 560
Komponen kesalahan 560
Kesalahan transaksional 561
Pesan dan kode kesalahan 561
Penanganan kesalahan dalam aplikasi Anda 565
Percobaan ulang kesalahan dan penundaan eksponensial 566
Operasi batch dan penanganan kesalahan 567

Bekerja dengan AWS SDKs ... 568
Bekerja dengan DynamoDB ... 570
Bekerja dengan tabel ... 570
Operasi dasar pada tabel ... 571
Pertimbangan saat memilih kelas tabel di DynamoDB ... 580
Tag dan label ... 581
Menggunakan tabel global ... 587
Mereplikasi data secara mulus di Wilayah dengan tabel global ... 588
Memberikan keamanan dan akses untuk tabel global Anda dengan AWS KMS ... 589
Cara kerjanya ... 590
Praktik terbaik dan persyaratan ... 595
Tutorial: Membuat tabel global ... 598
Memantau tabel global ... 604
Menggunakan IAM dengan tabel global DynamoDB ... 605
Menentukan versi ... 608
Meningkatkan tabel global ... 611
Penagihan tabel global ... 620
Bekerja dengan item ... 623
Ukuran dan format item ... 624
Membaca item ... 625
Menulis item ... 626
Nilai yang dikembalikan ... 629
Operasi batch ... 630
Penghitung atom ... 633
Penulisan bersyarat ... 634
Menggunakan ekspresi ... 640
Waktu untuk Hidup (TTL) ... 681
Tabel kueri ... 711
Memindai tabel ... 720
Bahasa kueri PartiQL ... 729
Bekerja dengan item: Java ... 776
Bekerja dengan item: .NET ... 808
Bekerja dengan indeks ... 841
Indeks sekunder global ... 846
Indeks Sekunder Lokal di DynamoDB ... 908
Bekerja dengan transactions ... 962

Cara kerjanya 963
Menggunakan IAM dengan transactions 973
Contoh kode 976
Bekerja dengan stream 980
Opsi 981
Bekerja dengan Kinesis Data Streams 982
Bekerja dengan DynamoDB Streams 1000
Akselerasi dalam memori dengan DAX 1079
Kasus penggunaan untuk DAX 1080
Catatan penggunaan DAX 1081
Cara kerjanya 1082
Cara DAX memproses permintaan 1084
Cache item 1085
Cache kueri 1086
Komponen klaster DAX 1087
Simpul 1088
Klaster 1088
Wilayah dan zona ketersediaan 1090
Grup parameter 1090
Grup keamanan 1090
ARN klaster 1091
Titik akhir klaster 1091
Titik akhir simpul 1091
Grup subnet 1092
Peristiwa 1092
Periode pemeliharaan 1092
Membuat klaster DAX 1094
Membuat peran layanan IAM untuk DAX agar dapat mengakses DynamoDB 1094
Menggunakan AWS CLI 1096
Menggunakan konsol 1102
Model konsistensi 1107
Konsistensi di antara simpul klaster DAX 1108
Perilaku cache item DAX 1108
Perilaku cache kueri DAX 1112
Bacaan sangat konsisten dan transaksional 1113
Caching negatif 1113

Strategi untuk penulisan 1114
Melakukan pengembangan dengan klien DAX 1117
Tutorial: Menjalankan aplikasi sampel 1117
Memodifikasi aplikasi yang ada untuk menggunakan DAX 1182
Mengelola klaster DAX 1184
Izin IAM untuk mengelola klaster DAX 1184
Menskalakan klaster DAX 1187
Menyesuaikan pengaturan klaster DAX 1188
Mengonfigurasi pengaturan TTL 1189
Dukungan penandaan untuk DAX 1191
AWS CloudTrail integrasi 1192
Menghapus klaster DAX 1192
Memantau DynamoDB Accelerator 1193
Alat pemantauan DAX 1193
Pemantauan CloudWatch dengan 1195
Mencatat operasi DAX menggunakan AWS CloudTrail 1222
Instans DAX T3/T2 yang dapat melonjak 1223
Rangkaian instans T2 DAX 1223
Rangkaian instans T3 DAX 1223
Kontrol akses DAX 1224
Peran layanan IAM untuk DAX 1225
Kebijakan IAM untuk mengizinkan akses klaster DAX 1227
Studi Kasus: Mengakses DynamoDB dan DAX 1228
Akses ke DynamoDB, tetapi tidak ada akses ke DAX 1229
Akses ke DynamoDB dan DAX 1231
Akses ke DynamoDB melalui DAX, tetapi tidak ada akses langsung ke DynamoDB 1237
Enkripsi DAX saat diam 1239
Mengaktifkan enkripsi saat diam menggunakan AWS Management Console 1241
Enkripsi bergerak DAX 1242
Menggunakan peran tertaut layanan untuk DAX 1243
Izin peran tertaut layanan untuk DAX 1244
Membuat peran tertaut layanan untuk DAX 1245
Mengedit peran tertaut layanan untuk DAX 1246
Menghapus peran tertaut layanan untuk DAX 1246
Mengakses DAX di seluruh akun AWS 1247
Menyiapkan IAM 1248

Menyiapkan VPC ... 1251
Mengubah klien DAX untuk mengizinkan akses lintas akun ... 1253
Panduan pengukuran klaster DAX ... 1257
Gambaran Umum ... 1258
Memperkirakan lalu lintas ... 1258
Pengujian beban ... 1259
Pemodelan data ... 1262
Bekerja dengan Kumpulan Item ... 1263
Mempercepat kueri dengan mengatur data Anda dengan koleksi item ... 1265
Fondasi pemodelan data ... 1265
Desain tabel tunggal ... 1266
Desain multitabel ... 1268
Blok bangunan pemodelan data ... 1270
Kunci urutan komposit ... 1270
Multi-penghunian ... 1272
Indeks jarang ... 1273
Waktu untuk tayang ... 1274
Pengarsipan waktu untuk tayang ... 1276
Partisi vertikal ... 1276
Sharding penulisan ... 1279
Paket desain skema pemodelan data ... 1280
Prasyarat ... 1281
Jejaring sosial ... 1282
Profil game ... 1291
Sistem manajemen pengaduan ... 1300
Pembayaran berulang ... 1319
Pembaruan status perangkat ... 1324
Toko online ... 1337
Pemodelan relasional ... 1361
Model basis data relasional traditional ... 1362
Cara DynamoDB menghilangkan kebutuhan akan operasi JOIN ... 1364
Bagaimana transaksi DynamoDB menghilangkan overhead ke proses tulis ... 1365
Langkah pertama ... 1366
Contoh ... 1368
Migrasi ke DynamoDB ... 1373
Alasan untuk bermigrasi ... 1373

Pertimbangan saat bermigrasi 1374
Cara kerjanya 1376
Alat Migrasi 1377
Memilih strategi migrasi 1378
Migrasi offline 1380
Migrasi hibrida 1383
Online - memigrasikan setiap tabel 1:1 1384
Online - migrasi dengan tabel penahapan khusus 1385
NoSQL Workbench 1388
Unduh 1389
Menginstal 1391
Pemodel data 1398
Membuat model baru 1398
Mengimpor model yang ada 1405
Mengekspor model 1408
Mengedit model yang ada 1410
Pemvisualisasi data 1414
Menambahkan data sampel 1414
Mengimpor dari CSV 1417
Faset 1418
Tampilan agregat 1421
Melakukan model data 1422
Pembangun operasi 1425
Menghubungkan ke set data 1426
Membangun operasi 1427
Tabel kloning 1439
Mengekspor ke CSV 1440
Model data sampel 1441
Model data karyawan 1441
Model data forum diskusi 1442
Model data pustaka musik 1442
Model data resor ski 1443
Model data penawaran kartu kredit 1443
Model data bookmark 1444
Riwayat rilis 1444
Pencadangan dan pemulihan 1451

Point-in-time cadangan 1453
Sebelum Anda mulai 1453
Aktifkan point-in-time pemulihan 1454
Pencadangan sesuai permintaan 1458
Cara kerjanya 1459
Mencadangkan tabel 1462
Memulihkan tabel 1465
Menghapus cadangan tabel 1471
Menggunakan IAM 1472
Penagihan untuk cadangan 1479
Cara kerjanya 1479
Contoh penagihan cadangan DynamoDB 1480
Memulihkan 1484
Memulihkan tabel menggunakan point-in-time pemulihan 1484
Memulihkan tabel DynamoDB ke titik waktu 1486
Menggunakan AWS Backup 1492
Cara kerjanya 1494
Membuat cadangan 1497
Menyalin cadangan 1498
Memulihkan tabel 1500
Menghapus cadangan 1501
Pencadangan sesuai permintaan yang dikelola oleh versus DynamoDB AWS Backup 1501
Contoh kode 1503
Hal-hal mendasar 1515
Halo DynamoDB 1516
Pelajari dasar-dasarnya 1525
Tindakan 1677
Skenario 2069
Mempercepat pembacaan dengan DAX 2072
Skenario Indeks Sekunder Global Tingkat Lanjut 2080
Membangun aplikasi untuk mengirimkan data ke tabel DynamoDB 2083
Bandingkan beberapa nilai dengan satu atribut 2084
Perbarui TTL item secara kondisional 2107
Connect ke instance lokal 2114
Hitung operator ekspresi 2115
Membuat API REST untuk melacak data COVID-19 2147

Membuat aplikasi messenger ... 2148
Membuat aplikasi nirserver untuk mengelola foto ... 2149
Buat tabel dengan indeks sekunder global ... 2154
Buat tabel dengan throughput hangat diaktifkan ... 2158
Membuat aplikasi web untuk melacak data DynamoDB ... 2168
Membuat aplikasi obrolan websocket ... 2170
Buat item dengan TTL ... 2170
Hapus data menggunakan PartiQL DELETE ... 2177
Mendeteksi APD dalam gambar ... 2183
Masukkan data menggunakan PartiQL INSERT ... 2184
Menginvokasi fungsi Lambda dari browser ... 2189
Kelola Indeks Sekunder Global ... 2189
Mengelola kebijakan berbasis sumber daya ... 2192
Pantau kinerja DynamoDB ... 2195
Lakukan operasi kueri lanjutan ... 2195
Lakukan operasi daftar ... 2230
Lakukan operasi peta ... 2263
Lakukan operasi set ... 2287
Melakukan kueri pada tabel menggunakan batch pernyataan PartiQL ... 2317
Melakukan kueri tabel menggunakan PartiQL ... 2379
Kueri tabel menggunakan Global Secondary Index ... 2434
Kueri tabel menggunakan kondisi begins_with ... 2443
Kueri tabel menggunakan rentang tanggal ... 2449
Kueri tabel dengan ekspresi filter yang kompleks ... 2458
Kueri tabel dengan ekspresi filter dinamis ... 2465
Kueri tabel dengan ekspresi filter dan batas ... 2474
Kueri tabel dengan atribut bersarang ... 2479
Kueri tabel dengan pagination ... 2488
Kueri tabel dengan pembacaan yang sangat konsisten ... 2499
Kueri data menggunakan PartiQL SELECT ... 2503
Kueri untuk item TTL ... 2509
Tabel kueri menggunakan pola tanggal dan waktu ... 2513
Menyimpan EXIF dan informasi gambar lainnya ... 2527
Mengatur Kontrol Akses Berbasis Atribut ... 2528
Memahami urutan ekspresi pembaruan ... 2531
Perbarui pengaturan throughput hangat tabel ... 2563

Perbarui TTL item ... 2568
Perbarui data menggunakan PartiQL UPDATE ... 2572
Menggunakan API Gateway untuk menginvokasi fungsi Lambda ... 2579
Menggunakan Step Functions untuk menginvokasi fungsi Lambda ... 2581
Menggunakan model dokumen ... 2581
Menggunakan model persistensi objek tingkat tinggi ... 2597
Gunakan operasi penghitung atom ... 2607
Gunakan operasi bersyarat ... 2630
Gunakan nama atribut ekspresi ... 2660
Menggunakan peristiwa terjadwal untuk menginvokasi fungsi Lambda ... 2687
Bekerja dengan Indeks Sekunder Lokal ... 2689
Bekerja dengan Streams dan Time-to-Live ... 2692
Bekerja dengan tabel global dan replikasi multi-wilayah ... 2695
Bekerja dengan penandaan sumber daya ... 2698
Bekerja dengan enkripsi tabel ... 2700
Contoh nirserver ... 2702
Memanggil fungsi Lambda dari pemicu DynamoDB ... 2702
Melaporkan kegagalan item batch untuk fungsi Lambda dengan pemicu DynamoDB ... 2712
AWS kontribusi komunitas ... 2723
Membangun dan menguji aplikasi tanpa server ... 2723
Keamanan ... 2726
AWS kebijakan terkelola ... 2727
AWS kebijakan terkelola ... 2727
AWS kebijakan terkelola: AmazonDynamo DBFull Access_v2 ... 2728
AmazonDynamoDBReadOnlyAccess ... 2728
DynamoDB memperbarui kebijakan terkelola AWS ... 2729
Kebijakan berbasis sumber daya ... 2731
Membuat tabel ... 2733
Lampirkan kebijakan berbasis sumber daya ... 2739
Lampirkan kebijakan ke aliran ... 2743
Hapus kebijakan berbasis sumber daya ... 2746
Akses lintas akun ... 2747
Memblokir akses publik ... 2748
Operasi API ... 2751
Otorisasi IAM ... 2760
Contoh ... 2760

Pertimbangan ... 2767
Praktik terbaik ... 2768
Kontrol akses berbasis atribut ... 2769
Mengapa saya harus menggunakan ABAC? ... 2771
Kunci syarat ... 2772
Pertimbangan ... 2772
Mengaktifkan ABAC di DynamoDB ... 2773
Menggunakan ABAC ... 2776
Contoh kasus penggunaan ... 2777
Pemecahan Masalah ... 2790
Perlindungan data ... 2792
Enkripsi saat diam ... 2792
Keamanan koneksi DynamoDB ... 2819
IAM ... 2820
Identity and Access Management ... 2821
Ketentuan penggunaan ... 2857
Validasi kepatuhan ... 2881
Ketahanan ... 2883
Keamanan infrastruktur ... 2884
Menggunakan titik akhir VPC ... 2884
AWS PrivateLink untuk DynamoDB ... 2894
Jenis titik akhir VPC Amazon ... 2895
Pertimbangan saat menggunakan AWS PrivateLink untuk Amazon DynamoDB ... 2896
Membuat titik akhir Amazon VPC ... 2896
Mengakses titik akhir antarmuka Amazon DynamoDB ... 2897
Mengakses tabel DynamoDB dan mengontrol operasi API dari titik akhir antarmuka DynamoDB ... 2897
Memperbarui konfigurasi DNS on-premise ... 2899
Membuat kebijakan titik akhir Amazon VPC ... 2901
AWS PrivateLink untuk DynamoDB Streams ... 2902
Konfigurasi dan analisis kerentanan ... 2908
Praktik terbaik keamanan ... 2908
Praktik terbaik keamanan pencegahan ... 2909
Praktik terbaik keamanan detektif ... 2911
Pencatatan dan pemantauan ... 2915
Rencana pemantauan ... 2915

Acuan dasar performa ... 2915
Layanan terintegrasi ... 2916
Alat-alat pemantauan otomatis ... 2916
Metrik pemantauan ... 2917
Bagaimana cara menggunakan metrik DynamoDB? ... 2917
Melihat metrik di konsol CloudWatch ... 2919
Melihat metrik di AWS CLI ... 2919
Metrik dan dimensi ... 2920
Membuat CloudWatch alarm di DynamoDB ... 2947
Operasi penebangan ... 2951
Informasi DynamoDB di CloudTrail ... 2951
Memahami entri file log DynamoDB ... 2955
Wawasan Kontributor ... 2974
Cara kerjanya ... 2975
Memulai ... 2981
Menggunakan IAM ... 2987
Praktik terbaik ... 2993
Desain NoSQL ... 2994
NoSQL vs. RDBMS ... 2994
Dua konsep utama ... 2995
Pendekatan umum ... 2995
NoSQL Workbench ... 2996
DynamoDB Well-Architected Lens ... 2997
Optimasi biaya ... 2997
Melakukan tinjauan Amazon DynamoDB Well-Architected Lens ... 3049
Pilar Amazon DynamoDB Well-Architected Lens ... 3049
Desain kunci partisi ... 3052
Mendistribusikan beban kerja ... 3053
Pembagian tulis ... 3054
Mengunggah data secara efisien ... 3056
Desain kunci urutan ... 3057
Kontrol versi ... 3058
Indeks sekunder ... 3059
Pedoman umum ... 3060
Indeks jarang ... 3063
Agregasi ... 3065

Muatan berlebih GSI ... 3066
Pembagian GSI ... 3067
Membuat replika ... 3072
Item besar ... 3073
Kompresi ... 3074
Partisi vertikal ... 3074
Menggunakan Amazon S3 ... 3075
Data deret waktu ... 3075
Pola desain untuk data deret waktu ... 3075
Contoh tabel deret waktu ... 3076
Many-to-many hubungan ... 3077
Daftar kedekatan ... 3078
Grafik terwujud ... 3079
Mengkueri dan memindai ... 3084
Performa pemindaian ... 3084
Menghindari lonjakan ... 3085
Pemindaian paralel ... 3088
Desain tabel ... 3089
Desain tabel global ... 3089
Desain tabel global ... 3090
Fakta kunci ... 3090
Kasus penggunaan ... 3092
Mode tulis ... 3093
Meminta perutean ... 3101
Mengevakuasi Wilayah ... 3110
Kapasitas throughput untuk tabel global ... 3113
Daftar periksa dan FAQ untuk tabel global ... 3115
Bidang kontrol ... 3122
Operasi data massal ... 3123
Pembaruan batch bersyarat ... 3123
Operasi massal yang efisien ... 3128
Menerapkan kontrol versi ... 3130
Kapan menggunakan pola ini ... 3130
Desain pola ... 3131
Menggunakan pola ... 3132
Laporan Penagihan dan Penggunaan ... 3134

Kapasitas Throughput ... 3138
Pengaliran ... 3142
Penyimpanan ... 3143
Pencadangan dan Pemulihan ... 3144
Transfer Data ... 3148
CloudWatch ... 3148
DAX ... 3150
Migrasi tabel DynamoDB dari satu akun ke akun lainnya ... 3151
Migrasi tabel menggunakan AWS Backup pencadangan dan pemulihan lintas akun ... 3152
Migrasikan tabel menggunakan ekspor ke S3 dan impor dari S3 ... 3154
Panduan preskriptif DAX ... 3156
Mengevaluasi kesesuaian DAX ... 3157
Mengkonfigurasi klien DAX Anda ... 3160
Mengonfigurasi klaster DAX Anda ... 3161
Mengukur kluster DAX Anda ... 3167
Menerapkan cluster ... 3174
Operasi cluster ... 3175
Memantau DAX ... 3178
Menggunakan DynamoDB dengan layanan lain AWS ... 3182
Melakukan integrasi dengan Amazon Cognito ... 3182
Mengintegrasikan dengan Amazon Redshift ... 3185
Integrasi nol-ETL dengan Amazon Redshift ... 3185
Memuat data dari DynamoDB ke Amazon Redshift dengan COPY ... 3193
Mengintegrasikan dengan Amazon EMR ... 3195
Gambaran Umum ... 3195
Tutorial: Menggunakan Amazon DynamoDB dan Apache Hive ... 3196
Membuat tabel eksternal di Hive ... 3205
Memproses pernyataan HiveQL ... 3209
Mengkueri data di DynamoDB ... 3210
Menyalin data ke dan dari Amazon DynamoDB ... 3212
Penyempurnaan performa ... 3226
Integrasi dengan S3 ... 3232
Mengimpor dari Amazon S3 ... 3233
Ekspor ke Amazon S3 ... 3255
Integrasi dengan Amazon SageMaker Lakehouse ... 3277
Integrasi nol-ETL dengan Amazon Lakehouse SageMaker ... 3277

Integrasi dengan Amazon Service OpenSearch ... 3279
Cara kerjanya ... 3280
Membuat integrasi ... 3280
Langkah selanjutnya ... 3281
Menangani perubahan yang menyebabkan gangguan ... 3281
Integrasi nol-ETL dengan Layanan OpenSearch ... 3285
Integrasi dengan Amazon EventBridge ... 3288
Cara kerjanya ... 3289
Membuat integrasi melalui konsol ... 3289
Langkah selanjutnya ... 3292
Integrasi dengan Amazon MSK ... 3292
Cara kerjanya ... 3293
Contoh integrasi ... 3294
Langkah selanjutnya ... 3300
Praktik terbaik integrasi ... 3301
Membuat snapshot ... 3301
Tangkapan data perubahan ... 3301
Menggunakan AI generatif dengan DynamoDB ... 3303
Kasus penggunaan AI generatif untuk DynamoDB ... 3303
Blog AI generatif untuk DynamoDB ... 3304
Memanfaatkan integrasi DynamoDB Zero-ETL dengan Layanan OpenSearch ... 3304
Kuota dan batasan ... 3306
Melakukan tugas manajemen kuota ... 3306
Mengakses kuota DynamoDB ... 3306
Melihat kuota saat ini di konsol ... 3307
Melihat kuota saat ini menggunakan AWS CLI ... 3307
Meminta peningkatan kuota ... 3309
Kuota ... 3310
Membaca/menulis throughput ... 3311
Kapasitas Terpesan ... 365
Tabel ... 3314
Tabel global ... 3314
Indeks sekunder ... 3315
Atribut indeks sekunder yang diproyeksikan ... 3315
DynamoDB Streams ... 3316
Mengimpor dari Amazon S3 ... 3316

Ekspor tabel ke Amazon S3 ... 3317
Pencadangan dan pemulihan ... 3317
Wawasan Kontributor ... 3317
Batasan ... 3317
Mode kapasitas baca/tulis ... 3318
Indeks sekunder ... 3315
Tombol partisi dan kunci urutan ... 3319
Peraturan penamaan ... 3320
Jenis Data ... 3321
Item ... 3321
Atribut ... 3322
Parameter ekspresi ... 3322
Transaksi DynamoDB ... 3323
DynamoDB Streams ... 3324
DynamoDB Accelerator (DAX) ... 3324
Kendala khusus API ... 3325
Enkripsi DynamoDB saat diam ... 3328
Referensi API ... 3329
Memecahkan masalah ... 3330
Kesalahan server internal ... 3330
Menyelidiki kesalahan server internal ... 3331
Meminimalkan dampak dari kesalahan server internal ... 3332
Meningkatkan kesadaran operasional ... 3332
Latensi ... 3336
Masalah pelambatan ... 3339
Mode yang disediakan ... 3340
Mode sesuai permintaan ... 3342
Menggunakan CloudWatch metrik ... 3344
Lampiran ... 3346
Memecahkan masalah pembuatan koneksi SSL/TLS dengan DynamoDB ... 3346
Menguji aplikasi atau layanan Anda ... 3346
Menguji browser klien Anda ... 3347
Memperbarui klien aplikasi perangkat lunak Anda ... 3347
Memperbarui browser klien Anda ... 3348
Memperbarui paketan sertifikat Anda secara manual ... 3348
Contoh tabel dan data untuk digunakan di DynamoDB ... 3349

File data sampel ... 3350
Membuat contoh tabel dan mengunggah data ... 3363
Membuat contoh tabel dan mengunggah data - Java ... 3363
Membuat contoh tabel dan mengunggah data - .NET ... 3373
Contoh aplikasi menggunakan AWS SDK untuk Python (Boto3) ... 3385
Langkah 1: Deploy dan uji secara lokal ... 3386
Langkah 2: Periksa model data dan detail implementasi ... 3391
Langkah 3: Deploy dalam produksi ... 3401
Langkah 4: Bersihkan Sumber Daya ... 3410
Kata-kata penggunaan khusus di DynamoDB ... 3411
AWS Contoh SDK for Java 1.x ... 3424
DAX dan Java SDK v1 ... 3424
Memodifikasi SDK untuk Java 1.x yang ada agar menggunakan DAX ... 3437
Mengkueri indeks sekunder global dengan SDK untuk Java 1.x ... 3442
AWS Contoh SDK for Go 1.x ... 3445
Go dan DAX ... 3446
AWS Contoh SDK untuk Node.js 2.x ... 3448
Node.js dan DAX ... 3448
Riwayat dokumen ... 3459
Pembaruan lebih awal ... 3495
Fitur warisan ... 3532
Versi tabel global 2017.11.29 (Legacy) ... 3532
Cara kerjanya ... 3533
Praktik Terbaik dan Persyaratan ... 3538
Membuat tabel global ... 3542
Memantau tabel global ... 3546
Menggunakan IAM dengan tabel global ... 3548
Versi API DynamoDB tingkat rendah sebelumnya (2011-12-05) ... 3551
BatchGetItem ... 3552
BatchWriteItem ... 3559
CreateTable ... 3567
DeleteItem ... 3574
DeleteTable ... 3581
DescribeTables ... 3585
GetItem ... 3589
ListTables ... 3593

PutItem 3596
Kueri 3603
Scan 3620
UpdateItem 3641
UpdateTable 3651
Parameter bersyarat DynamoDB lama 3656
AttributesToGet 3658
AttributeUpdates 3659
ConditionalOperator 3662
Expected 3662
KeyConditions 3668
QueryFilter 3671
ScanFilter 3673
Syarat penulisan dengan parameter warisan 3675
Fitur pratinjau 3685
Konsistensi kuat Multi-Region 3685
Mode konsistensi untuk tabel global 3686
Ketersediaan wilayah untuk pratinjau MRSC 3687
Pertimbangan pratinjau MRSC 3688
Mengelola tabel global MRSC 3689
..... mmmdcxcv

Apa itu Amazon DynamoDB?

Amazon DynamoDB adalah database tanpa server, NoSQL, yang dikelola sepenuhnya dengan kinerja milidetik satu digit pada skala apa pun.

DynamoDB memenuhi kebutuhan Anda untuk mengatasi penskalaan dan kompleksitas operasional database relasional. DynamoDB dibuat khusus dan dioptimalkan untuk beban kerja operasional yang memerlukan kinerja yang konsisten pada skala apa pun. Misalnya, DynamoDB memberikan kinerja milidetik satu digit yang konsisten untuk kasus penggunaan keranjang belanja, apakah Anda memiliki 10 atau 100 juta pengguna. Diluncurkan pada tahun 2012, DynamoDB terus membantu Anda menjauh dari database relasional sekaligus mengurangi biaya dan meningkatkan kinerja dalam skala besar.

Pelanggan di semua ukuran, industri, dan geografi menggunakan DynamoDB untuk membangun aplikasi modern tanpa server yang dapat dimulai dari skala kecil dan global. DynamoDB menskalakan untuk mendukung tabel dari hampir semua ukuran sambil memberikan kinerja milidetik satu digit yang konsisten dan ketersediaan tinggi.

Untuk acara, seperti Amazon Prime Day, DynamoDB mendukung beberapa properti dan sistem Amazon dengan lalu lintas tinggi, termasuk Alexa, situs Amazon.com, dan semua pusat pemenuhan Amazon. Untuk acara semacam itu, APIs DynamoDB telah menangani triliunan panggilan dari properti dan sistem Amazon. DynamoDB terus melayani ratusan pelanggan dengan tabel yang memiliki lalu lintas puncak lebih dari setengah juta permintaan per detik. Ini juga melayani ratusan pelanggan yang ukuran tabelnya melebihi 200 TB, dan memproses lebih dari satu miliar permintaan per jam.

Topik

- Karakteristik DynamoDB
- Kasus penggunaan DynamoDB
- Kemampuan DynamoDB
- Integrasi layanan
- Keamanan
- Ketahanan
- Mengakses DynamoDB
- Harga DynamoDB

- Mulai menggunakan DynamoDB

Karakteristik DynamoDB

Nirserver

Dengan DynamoDB, Anda tidak perlu menyediakan server apa pun, atau menambal, mengelola, menginstal, memelihara, atau mengoperasikan perangkat lunak apa pun. DynamoDB menyediakan pemeliharaan downtime nol. Ini tidak memiliki versi (mayor, minor, atau patch), dan tidak ada jendela pemeliharaan.

Mode kapasitas sesuai permintaan DynamoDB menawarkan pay-as-you-go harga untuk permintaan baca dan tulis sehingga Anda hanya membayar untuk apa yang Anda gunakan. Dengan sesuai permintaan, DynamoDB langsung menaikkan atau menurunkan tabel Anda untuk menyesuaikan kapasitas dan mempertahankan kinerja tanpa administrasi. Ini juga menurunkan skala ke nol sehingga Anda tidak membayar untuk throughput ketika tabel Anda tidak memiliki lalu lintas dan tidak ada start dingin.

NoSQL

Sebagai database NoSQL, DynamoDB dibuat khusus untuk memberikan peningkatan kinerja, skalabilitas, pengelolaan, dan fleksibilitas dibandingkan dengan database relasional tradisional. Untuk mendukung berbagai macam kasus penggunaan, DynamoDB mendukung model data nilai kunci dan dokumen.

Tidak seperti database relasional, DynamoDB tidak mendukung operator JOIN. Kami menyarankan Anda mendenormalisasi model data Anda untuk mengurangi perjalanan pulang pergi basis data dan daya pemrosesan yang diperlukan untuk menjawab pertanyaan. Sebagai database NoSQL, DynamoDB menyediakan konsistensi baca yang kuat dan transaksi ACID untuk membangun aplikasi kelas perusahaan.

Dikelola sepenuhnya

Sebagai layanan database yang dikelola sepenuhnya, DynamoDB menangani beban berat yang tidak terdiferensiasi dalam mengelola database sehingga Anda dapat fokus pada membangun nilai bagi pelanggan Anda. Ini menangani pengaturan, konfigurasi, pemeliharaan, ketersediaan tinggi, penyediaan perangkat keras, keamanan, cadangan, pemantauan, dan banyak lagi. Ini memastikan bahwa ketika Anda membuat tabel DynamoDB, itu langsung siap untuk beban kerja produksi.

DynamoDB terus meningkatkan ketersediaan, keandalan, kinerja, keamanan, dan fungsionalitasnya tanpa memerlukan upgrade atau downtime.

Kinerja milidetik satu digit pada skala apa pun

DynamoDB dibuat khusus untuk meningkatkan kinerja dan skalabilitas database relasional untuk memberikan kinerja milidetik satu digit pada skala apa pun. Untuk mencapai skala dan kinerja ini, DynamoDB dioptimalkan untuk beban kerja berkinerja tinggi dan APIs menyediakan yang mendorong penggunaan database yang efisien. Ini menghilangkan fitur yang tidak efisien dan tidak berkinerja dalam skala, misalnya, operasi JOIN. DynamoDB memberikan kinerja milidetik satu digit yang konsisten untuk aplikasi Anda, baik Anda memiliki 100 atau 100 juta pengguna.

Kasus penggunaan DynamoDB

Pelanggan di semua ukuran, industri, dan geografi menggunakan DynamoDB untuk membangun aplikasi modern tanpa server yang dapat dimulai dari skala kecil dan global. DynamoDB sangat ideal untuk kasus penggunaan yang memerlukan kinerja yang konsisten pada skala apa pun dengan overhead operasional yang sedikit hingga nol. Daftar berikut menyajikan beberapa kasus penggunaan di mana Anda dapat menggunakan DynamoDB:

- Aplikasi layanan keuangan — Misalkan Anda adalah perusahaan jasa keuangan yang membangun aplikasi, seperti perdagangan langsung dan perutean, manajemen pinjaman, pembuatan token, dan buku besar transaksi. Dengan tabel global DynamoDB, aplikasi Anda dapat merespons peristiwa dan melayani lalu lintas dari Wilayah AWS pilihan Anda dengan kinerja baca dan tulis lokal yang cepat.

  DynamoDB cocok untuk aplikasi dengan persyaratan ketersediaan paling ketat. Ini menghilangkan beban operasional instans penskalaan secara manual untuk peningkatan penyimpanan atau throughput, pembuatan versi, dan lisensi.

  Anda dapat menggunakan transaksi DynamoDB untuk mencapai atomisitas, konsistensi, isolasi, dan daya tahan (ACID) di satu atau beberapa tabel dengan satu permintaan. (ACID) transaksi sesuai dengan beban kerja yang mencakup pemrosesan transaksi keuangan atau pemenuhan pesanan. DynamoDB langsung mengakomodasi beban kerja Anda saat naik atau turun, memungkinkan Anda untuk meningkatkan skala database Anda secara efisien untuk kondisi pasar, seperti jam perdagangan.
- Aplikasi game — Sebagai perusahaan game, Anda dapat menggunakan DynamoDB untuk semua bagian platform game, misalnya, status game, data pemain, riwayat sesi, dan papan

peringkat. Pilih DynamoDB untuk skala, kinerja yang konsisten, dan kemudahan operasi yang disediakan oleh arsitektur tanpa servernya. DynamoDB sangat cocok untuk arsitektur scale-out yang diperlukan untuk mendukung game yang sukses. Ini dengan cepat meningkatkan throughput game Anda baik masuk maupun keluar (skala ke nol tanpa awal yang dingin). Skalabilitas ini mengoptimalkan efisiensi arsitektur Anda apakah Anda sedang meningkatkan lalu lintas puncak atau menskalakan kembali saat penggunaan gameplay rendah.

- Aplikasi streaming — Perusahaan media dan hiburan menggunakan DynamoDB sebagai indeks metadata untuk konten, layanan manajemen konten, atau untuk melayani statistik olahraga yang mendekati waktu nyata. Mereka juga menggunakan DynamoDB untuk menjalankan layanan daftar pantauan dan bookmark pengguna dan memproses miliaran acara pelanggan harian untuk menghasilkan rekomendasi. Pelanggan ini mendapat manfaat dari skalabilitas, kinerja, dan ketahanan DynamoDB. DynamoDB menskalakan perubahan beban kerja saat naik atau turun, memungkinkan kasus penggunaan media streaming yang dapat mendukung tingkat permintaan apa pun.

Untuk mempelajari lebih lanjut tentang cara pelanggan dari berbagai industri menggunakan DynamoDB, lihat Pelanggan Amazon DynamoDB dan Ini adalah Arsitektur Saya.

Kemampuan DynamoDB

Replikasi multi-aktif dengan tabel global

Tabel global menyediakan replikasi multi-aktif data Anda di seluruh pilihan Anda Wilayah AWS dengan ketersediaan 99,999%. Tabel global memberikan solusi yang dikelola sepenuhnya untuk menerapkan database multi-wilayah, multi-aktif, tanpa membangun dan memelihara solusi replikasi Anda sendiri. Dengan tabel global, Anda dapat menentukan Wilayah AWS di mana Anda ingin tabel tersedia. DynamoDB mereplikasi perubahan data yang sedang berlangsung ke semua tabel ini.

Aplikasi Anda yang didistribusikan secara global dapat mengakses data secara lokal di Wilayah yang dipilih untuk mencapai kinerja baca dan tulis milidetik satu digit. Karena tabel global bersifat multi-aktif, Anda tidak memerlukan tabel utama. Ini berarti tidak ada kegagalan yang rumit atau tertunda, atau downtime database saat gagal atas aplikasi antar Wilayah.

Transaksi ACID

DynamoDB dibangun untuk beban kerja mission-critical. Ini termasuk dukungan transaksi (ACID) untuk aplikasi yang membutuhkan logika bisnis yang kompleks. DynamoDB menyediakan dukungan

asli, sisi server untuk transaksi, menyederhanakan pengalaman pengembang membuat terkoordinasi all-or-nothing, perubahan ke beberapa item di dalam dan di seluruh tabel.

Ubah pengambilan data untuk arsitektur berbasis peristiwa

DynamoDB mendukung streaming rekaman penangkapan data perubahan tingkat item (CDC) dalam waktu hampir nyata. Ini menawarkan dua model streaming untuk CDC: DynamoDB Streams dan Kinesis Data Streams untuk DynamoDB. Setiap kali aplikasi membuat, memperbarui, atau menghapus item dalam tabel, stream merekam urutan urutan waktu dari setiap perubahan tingkat item dalam waktu nyaris real time. Hal ini membuat DynamoDB Streams ideal untuk aplikasi dengan arsitektur event-driven untuk mengkonsumsi dan bertindak atas perubahan.

Indeks sekunder

DynamoDB menawarkan opsi untuk membuat indeks sekunder global dan lokal, yang memungkinkan Anda menanyakan data tabel menggunakan kunci alternatif. Dengan indeks sekunder ini, Anda dapat mengakses data dengan atribut selain kunci utama, memberi Anda fleksibilitas maksimum dalam mengakses data Anda.

Integrasi layanan

DynamoDB terintegrasi secara luas dengan Layanan AWS beberapa untuk membantu Anda mendapatkan nilai lebih dari data Anda, menghilangkan pengangkatan berat yang tidak berdiferensiasi, dan mengoperasikan beban kerja Anda dalam skala besar. Beberapa contoh adalah: AWS CloudFormation, Amazon CloudWatch, Amazon S3, AWS Identity and Access Management (IAM), dan. AWS Auto Scaling Bagian berikut menjelaskan beberapa integrasi layanan yang dapat Anda lakukan menggunakan DynamoDB:

Integrasi tanpa server

Untuk membangun aplikasi end-to-end tanpa server, DynamoDB terintegrasi secara native dengan sejumlah tanpa server. Layanan AWS Misalnya, Anda dapat mengintegrasikan DynamoDB AWS Lambda dengan untuk membuat pemicu, yang merupakan potongan kode yang secara otomatis merespons peristiwa di DynamoDB Streams. Dengan pemicu, Anda dapat membangun aplikasi berbasis peristiwa yang bereaksi terhadap modifikasi data dalam tabel DynamoDB. Untuk pengoptimalan biaya, Anda dapat memfilter peristiwa yang diproses Lambda dari aliran DynamoDB.

Daftar berikut menyajikan beberapa contoh integrasi tanpa server dengan DynamoDB:

- AWS AppSyncuntuk membuat GraphQL APIs
- Amazon API Gateway untuk membuat REST APIs
- Lambda untuk komputasi tanpa server
- Amazon Kinesis Data Streams untuk pengambilan data perubahan (CDC)

Mengimpor dan mengekspor data ke Amazon S3

Mengintegrasikan DynamoDB dengan Amazon S3 memungkinkan Anda mengekspor data dengan mudah ke bucket Amazon S3 untuk analitik dan pembelajaran mesin. DynamoDB mendukung ekspor tabel penuh dan ekspor tambahan untuk mengekspor data yang diubah, diperbarui, atau dihapus antara periode waktu tertentu. Anda juga dapat mengimpor data dari Amazon S3 ke tabel DynamoDB baru.

Integrasi nol-ETL

DynamoDB mendukung integrasi nol-ETL dengan Amazon Redshift dan OpenSearch Menggunakan pipeline Ingestion dengan Amazon DynamoDB. Integrasi ini memungkinkan Anda menjalankan analisis kompleks dan menggunakan kemampuan pencarian lanjutan pada data tabel DynamoDB Anda. Misalnya, Anda dapat melakukan pencarian teks lengkap dan vektor, dan pencarian semantik pada data DynamoDB Anda. Integrasi nol-ETL tidak berdampak pada beban kerja produksi yang berjalan di DynamoDB.

Pembuatan cache

DynamoDB Accelerator (DAX) adalah layanan caching yang dikelola sepenuhnya dan sangat tersedia yang dibangun untuk DynamoDB. DAX memberikan peningkatan kinerja hingga 10 kali lipat - dari milidetik ke mikrodetik - bahkan pada jutaan permintaan per detik. DAX melakukan semua pekerjaan berat yang diperlukan untuk menambahkan akselerasi dalam memori ke tabel DynamoDB Anda, tanpa mengharuskan Anda mengelola pembatalan cache, populasi data, atau manajemen klaster.

Keamanan

DynamoDB menggunakan IAM untuk membantu Anda mengontrol akses ke sumber daya DynamoDB Anda dengan aman. Dengan IAM, Anda dapat mengelola izin secara terpusat yang mengontrol pengguna DynamoDB mana yang dapat mengakses sumber daya. Anda menggunakan

IAM untuk mengontrol siapa yang diautentikasi (masuk) dan diotorisasi (memiliki izin) untuk menggunakan sumber daya. Karena DynamoDB menggunakan IAM, tidak ada nama pengguna atau kata sandi untuk mengakses DynamoDB. Karena Anda tidak memiliki kebijakan rotasi kata sandi yang rumit untuk dikelola, ini menyederhanakan postur keamanan Anda. Dengan IAM, Anda juga dapat mengaktifkan kontrol akses berbutir halus untuk memberikan otorisasi pada tingkat atribut. Anda juga dapat menentukan kebijakan berbasis sumber daya dengan dukungan untuk IAM Access Analyzer dan Block Public Access (BPA) untuk menyederhanakan manajemen kebijakan.

Secara default, DynamoDB mengenkripsi semua data pelanggan saat istirahat. Enkripsi saat istirahat meningkatkan keamanan data Anda dengan menggunakan kunci enkripsi yang disimpan di AWS Key Management Service(AWS KMS). Dengan enkripsi saat diam, Anda dapat membuat aplikasi yang sensitif terhadap keamanan yang memenuhi persyaratan kepatuhan dan peraturan enkripsi yang ketat. Ketika Anda mengakses tabel terenkripsi, DynamoDB mendekripsi data tabel secara transparan. Anda tidak perlu mengubah kode atau aplikasi apa pun untuk menggunakan atau mengelola tabel terenkripsi. DynamoDB terus memberikan latensi milidetik satu digit yang sama dengan yang Anda harapkan, dan semua kueri DynamoDB bekerja dengan mulus pada data terenkripsi Anda.

Anda dapat menentukan apakah DynamoDB harus menggunakan Kunci milik AWS (tipe enkripsi default) Kunci yang dikelola AWS, atau kunci yang dikelola Pelanggan untuk mengenkripsi data pengguna. Enkripsi default menggunakan kunci KMS yang AWS dimiliki tersedia tanpa biaya tambahan. Untuk enkripsi sisi klien, Anda dapat menggunakan AWS Database Encryption SDK.

DynamoDB juga mematuhi beberapa standar kepatuhan, termasuk HIPAA, PCI DSS, dan GDPR, yang memungkinkan Anda memenuhi persyaratan peraturan.

Ketahanan

Secara default, DynamoDB secara otomatis mereplikasi data Anda di tiga Availability Zone untuk memberikan daya tahan tinggi dan SLA ketersediaan 99,99%. DynamoDB juga menyediakan kemampuan tambahan untuk membantu Anda mencapai kesinambungan bisnis dan tujuan pemulihan bencana.

DynamoDB menyertakan fitur-fitur berikut untuk membantu mendukung ketahanan data dan kebutuhan cadangan Anda:

Fitur

- Tabel global

- Pencadangan dan pemulihan berkelanjutan point-in-time
- Pencadangan dan pemulihan sesuai permintaan

Tabel global

Tabel global DynamoDB memungkinkan SLA ketersediaan 99,999% dan ketahanan Multi-wilayah. Ini membantu Anda membangun aplikasi yang tangguh dan mengoptimalkannya untuk tujuan waktu pemulihan terendah (RTO) dan tujuan titik pemulihan (RPO). Tabel global juga terintegrasi dengan AWS Fault Injection Service (AWS FIS) untuk melakukan eksperimen injeksi kesalahan pada beban kerja tabel global Anda. Misalnya, menjeda replikasi tabel global ke tabel replika apa pun.

Pencadangan dan pemulihan berkelanjutan point-in-time

Pencadangan berkelanjutan memberi Anda perincian per detik dan kemampuan untuk memulai pemulihan. point-in-time Dengan point-in-time pemulihan, Anda dapat mengembalikan tabel ke titik waktu mana pun hingga yang kedua selama 35 hari terakhir. Anda dapat mengatur periode pemulihan ke nilai apa pun antara 1 dan 35 hari.

Pencadangan berkelanjutan dan memulai point-in-time pemulihan tidak menggunakan kapasitas yang disediakan. Mereka juga tidak berdampak pada kinerja atau ketersediaan aplikasi Anda.

Pencadangan dan pemulihan sesuai permintaan

Pencadangan dan pemulihan sesuai permintaan memungkinkan Anda membuat cadangan lengkap tabel untuk retensi jangka panjang dan arsip untuk kebutuhan kepatuhan peraturan. Cadangan tidak memengaruhi kinerja tabel Anda dan Anda dapat mencadangkan tabel dengan ukuran berapa pun. Dengan AWS Backup integrasi, Anda dapat menggunakannya AWS Backup untuk menjadwalkan, menyalin, menandai, dan mengelola siklus hidup backup berdasarkan permintaan DynamoDB Anda secara otomatis. Dengan menggunakan AWS Backup, Anda dapat menyalin cadangan sesuai permintaan di seluruh akun dan Wilayah, dan mentransisikan cadangan lama ke penyimpanan dingin untuk pengoptimalan biaya.

Mengakses DynamoDB

Anda dapat bekerja dengan DynamoDB menggunakan, AWS Management Consolethe,, atau AWS Command Line InterfaceNoSQL Workbench untuk DynamoDBDynamoDB. APIs

Untuk informasi selengkapnya, lihat Mengakses DynamoDB.

Harga DynamoDB

DynamoDB mengenakan biaya untuk membaca, menulis, dan menyimpan data dalam tabel Anda, bersama dengan fitur opsional apa pun yang Anda pilih untuk diaktifkan. DynamoDB memiliki dua mode kapasitas dengan opsi penagihan masing-masing untuk memproses pembacaan dan penulisan di tabel Anda: sesuai permintaan dan disediakan.

DynamoDB juga termasuk dalam tingkat selalu gratis, menyediakan penyimpanan 25 GB. Tingkat Selalu gratis juga mencakup 25 Tulis yang disediakan dan 25 Unit Kapasitas Baca yang disediakan (WCU, RCU) yang cukup untuk menangani permintaan 200 M per bulan.

Untuk informasi selengkapnya, lihat harga Amazon DynamoDB.

Mulai menggunakan DynamoDB

Jika Anda pengguna pertama kali DynamoDB, kami sarankan Anda mulai dengan membaca topik berikut:

- Mulai menggunakan DynamoDB— Memandu Anda melalui proses pengaturan DynamoDB, membuat tabel sampel, dan mengunggah data. Topik ini juga memberikan informasi tentang melakukan beberapa operasi basis data dasar menggunakan,, NoSQL Workbench AWS Management Console AWS CLI, dan DynamoDB. APIs
- Komponen inti DynamoDB - Menjelaskan konsep DynamoDB dasar.
- Praktik terbaik untuk mendesain dan merancang dengan DynamoDB— Memberikan rekomendasi tentang desain NoSQL, DynamoDB Well-Architected Lens, desain meja dan beberapa fitur DynamoDB lainnya. Praktik terbaik ini membantu Anda memaksimalkan kinerja dan meminimalkan biaya throughput saat bekerja dengan DynamoDB.

Kami juga menyarankan Anda meninjau tutorial berikut yang menyajikan end-to-end prosedur lengkap untuk membiasakan diri dengan DynamoDB. Anda dapat menyelesaikan tutorial ini menggunakan fitur tier selalu gratis.

- Buat dan Kueri Tabel NoSQL dengan Amazon DynamoDB
- Membangun Aplikasi Menggunakan NoSQL Key-Value Data Store

Untuk informasi tentang sumber daya, alat, dan strategi untuk bermigrasi ke DynamoDB, lihat Migrasi ke DynamoDB. Untuk membaca blog dan whitepaper terbaru, lihat sumber daya Amazon DynamoDB.

Mulai menggunakan DynamoDB

Anda akan mempelajari cara menghubungkan, membuat, dan mengelola tabel DynamoDB di bagian berikut.

Sebelum memulai, Anda harus membiasakan diri dengan konsep dasar di Amazon DynamoDB. Anda bisa mendapatkan ikhtisar singkat Apa itu Amazon DynamoDB? dan tampilan yang lebih mendalam. Komponen inti dari Amazon DynamoDB Kemudian, lanjutkan ke Prasyarat.

> **Note**
>
> Ketika Anda mendaftar AWS, Anda dapat memulai dengan DynamoDB menggunakan AWS Tingkat Gratis. Jika Anda belum melampaui manfaat tingkat gratis untuk Amazon DynamoDB, Anda tidak perlu mengeluarkan biaya apa pun untuk melengkapi contoh di bagian ini. Jika tidak, Anda akan dikenakan biaya penggunaan DynamoDB standar sejak Anda membuat tabel hingga Anda menghapus tabel.
> Jika Anda tidak ingin mendaftar untuk akun tingkat gratis, Anda dapat mengatur DynamoDB lokal (versi yang dapat diunduh) di komputer Anda. Versi yang dapat diunduh memungkinkan Anda mengembangkan dan menguji aplikasi secara lokal tanpa mendaftar untuk AWS akun atau mengakses layanan web DynamoDB.

Topik

- Sumber daya Amazon DynamoDB untuk pengguna pertama kali
- Mengakses DynamoDB
- Prasyarat
- Menyiapkan DynamoDB
- Langkah 1: Buat tabel di DynamoDB
- Langkah 2: Menulis data ke tabel DynamoDB
- Langkah 3: Baca data dari tabel DynamoDB
- Langkah 4: Perbarui data dalam tabel DynamoDB
- Langkah 5: Kueri data dalam tabel DynamoDB
- Langkah 6: (Opsional) Hapus tabel DynamoDB Anda untuk membersihkan sumber daya

- Lanjutkan belajar tentang DynamoDB

Sumber daya Amazon DynamoDB untuk pengguna pertama kali

Kami menyarankan agar pengguna pertama kali memulai dengan membaca bagian berikut, dan merujuknya sesuai kebutuhan.

- Sorotan dan harga layanan - Halaman detail produk memberikan gambaran umum produk DynamoDB, kasus penggunaan umum, sorotan layanan, dan harga.
- Sumber daya DynamoDB - Bagian sumber daya DynamoDB memiliki video, tutorial, dan panduan preskriptif yang memperkenalkan Anda pada layanan, konsep pemodelan data, serta fitur dan kemampuan inti.
- Memulai — Mulai menggunakan DynamoDB Bagian ini mencakup informasi tentang pengaturan DynamoDB, membuat tabel sampel, dan mengunggah data. Ini juga memberikan informasi tentang melakukan operasi basis data dasar menggunakan,, NoSQL Workbench AWS Management Console AWS CLI, dan DynamoDB. APIs
- Kursus dasar DynamoDB — Kursus digital gratis yang mengajarkan dasar-dasar DynamoDB, termasuk desain tabel, tipe data, dan operasi dasar. Kursus ini sangat ideal untuk pengembang yang baru mengenal database DynamoDB dan NoSQL.
- DynamoDB Nuggets — Kumpulan tutorial video pendek dan terfokus yang menjelaskan konsep dan fitur DynamoDB utama. Video berukuran kecil ini mencakup topik-topik seperti pemodelan data, strategi partisi, pola akses, dan praktik terbaik dalam suatu format. easy-to-understand
- Repositori sampel kode DynamoDB - Menyediakan contoh kode DynamoDB praktis dalam berbagai bahasa pemrograman.
- Pelatihan DynamoDB gratis AWS - menyediakan kursus pelatihan digital gratis yang mencakup konsep, fitur, dan praktik terbaik DynamoDB. Seri kursus deep-dive DynamoDB membantu Anda mempelajari cara merancang model data yang efisien, menerapkan kunci partisi yang efektif, dan menggunakan indeks sekunder secara efektif.
- NoSQL Workbench for DynamoDB - NoSQL Workbench adalah alat visual terpadu yang menyediakan pemodelan data, visualisasi data, dan fitur pengembangan kueri. Ini membantu Anda merancang, membuat, menanyakan, dan mengelola tabel DynamoDB, membuatnya sangat berharga untuk mempelajari konsep pemodelan data DynamoDB dan menguji pola akses.
- Pola Desain DynamoDB — Contoh pemodelan Praktik terbaik A dan data yang menunjukkan praktik terbaik untuk kasus penggunaan yang berbeda. Termasuk contoh praktis dan contoh kode untuk menerapkan solusi yang efektif.

- Tutorial Hands-on — Step-by-step tutorial AWS Management Console yang memandu Anda melalui tugas DynamoDB umum, mulai dari membuat tabel hingga menerapkan kueri yang kompleks.
- Migrasi ke DynamoDB — Migrasi ke DynamoDB Ini memberikan gambaran umum tentang proses, alat, dan strategi untuk memigrasikan database ke DynamoDB.
- AWS Well-Architected Lens untuk DynamoDB - the section called “DynamoDB Well-Architected Lens” The menyediakan praktik terbaik arsitektur untuk merancang dan mengoperasikan aplikasi yang andal, aman, efisien, dan hemat biaya menggunakan DynamoDB. Ini mencakup prinsip-prinsip desain, pola arsitektur, dan panduan operasional yang selaras dengan enam pilar Kerangka Well-Architected.

Amazon DynamoDB praktik terbaik tambahan untuk pengguna pertama kali

Setelah Anda menyelesaikan bagian terdahulu, baca bagian ini:

- Kapasitas throughput DynamoDB

  Memberikan gambaran umum dari dua mode throughput yang tersedia untuk DynamoDB dan pertimbangan dalam memilih mode kapasitas yang sesuai untuk aplikasi Anda. Mode on-demand adalah opsi throughput default dan direkomendasikan untuk sebagian besar beban kerja DynamoDB.
- Praktik terbaik

  Identifikasi dan atasi masalah untuk memaksimalkan kinerja dan meminimalkan biaya saat bekerja dengan DynamoDB.

AWS CLI sumber daya

Jika Anda ingin menggunakan AWS Command Line Interface (AWS CLI), Anda dapat menggunakan dokumen-dokumen ini untuk membantu Anda memulai:

- Dokumentasi AWS CLI

  Bagian ini memberikan informasi tentang mengunduh AWS CLI, mendapatkan AWS CLI pekerjaan pada sistem Anda, dan memberikan AWS kredensil Anda.
- AWS CLI dokumentasi untuk DynamoDB

Dokumen terpisah ini mencakup semua perintah AWS CLI for DynamoDB, termasuk sintaks dan contoh.

Sumber daya pemrograman

Anda dapat menulis program aplikasi untuk menggunakan DynamoDB API dengan berbagai bahasa pemrograman populer. Berikut adalah beberapa sumber daya:

- Alat untuk Amazon Web Services

  AWS menyediakan sejumlah kit pengembangan perangkat lunak (SDKs) dengan dukungan untuk DynamoDB. Anda dapat membuat kode untuk DynamoDB menggunakan Java, .NET, PHP, Ruby, dan bahasa lainnya. Ini SDKs dapat sangat menyederhanakan pengembangan aplikasi Anda dengan memformat permintaan Anda ke DynamoDB, mengurai respons, dan menyediakan logika coba lagi dan penanganan kesalahan.

- DynamoDB API Referensi

  Jika Anda tidak ingin menggunakan AWS SDKs, Anda dapat berinteraksi dengan DynamoDB secara langsung menggunakan DynamoDB API. Dokumen ini mencakup semua operasi DynamoDB API, termasuk sintaks dan contoh. Pada bagian ini, Anda dapat menemukan tips pemecahan masalah dan informasi tentang membuat dan melakukan autentikasi atas permintaan serta menangani tanggapan.

Mengakses DynamoDB

Anda dapat mengakses Amazon DynamoDB menggunakan API, AWS Management Console the AWS CLI(), AWS Command Line Interface atau DynamoDB.

Topik

- Menggunakan konsol
- Menggunakan AWS CLI
- Menggunakan API
- Menggunakan NoSQL Workbench untuk DynamoDB
- Rentang alamat IP

Menggunakan konsol

Anda dapat mengakses Amazon https://console.aws.amazon.com/dynamodb/ DynamoDB AWS Management Console untuk di rumah.

Berikut adalah beberapa tindakan yang dapat Anda lakukan di konsol DynamoDB:

- Kelola tabel: Buat, perbarui, dan hapus tabel. Kalkulator kapasitas dapat membantu memperkirakan kebutuhan kapasitas.
- Berinteraksi dengan data: Melihat, menambah, memperbarui, dan menghapus item dalam tabel Anda. Kelola pengaturan Time to Live (TTL).
- Pantau dan analisis: Lihat dasbor, pantau dan atur alarm, serta analisis metrik dan peringatan untuk tabel DynamoDB Anda.
- Optimalkan dan perluas: Kelola indeks sekunder, aliran, pemicu, kapasitas cadangan, dan fitur canggih lainnya untuk meningkatkan penggunaan DynamoDB Anda.

Konsol DynamoDB menyediakan antarmuka yang komprehensif untuk mengelola sumber daya DynamoDB Anda. Kami mendorong Anda untuk mengakses konsol dan berinteraksi dengannya untuk mempelajari lebih lanjut.

Menggunakan AWS CLI

Anda dapat menggunakan AWS Command Line Interface (AWS CLI) untuk mengontrol beberapa AWS layanan dari baris perintah dan mengotomatiskannya melalui skrip. Anda dapat menggunakan operasi AWS CLI for ad hoc, seperti membuat tabel. Anda juga dapat menggunakannya untuk menanamkan operasi Amazon DynamoDB dalam skrip utilitas.

Sebelum Anda dapat menggunakan AWS CLI dengan DynamoDB, Anda harus mendapatkan ID kunci akses dan kunci akses rahasia. Untuk informasi selengkapnya, lihat Memberikan akses terprogram .

Untuk daftar lengkap semua perintah yang tersedia untuk DynamoDB di, lihat AWS CLIAWS CLI referensi perintah.

Mengunduh dan mengonfigurasi AWS CLI

AWS CLI Tersedia di http://aws.amazon.com/cli. Alat ini berjalan di Windows, macOS, atau Linux. Setelah Anda mengunduh AWS CLI, ikuti langkah-langkah berikut untuk menginstal dan mengkonfigurasinya:

1. Buka Panduan Pengguna AWS Command Line Interface.
2. Ikuti petunjuk untuk Menginstal AWS CLI dan Mengonfigurasi AWS CLI.

Menggunakan AWS CLI dengan DynamoDB

Format baris perintah terdiri dari nama operasi DynamoDB, diikuti oleh parameter untuk operasi tersebut. AWS CLI Mendukung sintaks singkatan untuk nilai parameter, serta JSON.

Misalnya, perintah berikut membuat tabel bernama Music. Kunci partisi adalah Artist, dan kunci sortir adalah SongTitle. (Agar mudah dibaca, perintah panjang di bagian ini dipecah menjadi baris terpisah.)

```
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
```

Perintah berikut menambahkan item baru ke tabel. Contoh-contoh ini menggunakan kombinasi sintaks singkatan dan 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
```

Pada baris perintah, menulis JSON yang valid bisa menjadi hal yang sulit. Namun, AWS CLI dapat membaca file JSON. Sebagai contoh, perhatikan potongan kode JSON berikut, yang disimpan dalam sebuah file bernama key-conditions.json.

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

Sekarang, Anda dapat menerbitkan permintaan `Query` menggunakan AWS CLI. Dalam contoh ini, isi file key-conditions.json digunakan untuk parameter `--key-conditions`.

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

Menggunakan AWS CLI dengan DynamoDB lokal

Ini juga AWS CLI dapat berinteraksi dengan DynamoDB lokal (versi yang dapat diunduh) yang berjalan di komputer Anda. Untuk mengaktifkan hal ini, tambahkan parameter berikut ke setiap perintah:

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

Contoh berikut AWS CLI menggunakan daftar tabel dalam database lokal.

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

Jika DynamoDB menggunakan nomor port selain nomor default (8000), modifikasi nilai `--endpoint-url` sebagaimana mestinya.

> **Note**
>
> Tidak AWS CLI dapat menggunakan DynamoDB lokal (versi yang dapat diunduh) sebagai titik akhir default. Oleh karena itu, Anda harus menetapkan `--endpoint-url` dengan setiap perintah.

Menggunakan API

Anda dapat menggunakan AWS Management Console dan AWS Command Line Interface untuk bekerja secara interaktif dengan Amazon DynamoDB. Namun, untuk mendapatkan hasil maksimal dari DynamoDB, Anda dapat menulis kode aplikasi menggunakan. AWS SDKs

AWS SDKs Memberikan dukungan luas untuk DynamoDB di JavaJavaScript , dibrowser, .NET, Node.js, PHP,Python, Ruby, C ++,Go, Android, dan iOS.

Sebelum Anda dapat menggunakan AWS SDKs dengan DynamoDB, Anda harus mendapatkan AWS ID kunci akses dan kunci akses rahasia. Untuk informasi selengkapnya, lihat Menyiapkan DynamoDB (layanan web) .

Untuk ikhtisar tingkat tinggi pemrograman aplikasi DynamoDB dengan, lihat. AWS SDKs Pemrograman dengan DynamoDB dan AWS SDKs

Menggunakan NoSQL Workbench untuk DynamoDB

Anda juga dapat mengakses DynamoDB dengan mengunduh dan menggunakan NoSQL Workbench untuk DynamoDB.

NoSQL Workbench untuk Amazon DynamoDB adalah aplikasi GUI sisi klien lintas platform yang dapat Anda gunakan untuk pengembangan dan operasi basis data modern. Ini tersedia untuk Windows, macOS, dan Linux. NoSQL Workbench adalah alat pengembangan visual yang menyediakan fitur pemodelan data, visualisasi data, dan pengembangan kueri untuk membantu Anda merancang, membuat, mengkueri, dan mengelola tabel DynamoDB. NoSQL Workbench kini menyertakan DynamoDB lokal sebagai bagian opsional dari proses instalasi, yang mempermudah Anda untuk memodelkan data Anda di DynamoDB lokal. Untuk mempelajari selengkapnya tentang DynamoDB lokal dan persyaratannya, lihat Menyiapkan DynamoDB lokal (versi yang dapat diunduh) .

> **Note**
>
> NoSQL Workbench untuk DynamoDB saat ini tidak AWS mendukung login yang dikonfigurasi dengan otentikasi dua faktor (2FA).

Pemodelan data

Dengan NoSQL Workbench untuk DynamoDB, Anda dapat membuat model data baru dari, atau mendesain model berdasarkan, model data yang sudah ada yang memenuhi pola akses data aplikasi Anda. Anda juga dapat mengimpor dan mengekspor model data yang didesain pada akhir proses. Untuk informasi selengkapnya, lihat Membangun Model Data dengan NoSQL Workbench.

Visualisasi data

Pemvisualisasi model data menyediakan kanvas tempat Anda dapat memetakan kueri dan memvisualisasikan pola akses (faset) aplikasi tanpa harus menulis kode. Setiap faset berhubungan dengan pola akses yang berbeda di DynamoDB. Anda dapat membuat data sampel secara otomatis untuk digunakan dalam model data Anda. Untuk informasi selengkapnya, lihat Memvisualisasikan pola akses data.

Pembangunan operasi

NoSQL Workbench menyediakan antarmuka pengguna grafis yang kaya bagi Anda untuk mengembangkan dan menguji kueri. Anda dapat menggunakan pembangun operasi untuk melihat, menjelajahi, dan mengkueri dataset langsung. Anda juga dapat menggunakan pembangun operasi terstruktur untuk membuat dan menjalankan operasi bidang data. Fitur ini mendukung proyeksi dan ekspresi kondisi, serta memungkinkan Anda menghasilkan kode sampel dalam berbagai bahasa. Untuk informasi selengkapnya, lihat Menjelajahi set data dan membangun operasi dengan NoSQL Workbench.

Rentang alamat IP

Amazon Web Services (AWS) menerbitkan rentang alamat IP saat ini dalam format JSON. Untuk melihat rentang saat ini, unduh ip-ranges.json. Untuk informasi selengkapnya, lihat Rentang alamat IP AWS di Referensi Umum AWS.

Untuk menemukan rentang alamat IP yang dapat Anda gunakan untuk mengakses tabel dan indeks DynamoDB, cari file ip-ranges.json untuk string berikut: `"service": "DYNAMODB"`.

> **Note**
>
> Rentang alamat IP tidak berlaku untuk DynamoDB Streams atau DynamoDB Accelerator (DAX).

Prasyarat

Sebelum memulai tutorial Amazon DynamoDB, pelajari tentang cara Anda dapat mengakses DynamoDB. Mengakses DynamoDB Kemudian, atur DynamoDB melalui layanan web atau versi yang diunduh secara lokal di. Menyiapkan DynamoDB Setelah itu, lanjutkan keLangkah 1: Buat tabel di DynamoDB.

> **Note**
>
> - Jika Anda berencana untuk berinteraksi dengan DynamoDB hanya melalui AWS Management Console, Anda tidak memerlukan AWS kunci akses. Selesaikan langkah-langkah dalam Mendaftar AWS, dan kemudian lanjutkan keLangkah 1: Buat tabel di DynamoDB.
> - Jika tidak ingin mendaftar untuk akun tingkat gratis, Anda dapat mengatur DynamoDB Lokal (Versi yang dapat diunduh). Kemudian lanjutkan ke Langkah 1: Buat tabel di DynamoDB.
> - Ada beberapa perbedaan saat menggunakan perintah CLI di terminal di Linux dan Windows. Panduan berikut menyajikan perintah yang diformat untuk terminal Linux (ini termasuk macOS), dan perintah yang diformat untuk Windows CMD. Pilih perintah yang paling sesuai dengan aplikasi terminal yang Anda gunakan.

Menyiapkan DynamoDB

Selain layanan web Amazon DynamoDB AWS , menyediakan versi DynamoDB yang dapat diunduh yang dapat Anda jalankan di komputer Anda. Versi yang dapat diunduh berguna untuk mengembangkan dan menguji kode Anda. Ini memungkinkan Anda menulis dan menguji aplikasi secara lokal tanpa mengakses layanan web DynamoDB.

Topik di bagian ini menjelaskan cara menyiapkan DynamoDB (versi yang dapat diunduh) dan layanan web DynamoDB.

Topik

- Menyiapkan DynamoDB (layanan web)
- Menyiapkan DynamoDB lokal (versi yang dapat diunduh)

Menyiapkan DynamoDB (layanan web)

Untuk menggunakan layanan web Amazon DynamoDB:

1. Daftar AWS. .
2. Dapatkan kunci AWS akses (digunakan untuk mengakses DynamoDB secara terprogram).

> **Note**
>
> Jika Anda berencana untuk berinteraksi dengan DynamoDB hanya melalui AWS Management Console, Anda tidak memerlukan AWS kunci akses, dan Anda dapat melompat ke depan. Menggunakan konsol

3. Konfigurasikan kredensial Anda (digunakan untuk mengakses DynamoDB secara terprogram).

Mendaftar untuk AWS

Untuk menggunakan layanan DynamoDB, Anda harus memiliki akun. AWS Jika belum memiliki akun, Anda diminta membuatnya saat mendaftar. Anda tidak dikenakan biaya untuk AWS layanan apa pun yang Anda daftarkan kecuali Anda menggunakannya.

Untuk mendaftar AWS

1. Buka https://portal.aws.amazon.com/billing/pendaftaran.
2. Ikuti petunjuk online.

   Bagian dari prosedur pendaftaran melibatkan menerima panggilan telepon atau pesan teks dan memasukkan kode verifikasi pada keypad telepon.

   Saat Anda mendaftar untuk sebuah Akun AWS, sebuah Pengguna root akun AWSdibuat. Pengguna root memiliki akses ke semua Layanan AWS dan sumber daya di akun. Sebagai praktik keamanan terbaik, tetapkan akses administratif ke pengguna, dan gunakan hanya pengguna root untuk melakukan tugas yang memerlukan akses pengguna root.

Memberikan akses terprogram

Sebelum Anda dapat mengakses DynamoDB secara terprogram atau melalui AWS CLI(), Anda harus memiliki akses AWS Command Line Interface terprogram. Anda tidak memerlukan akses terprogram jika Anda berencana menggunakan hanya konsol DynamoDB.

Pengguna membutuhkan akses terprogram jika mereka ingin berinteraksi dengan AWS luar. AWS Management Console Cara untuk memberikan akses terprogram tergantung pada jenis pengguna yang mengakses AWS.

Untuk memberi pengguna akses programatis, pilih salah satu opsi berikut.

Pengguna mana yang membutuhkan akses programatis?	Untuk	Oleh
Identitas tenaga kerja (Pengguna yang dikelola di Pusat Identitas IAM)	Gunakan kredensi sementara untuk menandatangani permintaan terprogram ke AWS CLI,, AWS SDKs atau. AWS APIs	Mengikuti petunjuk untuk antarmuka yang ingin Anda gunakan. • Untuk AWS CLI, lihat Mengkonfigurasi yang akan AWS CLI digunakan AWS IAM Identity Center dalam Panduan AWS Command Line Interface Pengguna. • Untuk AWS SDKs, alat, dan AWS APIs, lihat Autentikasi Pusat Identitas IAM di Panduan Referensi Alat AWS SDKs dan Alat.
IAM	Gunakan kredensi sementara untuk menandatangani permintaan terprogram ke AWS CLI,, AWS SDKs atau. AWS APIs	Mengikuti petunjuk dalam Menggunakan kredensil sementara dengan AWS sumber daya di Panduan Pengguna IAM.

Pengguna mana yang membutuhkan akses programatis?	Untuk	Oleh
IAM	(Tidak direkomendasikan) Gunakan kredensi jangka panjang untuk menandatangani permintaan terprogram ke AWS CLI,, AWS SDKs atau. AWS APIs	Mengikuti petunjuk untuk antarmuka yang ingin Anda gunakan. • Untuk mengetahui AWS CLI, lihat Mengautentikasi menggunakan kredensil pengguna IAM di Panduan Pengguna.AWS Command Line Interface • Untuk AWS SDKs dan alat, lihat Mengautentikasi menggunakan kredensil jangka panjang di Panduan Referensi Alat AWS SDKs dan Alat. • Untuk AWS APIs, lihat Mengelola kunci akses untuk pengguna IAM di Panduan Pengguna IAM.

Mengonfigurasi kredensial Anda

Sebelum Anda dapat mengakses DynamoDB secara terprogram atau melalui AWS CLI, Anda harus mengonfigurasi kredensialnya untuk mengaktifkan otorisasi untuk aplikasi Anda.

Ada beberapa cara untuk melakukan ini. Misalnya, Anda dapat secara manual membuat file kredensial untuk menyimpan ID kunci akses dan kunci akses rahasia Anda. Anda juga dapat menggunakan AWS CLI perintah `aws configure` untuk membuat file secara otomatis. Alternatifnya, Anda dapat menggunakan variabel lingkungan. Untuk informasi selengkapnya tentang mengonfigurasi kredensional Anda, lihat panduan pengembang SDK khusus pemrograman AWS .

Untuk menginstal dan mengkonfigurasi AWS CLI, lihatMenggunakan AWS CLI.

Integrasi dengan layanan DynamoDB lainnya

Anda dapat mengintegrasikan DynamoDB dengan banyak layanan lainnya. AWS Untuk informasi selengkapnya, lihat berikut ini:

- Menggunakan DynamoDB dengan layanan lain AWS
- AWS CloudFormation untuk DynamoDB
- Menggunakan AWS Backup dengan DynamoDB
- AWS Identity and Access Management (IAM) dan DynamoDB
- Menggunakan AWS Lambda dengan Amazon DynamoDB

Menyiapkan DynamoDB lokal (versi yang dapat diunduh)

Dengan versi Amazon DynamoDB yang dapat diunduh, Anda dapat mengembangkan dan menguji aplikasi tanpa mengakses layanan web DynamoDB. Sebaliknya, basis data bersifat mandiri di komputer Anda. Saat Anda siap untuk menyebarkan aplikasi Anda dalam produksi, Anda menghapus titik akhir lokal dalam kode, dan kemudian menunjuk ke layanan web DynamoDB.

Memiliki versi lokal ini membantu Anda menghemat throughput, penyimpanan data, dan biaya transfer data. Selain itu, Anda tidak memerlukan koneksi internet saat mengembangkan aplikasi.

DynamoDB lokal tersedia sebagai unduhan (memerlukan JRE), sebagai dependensi Apache Maven, atau sebagai citra Docker.

Jika Anda lebih memilih menggunakan layanan web Amazon DynamoDB, lihat Menyiapkan DynamoDB (layanan web) .

Topik

- Men-deploy DynamoDB secara lokal pada komputer Anda
- Catatan penggunaan DynamoDB lokal
- Riwayat rilis DynamoDB lokal
- Telemetri di DynamoDB lokal

Men-deploy DynamoDB secara lokal pada komputer Anda

Important

Toples lokal DynamoDB dapat diunduh dari tautan distribusi AWS CloudFront kami yang direferensikan di sini. Mulai 1 Januari 2025, bucket distribusi S3 lama tidak lagi aktif dan DynamoDB lokal akan didistribusikan melalui tautan distribusi saja. CloudFront

Note

- Ada dua versi utama DynamoDB lokal yang tersedia: DynamoDB local v2.x (Current) dan DynamoDB local v1.x (Legacy). Pelanggan harus menggunakan versi 2.x (Current) bila memungkinkan, karena mendukung versi terbaru Java Runtime Environment dan kompatibel dengan namespace jakarta.* untuk proyek Maven. DynamoDB local v1.x akan mencapai akhir dukungan standar mulai 1 Januari 2025. Setelah tanggal ini, v1.x tidak akan lagi menerima pembaruan atau perbaikan bug.
- DynamoDB lokal `AWS_ACCESS_KEY_ID` hanya dapat berisi huruf (A–Z, a–z) dan angka (0–9).

Unduh DynamoDB lokal

Ikuti langkah berikut untuk menyiapkan dan menjalankan DynamoDB pada komputer Anda.

Untuk menyiapkan DynamoDB pada komputer Anda

1. Unduh DynamoDB lokal secara gratis dari salah satu lokasi berikut.

Tautan Unduhan	Checksum
.tar.gz | .zip	.tar.gz.sha256 | .zip.sha256

> **Important**
>
> Untuk menjalankan DynamoDB v2.6.0 atau lebih tinggi di komputer Anda, Anda harus memiliki Java Runtime Environment (JRE) versi 17.x atau yang lebih baru. Aplikasi tidak berjalan pada versi JRE sebelumnya.

2. Setelah Anda mengunduh arsip, ekstrak isinya dan salin direktori yang diekstrak ke lokasi pilihan Anda.
3. Untuk memulai DynamoDB di komputer Anda, buka jendela prompt perintah, navigasikan ke direktori tempat Anda mengekstrak `DynamoDBLocal.jar`, dan ketik perintah berikut.

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

> **Note**
>
> Jika Anda menggunakan Windows PowerShell, pastikan untuk melampirkan nama parameter atau seluruh nama dan nilai seperti ini:
> `java -D"java.library.path=./DynamoDBLocal_lib" -jar DynamoDBLocal.jar`
> DynamoDB memproses permintaan masuk hingga Anda menghentikannya. Untuk menghentikan DynamoDB, tekan Ctrl+C pada prompt perintah.
> DynamoDB menggunakan port 8000 secara default. Jika port 8000 tidak tersedia, perintah ini memunculkan pengecualian. Untuk daftar lengkap opsi runtime DynamoDB, termasuk `-port`, masukkan perintah ini.
> `java -Djava.library.path=./DynamoDBLocal_lib -jar DynamoDBLocal.jar -help`

4. Sebelum Anda dapat mengakses DynamoDB secara terprogram atau melalui AWS Command Line Interface (AWS CLI), Anda harus mengonfigurasi kredensial Anda untuk mengaktifkan otorisasi bagi aplikasi Anda. DynamoDB yang dapat diunduh memerlukan kredensial untuk bisa berfungsi, seperti yang ditunjukkan dalam contoh berikut.

```
AWS Access Key ID: "fakeMyKeyId"
AWS Secret Access Key: "fakeSecretAccessKey"
Default Region Name: "fakeRegion"
```

Anda dapat menggunakan perintah `aws configure` dari AWS CLI untuk menyiapkan kredensial. Untuk informasi selengkapnya, lihat Menggunakan AWS CLI.

5. Mulai menulis aplikasi. Untuk mengakses DynamoDB berjalan secara lokal dengan, gunakan parameter AWS CLI. `--endpoint-url` Misalnya, gunakan perintah berikut untuk membuat daftar tabel DynamoDB.

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

Jalankan DynamoDB lokal sebagai image Docker

Versi Amazon DynamoDB yang dapat diunduh tersedia sebagai citra Docker. Untuk informasi selengkapnya, lihat dynamodb-local. Untuk melihat versi lokal DynamoDB Anda saat ini, masukkan perintah berikut:

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

Untuk contoh penggunaan DynamoDB lokal sebagai bagian dari aplikasi REST yang dibangun di atas AWS Serverless Application Model AWS SAM(), lihat aplikasi SAM DynamoDB untuk mengelola pesanan. Contoh aplikasi ini menunjukkan cara menggunakan DynamoDB lokal untuk pengujian.

Jika Anda ingin menjalankan aplikasi multi-kontainer yang juga menggunakan kontainer lokal DynamoDB, gunakan Docker Compose untuk menentukan dan menjalankan semua layanan di aplikasi Anda, termasuk DynamoDB lokal.

Untuk menginstal dan menjalankan DynamoDB lokal dengan penulisan Docker:

1. Unduh dan instal desktop Docker.
2. Salin kode berikut ke file dan simpan sebagai `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
```

Jika Anda ingin aplikasi Anda dan DynamoDB lokal berada dalam kontainer terpisah, gunakan file yaml berikut.

```
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
```

Skrip docker-compose.yml ini membuat kontainer `app-node` dan kontainer `dynamodb-local`. Skrip menjalankan perintah di kontainer `app-node` yang menggunakan AWS CLI untuk terhubung ke kontainer `dynamodb-local` dan menjelaskan batas akun dan tabel.

Untuk digunakan dengan gambar aplikasi Anda sendiri, ganti nilai `image` pada contoh di bawah dengan nilai gambar aplikasi Anda.

```
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**
>
> Skrip YAMB mengharuskan Anda menentukan kunci AWS akses dan kunci AWS rahasia, tetapi skrip tersebut tidak diharuskan menjadi kunci yang valid AWS bagi Anda untuk mengakses DynamoDB lokal.

3. Jalankan perintah baris perintah berikut:

```
docker-compose up
```

Jalankan DynamoDB lokal sebagai dependensi Apache Maven

Ikuti langkah-langkah berikut untuk menggunakan Amazon DynamoDB dalam aplikasi Anda sebagai dependensi.

Untuk men-deploy DynamoDB sebagai repositori Apache Maven

1. Unduh dan instal Apache Maven. Untuk informasi selengkapnya, lihat Mengunduh Apache Maven dan Menginstal Apache Maven.
2. Tambahkan repositori DynamoDB Maven ke file Project Object Model (POM) aplikasi Anda.

```
<!--Dependency:-->
<dependencies>
   <dependency>
      <groupId>com.amazonaws</groupId>
      <artifactId>DynamoDBLocal</artifactId>
      <version>2.6.0</version>
   </dependency>
</dependencies>
```

Contoh template untuk digunakan dengan 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>3.0.1</version>
   </parent>

<dependencies>
```

```
    <dependency>
        <groupId>com.amazonaws</groupId>
        <artifactId>DynamoDBLocal</artifactId>
        <version>2.6.0</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**
>
> Anda juga dapat menggunakan URL repositori pusat Maven.

Jalankan DynamoDB lokal di AWS CloudShell

AWS CloudShell adalah shell pra-otentikasi berbasis browser yang dapat Anda luncurkan langsung dari file. AWS Management Console Anda dapat menavigasi ke AWS CloudShell dari AWS Management Console beberapa cara yang berbeda. Untuk informasi selengkapnya, lihat Memulai dengan AWS CloudShell.

Ikuti langkah-langkah ini untuk menjalankan DynamoDB lokal di mana saja di AWS CloudShell . AWS Management Console

Untuk menjalankan DynamoDB lokal di Anda di AWS CloudShellAWS Management Console

1. Luncurkan AWS CloudShell dari antarmuka konsol, pilih yang tersedia Wilayah AWS, dan beralih ke shell pilihan Anda, seperti Bash, PowerShell, atau Z shell.
2. Untuk memilih Wilayah AWS, buka menu Pilih Wilayah dan pilih yang didukung Wilayah AWS. (Wilayah yang Tersedia disorot.)
3. Dari AWS Management Console, luncurkan AWS CloudShell dengan memilih salah satu opsi berikut:
   a. Pada bilah navigasi, pilih AWS CloudShellikon.
   b. Di kotak Cari, masukkan kata CloudShell, lalu pilih CloudShell.
   c. Di widget yang baru dikunjungi, pilih CloudShell.
   d. Dari toolbar konsol, pilih CloudShell.
4. Untuk menjalankan DynamoDB lokal AWS CloudShell di Anda dapat menggunakan alias. `dynamodb-local` Anda dapat menentukan opsi baris perintah tambahan untuk mengubah pengaturan lokal DynamoDB. Lihat the section called “Catatan penggunaan” opsi yang tersedia.

> **Note**
>
> Untuk menjalankan DynamoDB lokal di latar belakang, jalankan DynamoDB lokal menggunakan:. AWS CloudShell `dynamodb-local &`

5. Untuk mengakses DynamoDB berjalan secara lokal dengan, AWS CloudShell gunakan parameter AWS CLI. `--endpoint-url` Misalnya, gunakan perintah berikut untuk daftar tabel DynamoDB:

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

Untuk contoh proyek sampel yang menampilkan beberapa pendekatan untuk menyiapkan dan menggunakan DynamoDB lokal, termasuk mengunduh file JAR, menjalankannya sebagai image Docker, dan menggunakannya sebagai dependensi Maven, lihat DynamoDB Local Sample Java Project.

Catatan penggunaan DynamoDB lokal

Kecuali titik akhir, aplikasi yang berjalan dengan versi Amazon DynamoDB yang dapat diunduh juga harus bekerja dengan layanan web DynamoDB. Namun, saat menggunakan DynamoDB secara lokal, Anda harus mengetahui hal berikut:

- Jika Anda menggunakan *`-sharedDb`* opsi, DynamoDB membuat file database tunggal bernama .db. shared-local-instance Setiap program yang terhubung ke DynamoDB mengakses file ini. Jika Anda menghapus file, Anda kehilangan semua data yang Anda simpan di dalamnya.
- Jika Anda menghilangkan`-sharedDb`, file database bernama myaccesskeyid_region.db, dengan ID kunci AWS akses dan AWS Region seperti yang muncul dalam konfigurasi aplikasi Anda. Jika Anda menghapus file, Anda kehilangan semua data yang Anda simpan di dalamnya.
- Jika Anda menggunakan opsi `-inMemory`, DynamoDB tidak menulis file basis data apa pun sama sekali. Sebaliknya, semua data ditulis ke memori, dan data tidak disimpan saat Anda menghentikan DynamoDB.
- Jika Anda menggunakan opsi `-inMemory`, opsi `-sharedDb` juga diperlukan.
- Jika Anda menggunakan opsi `-optimizeDbBeforeStartup`, Anda juga harus menentukan parameter `-dbPath` agar DynamoDB dapat menemukan file basis datanya.
- AWS SDKs Untuk DynamoDB mengharuskan konfigurasi aplikasi Anda menentukan nilai kunci akses dan AWS nilai Region. Kecuali Anda menggunakan opsi `-sharedDb` atau `-inMemory`, DynamoDB menggunakan nilai-nilai ini untuk memberi nama file basis data lokal. Nilai-nilai ini tidak harus berupa AWS nilai yang valid untuk dijalankan secara lokal. Namun, Anda mungkin merasa nyaman menggunakan nilai yang valid sehingga Anda dapat menjalankan kode di cloud di lain waktu dengan mengubah titik akhir yang Anda gunakan.
- DynamoDB lokal selalu mengembalikan null untuk `billingModeSummary.`

- DynamoDB lokal `AWS_ACCESS_KEY_ID` hanya dapat berisi huruf (A–Z, a–z) dan angka (0–9).
- DynamoDB lokal tidak Point-in-time mendukung pemulihan (PITR).

Topik

- Opsi baris perintah
- Mengatur Titik Akhir Lokal
- Perbedaan antara DynamoDB yang dapat diunduh dan layanan web DynamoDB

Opsi baris perintah

Anda dapat menggunakan opsi baris perintah berikut dengan versi DynamoDB yang dapat diunduh:

- `-corsvalue`— Memungkinkan dukungan untuk berbagi sumber daya lintas asal (CORS) untuk. JavaScript Anda harus memberikan daftar domain tertentu yang "diizinkan" yang dipisahkan koma. Pengaturan default untuk `-cors` adalah tanda bintang (*), yang memungkinkan akses publik.
- `-dbPath value` — Direktori tempat DynamoDB menulis file basis datanya. Jika Anda tidak menentukan opsi ini, file akan ditulis ke direktori saat ini. Anda tidak dapat menentukan `-dbPath` dan `-inMemory` sekaligus.
- `-delayTransientStatuses` — Menyebabkan DynamoDB menimbulkan penundaan untuk operasi tertentu. DynamoDB (versi download) dapat melakukan beberapa tugas hampir seketika, seperti operasi pada tabel dan indeks. create/update/delete Namun, layanan DynamoDB memerlukan lebih banyak waktu untuk tugas ini. Menyetel parameter ini membantu DynamoDB yang berjalan di komputer Anda menyimulasikan perilaku layanan web DynamoDB dengan lebih dekat. (Saat ini, parameter ini menyebabkan penundaan hanya untuk indeks sekunder global yang berstatus CREATING atau DELETING.)
- `-help` — Mencetak ringkasan dan opsi penggunaan.
- `-inMemory` — DynamoDB berjalan di memori alih-alih menggunakan file basis data. Saat Anda menghentikan DynamoDB, tidak ada data yang disimpan. Anda tidak dapat menentukan `-dbPath` dan `-inMemory` sekaligus.
- `-optimizeDbBeforeStartup` — Mengoptimalkan tabel basis data yang mendasarinya sebelum memulai DynamoDB di komputer Anda. Anda juga harus menentukan `-dbPath` saat menggunakan parameter ini.
- `-port value` — Nomor port yang digunakan DynamoDB untuk berkomunikasi dengan aplikasi Anda. Jika Anda tidak menentukan opsi ini, port defaultnya adalah `8000`.

> **Note**
>
> DynamoDB menggunakan port 8000 secara default. Jika port 8000 tidak tersedia, perintah ini memunculkan pengecualian. Anda dapat menggunakan opsi `-port` untuk menentukan nomor port yang berbeda. Untuk daftar lengkap opsi runtime DynamoDB, termasuk `-port`, masukkan perintah ini:
> `java -Djava.library.path=./DynamoDBLocal_lib -jar DynamoDBLocal.jar -help`

- `-sharedDb` — Jika Anda menentukan `-sharedDb`, DynamoDB menggunakan file basis data tunggal, bukan file terpisah untuk setiap kredensial dan Wilayah.
- `-disableTelemetry` — Ketika ditentukan, DynamoDB lokal tidak akan mengirim telemetri apa pun.
- `-version`— Mencetak versi DynamoDB lokal.

Mengatur Titik Akhir Lokal

Secara default, AWS SDKs dan alat menggunakan titik akhir untuk layanan web Amazon DynamoDB. Untuk menggunakan SDKs dan alat dengan versi DynamoDB yang dapat diunduh, Anda harus menentukan titik akhir lokal:

`http://localhost:8000`

AWS Command Line Interface

Anda dapat menggunakan AWS Command Line Interface (AWS CLI) untuk berinteraksi dengan DynamoDB yang dapat diunduh.

Untuk mengakses DynamoDB yang berjalan secara lokal, gunakan parameter `--endpoint-url`. Berikut ini adalah contoh menggunakan daftar tabel di AWS CLI DynamoDB di komputer Anda.

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

> **Note**
>
> Tidak AWS CLI dapat menggunakan versi DynamoDB yang dapat diunduh sebagai titik akhir default. Karena itu, Anda harus menentukan `--endpoint-url` dengan setiap AWS CLI perintah.

AWS SDKs

Cara Anda menentukan titik akhir bergantung pada bahasa pemrograman dan AWS SDK yang Anda gunakan. Bagian berikut menjelaskan cara melakukannya:

- Java: Mengatur AWS Wilayah dan titik akhir(DynamoDB lokal mendukung SDK AWS for Java V1 dan V2)
- CodeSamples.Jawa. RegionAndEndpoint .NET: Mengatur Wilayah dan titik akhir AWS

Perbedaan antara DynamoDB yang dapat diunduh dan layanan web DynamoDB

Versi DynamoDB yang dapat diunduh ditujukan hanya untuk pengembangan dan pengujian. Sebagai perbandingan, layanan web DynamoDB adalah layanan terkelola dengan skalabilitas, ketersediaan, dan fitur daya tahan yang membuatnya sangat cocok untuk penggunaan produksi.

Versi DynamoDB yang dapat diunduh berbeda dengan layanan web dengan cara berikut:

- Wilayah AWS dan Akun AWS berbeda tidak didukung di tingkat klien.
- Pengaturan throughput yang disediakan diabaikan dalam DynamoDB yang dapat diunduh, meskipun operasi `CreateTable` memerlukannya. Untuk `CreateTable`, Anda dapat menentukan nomor yang Anda inginkan untuk throughput baca dan tulis yang disediakan, meskipun angka-angka ini tidak digunakan. Anda dapat memanggil `UpdateTable` sebanyak yang Anda inginkan per hari. Namun, perubahan nilai throughput yang disediakan akan diabaikan.
- Operasi `Scan` dilakukan secara berurutan. Pemindaian paralel tidak didukung. Parameter `Segment` dan `TotalSegments` dari operasi `Scan` akan diabaikan.
- Kecepatan operasi baca dan tulis pada data tabel hanya dibatasi oleh kecepatan komputer Anda. Operasi `CreateTable`, `UpdateTable`, dan `DeleteTable` terjadi segera, dan status tabel selalu ACTIVE. Operasi `UpdateTable` yang hanya mengubah pengaturan throughput yang disediakan pada tabel atau indeks sekunder global terjadi segera. Jika operasi `UpdateTable` membuat atau menghapus setiap indeks sekunder global, indeks tersebut melakukan transisi melalui status

normal (seperti CREATING dan DELETING) sebelum status indeks menjadi ACTIVE. Tabel tetap ACTIVE selama waktu ini.

- Operasi baca bersifat konsisten pada akhirnya. Namun, karena kecepatan DynamoDB lokal berjalan di komputer Anda, sebagian besar pembacaan tampaknya sangat konsisten.
- Metrik koleksi item dan ukuran koleksi item tidak dilacak. Dalam respons operasi, null akan dikembalikan, bukan metrik koleksi item.
- Dalam DynamoDB, terdapat batas data sebesar 1 MB yang dikembalikan per set hasil. Layanan web DynamoDB dan versi yang dapat diunduh menerapkan batas ini. Namun, ketika mengkueri indeks, layanan DynamoDB menghitung hanya ukuran kunci dan atribut terproyeksi. Sebaliknya, versi DynamoDB yang dapat diunduh menghitung ukuran keseluruhan item.
- Jika Anda menggunakan DynamoDB Streams, laju serpihan yang dibuat mungkin berbeda. Dalam layanan web DynamoDB, perilaku pembuatan serpihan sebagian dipengaruhi oleh aktivitas partisi tabel. Ketika Anda menjalankan DynamoDB secara lokal, tidak ada pembuatan partisi tabel. Dalam kedua kasus, serpihan bersifat fana, sehingga aplikasi Anda tidak harus bergantung pada perilaku serpihan.
- `TransactionConflictExceptions`tidak dilemparkan oleh DynamoDB yang dapat diunduh untuk transaksional. APIs Sebaiknya Anda menggunakan kerangka kerja mocking Java untuk menyimulasikan `TransactionConflictExceptions` di handler DynamoDB untuk menguji cara aplikasi Anda merespons transaksi yang bertentangan.
- Di layanan web DynamoDB, apakah diakses melalui konsol atau, nama tabel peka huruf AWS CLI besar/kecil. Tabel bernama `Authors` dan yang bernama `authors` bisa ada sebagai tabel terpisah. Dalam versi yang dapat diunduh, nama tabel tidak peka huruf besar-kecil, dan upaya membuat kedua tabel ini akan menghasilkan kesalahan.
- Penandaan tidak didukung dalam versi DynamoDB yang dapat diunduh.
- Versi DynamoDB yang dapat diunduh mengabaikan parameter Limit di. ExecuteStatement

Riwayat rilis DynamoDB lokal

Tabel berikut menjelaskan perubahan penting dalam setiap rilis DynamoDB lokal.

Versi	Perubahan	Deskripsi	Tanggal
2.6.0	Support tabel ARN sebagai nama tabel di DynamoDB APIs	• Ditambahkan dukungan untuk menggunakan tabel	13 Maret 2025

Versi	Perubahan	Deskripsi	Tanggal
	Perbaikan kinerja dan pembaruan keamanan	ARN sebagai nama tabel di beberapa DynamoDB APIs • Memperbaiki `CreateStreamTable` bug pada mesin berkinerja tinggi, seperti Mac M3 • Memutakhirkan dependensi untuk memperbaiki masalah kerentanan (CVE-2022-49043, CVE-2024-56732, CVE-2020-29582, CVE-2025-21502, CVE-2024-50602, CVE-2025-24970, CVE-2025-25193)	
2.5.4	Upgrade ke Dependensi Jetty	• Upgrade dari Jetty 12.0.8 ke Jetty 12.0.14 (Menyelesaikan CVE-2024-6763, CVE-2024-8184, CVE-2024-47535) Mitigasi untuk (CVE-2024-21634)	Desember 12, 2024

Versi	Perubahan	Deskripsi	Tanggal
2.5.3	Memutakhirkan Dependensi Jackson ke 2.17.x di Log4j Core (Menyelesaikan CVE-2022-1471)	• Memutakhirkan Dependensi Jackson ke 2.17.x di Log4j Core (Menyelesaikan CVE-2022-1471) untuk mengatasi kerentanan keamanan kritis di perpustak aan SnakeYaml, yang merupakan ketergantungan transitif	6 November 2024
2.5.2	Perbaikan bug untuk alur kerja tabel Perbarui	• Perbaikan bug untuk alur kerja saat tabel pembaruan mencoba memperbarui tabel dengan Mode Penagihan Ondemand ke Disediakan Dengan GSI	Juni 20, 2024
2.5.1	Patch untuk bug yang diperkenalkan dalam `0ndemandT hroughPut` fitur	• Memperbaiki beberapa bug yang terkait dengan `0ndemandT hroughPut`	Juni 5, 2024

Versi	Perubahan	Deskripsi	Tanggal
2.5.0	Support untuk throughput maksimum yang dapat dikonfigurasi untuk tabel sesuai permintaan,,, dan `ReturnValuesOnConditionCheckFailure` `BatchExecuteStatement` `ExecuteTransaction` `Request`	• Menambahkan telemetri ke Mode Tertanam • Memperbaiki SDKv2 terjemahan untuk `ConditionalCheckException`	28 Mei 2024
2.4.0	Dukungan untuk `ReturnValuesOnConditionCheckFailure` - Mode Tertanam	• Perbaikan Mode Tertanam `TrimmedDataAccessException` untuk Operasi di Beberapa Aliran • Memperbaiki terjemahan pengecualian untuk SDKv2 dalam Mode Tertanam	April 17, 2024
2.3.0	Peningkatan Dermaga dan JDK	• Upgrade ke Jetty 12.0.2 • Upgrade ke JDK 17 • Upgrade ANTLR4 ke 4.10.1	Maret 14, 2024

Versi	Perubahan	Deskripsi	Tanggal
2.2.0	Ditambahkan dukungan untuk perlindungan penghapusan tabel dan parameter `ReturnValuesOnConditionCheckFailure`	• Menambahkan dukungan perlindungan Hapus Tabel • Ditambahkan dukungan untuk ReturnValuesOnConditionCheckFailure • Ditambahkan dukungan untuk bendera -version	14 Desember 2023
2.1.0	Support untuk SQLLite Native Libraries untuk proyek Maven dan menambahkan telemetri	• Menambahkan telemetri ke DynamoDB lokal • Menyalin Perpustakaan SQLLite Asli secara dinamis untuk proyek Maven • Menghapus pustaka io.github.ganadist.sqlite4java dari dependensi Maven • Meningkatkan ke GoogleGuava 32.1.1-jre	23 Oktober 2023

Versi	Perubahan	Deskripsi	Tanggal
2.0.0	Migrasi dari javax ke jakarta namespace dan Support JDK11	• Migrasi dari javax ke jakarta namespace dan dukungan JDK11 • Perbaikan untuk menangani akses tidak valid dan kunci rahasia saat server dinyalakan • Memperbaiki kerentanan yang diidentifikasi Maven dengan memperbar ui dependensi	5 Juli 2023
1.25.1	Memutakhirkan Dependensi Jackson ke 2.17.x di Log4j Core (Menyelesaikan CVE-2022-1471)	Memutakhirkan Dependensi Jackson ke 2.17.x di Log4j Core (Menyelesaikan CVE-2022-1471) untuk mengatasi kerentanan keamanan kritis di perpustakaan SnakeYaml, yang merupakan ketergant ungan transitif	6 November 2024

Versi	Perubahan	Deskripsi	Tanggal
1.25.0	Ditambahkan dukungan untuk perlindungan penghapusan tabel dan parameter `ReturnValuesOnConditionCheckFailure`	• Menambahkan dukungan perlindungan Hapus Tabel • Ditambahkan dukungan untuk ReturnValuesOnConditionCheckFailure • Ditambahkan dukungan untuk bendera -version	18 Desember 2023
1.24.0	Support untuk SQLLite Native Libraries untuk proyek Maven dan menambahkan telemetri	• Menambahkan telemetri ke DynamoDB lokal • Menyalin Perpustakaan SQLLite Asli secara dinamis untuk proyek Maven • Menghapus pustaka io.github.ganadist.sqlite4java dari dependensi Maven • Meningkatkan ke GoogleGuava 32.1.1-jre	23 Oktober 2023

Versi	Perubahan	Deskripsi	Tanggal
1.23.0	Penanganan akses tidak valid dan kunci rahasia saat server dinyalakan	• Perbaikan untuk menangani akses tidak valid dan kunci rahasia saat server dinyalakan • Memperbaiki kerentanan yang diidentifikasi Maven dengan memperbarui dependensi	28 Juni 2023
1.22.0	Dukungan Operasi Batas untuk PartiQL	• Mengoptimalkan klausul IN untuk PartiQL • Dukungan untuk Operasi Batas • Dukungan M1 untuk proyek Maven	8 Juni 2023
1.21.0	Dukungan untuk 100 tindakan per transaksi	• Meningkatkan tindakan per transaksi dari 25 menjadi 100 • Meningkatkan citra docker Open JDK ke 11 • Memperbaiki paritas untuk pengecualian yang dilemparkan saat menduplikasi item di BatchExecuteStatement	26 Januari 2023

Versi	Perubahan	Deskripsi	Tanggal
1.20.0	Menambahkan dukungan untuk M1 Mac	• Menambahkan dukungan untuk M1 Mac • Meningkatkan dependensi Jetty ke 9.4.48.v20220622	12 September 2022
1.19.0	Peningkatan Parser PartiQL	Meningkatkan Parser PartiQL dan pustaka terkait lainnya	27 Juli 2022
1.18.0	Peningkatan log4j-core dan Jackson-core	Meningkatkan log4j-core ke 2.17.1 dan Jackson-core 2.10.x ke 2.12.0	10 Januari 2022
1.17.2	Peningkatan log4j-core	Meningkatkan dependensi log4j-core ke versi 2.16	16 Januari 2021
1.17.1	Peningkatan log4j-core	Pembaruan dependensi log4j-core untuk mem-patch eksploitasi zero-day guna mencegah eksekusi kode jarak jauh - Log4Shel	10 Januari 2021

Versi	Perubahan	Deskripsi	Tanggal
1.17.0	Menghentikan penggunaan Javascript Web Shell	• Memperbarui ketergantungan AWS SDK ke AWS SDK for Java 1.12.x • Menghentikan penggunaan Javascript Web Shell	8 Januari 2021

Telemetri di DynamoDB lokal

Di AWS, kami mengembangkan dan meluncurkan layanan berdasarkan apa yang kami pelajari dari interaksi dengan pelanggan, dan kami menggunakan umpan balik pelanggan untuk mengulangi produk kami. Telemetri adalah informasi tambahan yang membantu kami lebih memahami kebutuhan pelanggan, mendiagnosis masalah, dan menghadirkan fitur yang meningkatkan pengalaman pelanggan.

DynamoDB lokal mengumpulkan telemetri, seperti metrik penggunaan umum, informasi sistem dan lingkungan, serta kesalahan. Untuk detail tentang jenis telemetri yang dikumpulkan, lihat Jenis informasi yang dikumpulkan.

DynamoDB lokal tidak mengumpulkan informasi pribadi, seperti nama pengguna atau alamat email. DynamoDB lokal juga tidak mengekstraksi informasi tingkat proyek yang sensitif.

Sebagai pelanggan, Anda mengontrol apakah telemetri diaktifkan, dan Anda dapat mengubah pengaturan kapan saja. Jika telemetri tetap aktif, DynamoDB lokal mengirimkan data telemetri di latar belakang tanpa memerlukan interaksi pelanggan tambahan.

Menonaktifkan telemetri menggunakan opsi baris perintah

Anda dapat menonaktifkan telemetri menggunakan opsi baris perintah saat memulai DynamoDB lokal menggunakan opsi `-disableTelemetry`. Untuk informasi selengkapnya, lihat Opsi baris perintah .

Menonaktifkan telemetri untuk satu sesi

Di sistem operasi macOS dan Linux, Anda dapat menonaktifkan telemetri untuk satu sesi. Untuk menonaktifkan telemetri untuk sesi Anda saat ini, jalankan perintah berikut untuk mengatur variabel

lingkungan DDB_LOCAL_TELEMETRY ke `false`. Ulangi perintah tersebut untuk setiap sesi atau terminal baru.

```
export DDB_LOCAL_TELEMETRY=0
```

Menonaktifkan telemetri untuk profil Anda di semua sesi

Jalankan perintah berikut guna menonaktifkan telemetri untuk semua sesi saat Anda menjalankan DynamoDB lokal di sistem operasi Anda.

Untuk menonaktifkan telemetri di Linux

1. Jalankan:

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

2. Jalankan:

   ```
   source ~/.profile
   ```

Untuk menonaktifkan telemetri di macOS

1. Jalankan:

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

2. Jalankan:

   ```
   source ~/.profile
   ```

Untuk menonaktifkan telemetri di Windows

1. Jalankan:

   ```
   setx DDB_LOCAL_TELEMETRY 0
   ```

2. Jalankan:

```
refreshenv
```

Matikan telemetri menggunakan DynamoDB lokal yang disematkan pada proyek Maven

Anda dapat mematikan telemetri menggunakan DynamoDB lokal tertanam pada proyek Maven.

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

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

Jenis informasi yang dikumpulkan

- Informasi penggunaan — Telemetri umum seperti server mulai/berhenti dan panggilan API atau Operasi.
- Informasi sistem dan lingkungan — Versi Java, sistem operasi (Windows, Linux, atau macOS), lingkungan tempat DynamoDB lokal dijalankan (misalnya, Stand alone JAR, Kontainer Docker, atau sebagai Dependensi Maven), dan nilai hash atribut penggunaan.

Pelajari selengkapnya

Data telemetri yang dikumpulkan DynamoDB lokal mematuhi kebijakan privasi data. AWS Untuk informasi selengkapnya, lihat berikut ini:

- AWS ketentuan layanan
- FAQ privasi data

Langkah 1: Buat tabel di DynamoDB

Pada langkah ini, Anda membuat tabel `Music` di Amazon DynamoDB. Tabel ini memiliki detail berikut:

- Kunci partisi — `Artist`

- Kunci urutan — `SongTitle`

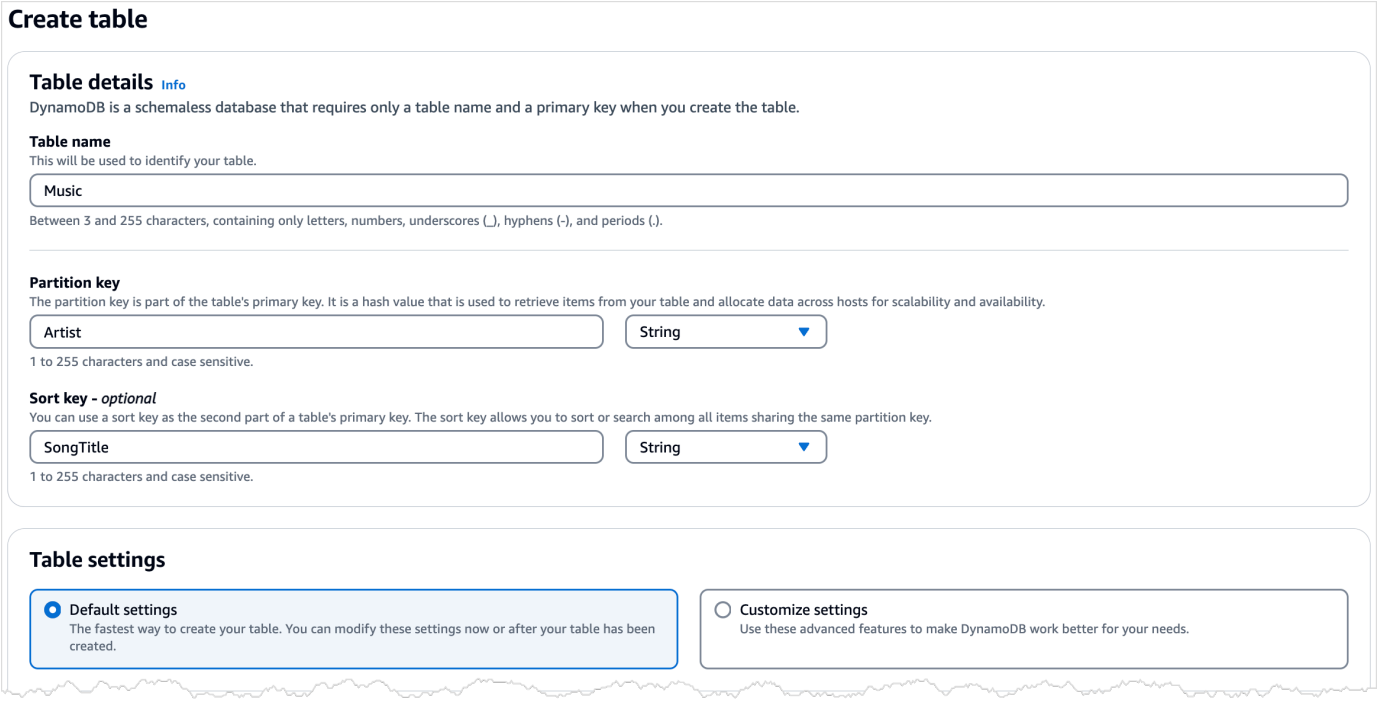
Untuk informasi selengkapnya tentang operasi tabel, lihat Bekerja dengan tabel dan data di DynamoDB.

> **Note**
>
> Sebelum memulai, pastikan Anda mengikuti langkah-langkah dalam Prasyarat.

AWS Management Console

Untuk membuat tabel `Music` baru menggunakan konsol DynamoDB:

1. Masuk ke AWS Management Console dan buka konsol DynamoDB di. https://console.aws.amazon.com/dynamodb/
2. Di panel navigasi kiri, pilih Tabel.
3. Pilih Buat tabel.
4. Masukkan rincian Tabel sebagai berikut:
   a. Untuk Nama tabel, masukkan **`Music`**.
   b. Untuk kunci Partisi, masukkan**`Artist`**.
   c. Untuk tombol Sortir, masukkan**`SongTitle`**.
5. Untuk pengaturan Tabel, pertahankan pilihan default pengaturan Default.
6. Pilih Buat tabel untuk membuat tabel.



7. Setelah tabel dalam `ACTIVE` status, kami sarankan Anda mengaktifkan Point-in-time backup untuk DynamoDB di atas meja dengan melakukan langkah-langkah berikut:

   a. Pilih nama tabel untuk membuka tabel.

   b. Pilih Backup.

   c. Pilih Edit di bagian Point-in-time pemulihan (PITR).

   d. Pada halaman Edit pengaturan point-in-time pemulihan, pilih Aktifkan point-in-time pemulihan.

   e. Pilih Simpan perubahan.

AWS CLI

AWS CLI Contoh berikut membuat `Music` tabel baru menggunakan`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
```

Menggunakan `create-table` akan menampilkan hasil sampel berikut.

```
{
    "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"
        }
    }
}
}
```

Perhatikan bahwa nilai bidang `TableStatus` diatur ke `CREATING`.

Untuk memverifikasi bahwa DynamoDB selesai membuat tabel `Music`, gunakan perintah `describe-table`.

Linux

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

Windows CMD

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

Perintah ini akan menampilkan hasil berikut. Setelah DynamoDB selesai membuat tabel, nilai bidang `TableStatus` diatur ke `ACTIVE`.

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

Setelah tabel berstatus `ACTIVE`, praktik terbaiknya adalah mengaktifkan Point-in-time backup untuk DynamoDB pada tabel dengan menjalankan perintah berikut:

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
```

Perintah ini akan menampilkan hasil berikut.

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

> **Note**
>
> Ada implikasi biaya untuk memungkinkan pencadangan berkelanjutan dengan pemulihan. point-in-time Untuk informasi selengkapnya tentang harga, lihat Harga Amazon DynamoDB.

AWS SDK

Contoh kode berikut menunjukkan cara membuat tabel DynamoDB menggunakan AWS SDK.

.NET

SDK untuk .NET

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
        /// <summary>
        /// Creates a new Amazon DynamoDB table and then waits for the new
        /// table to become active.
        /// </summary>
        /// <param name="client">An initialized Amazon DynamoDB client object.</
param>
        /// <param name="tableName">The name of the table to create.</param>
        /// <returns>A Boolean value indicating the success of the operation.</
returns>
        public static async Task<bool> CreateMovieTableAsync(AmazonDynamoDBClient
 client, string tableName)
        {
            var response = await client.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
            {
                System.Threading.Thread.Sleep(sleepDuration);

                var describeTableResponse = await
client.DescribeTableAsync(request);
                status = describeTableResponse.Table.TableStatus;

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

            return status == TableStatus.ACTIVE;
        }
```

- Untuk detail API, lihat CreateTabledi Referensi AWS SDK untuk .NET API.

Bash

AWS CLI dengan skrip Bash

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
###############################################################################
# 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
}
```

Fungsi utilitas yang digunakan dalam contoh ini.

```
###############################################################################
# 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
}
```

- Untuk detail API, lihat CreateTabledi Referensi AWS CLI Perintah.

C++

SDK untuk C++

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
//! 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);
}
```

Kode yang menunggu tabel menjadi aktif.

```
//! 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;
}
```

- Untuk detail API, lihat CreateTabledi Referensi AWS SDK untuk C++ API.

CLI

AWS CLI

Contoh 1: Untuk membuat tabel dengan tag

`create-table`Contoh berikut menggunakan atribut tertentu dan skema kunci untuk membuat tabel bernama`MusicCollection`. Tabel ini menggunakan throughput yang disediakan dan dienkripsi saat istirahat menggunakan CMK yang dimiliki default. AWS Perintah ini juga menerapkan tag ke tabel, dengan kunci dari `Owner` dan nilai`blueTeam`.

```
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
```

Output:

```
{
    "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"
    }
}
```

Untuk informasi selengkapnya, lihat Operasi Dasar untuk Tabel dalam Panduan Developer Amazon DynamoDB.

Contoh 2: Untuk membuat tabel dalam Mode On-Demand

Contoh berikut membuat tabel yang disebut `MusicCollection` menggunakan mode on-demand, bukan mode throughput yang disediakan. Ini berguna untuk tabel dengan beban kerja yang tidak terduga.

```
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
```

Output:

```
{
    "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"
        }
    }
}
```

Untuk informasi selengkapnya, lihat Operasi Dasar untuk Tabel dalam Panduan Developer Amazon DynamoDB.

Contoh 3: Untuk membuat tabel dan mengenkripsi dengan Customer Managed CMK

Contoh berikut membuat tabel bernama `MusicCollection` dan mengenkripsi menggunakan CMK yang dikelola pelanggan.

```
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
```

Output:

```
{
    "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"
        }
    }
```

```
}
```

Untuk informasi selengkapnya, lihat Operasi Dasar untuk Tabel dalam Panduan Developer Amazon DynamoDB.

Contoh 4: Untuk membuat tabel dengan Indeks Sekunder Lokal

Contoh berikut menggunakan atribut tertentu dan skema kunci untuk membuat tabel bernama `MusicCollection` dengan Indeks Sekunder Lokal bernama`AlbumTitleIndex`.

```
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\"]
                }
            }
        ]"
```

Output:

```
{
    "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"
            }
        ]
    }
}
```

Untuk informasi selengkapnya, lihat Operasi Dasar untuk Tabel dalam Panduan Developer Amazon DynamoDB.

Contoh 5: Untuk membuat tabel dengan Indeks Sekunder Global

Contoh berikut membuat tabel bernama `GameScores` dengan Global Secondary Index disebut`GameTitleIndex`. Tabel dasar memiliki kunci partisi `UserId` dan kunci urutan `GameTitle`, sehingga Anda dapat menemukan skor terbaik pengguna individu untuk game tertentu secara efisien, sedangkan GSI memiliki kunci partisi `GameTitle` dan kunci urutan `TopScore`, memungkinkan Anda untuk cepat menemukan skor tertinggi secara keseluruhan untuk game tertentu.

```
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
                }
            }
        ]"
```

Output:

```
{
    "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"
            }
        ]
    }
}
```

Untuk informasi selengkapnya, lihat Operasi Dasar untuk Tabel dalam Panduan Developer Amazon DynamoDB.

Contoh 6: Untuk membuat tabel dengan beberapa Indeks Sekunder Global sekaligus

Contoh berikut membuat tabel bernama `GameScores` dengan dua Global Secondary Indexes. Skema GSI diteruskan melalui file, bukan pada baris perintah.

```
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
```

Isi dari `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
        }
    }
]
```

Output:

```
{
    "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"
            }
        ]
    }
}
```

Untuk informasi selengkapnya, lihat Operasi Dasar untuk Tabel dalam Panduan Developer Amazon DynamoDB.

Contoh 7: Untuk membuat tabel dengan Streams diaktifkan

Contoh berikut membuat tabel yang disebut `GameScores` dengan DynamoDB Streams diaktifkan. Gambar baru dan lama dari setiap item akan ditulis ke aliran.

```
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
```

Output:

```
{
    "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"
    }
}
```

Untuk informasi selengkapnya, lihat Operasi Dasar untuk Tabel dalam Panduan Developer Amazon DynamoDB.

Contoh 8: Untuk membuat tabel dengan Keys-Only Stream diaktifkan

Contoh berikut membuat tabel yang disebut `GameScores` dengan DynamoDB Streams diaktifkan. Hanya atribut kunci dari item yang dimodifikasi yang ditulis ke aliran.

```
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
```

Output:

```
{
    "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
    }
}
```

Untuk informasi selengkapnya, lihat Mengubah pengambilan data untuk DynamoDB Streams di Panduan Pengembang Amazon DynamoDB.

Contoh 9: Untuk membuat tabel dengan kelas Standard Infrequent Access

Contoh berikut membuat tabel yang disebut `GameScores` dan menetapkan kelas tabel Standard-Infrequent Access (DynamoDB Standard-IA). Kelas tabel ini dioptimalkan untuk penyimpanan menjadi biaya dominan.

```
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
```

Output:

```
{
    "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
    }
}
```

Untuk informasi selengkapnya, lihat Kelas tabel di Panduan Pengembang Amazon DynamoDB.

Contoh 10: Membuat tabel dengan Hapus Perlindungan diaktifkan

Contoh berikut membuat tabel yang disebut `GameScores` dan memungkinkan perlindungan penghapusan.

```
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
```

Output:

```
{
    "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
    }
}
```

Untuk informasi selengkapnya, lihat Menggunakan perlindungan penghapusan di Panduan Pengembang Amazon DynamoDB.

- Untuk detail API, lihat CreateTabledi Referensi AWS CLI Perintah.

Go

SDK untuk Go V2

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
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
}
```

- Untuk detail API, lihat CreateTabledi Referensi AWS SDK untuk Go API.

Java

SDK untuk Java 2.x

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
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 "";
```

```
    }
}
```

- Untuk detail API, lihat CreateTabledi Referensi AWS SDK for Java 2.x API.

JavaScript

SDK untuk JavaScript (v3)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
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;
};
```

- Untuk informasi selengkapnya, silakan lihat Panduan Developer AWS SDK untuk JavaScript.
- Untuk detail API, lihat CreateTabledi Referensi AWS SDK untuk JavaScript API.

SDK untuk JavaScript (v2)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
// 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);
  }
});
```

- Untuk informasi selengkapnya, silakan lihat Panduan Developer AWS SDK untuk JavaScript.
- Untuk detail API, lihat CreateTabledi Referensi AWS SDK untuk JavaScript API.

Kotlin

SDK untuk Kotlin

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
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 { 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
    }
}
```

- Untuk detail API, lihat CreateTabledi AWS SDK untuk referensi API Kotlin.

PHP

SDK untuk PHP

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

Buat tabel.

```
        $tableName = "ddb_demo_table_$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],
        ]);
    }
```

- Untuk detail API, lihat CreateTabledi Referensi AWS SDK untuk PHP API.

PowerShell

Alat untuk PowerShell V4

Contoh 1: Contoh ini membuat tabel bernama Thread yang memiliki kunci utama yang terdiri dari 'ForumName' (hash tipe kunci) dan 'Subject' (rentang tipe kunci). Skema yang digunakan untuk membangun tabel dapat disalurkan ke setiap cmdlet seperti yang ditunjukkan atau ditentukan menggunakan parameter -Schema.

```
$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
```

Output:

```
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  : {}
```

Contoh 2: Contoh ini membuat tabel bernama Thread yang memiliki kunci utama yang terdiri dari 'ForumName' (hash tipe kunci) dan 'Subject' (rentang tipe kunci). Indeks sekunder lokal juga didefinisikan. Kunci indeks sekunder lokal akan diatur secara otomatis dari kunci hash utama pada tabel (ForumName). Skema yang digunakan untuk membangun tabel dapat disalurkan ke setiap cmdlet seperti yang ditunjukkan atau ditentukan menggunakan parameter -Schema.

```
$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
```

Output:

```
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}
```

Contoh 3: Contoh ini menunjukkan cara menggunakan pipeline tunggal untuk membuat tabel bernama Thread yang memiliki kunci utama yang terdiri dari 'ForumName' (hash tipe kunci) dan 'Subjek' (rentang tipe kunci) dan indeks sekunder lokal. DDBKeyAdd-Schema dan Add-DDBIndex Schema membuat TableSchema objek baru untuk Anda jika tidak disediakan dari pipeline atau parameter -Schema.

```
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
```

Output:

```
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}
```

- Untuk detail API, lihat CreateTabledi Referensi Alat AWS untuk PowerShell Cmdlet (V4).

Python

SDK untuk Python (Boto3)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

Buat tabel untuk menyimpan data film.

```
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
```

- Untuk detail API, lihat CreateTabledi AWS SDK for Python (Boto3) Referensi API.

Ruby

SDK untuk Ruby

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
# 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
```

- Untuk detail API, lihat CreateTabledi Referensi AWS SDK untuk Ruby API.

Rust

SDK untuk Rust

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
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))
        }
    }
}
```

- Untuk detail API, lihat CreateTablereferensi AWS SDK for Rust API.

SAP ABAP

SDK untuk SAP ABAP

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
    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.
```

- Untuk detail API, lihat CreateTabledi AWS SDK untuk referensi SAP ABAP API.

Swift

SDK untuk Swift

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
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
        }
    }
```

- Untuk detail API, lihat referensi CreateTable AWSSDK untuk Swift API.

Untuk contoh DynamoDB lainnya, lihat Contoh kode untuk DynamoDB menggunakan AWS SDKs.

Setelah membuat tabel baru, lanjutkan ke Langkah 2: Menulis data ke tabel DynamoDB.

Langkah 2: Menulis data ke tabel DynamoDB

Pada langkah ini, Anda akan memasukkan beberapa item ke tabel `Music` yang Anda buat di Langkah 1: Buat tabel di DynamoDB.

Untuk informasi selengkapnya tentang operasi tulis, lihat Menulis item.

AWS Management Console

Ikuti langkah-langkah ini untuk menulis data ke tabel `Music` menggunakan konsol DynamoDB.

1. Buka konsol DynamoDB di. https://console.aws.amazon.com/dynamodb/
2. Di panel navigasi kiri, pilih Tabel.
3. Pada halaman Tabel, pilih tabel Musik.
4. Pilih Jelajahi item tabel.
5. Di bagian Item yang dikembalikan, pilih Buat item.
6. Pada halaman Buat item, lakukan hal berikut untuk menambahkan item ke tabel Anda:
    a. Pilih Tambahkan atribut baru, lalu pilih Nomor.
    b. Untuk nama Atribut, masukkan**`Awards`**.
    c. Ulangi proses ini untuk membuat **`AlbumTitle`** jenis String.
    d. Masukkan nilai berikut untuk item Anda:
        i. Untuk Artist, masukkan **`No One You Know`**.
        ii. Untuk SongTitle, masukkan **`Call Me Today`**.
        iii. Untuk AlbumTitle, masukkan **`Somewhat Famous`**.
        iv. Untuk Awards, masukkan **`1`**.
7. Pilih Buat item.
8. Ulangi proses ini dan buat item lain dengan nilai berikut:
    a. Untuk Artist, masukkan **`Acme Band`**.

b. Untuk SongTitlemasuk**`Happy Day`**.

c. Untuk AlbumTitle, masukkan **`Songs About Life`**.

d. Untuk Awards, masukkan **`10`**.

9. Lakukan ini sekali lagi untuk membuat item lain dengan Artist yang sama seperti pada langkah sebelumnya, tetapi nilai untuk atribut lainnya berbeda:

a. Untuk Artist, masukkan **`Acme Band`**.

b. Untuk SongTitlemasuk**`PartiQL Rocks`**.

c. Untuk AlbumTitle, masukkan **`Another Album Title`**.

d. Untuk Awards, masukkan **`8`**.

AWS CLI

AWS CLI Contoh berikut menciptakan beberapa item baru dalam `Music` tabel. Anda dapat melakukan ini melalui DynamoDB API atau PartiQL, bahasa kueri yang kompatibel dengan SQL untuk 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'}"
```

Untuk informasi selengkapnya tentang menulis data dengan PartiQL, lihat Pernyataan sisipkan PartiQL.

Untuk informasi selengkapnya tentang jenis data yang didukung di DynamoDB, lihat Jenis data.

Untuk informasi selengkapnya tentang cara mewakili jenis data DynamoDB di JSON, lihat Nilai atribut.

AWS SDK

Contoh kode berikut menunjukkan cara menulis item ke tabel DynamoDB menggunakan SDK. AWS

.NET

SDK untuk .NET

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
        /// <summary>
        /// Adds a new item to the table.
        /// </summary>
        /// <param name="client">An initialized Amazon DynamoDB client object.</
param>
        /// <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 static async Task<bool> PutItemAsync(AmazonDynamoDBClient client,
 Movie newMovie, string tableName)
        {
            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,
            };

            var response = await client.PutItemAsync(request);
            return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
        }
```

- Untuk detail API, lihat PutItemdi Referensi AWS SDK untuk .NET API.

Bash

AWS CLI dengan skrip Bash

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

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

}
```

Fungsi utilitas yang digunakan dalam contoh ini.

```
##############################################################################
# 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
}
```

- Untuk detail API, lihat PutItemdi Referensi AWS CLI Perintah.

C++

SDK untuk C++

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
//! 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);
}
```

Kode yang menunggu tabel menjadi aktif.

```
//! 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;
}
```

- Untuk detail API, lihat PutItemdi Referensi AWS SDK untuk C++ API.

CLI

AWS CLI

Contoh 1: Untuk menambahkan item ke tabel

`put-item`Contoh berikut menambahkan item baru ke MusicCollectiontabel.

```
aws dynamodb put-item \
    --table-name MusicCollection \
    --item file://item.json \
    --return-consumed-capacity TOTAL \
    --return-item-collection-metrics SIZE
```

Isi dari `item.json`:

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

Output:

```
{
    "ConsumedCapacity": {
        "TableName": "MusicCollection",
        "CapacityUnits": 1.0
```

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

Untuk informasi selengkapnya, lihat Menulis Item di Panduan Pengembang Amazon DynamoDB.

Contoh 2: Untuk menimpa item secara kondisional dalam tabel

`put-item`Contoh berikut menimpa item yang ada dalam `MusicCollection` tabel hanya jika item yang ada memiliki `AlbumTitle` atribut dengan nilai. `Greatest Hits` Perintah mengembalikan nilai item sebelumnya.

```
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
```

Isi dari `item.json`:

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

Isi dari `names.json`:

```
{
```

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

Isi dari `values.json`:

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

Output:

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

Jika kunci sudah ada, Anda akan melihat output berikut:

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

Untuk informasi selengkapnya, lihat Menulis Item di Panduan Pengembang Amazon DynamoDB.

- Untuk detail API, lihat PutItemdi Referensi AWS CLI Perintah.

Go

SDK untuk Go V2

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
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
}
```

Tentukan struct Movie yang digunakan dalam contoh ini.

```
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"])
}
```

- Untuk detail API, lihat PutItemdi Referensi AWS SDK untuk Go API.

Java

SDK untuk Java 2.x

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

Menempatkan item ke dalam tabel menggunakan 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);
        }
    }
}
```

- Untuk detail API, lihat PutItemdi Referensi AWS SDK for Java 2.x API.

JavaScript

SDK untuk JavaScript (v3)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

Contoh ini menggunakan klien dokumen untuk menyederhanakan penggunaan item di DynamoDB. Untuk detail API, lihat 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;
};
```

- Untuk detail API, lihat PutItemdi Referensi AWS SDK untuk JavaScript API.

SDK untuk JavaScript (v2)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

Menempatkan item dalam tabel.

```
// 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);
  }
});
```

Tempatkan item dalam tabel menggunakan klien dokumen DynamoDB.

```
// 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);
  }
});
```

- Untuk informasi selengkapnya, silakan lihat Panduan Developer AWS SDK untuk JavaScript.
- Untuk detail API, lihat PutItemdi Referensi AWS SDK untuk JavaScript API.

Kotlin

SDK untuk Kotlin

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

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

- Untuk detail API, lihat PutItemdi AWS SDK untuk referensi API Kotlin.

PHP

SDK untuk PHP

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
        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);
    }
```

- Untuk detail API, lihat PutItemdi Referensi AWS SDK untuk PHP API.

PowerShell

Alat untuk PowerShell V4

Contoh 1: Membuat item baru, atau mengganti item yang sudah ada dengan item baru.

```
$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
```

- Untuk detail API, lihat PutItemdi Referensi Alat AWS untuk PowerShell Cmdlet (V4).

Python

SDK untuk Python (Boto3)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
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
```

- Untuk detail API, lihat PutItemdi AWS SDK for Python (Boto3) Referensi API.

Ruby

SDK untuk Ruby

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
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
```

- Untuk detail API, lihat PutItemdi Referensi AWS SDK untuk Ruby API.

Rust

SDK untuk Rust

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
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,
    })
}
```

- Untuk detail API, lihat PutItemreferensi AWS SDK for Rust API.

SAP ABAP

SDK untuk SAP ABAP

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
    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.
```

- Untuk detail API, lihat PutItemdi AWS SDK untuk referensi SAP ABAP API.

Swift

SDK untuk Swift

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
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
    }
```

- Untuk detail API, lihat referensi PutItem AWSSDK untuk Swift API.

Untuk contoh DynamoDB lainnya, lihat Contoh kode untuk DynamoDB menggunakan AWS SDKs.

Setelah menulis data ke tabel Anda, lanjutkan ke Langkah 3: Baca data dari tabel DynamoDB.

Langkah 3: Baca data dari tabel DynamoDB

Pada langkah ini, Anda akan membaca kembali salah satu item yang Anda buatLangkah 2: Menulis data ke tabel DynamoDB. Anda dapat menggunakan konsol DynamoDB atau AWS CLI untuk membaca item dari tabel dengan menentukan `Music` dan. `Artist` `SongTitle`

Untuk informasi selengkapnya tentang operasi baca di DynamoDB, lihat Membaca item.

AWS Management Console

Ikuti langkah-langkah ini untuk membaca data dari tabel `Music` menggunakan konsol DynamoDB.

1. Buka konsol DynamoDB di. https://console.aws.amazon.com/dynamodb/
2. Di panel navigasi kiri, pilih Tabel.
3. Pada halaman Tabel, pilih tabel Musik.
4. Pilih Jelajahi item tabel.
5. Pada bagian Item yang dikembalikan, lihat daftar item yang disimpan dalam tabel, diurutkan berdasarkan `Artist` dan`SongTitle`. Item pertama dalam daftar adalah item dengan Artis bernama Acme Band dan SongTitlePartiQL Rocks.

AWS CLI

AWS CLI Contoh berikut membaca item dari`Music`. Anda dapat melakukan ini melalui DynamoDB API atau PartiQL, bahasa kueri yang kompatibel dengan SQL untuk DynamoDB.

DynamoDB API

> **Note**
>
> Perilaku default untuk DynamoDB adalah bacaan akhir konsisten. Parameter `consistent-read` digunakan di bawah ini untuk menunjukkan bacaan sangat konsisten.

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
\"}}"
```

Menggunakan `get-item` akan menampilkan hasil sampel berikut.

```
{
    "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'"
```

Menggunakan pernyataan `Select` PartiQL akan menampilkan hasil sampel berikut.

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

Untuk informasi selengkapnya tentang membaca data dengan PartiQL, lihat Pernyataan pilih PartiQL.

AWS SDK

Contoh kode berikut menunjukkan cara membaca item dari tabel DynamoDB menggunakan SDK. AWS

.NET

SDK untuk .NET

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
        /// <summary>
```

```
        /// Gets information about an existing movie from the table.
        /// </summary>
        /// <param name="client">An initialized Amazon DynamoDB client object.</
param>
        /// <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 static async Task<Dictionary<string, AttributeValue>>
 GetItemAsync(AmazonDynamoDBClient client, Movie newMovie, string tableName)
        {
            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 client.GetItemAsync(request);
            return response.Item;
        }
```

- Untuk detail API, lihat GetItemdi Referensi AWS SDK untuk .NET API.

Bash

AWS CLI dengan skrip Bash

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
###############################################################################
# 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
}
```

Fungsi utilitas yang digunakan dalam contoh ini.

```
########################################################################################
# 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
}
```

- Untuk detail API, lihat GetItemdi Referensi AWS CLI Perintah.

C++

SDK untuk C++

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
//! 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::cerr << "Failed to get item: " << outcome.GetError().GetMessage();
    }

    return outcome.IsSuccess();
}
```

- Untuk detail API, lihat GetItemdi Referensi AWS SDK untuk C++ API.

CLI

AWS CLI

Contoh 1: Untuk membaca item dalam tabel

`get-item`Contoh berikut mengambil item dari `MusicCollection` tabel. Tabel memiliki kunci hash-and-range utama (`Artist`dan`SongTitle`), jadi Anda harus menentukan kedua atribut ini. Perintah tersebut juga meminta informasi tentang kapasitas baca yang dikonsumsi oleh operasi.

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

Isi dari `key.json`:

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

Output:

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

Untuk informasi selengkapnya, lihat Membaca Item di Panduan Pengembang Amazon DynamoDB.

Contoh 2: Untuk membaca item menggunakan pembacaan yang konsisten

Contoh berikut mengambil item dari `MusicCollection` tabel menggunakan pembacaan yang sangat konsisten.

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

Isi dari `key.json`:

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

Output:

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

Untuk informasi selengkapnya, lihat Membaca Item di Panduan Pengembang Amazon DynamoDB.

Contoh 3: Untuk mengambil atribut tertentu dari suatu item

Contoh berikut menggunakan ekspresi proyeksi untuk mengambil hanya tiga atribut dari item yang diinginkan.

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

Isi dari `names.json`:

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

Output:

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

Untuk informasi selengkapnya, lihat Membaca Item di Panduan Pengembang Amazon DynamoDB.

- Untuk detail API, lihat GetItemdi Referensi AWS CLI Perintah.

Go

SDK untuk Go V2

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
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
}
```

Tentukan struct Movie yang digunakan dalam contoh ini.

```
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"])
}
```

- Untuk detail API, lihat GetItemdi Referensi AWS SDK untuk Go API.

Java

SDK untuk Java 2.x

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

Mendapat item dari tabel dengan menggunakan 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);
        }
    }
}
```

- Untuk detail API, lihat GetItemdi Referensi AWS SDK for Java 2.x API.

JavaScript

SDK untuk JavaScript (v3)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

Contoh ini menggunakan klien dokumen untuk menyederhanakan penggunaan item di DynamoDB. Untuk detail API, lihat 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;
};
```

- Untuk detail API, lihat GetItemdi Referensi AWS SDK untuk JavaScript API.

SDK untuk JavaScript (v2)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

Dapatkan item dari tabel.

```
// 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);
  }
});
```

Dapatkan item dari tabel menggunakan klien dokumen DynamoDB.

```
// 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);
  }
});
```

- Untuk informasi selengkapnya, silakan lihat Panduan Developer AWS SDK untuk JavaScript.
- Untuk detail API, lihat GetItemdi Referensi AWS SDK untuk JavaScript API.

Kotlin

SDK untuk Kotlin

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

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

```
            println(key1.key)
            println(key1.value)
        }
    }
}
```

- Untuk detail API, lihat GetItemdi AWS SDK untuk referensi API Kotlin.

PHP

SDK untuk PHP

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
        $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,
        ]);
    }
```

- Untuk detail API, lihat GetItemdi Referensi AWS SDK untuk PHP API.

PowerShell

Alat untuk PowerShell V4

Contoh 1: Mengembalikan item DynamoDB dengan SongTitle kunci partisi dan kunci sort Artist.

```
$key = @{
  SongTitle = 'Somewhere Down The Road'
  Artist = 'No One You Know'
} | ConvertTo-DDBItem

Get-DDBItem -TableName 'Music' -Key $key | ConvertFrom-DDBItem
```

Output:

```
Name                           Value
----                           -----
Genre                          Country
SongTitle                      Somewhere Down The Road
Price                          1.94
Artist                         No One You Know
CriticRating                   9
AlbumTitle                     Somewhat Famous
```

- Untuk detail API, lihat GetItemdi Referensi Alat AWS untuk PowerShell Cmdlet (V4).

Python

SDK untuk Python (Boto3)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
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"]
```

- Untuk detail API, lihat GetItemdi AWS SDK for Python (Boto3) Referensi API.

Ruby

SDK untuk Ruby

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
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
```

- Untuk detail API, lihat GetItemdi Referensi AWS SDK untuk Ruby API.

SAP ABAP

SDK untuk SAP ABAP

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
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.
```

- Untuk detail API, lihat GetItemdi AWS SDK untuk referensi SAP ABAP API.

Swift

SDK untuk Swift

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
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
        }
    }
```

- Untuk detail API, lihat referensi GetItem AWSSDK untuk Swift API.

Untuk contoh DynamoDB lainnya, lihat Contoh kode untuk DynamoDB menggunakan AWS SDKs.

Untuk memperbarui data di tabel Anda, lanjutkan ke Langkah 4: Perbarui data dalam tabel DynamoDB.

Langkah 4: Perbarui data dalam tabel DynamoDB

Pada langkah ini, Anda akan memperbarui item yang Anda buat di Langkah 2: Menulis data ke tabel DynamoDB. Anda dapat menggunakan konsol DynamoDB atau AWS CLI untuk memperbarui `AlbumTitle` item dalam `Music` tabel dengan `Artist` menentukan`SongTitle`,, dan diperbarui. `AlbumTitle`

Untuk informasi selengkapnya tentang operasi tulis, lihat Menulis item.

AWS Management Console

Anda dapat menggunakan konsol DynamoDB untuk memperbarui data dalam tabel `Music`.

1. Buka konsol DynamoDB di. https://console.aws.amazon.com/dynamodb/
2. Di panel navigasi kiri, pilih Tabel.
3. Pilih tabel Musik dari daftar tabel.
4. Pilih Jelajahi item tabel.
5. Di Item yang dikembalikan, untuk baris item dengan Acme Band Artist dan Happy Day SongTitle, lakukan hal berikut:

    a. Tempatkan kursor Anda pada Songs About Life yang AlbumTitlebernama.

    b. Pilih ikon Edit.

    c. Di jendela popup Edit String, masukkan. **`Songs of Twilight`**

d. Pilih Simpan.

> **Tip**
>
> Atau, untuk memperbarui item, lakukan hal berikut di bagian Item yang dikembalikan:
>
> 1. Pilih baris item dengan Artis bernama Acme Band dan SongTitlediberi nama Happy Day.
> 2. Dari daftar dropdown Tindakan, pilih Edit item.
> 3. Untuk masuk AlbumTitle, masukkan**`Songs of Twilight`**.
> 4. Pilih Save and close (Simpan dan pilih).

AWS CLI

AWS CLI Contoh berikut memperbarui item dalam `Music` tabel. Anda dapat melakukan ini melalui DynamoDB API atau PartiQL, bahasa kueri yang kompatibel dengan SQL untuk 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
```

Menggunakan `update-item` akan menampilkan hasil sampel berikut karena `return-values ALL_NEW` ditentukan.

```
{
    "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
 *"
```

Menggunakan pernyataan `Update` akan menampilkan hasil sampel berikut karena `RETURNING ALL NEW *` ditentukan.

```
{
    "Items": [
        {
            "AlbumTitle": {
```

```
                "S": "Updated Album Title"
            },
            "Awards": {
                "S": "10"
            },
            "Artist": {
                "S": "Acme Band"
            },
            "SongTitle": {
                "S": "Happy Day"
            }
        }
    ]
}
```

Untuk informasi selengkapnya tentang memperbarui data dengan PartiQL, lihat Pernyataan perbarui PartiQL.

AWS SDK

Contoh kode berikut menunjukkan cara memperbarui item dalam tabel DynamoDB menggunakan SDK. AWS

.NET

SDK untuk .NET

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
        /// <summary>
        /// Updates an existing item in the movies table.
        /// </summary>
        /// <param name="client">An initialized Amazon DynamoDB client object.</
param>
        /// <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 static async Task<bool> UpdateItemAsync(
            AmazonDynamoDBClient client,
            Movie newMovie,
            MovieInfo newInfo,
            string tableName)
        {
            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,
            };

            var response = await client.UpdateItemAsync(request);

            return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
        }
```

- Untuk detail API, lihat UpdateItemdi Referensi AWS SDK untuk .NET API.

Bash

AWS CLI dengan skrip Bash

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

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

}
```

Fungsi utilitas yang digunakan dalam contoh ini.

```
###############################################################################
# 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
}
```

- Untuk detail API, lihat UpdateItemdi Referensi AWS CLI Perintah.

C++

SDK untuk C++

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
//! 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);
}
```

Kode yang menunggu tabel menjadi aktif.

```
//! 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;
}
```

- Untuk detail API, lihat UpdateItemdi Referensi AWS SDK untuk C++ API.

CLI

AWS CLI

Contoh 1: Untuk memperbarui item dalam tabel

Contoh `update-item` berikut memperbarui item dalam tabel `MusicCollection`. Ia menambahkan atribut baru (`Year`) dan memodifikasi `AlbumTitle` atribut. Semua atribut dalam item, seperti yang muncul setelah pembaruan, dikembalikan sebagai respons.

```
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
```

Isi dari `key.json`:

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

Isi dari `expression-attribute-names.json`:

```
{
    "#Y":"Year", "#AT":"AlbumTitle"
}
```

Isi dari `expression-attribute-values.json`:

```
{
    ":y":{"N": "2015"},
    ":t":{"S": "Louder Than Ever"}
}
```

Output:

```
{
    "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
        ]
    }
}
```

Untuk informasi selengkapnya, lihat Menulis Item di Panduan Pengembang Amazon DynamoDB.

Contoh 2: Untuk memperbarui item secara kondisional

Contoh berikut memperbarui item dalam `MusicCollection` tabel, tetapi hanya jika item yang ada belum memiliki `Year` atribut.

```
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)"
```

Isi dari `key.json`:

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

Isi dari `expression-attribute-names.json`:

```
{
    "#Y":"Year",
    "#AT":"AlbumTitle"
}
```

Isi dari `expression-attribute-values.json`:

```
{
    ":y":{"N": "2015"},
    ":t":{"S": "Louder Than Ever"}
}
```

Jika item sudah memiliki `Year` atribut, DynamoDB mengembalikan output berikut.

```
An error occurred (ConditionalCheckFailedException) when calling the UpdateItem
 operation: The conditional request failed
```

Untuk informasi selengkapnya, lihat Menulis Item di Panduan Pengembang Amazon DynamoDB.

- Untuk detail API, lihat UpdateItemdi Referensi AWS CLI Perintah.

Go

SDK untuk Go V2

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
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
}
```

Tentukan struct Movie yang digunakan dalam contoh ini.

```
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"])
}
```

- Untuk detail API, lihat UpdateItemdi Referensi AWS SDK untuk Go API.

Java

SDK untuk Java 2.x

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

Memperbarui item dalam tabel menggunakan 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!");
    }
}
```

- Untuk detail API, lihat UpdateItemdi Referensi AWS SDK for Java 2.x API.

JavaScript

SDK untuk JavaScript (v3)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

Contoh ini menggunakan klien dokumen untuk menyederhanakan penggunaan item di DynamoDB. Untuk detail API, lihat 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;
};
```

- Untuk detail API, lihat UpdateItemdi Referensi AWS SDK untuk JavaScript API.

Kotlin

SDK untuk Kotlin

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
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 { region = "us-east-1" }.use { ddb ->
        ddb.updateItem(request)
        println("Item in $tableNameVal was updated")
    }
}
```

- Untuk detail API, lihat UpdateItemdi AWS SDK untuk referensi API Kotlin.

PHP

SDK untuk PHP

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
        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
                ]
            ],
        ]);
    }
```

- Untuk detail API, lihat UpdateItemdi Referensi AWS SDK untuk PHP API.

PowerShell

Alat untuk PowerShell V4

Contoh 1: Menetapkan atribut genre ke 'Rap' pada item DynamoDB dengan SongTitle kunci partisi dan Artis kunci sortir.

```
$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
```

Output:

```
Name                           Value
----                           -----
Genre                          Rap
```

- Untuk detail API, lihat UpdateItemdi Referensi Alat AWS untuk PowerShell Cmdlet (V4).

Python

SDK untuk Python (Boto3)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

Perbarui item menggunakan ekspresi pembaruan.

```
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"]
```

Perbarui item menggunakan ekspresi pembaruan yang menyertakan operasi aritmatika.

```
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"]
```

Perbarui item hanya jika memenuhi persyaratan tertentu.

```
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"]
```

- Untuk detail API, lihat UpdateItemdi AWS SDK for Python (Boto3) Referensi API.

Ruby

SDK untuk Ruby

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
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
```

- Untuk detail API, lihat UpdateItemdi Referensi AWS SDK untuk Ruby API.

SAP ABAP

SDK untuk SAP ABAP

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
    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.
```

- Untuk detail API, lihat UpdateItemdi AWS SDK untuk referensi SAP ABAP API.

Swift

SDK untuk Swift

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
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
        }
    }
```

- Untuk detail API, lihat referensi UpdateItem AWSSDK untuk Swift API.

Untuk contoh DynamoDB lainnya, lihat Contoh kode untuk DynamoDB menggunakan AWS SDKs.

Untuk mengkueri data dalam tabel `Music`, lanjutkan ke Langkah 5: Kueri data dalam tabel DynamoDB.

Langkah 5: Kueri data dalam tabel DynamoDB

Pada langkah ini, Anda akan mengkueri data yang ditulis ke tabel `Music` di the section called "Langkah 2: Tulis data" dengan menentukan `Artist`. Ini akan menampilkan semua lagu yang terkait dengan kunci partisi: `Artist`.

Untuk informasi selengkapnya tentang operasi kueri, lihat Menanyakan tabel di DynamoDB.

AWS Management Console

Ikuti langkah-langkah ini untuk menggunakan konsol DynamoDB guna mengkueri data di tabel `Music`.

1. Buka konsol DynamoDB di. https://console.aws.amazon.com/dynamodb/
2. Di panel navigasi kiri, pilih Tabel.
3. Pilih tabel Musik dari daftar tabel.
4. Pilih Jelajahi item tabel.
5. Di item Pindai atau kueri, pastikan bahwa Kueri dipilih.
6. Untuk kunci partisi, masukkan **`Acme Band`**, lalu pilih Jalankan.

AWS CLI

AWS CLI Contoh berikut query item dalam `Music` tabel. Anda dapat melakukan ini melalui DynamoDB API atau PartiQL, bahasa kueri yang kompatibel dengan SQL untuk DynamoDB.

DynamoDB API

Anda akan mengkueri item melalui DynamoDB API menggunakan `query` dan menyediakan kunci partisi.

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\"}}"
```

Menggunakan `query` akan menampilkan semua lagu yang terkait dengan `Artist` ini.

```
{
    "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

Anda akan mengkueri item melalui PartiQL menggunakan pernyataan `Select` dan menyediakan kunci partisi.

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'"
```

Menggunakan pernyataan `Select` dengan cara ini akan menampilkan semua lagu yang terkait dengan `Artist` ini.

```
{
    "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"
            }
        }
    ]
}
```

Untuk informasi selengkapnya tentang mengkueri data dengan PartiQL, lihat Pernyataan pilih PartiQL.

AWS SDK

Contoh kode berikut menunjukkan cara mengkueri tabel DynamoDB menggunakan AWS SDK.

.NET

SDK untuk .NET

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
        /// <summary>
        /// Queries the table for movies released in a particular year and
        /// then displays the information for the movies returned.
        /// </summary>
        /// <param name="client">The initialized DynamoDB client object.</param>
        /// <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 static async Task<int> QueryMoviesAsync(AmazonDynamoDBClient
client, string tableName, int year)
        {
            var movieTable = Table.LoadTable(client, tableName);
            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;

            Search 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;
        }
```

- Untuk detail API, lihat Kueri di Referensi API AWS SDK untuk .NET .

Bash

AWS CLI dengan skrip Bash

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
###############################################################################
# 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
}
```

Fungsi utilitas yang digunakan dalam contoh ini.

```
###############################################################################
# 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
}
```

- Untuk detail API, lihat Kueri di Referensi Perintah AWS CLI .

C++

SDK untuk C++

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
//! 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;
}
```

- Untuk detail API, lihat Kueri di Referensi API AWS SDK untuk C++ .

CLI

AWS CLI

Contoh 1: Untuk menanyakan tabel

`query`Contoh berikut query item dalam `MusicCollection` tabel. Tabel memiliki kunci hash-and-range utama (`Artist`dan`SongTitle`), tetapi kueri ini hanya menentukan nilai kunci hash. Ini mengembalikan judul lagu oleh artis bernama “No One You Know”.

```
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
```

Isi dari `expression-attributes.json`:

```
{
    ":v1": {"S": "No One You Know"}
}
```

Output:

```
{
    "Items": [
        {
            "SongTitle": {
                "S": "Call Me Today"
            },
            "SongTitle": {
                "S": "Scared of My Shadow"
            }
        }
    ],
    "Count": 2,
    "ScannedCount": 2,
    "ConsumedCapacity": {
        "TableName": "MusicCollection",
        "CapacityUnits": 0.5
    }
}
```

Untuk informasi selengkapnya, lihat Bekerja dengan Kueri di DynamoDB di Panduan Pengembang Amazon DynamoDB.

Contoh 2: Untuk menanyakan tabel menggunakan pembacaan yang sangat konsisten dan melintasi indeks dalam urutan menurun

Contoh berikut melakukan kueri yang sama dengan contoh pertama, tetapi mengembalikan hasil dalam urutan terbalik dan menggunakan pembacaan yang sangat konsisten.

```
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
```

Isi dari `expression-attributes.json`:

```
{
    ":v1": {"S": "No One You Know"}
}
```

Output:

```
{
    "Items": [
        {
            "SongTitle": {
                "S": "Scared of My Shadow"
            }
        },
        {
            "SongTitle": {
                "S": "Call Me Today"
            }
        }
    ],
    "Count": 2,
    "ScannedCount": 2,
    "ConsumedCapacity": {
        "TableName": "MusicCollection",
        "CapacityUnits": 1.0
    }
}
```

Untuk informasi selengkapnya, lihat Bekerja dengan Kueri di DynamoDB di Panduan Pengembang Amazon DynamoDB.

Contoh 3: Untuk menyaring hasil tertentu

Contoh berikut query `MusicCollection` tetapi mengecualikan hasil dengan nilai-nilai tertentu dalam atribut. `AlbumTitle` Perhatikan bahwa ini tidak mempengaruhi `ScannedCount` atau`ConsumedCapacity`, karena filter diterapkan setelah item telah dibaca.

```
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
```

Isi dari `values.json`:

```
{
    ":v1": {"S": "No One You Know"},
    ":v2": {"S": "Blue Sky Blues"},
    ":v3": {"S": "Greatest Hits"}
}
```

Isi dari `names.json`:

```
{
    "#n1": "Artist",
    "#n2": "AlbumTitle"
}
```

Output:

```
{
    "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
    }
}
```

Untuk informasi selengkapnya, lihat Bekerja dengan Kueri di DynamoDB di Panduan Pengembang Amazon DynamoDB.

Contoh 4: Untuk mengambil hanya jumlah item

Contoh berikut mengambil hitungan item yang cocok dengan query, tetapi tidak mengambil salah satu item itu sendiri.

```
aws dynamodb query \
    --table-name MusicCollection \
    --select COUNT \
    --key-condition-expression "Artist = :v1" \
    --expression-attribute-values file://expression-attributes.json
```

Isi dari `expression-attributes.json`:

```
{
    ":v1": {"S": "No One You Know"}
}
```

Output:

```
{
    "Count": 2,
    "ScannedCount": 2,
    "ConsumedCapacity": null
}
```

Untuk informasi selengkapnya, lihat Bekerja dengan Kueri di DynamoDB di Panduan Pengembang Amazon DynamoDB.

Contoh 5: Untuk menanyakan indeks

Contoh berikut query indeks `AlbumTitleIndex` sekunder lokal. Query mengembalikan semua atribut dari tabel dasar yang telah diproyeksikan ke indeks sekunder lokal. Perhatikan bahwa saat menanyakan indeks sekunder lokal atau indeks sekunder global, Anda juga harus memberikan nama tabel dasar menggunakan `table-name` parameter.

```
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
```

Isi dari `expression-attributes.json`:

```
{
    ":v1": {"S": "No One You Know"}
}
```

Output:

```
{
    "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
            }
        }
    }
}
```

Untuk informasi selengkapnya, lihat Bekerja dengan Kueri di DynamoDB di Panduan Pengembang Amazon DynamoDB.

- Untuk detail API, lihat Kueri di Referensi Perintah AWS CLI .

Go

SDK untuk Go V2

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
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
}
```

Tentukan struct Movie yang digunakan dalam contoh ini.

```
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"])
}
```

- Untuk detail API, lihat Kueri di Referensi API AWS SDK untuk Go .

Java

SDK untuk Java 2.x

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

Kueri tabel dengan menggunakan 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;
```

```
    }
}
```

Melakukan tabel menggunakan `DynamoDbClient` dan indeks sekunder.

```
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);
        }
    }
}
```

- Untuk detail API, lihat Kueri di Referensi API AWS SDK for Java 2.x .

JavaScript

SDK untuk JavaScript (v3)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

Contoh ini menggunakan klien dokumen untuk menyederhanakan penggunaan item di DynamoDB. Untuk detail API, lihat 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;
};
```

- Untuk informasi selengkapnya, silakan lihat Panduan Developer AWS SDK untuk JavaScript.
- Untuk detail API, lihat Kueri di Referensi API AWS SDK untuk JavaScript .

SDK untuk JavaScript (v2)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
// 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);
  }
});
```

- Untuk informasi selengkapnya, silakan lihat Panduan Developer AWS SDK untuk JavaScript.
- Untuk detail API, lihat Kueri di Referensi API AWS SDK untuk JavaScript .

Kotlin

SDK untuk Kotlin

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
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 { region = "us-east-1" }.use { ddb ->
        val response = ddb.query(request)
        return response.count
    }
}
```

- Untuk detail API, lihat Kueri di Referensi API AWS SDK untuk Kotlin.

PHP

SDK untuk PHP

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
        $birthKey = [
            'Key' => [
                'year' => [
                    'N' => "$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);
    }
```

- Untuk detail API, lihat Kueri di Referensi API AWS SDK untuk PHP .

PowerShell

Alat untuk PowerShell V4

Contoh 1: Memanggil query yang mengembalikan item DynamoDB dengan yang ditentukan dan Artist. SongTitle

```
$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
```

Output:

```
Name                           Value
----                           -----
Genre                          Country
Artist                         No One You Know
Price                          1.94
CriticRating                   9
SongTitle                      Somewhere Down The Road
AlbumTitle                     Somewhat Famous
```

- Untuk detail API, lihat Kueri di Referensi Alat AWS untuk PowerShell Cmdlet (V4).

Python

SDK untuk Python (Boto3)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

Kueri item menggunakan ekspresi kondisi kunci.

```
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"]
```

Kueri dan proyeksikan item untuk mengembalikan subset data.

```
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"]
```

- Untuk detail API, lihat Kueri di Referensi API AWS SDK untuk Python (Boto3).

Ruby

SDK untuk Ruby

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
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
```

- Untuk detail API, lihat Kueri di Referensi API AWS SDK untuk Ruby .

Rust

SDK untuk Rust

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

Temukan film yang dibuat pada tahun tertentu.

```
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![])
    }
```

```
}
```

- Untuk detail API, lihat Kueri di referensi API AWS SDK untuk Rust.

SAP ABAP

SDK untuk SAP ABAP

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
    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.
```

- Untuk detail API, lihat Kueri di referensi API AWS SDK untuk SAP ABAP.

Swift

SDK untuk Swift

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
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
        }
    }
```

- Untuk detail API, lihat Kueri di Referensi API AWS SDK untuk Swift.

Untuk contoh DynamoDB lainnya, lihat Contoh kode untuk DynamoDB menggunakan AWS SDKs.

Untuk membuat indeks sekunder global untuk tabel Anda, lanjutkan ke Langkah 6: (Opsional) Hapus tabel DynamoDB Anda untuk membersihkan sumber daya.

Langkah 6: (Opsional) Hapus tabel DynamoDB Anda untuk membersihkan sumber daya

Jika tidak lagi memerlukan tabel Amazon DynamoDB yang dibuat untuk tutorial, Anda dapat menghapusnya. Langkah ini membantu memastikan bahwa Anda tidak akan dikenakan biaya untuk

sumber daya yang tidak Anda gunakan. Anda dapat menggunakan konsol DynamoDB atau AWS CLI untuk menghapus tabel `Music` yang Anda buat. Langkah 1: Buat tabel di DynamoDB

Untuk informasi selengkapnya tentang operasi tabel di DynamoDB, lihat Bekerja dengan tabel dan data di DynamoDB.

AWS Management Console

Untuk menghapus tabel `Music` menggunakan konsol:

1. Buka konsol DynamoDB di. https://console.aws.amazon.com/dynamodb/
2. Di panel navigasi kiri, pilih Tabel.
3. Pilih kotak centang di samping tabel Musik dalam daftar tabel.
4. Pilih Hapus.

AWS CLI

AWS CLI Contoh berikut menghapus `Music` tabel menggunakan`delete-table`.

```
aws dynamodb delete-table --table-name Music
```

AWS SDK

Contoh kode berikut menunjukkan cara menghapus tabel DynamoDB menggunakan SDK. AWS

.NET

SDK untuk .NET

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
        public static async Task<bool> DeleteTableAsync(AmazonDynamoDBClient
client, string tableName)
        {
            var request = new DeleteTableRequest
            {
```

```
                TableName = tableName,
            };

            var response = await client.DeleteTableAsync(request);
            if (response.HttpStatusCode == System.Net.HttpStatusCode.OK)
            {
                Console.WriteLine($"Table {response.TableDescription.TableName}
 successfully deleted.");
                return true;
            }
            else
            {
                Console.WriteLine("Could not delete table.");
                return false;
            }
        }
```

- Untuk detail API, lihat DeleteTabledi Referensi AWS SDK untuk .NET API.

Bash

AWS CLI dengan skrip Bash

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
###############################################################################
# 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
}
```

Fungsi utilitas yang digunakan dalam contoh ini.

```
###############################################################################
# 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
}
```

- Untuk detail API, lihat DeleteTabledi Referensi AWS CLI Perintah.

C++

SDK untuk C++

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
//! 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();
}
```

- Untuk detail API, lihat DeleteTabledi Referensi AWS SDK untuk C++ API.

CLI

AWS CLI

Untuk menghapus tabel

`delete-table`Contoh berikut menghapus `MusicCollection` tabel.

```
aws dynamodb delete-table \
```

```
    --table-name MusicCollection
```

Output:

```
{
    "TableDescription": {
        "TableStatus": "DELETING",
        "TableSizeBytes": 0,
        "ItemCount": 0,
        "TableName": "MusicCollection",
        "ProvisionedThroughput": {
            "NumberOfDecreasesToday": 0,
            "WriteCapacityUnits": 5,
            "ReadCapacityUnits": 5
        }
    }
}
```

Untuk informasi selengkapnya, lihat Menghapus Tabel di Panduan Pengembang Amazon DynamoDB.

- Untuk detail API, lihat DeleteTabledi Referensi AWS CLI Perintah.

Go

SDK untuk Go V2

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
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
}
```

- Untuk detail API, lihat DeleteTabledi Referensi AWS SDK untuk Go API.

Java

SDK untuk Java 2.x

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
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!");
    }
}
```

- Untuk detail API, lihat DeleteTabledi Referensi AWS SDK for Java 2.x API.

JavaScript

SDK untuk JavaScript (v3)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
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;
};
```

- Untuk detail API, lihat DeleteTabledi Referensi AWS SDK untuk JavaScript API.

SDK untuk JavaScript (v2)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
// 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);
  }
});
```

- Untuk informasi selengkapnya, silakan lihat Panduan Developer AWS SDK untuk JavaScript.
- Untuk detail API, lihat DeleteTabledi Referensi AWS SDK untuk JavaScript API.

Kotlin

SDK untuk Kotlin

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
suspend fun deleteDynamoDBTable(tableNameVal: String) {
    val request =
        DeleteTableRequest {
            tableName = tableNameVal
        }

    DynamoDbClient { region = "us-east-1" }.use { ddb ->
        ddb.deleteTable(request)
        println("$tableNameVal was deleted")
    }
}
```

- Untuk detail API, lihat DeleteTabledi AWS SDK untuk referensi API Kotlin.

PHP

SDK untuk PHP

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
    public function deleteTable(string $TableName)
    {
        $this->customWaiter(function () use ($TableName) {
            return $this->dynamoDbClient->deleteTable([
                'TableName' => $TableName,
```

```
            ]);
        });
    }
```

- Untuk detail API, lihat DeleteTabledi Referensi AWS SDK untuk PHP API.

PowerShell

Alat untuk PowerShell V4

Contoh 1: Menghapus tabel yang ditentukan. Anda diminta untuk konfirmasi sebelum operasi berlangsung.

```
Remove-DDBTable -TableName "myTable"
```

Contoh 2: Menghapus tabel yang ditentukan. Anda tidak diminta untuk konfirmasi sebelum operasi berlangsung.

```
Remove-DDBTable -TableName "myTable" -Force
```

- Untuk detail API, lihat DeleteTabledi Referensi Alat AWS untuk PowerShell Cmdlet (V4).

Python

SDK untuk Python (Boto3)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
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
```

- Untuk detail API, lihat DeleteTabledi AWS SDK for Python (Boto3) Referensi API.

Ruby

SDK untuk Ruby

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
# 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
```

- Untuk detail API, lihat DeleteTabledi Referensi AWS SDK untuk Ruby API.

Rust

SDK untuk Rust

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
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())),
    }
}
```

- Untuk detail API, lihat DeleteTablereferensi AWS SDK for Rust API.

SAP ABAP

SDK untuk SAP ABAP

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
    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.
```

- Untuk detail API, lihat DeleteTabledi AWS SDK untuk referensi SAP ABAP API.

Swift

SDK untuk Swift

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
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
        }
```

```
    }
```

- Untuk detail API, lihat referensi DeleteTable AWSSDK untuk Swift API.

Untuk contoh DynamoDB lainnya, lihat Contoh kode untuk DynamoDB menggunakan AWS SDKs.

Lanjutkan belajar tentang DynamoDB

Untuk informasi selengkapnya tentang cara menggunakan Amazon DynamoDB, lihat topik berikut:

- Bekerja dengan tabel dan data di DynamoDB
- Bekerja dengan item dan atribut di DynamoDB
- Menanyakan tabel di DynamoDB
- Menggunakan Indeks Sekunder Global di DynamoDB
- Bekerja dengan transactions
- Akselerasi dalam memori dengan DynamoDB Accelerator (DAX)
- Pemrograman dengan DynamoDB dan AWS SDKs

Amazon DynamoDB: Cara kerjanya

Bagian berikut memberikan gambaran umum tentang komponen layanan Amazon DynamoDB dan bagaimana mereka berinteraksi.

Topik

- Kisi-kisi untuk DynamoDB
- Komponen inti dari Amazon DynamoDB
- DynamoDB API
- Jenis data dan aturan penamaan yang didukung di Amazon DynamoDB
- Kelas tabel DynamoDB
- Partisi dan distribusi data di DynamoDB
- Pelajari cara beralih dari SQL ke NoSQL
- Sumber daya dan alat pembelajaran Amazon DynamoDB

Kisi-kisi untuk DynamoDB

Lembar contekan ini memberikan referensi cepat untuk bekerja dengan Amazon DynamoDB dan berbagai nya. AWS SDKs

Pengaturan awal

1. Mendaftar untuk AWS.
2. Dapatkan kunci akses AWS untuk mengakses DynamoDB secara terprogram.
3. Konfigurasikan kredenal DynamoDB Anda.

Lihat juga:

- Menyiapkan DynamoDB (layanan web)
- Mulai menggunakan DynamoDB
- Gambaran umum dasar komponen inti

SDK atau CLI

Pilih SDK pilihan Anda, atau atur AWS CLI.

> **Note**
>
> Saat Anda menggunakan AWS CLI on Windows, garis miring terbalik (\) yang tidak ada di dalam kutipan diperlakukan sebagai carriage return. Selain itu, Anda harus menghilangkan tanda kutip dan kurung kurawal apa pun di dalam tanda kutip lainnya. Sebagai contoh, lihat tab Windows di “Buat tabel” di bagian berikutnya.

Lihat juga:

- AWS CLI dengan DynamoDB
- Mulai menggunakan DynamoDB - langkah 2

Tindakan dasar

Bagian ini menyediakan kode untuk tugas DynamoDB dasar. Untuk informasi selengkapnya tentang tugas ini, lihat Memulai DynamoDB dan file. AWS SDKs

Membuat tabel

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
```

Tulis item ke tabel

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

Baca item dari tabel

```
aws dynamodb get-item \ --table-name Music \ --item file://item.json
```

Hapus item dari tabel

```
aws dynamodb delete-item --table-name Music --key file://key.json
```

Mengkueri Tabel

```
aws dynamodb query --table-name Music
--key-condition-expression "ArtistName=:Artist and SongName=:Songtitle"
```

Menghapus tabel

```
aws dynamodb delete-table --table-name Music
```

Daftar nama tabel

```
aws dynamodb list-tables
```

Peraturan penamaan

- Semua nama harus dikodekan menggunakan UTF-8 dan peka huruf besar-kecil.

- Nama tabel dan nama indeks panjangnya harus antara 3 hingga 255 karakter, dan hanya dapat berisi karakter berikut:
  - `a-z`
  - `A-Z`
  - `0-9`
  - _(garis bawah)
  - `-`(dasbor)
  - `.`(titik)
- Nama atribut harus memiliki panjang minimal satu karakter dan berukuran kurang dari 64 KB.

Untuk informasi selengkapnya, lihat Aturan penamaan.

Dasar-dasar kuota layanan

Membaca dan menulis unit

- Unit kapasitas baca (RCU) – Satu pembacaan sangat konsisten per detik, atau dua pembacaan akhirnya konsisten per detik, untuk item berukuran hingga 4 KB.
- Unit kapasitas tulis (WCU) – Satu tulis per detik, untuk item berukuran hingga 1 KB.

Batasan tabel

- Ukuran tabel — Tidak ada batas praktis pada ukuran tabel. Tabel tidak dibatasi dalam jumlah item atau jumlah byte.
- Jumlah tabel — Untuk AWS akun apa pun, ada kuota awal 2.500 tabel per AWS Wilayah.
- Batas ukuran halaman untuk kueri dan pemindaian – Ada batas 1 MB per halaman, per kueri atau pemindaian. Jika parameter kueri atau operasi pemindaian Anda pada tabel menghasilkan lebih dari 1 MB data, DynamoDB mengembalikan item awal yang cocok. Ini juga mengembalikan properti `LastEvaluatedKey` yang dapat Anda gunakan dalam permintaan baru untuk membaca halaman berikutnya.

Indeks

- Indeks sekunder lokal (LSIs) - Anda dapat menentukan maksimum lima indeks sekunder lokal. LSIs terutama berguna ketika indeks harus memiliki konsistensi yang kuat dengan tabel dasar.

- Indeks sekunder global (GSIs) — Ada kuota default 20 indeks sekunder global per tabel.
- Atribut indeks sekunder yang diproyeksikan per tabel – Anda dapat memproyeksikan total hingga 100 atribut ke semua indeks lokal tabel dan sekunder global. Ini hanya berlaku untuk atribut proyeksi yang ditentukan pengguna.

Kunci partisi

- Panjang minimum nilai kunci partisi adalah 1 byte. Panjang maksimum adalah 2048 byte.
- Tidak ada batasan praktis untuk jumlah nilai kunci partisi yang berbeda, untuk tabel atau indeks sekunder.
- Panjang minimum nilai kunci urutan adalah 1 byte. Panjang maksimum adalah 1024 byte.
- Secara umum, tidak ada batas praktis pada jumlah nilai kunci urutan berbeda per nilai kunci partisi. Pengecualian ini adalah untuk tabel dengan indeks sekunder.

Untuk informasi selengkapnya tentang indeks sekunder, desain kunci partisi, dan desain kunci urutan, lihat Praktik terbaik.

Batas untuk jenis data yang umum digunakan

- String – Panjang string dibatasi oleh ukuran item maksimum 400 KB. String adalah Unicode dengan pengodean biner UTF-8.
- Angka – Angka dapat memiliki hingga 38 digit presisi, dan dapat positif, negatif, atau nol.
- Binary – Panjang biner dibatasi oleh ukuran item maksimum 400 KB. Aplikasi yang bekerja dengan atribut binari harus mengodekan data dalam pengodean base64 sebelum mengirimnya ke DynamoDB.

Untuk daftar jenis daya yang didukung, lihat Jenis data. Untuk informasi selengkapnya, lihat Kuota layanan.

Item, atribut, dan parameter ekspresi

Ukuran item maksimum di DynamoDB adalah 400 KB, yang mencakup panjang biner nama atribut (UTF-8 panjang) dan panjang biner nilai atribut (UTF-8 panjang). Nama atribut dihitung terhadap batas ukuran.

Tidak ada batas pada jumlah nilai dalam daftar, peta, atau set, selama item yang berisi nilai-nilai cocok dalam batas ukuran item 400-KB.

Untuk parameter ekspresi, panjang maksimum string ekspresi adalah 4 KB.

Untuk informasi selengkapnya tentang ukuran item, atribut, dan parameter ekspresi, lihat Kuota layanan.

Informasi lain

- Keamanan
- Pencatatan dan pemantauan
- Bekerja dengan stream
- Cadangan dan pemulihan Point-in-time
- Integrasi dengan layanan lain AWS
- Referensi API
- Pusat Arsitektur: Praktik Terbaik Basis Data
- Tutorial video
- Forum DynamoDB

Komponen inti dari Amazon DynamoDB

Dalam DynamoDB, tabel, item, dan atribut adalah komponen inti yang Anda kerjakan. Sebuah tabel adalah koleksi item, dan setiap item adalah koleksi atribut. DynamoDB menggunakan kunci utama untuk mengidentifikasi secara unik setiap item dalam tabel. Anda dapat menggunakan DynamoDB Streams untuk menangkap peristiwa modifikasi data dalam tabel DynamoDB.

Ada batas dalam DynamoDB. Untuk informasi selengkapnya, lihat Kuota di Amazon DynamoDB.

Video berikut ini akan memberi Anda tampilan pengantar tabel, item, dan atribut.

Tabel, item, dan atribut

Tabel, item, dan atribut

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"}

Berikut ini adalah komponen DynamoDB dasar:

- Tabel – Mirip dengan sistem basis data lainnya, DynamoDB menyimpan data dalam tabel. Sebuah tabel adalah koleksi data. Misalnya, lihat tabel contoh yang disebut Orang yang dapat Anda gunakan untuk menyimpan informasi kontak pribadi tentang teman, keluarga, atau orang lain yang menarik. Anda juga dapat memiliki tabel Mobil untuk menyimpan informasi tentang kendaraan yang dikendarai orang.
- Item — Setiap tabel berisi nol item atau lebih. Sebuah item adalah sekelompok atribut yang dapat diidentifikasi secara unik di antara semua item lainnya. Dalam tabel Orang, setiap item mewakili satu orang. Untuk tabel Mobil, setiap item mewakili satu kendaraan. Item dalam DynamoDB serupa dalam banyak hal dengan baris, catatan, atau tupel dalam sistem basis data lain. Dalam DynamoDB, tidak ada batasan jumlah item yang dapat Anda simpan dalam tabel.
- Atribut — Setiap item terdiri dari satu atau beberapa atribut. Sebuah atribut adalah elemen data fundamental, sesuatu yang tidak perlu dipecah lebih jauh. Misalnya, item dalam tabel People berisi atribut yang disebut personId,, LastNameFirstName, dan sebagainya. Untuk tabel Departemen, item mungkin memiliki atribut seperti DepartmentID, Name, Manager, dan sebagainya. Atribut di DynamoDB serupa dalam banyak hal dengan bidang atau kolom dalam sistem basis data lain.

Diagram berikut menunjukkan tabel bernama Orang dengan beberapa item dan atribut contoh.

```
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"
}
```

Perhatikan hal berikut tentang tabel Orang:

- Setiap item dalam tabel memiliki pengidentifikasi unik, atau kunci primer, yang membedakan item dari semua hal lain dalam tabel. Dalam tabel Orang, kunci primer terdiri dari satu atribut (PersonID).
- Selain kunci primer, tabel Orang tidak berskema, yang berarti bahwa baik atribut maupun jenis datanya perlu didefinisikan terlebih dahulu. Setiap item dapat memiliki atribut tersendiri.

- Sebagian besar atribut skalar, yang berarti bahwa mereka hanya dapat memiliki satu nilai. String dan angka adalah contoh umum dari skalar.
- Beberapa item memiliki atribut bersarang (Alamat). DynamoDB mendukung atribut bersarang hingga sedalam 32 tingkat.

Berikut ini adalah tabel contoh lain bernama Musik yang dapat Anda gunakan untuk melacak koleksi musik Anda.

```
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"
}
```

Perhatikan hal berikut tentang tabel Musik:

- Kunci utama untuk Musik terdiri dari dua atribut (Artis dan SongTitle). Setiap item dalam tabel harus memiliki dua atribut ini. Kombinasi Artis dan SongTitlemembedakan setiap item dalam tabel dari yang lainnya.
- Selain kunci primer, tabel Musik tidak berskema, yang berarti bahwa baik atribut maupun jenis datanya perlu didefinisikan terlebih dahulu. Setiap item dapat memiliki atribut tersendiri.
- Salah satu item memiliki atribut bersarang (PromotionInfo), yang berisi atribut bersarang lainnya. DynamoDB mendukung atribut bersarang hingga sedalam 32 tingkat.

Untuk informasi selengkapnya, lihat Bekerja dengan tabel dan data di DynamoDB.

Kunci primer

Ketika Anda membuat tabel, selain nama tabel, Anda harus menentukan kunci primer dari tabel. Kunci utama secara unik mengidentifikasi setiap item dalam tabel, sehingga tidak ada dua item dapat memiliki kunci yang sama.

DynamoDB mendukung dua jenis kunci primer:

- Kunci partisi — Sebuah kunci primer sederhana, terdiri dari satu atribut yang dikenal sebagai kunci partisi.

DynamoDB menggunakan nilai kunci partisi sebagai masukan untuk fungsi hash internal. Output dari fungsi hash menentukan partisi (penyimpanan fisik internal pada DynamoDB) di mana item akan disimpan.

Dalam tabel yang hanya memiliki kunci partisi, tidak ada dua item dapat memiliki nilai kunci partisi yang sama.

Tabel Orang yang dijelaskan dalam Tabel, item, dan atribut adalah contoh dari sebuah tabel dengan kunci primer sederhana (PersonID). Anda dapat mengakses item apa pun di tabel Orang secara langsung dengan memberikan PersonIdnilai untuk item tersebut.

- Kunci partisi dan kunci urutan – Disebut sebagai kunci primer komposit, jenis kunci ini terdiri dari dua atribut. Atribut pertama adalah kunci partisi, dan atribut kedua adalah kunci urutan.

  DynamoDB menggunakan nilai kunci partisi sebagai input untuk fungsi hash internal. Output dari fungsi hash menentukan partisi (penyimpanan fisik internal pada DynamoDB) di mana item akan disimpan. Semua item dengan nilai kunci partisi yang sama disimpan bersama-sama, dalam urutan yang diurutkan berdasarkan nilai kunci urutan.

  Dalam tabel yang memiliki kunci partisi dan kunci urutan, mungkin bagi beberapa item memiliki nilai kunci partisi yang sama. Namun, item tersebut harus memiliki nilai kunci urutan yang berbeda.

  Tabel Musik yang dijelaskan dalam Tabel, item, dan atribut adalah contoh tabel dengan kunci primer komposit (Artis dan SongTitle). Anda dapat mengakses item apa pun di tabel Musik secara langsung, jika Anda memberikan Artis dan SongTitlenilai untuk item tersebut.

  Kunci primer komposit memberi Anda fleksibilitas tambahan saat mengkueri data. Misalnya, jika Anda hanya memberikan nilai untuk Artist, DynamoDB mengambil semua lagu oleh penyanyi tersebut. Untuk mengambil hanya subset lagu oleh artis tertentu, Anda dapat memberikan nilai untuk Artis bersama dengan berbagai nilai untuk. SongTitle

> **Note**
>
> Kunci partisi dari item juga dikenal sebagai atribut hash. Istilah atribut hash berasal dari penggunaan fungsi hash internal di DynamoDB yang mendistribusikan item data secara merata di seluruh partisi, berdasarkan nilai kunci partisi mereka.

Kunci urutan item juga dikenal sebagai atribut rentang. Istilah atribut rentang berasal dari cara DynamoDB menyimpan item dengan kunci partisi yang sama yang secara fisik berdekatan, dalam urutan berdasarkan nilai kunci urutan.

Setiap atribut kunci primer harus skalar (yang berarti bahwa atribut itu hanya dapat menyimpan nilai tunggal). Satu-satunya jenis data yang diperbolehkan untuk atribut kunci primer adalah string, nomor, atau biner. Tidak ada batasan seperti itu untuk atribut non-kunci lainnya.

Indeks sekunder

Anda dapat membuat satu atau beberapa indeks sekunder pada tabel. Indeks sekunder memungkinkan Anda mengkueri data dalam tabel menggunakan kunci alternatif, selain kueri terhadap kunci primer. DynamoDB tidak mengharuskan Anda menggunakan indeks, tetapi mereka memberikan aplikasi Anda fleksibilitas lebih ketika mengkueri pada data Anda. Setelah Anda membuat indeks sekunder pada tabel, Anda bisa membaca data dari indeks dengan cara yang sama seperti yang Anda lakukan dari tabel.

DynamoDB mendukung dua jenis indeks:

- Indeks sekunder global – Indeks dengan kunci partisi dan kunci urutan yang mungkin berbeda dari yang ada di tabel. Nilai kunci primer dalam indeks sekunder global tidak perlu unik.
- Indeks sekunder lokal – Indeks yang memiliki kunci partisi yang sama dengan tabel, namun kunci urutan berbeda.

Di DynamoDB, indeks sekunder global GSIs () adalah indeks yang menjangkau seluruh tabel, memungkinkan Anda untuk melakukan kueri di semua kunci partisi. Indeks sekunder lokal (LSIs) adalah indeks yang memiliki kunci partisi yang sama dengan tabel dasar tetapi kunci pengurutan yang berbeda.

Setiap tabel di DynamoDB memiliki kuota 20 indeks sekunder global (kuota default) dan 5 indeks sekunder lokal.

Dalam contoh tabel Musik yang ditampilkan sebelumnya, Anda dapat meminta item data berdasarkan Artis (kunci partisi) atau oleh Artis dan SongTitle(kunci partisi dan kunci sortir). Bagaimana jika Anda juga ingin menanyakan data berdasarkan Genre dan AlbumTitle? Untuk melakukan ini, Anda dapat membuat indeks pada Genre dan AlbumTitle, lalu menanyakan indeks dengan cara yang sama seperti Anda menanyakan tabel Musik.

Diagram berikut menunjukkan contoh tabel Musik, dengan indeks baru yang disebut GenreAlbumTitle. Dalam indeks, Genre adalah kunci partisi dan AlbumTitlemerupakan kunci sortir.

Tabel Musik	GenreAlbumTitle

{
"Artist": "No One You Know",
"SongTitle": "My Dog Spot",
"AlbumTitle": "Hey Now",
"Price": 1.98,
"Genre": "Country",
"CriticRating": 8.4
}
	{
	"Genre": "Country",
	"AlbumTitle": "Hey Now",
	"Artist": "No One You Know",
	"SongTitle": "My Dog Spot"
	}

{
"Artist": "No One You Know",
"SongTitle": "Somewhere Down The Road",
"AlbumTitle": "Somewhat Famous",
"Genre": "Country",
"CriticRating": 8.4,
"Year": 1984
}
	{
	"Genre": "Country",
	"AlbumTitle": "Somewhat Famous",
	"Artist": "No One You Know",
	"SongTitle": "Somewhere Down The Road"
	}

{
"Artist": "The Acme Band",
"SongTitle": "Still in Love",
"AlbumTitle": "The Buck Starts Here",
"Price": 2.47,
"Genre": "Rock",
"PromotionInfo": {
"RadioStationsPlaying": {
"KHCR",
"KQBX",
"WTNR",
	{
	"Genre": "Rock",
	"AlbumTitle": "The Buck Starts Here",
	"Artist": "The Acme Band",
	"SongTitle": "Still In Love"
	}

Tabel Musik	GenreAlbumTitle

<pre>"WJJH"
},
"TourDates": {
"Seattle": "20150622",
"Cleveland": "20150630"
},
"Rotation": "Heavy"
}
}</pre>	

<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>

Perhatikan hal berikut tentang GenreAlbumTitleindeks:

- Setiap indeks adalah milik sebuah tabel, yang disebut tabel dasar untuk indeks. Dalam contoh sebelumnya, Musik adalah tabel dasar untuk indeks. GenreAlbumTitle
- DynamoDB mengelola indeks secara otomatis. Saat Anda menambahkan, memperbarui, atau menghapus item di tabel dasar, DynamoDB menambahkan, memperbarui, atau menghapus item terkait di indeks mana pun yang termasuk dalam tabel tersebut.
- Saat Anda membuat indeks, Anda menentukan atribut mana yang akan disalin, atau diproyeksikan, dari tabel dasar ke indeks. Minimal, DynamoDB memproyeksikan atribut kunci dari tabel dasar ke indeks. Hal ini terjadi pada `GenreAlbumTitle`, dimana hanya atribut kunci dari tabel yang `Music` diproyeksikan ke dalam indeks.

Anda dapat menanyakan GenreAlbumTitleindeks untuk menemukan semua album dari genre tertentu (misalnya, semua album Rock). Anda juga dapat menanyakan indeks untuk menemukan

semua album dalam genre tertentu yang memiliki judul album tertentu (misalnya, semua album Country dengan judul yang dimulai dengan huruf H).

Untuk informasi selengkapnya, lihat Meningkatkan akses data dengan indeks sekunder di DynamoDB.

DynamoDB Streams

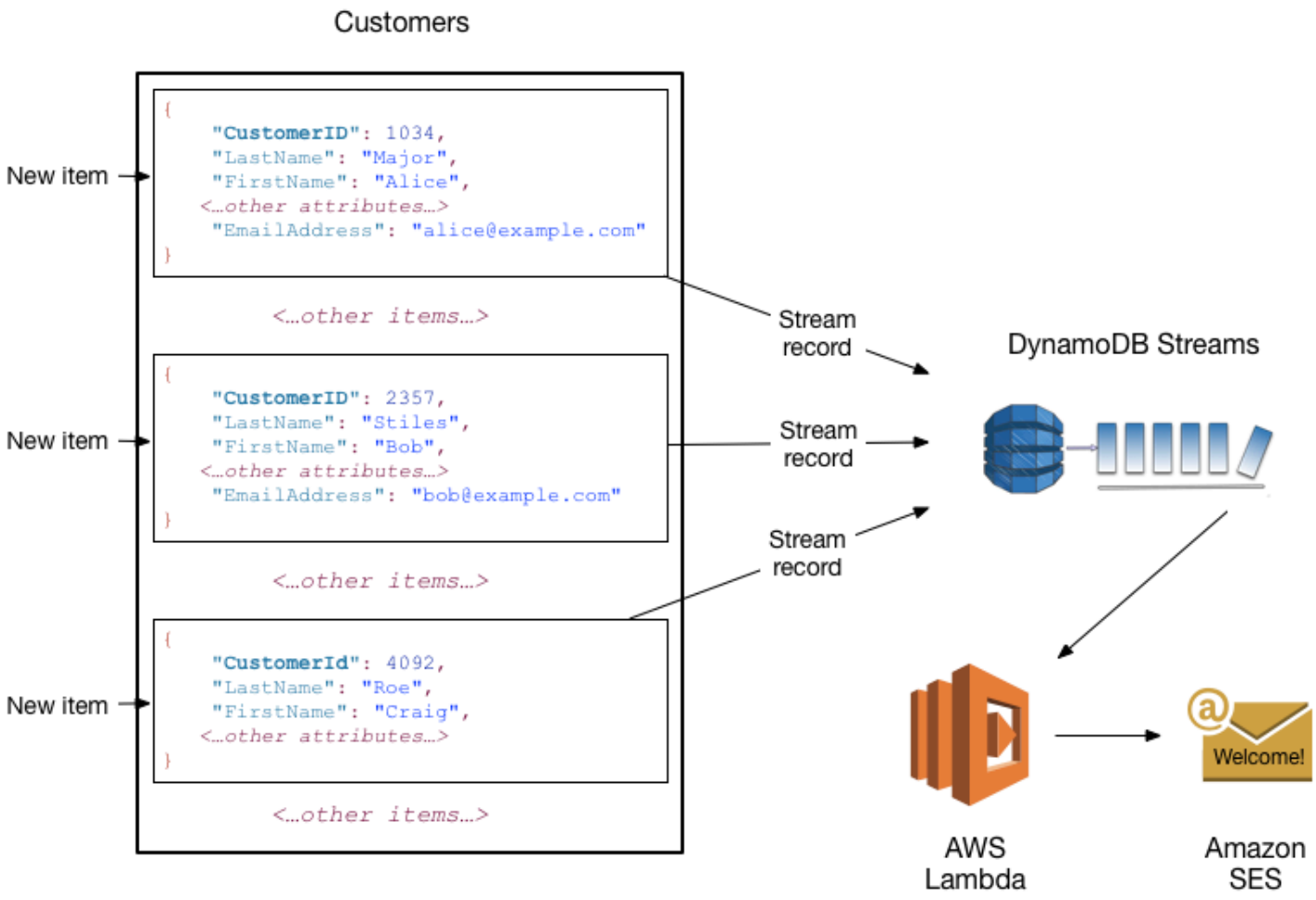
DynamoDB Streams adalah fitur opsional yang menangkap peristiwa modifikasi data dalam tabel DynamoDB. Data tentang peristiwa ini muncul di aliran hampir dalam waktu nyata, dan sesuai urutan terjadinya peristiwa tersebut.

Setiap acara diwakili oleh catatan aliran. Jika Anda mengaktifkan aliran pada tabel, DynamoDB Streams menulis catatan aliran setiap kali salah satu peristiwa berikut terjadi:

- Item baru ditambahkan ke tabel: Aliran mengambil gambar keseluruhan item, termasuk semua atributnya.
- Item diperbarui: Aliran menangkap gambar "sebelum" dan "sesudah" dari setiap atribut yang diubah dalam item.
- Item dihapus dari tabel: Aliran mengambil gambar keseluruhan item sebelum dihapus.

Setiap rekaman aliran juga berisi nama tabel, stempel waktu peristiwa, dan metadata lainnya. Rekaman streaming memiliki masa pakai 24 jam; setelah itu, mereka secara otomatis dihapus dari aliran.

Anda dapat menggunakan DynamoDB Streams AWS Lambda bersama-sama untuk membuat pemicu —kode yang berjalan secara otomatis setiap kali peristiwa yang diinginkan muncul dalam aliran. Misalnya, pertimbangkan tabel Pelanggan yang berisi informasi pelanggan untuk sebuah perusahaan. Misalkan Anda ingin mengirim email "selamat datang" ke setiap pelanggan baru. Anda dapat mengaktifkan aliran pada tabel itu, dan kemudian mengaitkan aliran tersebut dengan fungsi Lambda. Fungsi Lambda akan berjalan setiap kali rekaman aliran baru muncul, namun hanya memproses item baru yang ditambahkan ke tabel Pelanggan. Untuk item apa pun yang memiliki atribut `EmailAddress`, fungsi Lambda akan memanggil Amazon Simple Email Service (Amazon SES) untuk mengirim email ke alamat tersebut.



> **Note**
>
> Dalam contoh ini, pelanggan terakhir, Craig Roe, tidak akan menerima email karena dia tidak memiliki `EmailAddress`.

Selain pemicu, DynamoDB Streams memungkinkan solusi canggih seperti replikasi data di dalam dan AWS di seluruh Wilayah, tampilan data yang terwujud dalam tabel DynamoDB, analisis data menggunakan tampilan terwujud Kinesis, dan banyak lagi.

Untuk informasi selengkapnya, lihat Tangkapan data perubahan DynamoDB Streams.

DynamoDB API

Untuk bekerja dengan Amazon DynamoDB, aplikasi Anda harus menggunakan beberapa operasi API sederhana. Berikut ini adalah ringkasan operasi-operasi ini, yang disusun berdasarkan kategori.

> **Note**
>
> Untuk daftar lengkap operasi API, lihat Referensi API Amazon DynamoDB.

Topik

- Bidang kontrol
- Bidang data
- DynamoDB Streams
- Transaksi

Bidang kontrol

Operasi bidang kontrol memungkinkan Anda membuat dan mengelola tabel DynamoDB. Operasi ini juga memungkinkan Anda bekerja dengan indeks, aliran, dan objek lain yang tergantung pada tabel.

- `CreateTable` — Membuat tabel baru. Selain itu, Anda dapat membuat satu atau beberapa indeks sekunder, dan mengaktifkan DynamoDB Streams untuk tabel.
- `DescribeTable` — Mengembalikan informasi tentang tabel, seperti skema kunci primer, pengaturan throughput, dan informasi indeks.
- `ListTables` — Mengembalikan nama semua tabel Anda dalam daftar.
- `UpdateTable` — Memodifikasi pengaturan tabel atau indeks, menciptakan atau menghapus indeks baru pada tabel, atau memodifikasi pengaturan DynamoDB Streams untuk tabel.
- `DeleteTable` — Menghapus meja dan semua objek yang bergantung dari DynamoDB.

Bidang data

Operasi bidang data memungkinkan Anda membuat, membaca, memperbarui, dan menghapus tindakan (juga disebut CRUD) pada data dalam tabel. Beberapa operasi bidang data juga memungkinkan Anda membaca data dari indeks sekunder.

Anda dapat menggunakanPartiQL - Bahasa kueri yang kompatibel dengan SQL untuk Amazon DynamoDB, untuk melakukan operasi CRUD ini atau Anda dapat menggunakan CRUD klasik DynamoDB APIs yang memisahkan setiap operasi menjadi panggilan API yang berbeda.

PartiQL - Bahasa kueri yang kompatibel dengan SQL

- `ExecuteStatement` – Membaca beberapa item dari tabel. Anda juga dapat menulis atau memperbarui satu item dari tabel. Saat menulis atau memperbarui satu item, Anda harus menentukan atribut kunci primer.
- `BatchExecuteStatement` – Menulis, memperbarui, atau membaca beberapa item dari tabel. Ini lebih efisien dari `ExecuteStatement` karena aplikasi Anda hanya memerlukan satu jaringan bolak-balik untuk menulis atau membaca item.

Klasik APIs

Membuat data

- `PutItem` – Menulis satu item pada tabel. Anda harus menentukan atribut kunci primer, tetapi Anda tidak perlu menentukan atribut lainnya.
- `BatchWriteItem` – Menulis hingga 25 item pada tabel. Ini lebih efisien dari menjalankan `PutItem` beberapa kali karena aplikasi Anda hanya memerlukan satu jaringan bolak-balik untuk menulis item.

Membaca data

- `GetItem` – Mengambil satu item dari tabel. Anda harus menentukan kunci primer untuk item yang Anda inginkan. Anda dapat mengambil seluruh item, atau hanya sebagian dari atributnya.
- `BatchGetItem` – Mengambil hingga 100 item dari satu atau beberapa tabel. Ini lebih efisien dari menjalankan `GetItem` beberapa kali karena aplikasi Anda hanya memerlukan satu jaringan bolak-balik untuk membaca item.
- `Query` – Mengambil semua item yang memiliki kunci partisi tertentu. Anda harus menentukan nilai kunci partisi. Anda dapat mengambil seluruh item, atau hanya sebagian dari atributnya. Secara opsional, Anda dapat menerapkan ketentuan pada nilai kunci urutan sehingga Anda hanya mengambil subkumpulan data yang memiliki kunci partisi yang sama. Anda dapat menggunakan operasi ini pada tabel, asalkan tabel tersebut memiliki kunci partisi dan kunci urutan. Anda juga dapat menggunakan operasi ini pada indeks, asalkan indeks tersebut memiliki kunci partisi dan kunci urutan.
- `Scan` – Mengambil semua item dalam tabel atau indeks yang ditentukan. Anda dapat mengambil seluruh item, atau hanya sebagian dari atributnya. Secara opsional, Anda dapat menerapkan kondisi pemfilteran untuk hanya mengembalikan nilai yang Anda minati dan membuang sisanya.

Memperbarui data

- `UpdateItem` — Memodifikasi satu atau beberapa atribut dalam item. Memodifikasi satu atau lebih atribut dalam suatu item. Anda harus menentukan kunci primer untuk item yang ingin Anda ubah. Anda dapat menambahkan atribut baru dan mengubah atau menghapus atribut yang sudah ada. Anda juga dapat melakukan pembaruan bersyarat, sehingga pembaruan hanya berhasil bila kondisi yang ditentukan pengguna terpenuhi. Secara opsional, Anda dapat menerapkan penghitung atom, yang menambah atau mengurangi atribut numerik tanpa mengganggu permintaan tulis lainnya.

Menghapus data

- `DeleteItem` – Menghapus satu item dari tabel. Anda harus menentukan kunci primer untuk item yang ingin Anda hapus.
- `BatchWriteItem` – Menghapus hingga 25 item dari satu atau beberapa tabel. Ini lebih efisien dari menjalankan `DeleteItem` beberapa kali karena aplikasi Anda hanya memerlukan satu jaringan bolak-balik untuk menghapus item.

> **Note**
>
> Anda dapat menggunakan `BatchWriteItem` untuk membuat data dan menghapus data.

DynamoDB Streams

Operasi DynamoDB Streams memungkinkan Anda mengaktifkan atau menonaktifkan stream pada tabel, dan memungkinkan akses ke catatan modifikasi data yang terkandung dalam aliran.

- `ListStreams` – Menampilkan daftar semua stream Anda, atau hanya stream untuk tabel tertentu.
- `DescribeStream` – Menampilkan informasi tentang stream, seperti Amazon Resource Name (ARN) dan di mana aplikasi Anda dapat mulai membaca beberapa catatan stream pertama.
- `GetShardIterator` – Menampilkan iterator serpihan, yang merupakan struktur data yang menggunakan aplikasi Anda untuk mengambil catatan dari stream.
- `GetRecords` – Mengambil satu atau lebih catatan stream, menggunakan iterator serpihan yang diberikan.

Transaksi

Transaksi memberikan atomisitas, konsistensi, isolasi, dan daya tahan (ACID) yang memungkinkan Anda menjaga kebenaran data dalam aplikasi Anda dengan lebih mudah.

Anda dapat menggunakanPartiQL - Bahasa kueri yang kompatibel dengan SQL untuk Amazon DynamoDB, untuk melakukan operasi transaksional atau Anda dapat menggunakan CRUD klasik DynamoDB APIs yang memisahkan setiap operasi menjadi panggilan API yang berbeda.

PartiQL - Bahasa kueri yang kompatibel dengan SQL

- `ExecuteTransaction`— Operasi batch yang memungkinkan operasi CRUD ke beberapa item baik di dalam maupun di seluruh tabel dengan all-or-nothing hasil yang dijamin.

Klasik APIs

- `TransactWriteItems`— Operasi batch yang memungkinkan`Put`,`Update`, dan `Delete` operasi ke beberapa item baik di dalam maupun di seluruh tabel dengan all-or-nothing hasil yang dijamin.
- `TransactGetItems` – Sebuah operasi batch yang memungkinkan operasi `Get` untuk mengambil beberapa item dari satu atau beberapa tabel.

Jenis data dan aturan penamaan yang didukung di Amazon DynamoDB

Bagian ini menjelaskan aturan penamaan Amazon DynamoDB dan berbagai jenis daya yang didukung DynamoDB. Ada batasan yang berlaku untuk jenis daya. Untuk informasi selengkapnya, lihat Jenis Data.

Topik

- Peraturan penamaan
- Jenis Data
- Deskriptor jenis daya

Peraturan penamaan

Tabel, atribut, dan objek lain di DynamoDB harus memiliki nama. Nama harus bermakna dan ringkas-misalnya, nama-nama seperti Produk, Buku, dan Penulis sudah cukup jelas.

Berikut ini adalah aturan penamaan untuk DynamoDB:

- Semua nama harus dikodekan menggunakan UTF-8, dan peka huruf kapital.
- Nama tabel dan nama indeks panjangnya harus antara 3 hingga 255 karakter, dan hanya dapat berisi karakter berikut:
  - `a-z`
  - `A-Z`
  - `0-9`
  - `_` (garis bawah)
  - `-` (dasbor)
  - `.` (titik)
- Nama atribut harus memiliki panjang minimal satu karakter dan berukuran kurang dari 64 KB. Hal ini dianggap sebagai praktik terbaik untuk menjaga nama atribut Anda sesingkat mungkin. Hal ini membantu mengurangi penggunaan unit permintaan baca, karena nama atribut disertakan dalam pengukuran penyimpanan dan penggunaan throughput.

  Berikut ini adalah pengecualiannya. Panjang nama atribut berikut tidak boleh lebih dari 255 karakter:
  - Nama kunci partisi indeks sekunder
  - Nama kunci urutan indeks sekunder
  - Nama atribut proyeksi yang ditentukan pengguna (hanya berlaku untuk indeks sekunder lokal)

Kata-kata khusus dan karakter khusus

DynamoDB memiliki daftar kata-kata khusus dan karakter khusus. Untuk daftar lengkap, lihat Kata-kata penggunaan khusus di DynamoDB. Selain itu, karakter berikut memiliki arti khusus dalam DynamoDB: # (hash) dan :(titik dua).

Meskipun DynamoDB mengizinkan Anda menggunakan kata-kata khusus dan karakter khusus ini untuk nama, kami menyarankan agar Anda menghindari melakukannya karena Anda harus

menentukan variabel placeholder setiap kali Anda menggunakan nama-nama ini dalam sebuah ekspresi. Untuk informasi selengkapnya, lihat Nama atribut ekspresi (alias) di DynamoDB.

Jenis Data

DynamoDB mendukung banyak jenis data berbeda untuk atribut dalam tabel. Hal tersebut dapat dikategorikan sebagai berikut:

- Jenis Skalar - Jenis skalar dapat mewakili tepat satu nilai. Jenis skalar adalah angka, string, biner, Boolean, dan null.
- Jenis Dokumen — Jenis dokumen dapat mewakili struktur kompleks dengan atribut bertingkat, seperti yang Anda temukan di dokumen JSON. Jenis dokumen adalah daftar dan peta.
- Jenis Set - Jenis set dapat mewakili beberapa nilai skalar. Jenis set adalah set string, set angka, dan set biner.

Saat Anda membuat tabel atau indeks sekunder, Anda harus menentukan nama dan jenis data dari setiap atribut kunci primer (kunci partisi dan kunci urutan). Selanjutnya, setiap atribut kunci primer harus didefinisikan sebagai jenis string, angka, atau biner.

DynamoDB adalah basis data NoSQL dan tidak berskema. Artinya, selain atribut kunci primer, Anda tidak perlu mendefinisikan atribut atau jenis data apa pun saat membuat tabel. Sebagai perbandingan, basis data relasional mengharuskan Anda menentukan nama dan jenis data setiap kolom saat Anda membuat tabel.

Berikut penjelasan masing-masing jenis data beserta contohnya dalam format JSON.

Jenis skalar

Jenis skalar adalah angka, string, biner, Boolean, dan null.

Angka

Angka bisa positif, negatif, atau nol. Angka dapat memiliki hingga 38 digit presisi. Melebihi hasil ini dalam pengecualian. Jika Anda membutuhkan presisi lebih dari 38 digit, Anda dapat menggunakan string.

- Kisaran positif: 1E-130 hingga 9,99999999999999999999999999999999999999E+125
- Kisaran negatif: -9,99999999999999999999999999999999999999E+125 hingga -1E-130

Di DynamoDB, angka direpresentasikan sebagai panjang variabel. Angka nol di depan dan di belakang dipangkas.

Semua angka dikirim melalui jaringan untuk DynamoDB sebagai string, untuk memaksimalkan kompatibilitas di seluruh bahasa dan pustaka. Namun, DynamoDB memperlakukannya sebagai atribut jenis angka untuk operasi matematika.

Anda dapat menggunakan jenis data angka untuk mewakili tanggal atau stempel waktu. Salah satu cara untuk melakukannya adalah dengan menggunakan epoch time—jumlah detik sejak 00:00:00 UTC pada tanggal 1 Januari 1970. Sebagai contoh, jangka waktu `1437136300` mewakili 12:31:40 PM UTC pada tanggal 17 Juli 2015.

Untuk informasi lebih lanjut, lihat http://en.wikipedia. org/wiki/Unix_waktu.

String

String adalah Unicode dengan pengodean biner UTF-8. Panjang minimum string bisa nol, jika atribut tidak digunakan sebagai kunci untuk indeks atau tabel, dan dibatasi oleh batas ukuran item DynamoDB maksimum sebesar 400 KB.

Batasan tambahan berikut berlaku untuk atribut kunci primer yang didefinisikan sebagai jenis string:

- Untuk kunci primer sederhana, panjang maksimum nilai atribut pertama (kunci partisi) adalah 2048 byte.
- Untuk kunci primer komposit, panjang maksimum nilai atribut kedua (kunci urutan) adalah 1024 byte.

DynamoDB menyusun dan membandingkan string menggunakan byte pengodean string UTF-8 yang mendasarinya. Misalnya, "a" (0x61) lebih besar dari "A" (0x41), dan "¿" (0xC2BF) lebih besar dari "z" (0x7A).

Anda dapat menggunakan jenis data string untuk mewakili tanggal atau stempel waktu. Salah satu cara untuk melakukannya adalah dengan menggunakan string ISO 8601, seperti yang ditunjukkan dalam contoh berikut:

- `2016-02-15`
- `2015-12-21T17:42:34Z`
- `20150311T122706Z`

Untuk informasi lebih lanjut, lihat http://en.wikipedia. org/wiki/ISO_8601.

> **Note**
>
> Tidak seperti basis data relasional konvensional, DynamoDB tidak mendukung jenis daya tanggal dan waktu. Akan lebih berguna jika menyimpan data tanggal dan waktu sebagai jenis data angka, menggunakan jangka waktu Unix.

Biner

Atribut jenis biner dapat menyimpan data biner apa pun, seperti teks terkompresi, data terenkripsi, atau gambar. Setiap kali DynamoDB membandingkan nilai biner, DynamoDB memperlakukan setiap byte data biner sebagai tidak terdaftar.

Panjang atribut binari bisa nol, jika atribut tersebut tidak digunakan sebagai kunci untuk indeks atau tabel, dan dibatasi oleh batas ukuran item DynamoDB maksimum sebesar 400 KB.

Jika Anda mendefinisikan atribut kunci primer sebagai atribut jenis biner, batasan tambahan berikut akan berlaku:

- Untuk kunci primer sederhana, panjang maksimum nilai atribut pertama (kunci partisi) adalah 2048 byte.
- Untuk kunci primer komposit, panjang maksimum nilai atribut kedua (kunci urutan) adalah 1024 byte.

Aplikasi Anda harus mengodekan nilai biner dalam format yang dikodekan base64 sebelum mengirimkannya ke DynamoDB. Setelah menerima nilai-nilai ini, DynamoDB mendekodekan data ke dalam array byte yang tidak ditandatangani dan menggunakannya sebagai panjang atribut binari.

Contoh berikut adalah atribut binari, menggunakan teks berkode base64.

```
dGhpcyB0ZXh0IGlzIGJhc2U2NC1lbmNvZGVk
```

Boolean

Atribut jenis Boolean dapat menyimpan salah satu `true` atau `false`.

Null

Null melambangkan atribut dengan status tidak diketahui atau tidak ditentukan.

Jenis dokumen

Jenis dokumen adalah daftar dan peta. Jenis data ini dapat disarangkan satu sama lain, untuk mewakili struktur data kompleks hingga sedalam 32.

Tidak ada batasan jumlah nilai dalam daftar atau peta, selama item yang berisi nilai tersebut sesuai dengan batas ukuran item DynamoDB (400 KB).

Nilai atribut dapat berupa string kosong atau nilai biner kosong jika atribut tersebut tidak digunakan untuk tabel atau kunci indeks. Nilai atribut tidak boleh berupa set kosong (set string, set angka, atau set biner), namun daftar dan peta kosong diperbolehkan. String kosong dan nilai biner diperbolehkan dalam daftar dan peta. Untuk informasi selengkapnya, lihat Atribut.

Daftar

Atribut jenis daftar dapat menyimpan kumpulan nilai yang diurutkan. Daftar diapit tanda kurung siku: `[ ... ]`

Daftar mirip dengan array JSON. Tidak ada batasan pada jenis daya yang dapat disimpan dalam elemen daftar, dan elemen dalam elemen daftar tidak harus berjenis sama.

Contoh berikut memperlihatkan daftar yang berisi dua string dan sebuah angka.

```
FavoriteThings: ["Cookies", "Coffee", 3.14159]
```

> **Note**
>
> DynamoDB memungkinkan Anda bekerja dengan elemen individual dalam daftar, meskipun elemen tersebut sangat bersarang. Untuk informasi selengkapnya, lihat Menggunakan ekspresi di DynamoDB.

Peta

Atribut jenis peta dapat menyimpan kumpulan pasangan nama-nilai yang tidak berurutan. Peta diapit kurung kurawal: `{ ... }`

Peta serupa dengan objek JSON. Tidak ada batasan pada jenis data yang dapat disimpan dalam elemen peta, dan elemen dalam peta tidak harus berjenis sama.

Peta ideal untuk menyimpan dokumen JSON di DynamoDB. Contoh berikut memperlihatkan peta yang berisi string, angka, dan daftar bertumpuk yang berisi peta lain.

```
{
    Day: "Monday",
    UnreadEmails: 42,
    ItemsOnMyDesk: [
        "Coffee Cup",
        "Telephone",
        {
            Pens: { Quantity : 3},
            Pencils: { Quantity : 2},
            Erasers: { Quantity : 1}
        }
    ]
}
```

> **Note**
>
> DynamoDB memungkinkan Anda bekerja dengan elemen individual dalam peta, meskipun elemen tersebut sangat bersarang. Untuk informasi selengkapnya, lihat Menggunakan ekspresi di DynamoDB.

Set

DynamoDB mendukung jenis yang mewakili set angka, string, atau nilai-nilai biner. Semua elemen di dalam set harus dari jenis yang sama. Misalnya, Set Angka hanya dapat berisi angka dan Set String hanya dapat berisi string.

Tidak ada batasan jumlah nilai dalam set, selama item yang berisi nilai tersebut sesuai dengan batas ukuran item DynamoDB (400 KB).

Setiap nilai dalam suatu set harus unik. Urutan nilai dalam suatu set tidak dipertahankan. Oleh karena itu, aplikasi Anda tidak boleh bergantung pada urutan elemen tertentu dalam set tersebut. DynamoDB tidak mendukung set kosong, namun string kosong dan nilai biner diperbolehkan dalam satu set.

Contoh berikut menunjukkan set string, set angka, dan set biner:

```
["Black", "Green", "Red"]

[42.2, -19, 7.5, 3.14]

["U3Vubnk=", "UmFpbnk=", "U25vd3k="]
```

Deskriptor jenis daya

Protokol API DynamoDB tingkat rendah menggunakan Deskriptor jenis data sebagai token yang memberi tahu DynamoDB cara menafsirkan setiap atribut.

Berikut ini adalah daftar lengkap deskriptor jenis data DynamoDB:

- **`S`** – String
- **`N`** – Nomor
- **`B`** – Biner
- **`BOOL`** – Boolean
- **`NULL`** – Null
- **`M`** – Peta
- **`L`** – Daftar
- **`SS`** – Set String
- **`NS`** – Set Nomor
- **`BS`** – Set Biner

Kelas tabel DynamoDB

DynamoDB menawarkan dua kelas tabel yang dirancang untuk membantu Anda mengoptimalkan biaya. Kelas tabel Standar DynamoDB adalah defaultnya, dan direkomendasikan untuk sebagian besar beban kerja. Kelas tabel DynamoDB Standard-Infrequent Access (DynamoDB Standard-IA) dioptimalkan untuk tabel yang biayanya didominasi oleh penyimpanan. Misalnya, tabel yang menyimpan data yang jarang diakses, seperti log aplikasi, postingan media sosial lama, riwayat pesanan e-niaga, dan pencapaian game sebelumnya, merupakan kandidat yang baik untuk kelas tabel Standard-IA. Lihat Harga Amazon DynamoDB untuk mendapatkan detail harga.

Setiap tabel DynamoDB dikaitkan dengan kelas tabel (DynamoDB Standard secara default). Semua indeks sekunder yang terkait dengan tabel menggunakan kelas tabel yang sama. Setiap kelas tabel menawarkan harga berbeda untuk penyimpanan data serta permintaan baca dan tulis. Anda dapat memilih kelas tabel yang paling hemat biaya untuk tabel Anda berdasarkan pola penggunaan penyimpanan dan throughputnya.

Pilihan kelas tabel tidak permanen—Anda dapat mengubah pengaturan ini menggunakan, AWS Management Console AWS CLI, atau SDK. AWS DynamoDB juga mendukung pengelolaan kelas tabel Anda AWS CloudFormation menggunakan tabel Single-region dan tabel global. Untuk mempelajari selengkapnya tentang memilih kelas tabel Anda, lihat Pertimbangan saat memilih kelas tabel di DynamoDB.

Partisi dan distribusi data di DynamoDB

Amazon DynamoDB menyimpan data dalam partisi. Partisi adalah alokasi penyimpanan untuk tabel, didukung oleh solid state drive (SSDs) dan secara otomatis direplikasi di beberapa Availability Zone dalam suatu AWS Region. Manajemen partisi ditangani sepenuhnya oleh DynamoDB—Anda tidak perlu mengelola partisi sendiri.

Saat Anda membuat tabel, status awal tabel tersebut adalah `CREATING`. Selama fase ini, DynamoDB mengalokasikan partisi yang memadai ke tabel sehingga dapat menangani persyaratan throughput yang Anda sediakan. Anda dapat mulai menulis dan membaca data tabel setelah status tabel berubah menjadi `ACTIVE`.

DynamoDB mengalokasikan partisi tambahan ke tabel dalam situasi berikut:

- Jika Anda meningkatkan pengaturan throughput tabel yang tersedia melebihi apa yang dapat didukung oleh partisi yang ada.
- Jika partisi yang ada memenuhi kapasitas dan diperlukan lebih banyak ruang penyimpanan.

Manajemen partisi terjadi secara otomatis di latar belakang dan transparan bagi aplikasi Anda. Tabel Anda tetap tersedia dan sepenuhnya mendukung persyaratan throughput yang Anda sediakan.

Untuk rincian selengkapnya, lihat Desain kunci partisi.

Indeks sekunder global di DynamoDB juga terdiri dari partisi. Data dalam indeks sekunder global disimpan secara terpisah dari data dalam tabel dasarnya, namun partisi indeks berperilaku sama seperti partisi tabel.

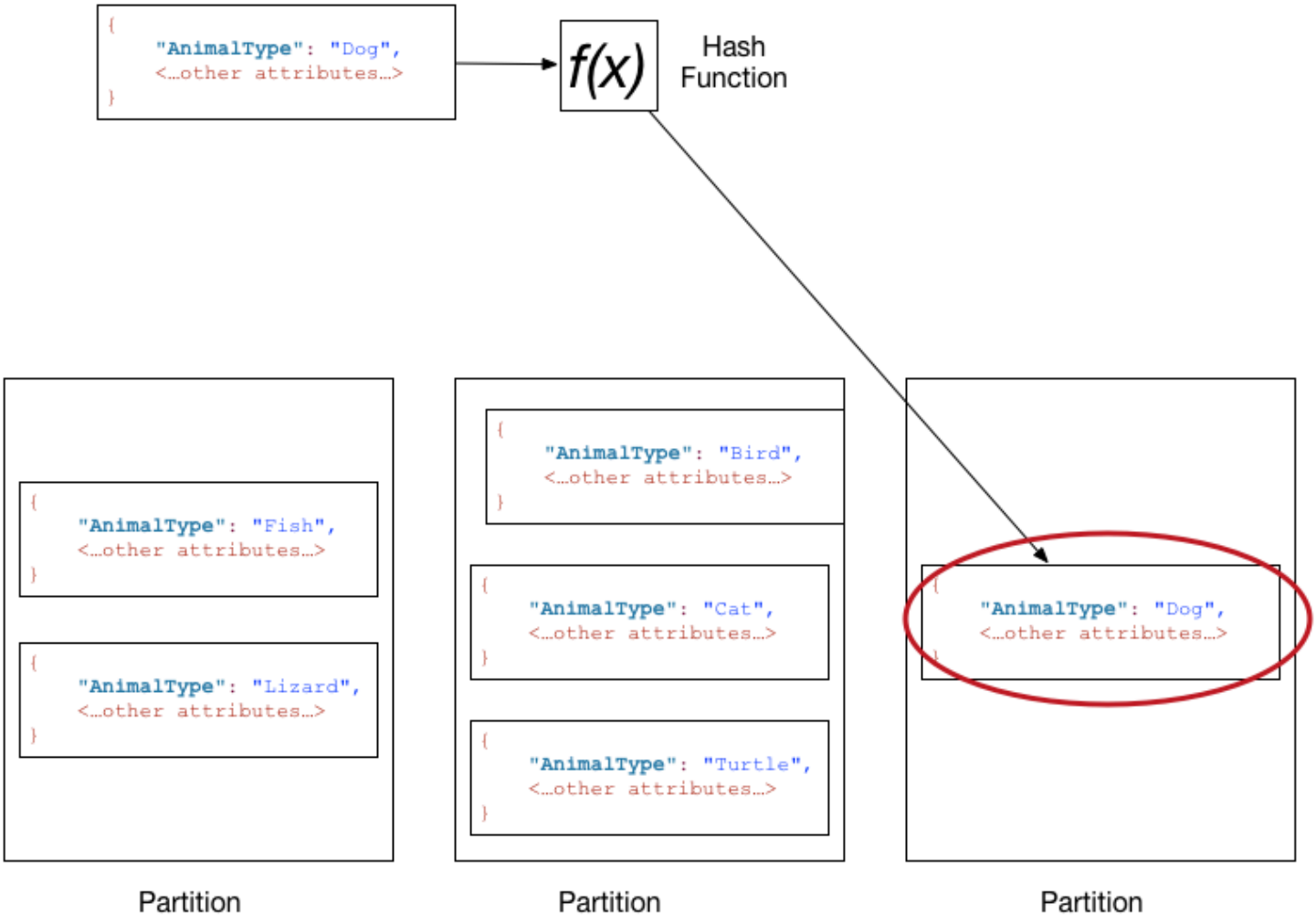
Distribusi data: Kunci partisi

Jika tabel Anda memiliki kunci primer sederhana (hanya kunci partisi), DynamoDB menyimpan dan mengambil setiap item berdasarkan nilai kunci partisinya.

Untuk menulis item ke tabel, DynamoDB menggunakan nilai kunci partisi sebagai input ke fungsi hash internal. Nilai keluaran dari fungsi hash menentukan partisi tempat item akan disimpan.

Untuk membaca item dari tabel, Anda harus menentukan nilai kunci partisi untuk item tersebut. DynamoDB menggunakan nilai ini sebagai masukan ke fungsi hashnya, menghasilkan partisi tempat item dapat ditemukan.

Diagram berikut menunjukkan tabel bernama Pets, yang mencakup beberapa partisi. Kunci utama tabel adalah AnimalType(hanya atribut kunci ini yang ditampilkan). DynamoDB menggunakan fungsi hashnya untuk menentukan tempat menyimpan item baru, dalam hal ini berdasarkan nilai hash string Dog. Perhatikan bahwa item tidak disimpan dalam urutan yang diurutkan. Lokasi setiap item ditentukan oleh nilai hash kunci partisinya.



> **Note**
>
> DynamoDB dioptimalkan untuk distribusi item yang seragam di seluruh partisi tabel, tidak peduli berapa banyak partisi yang ada. Kami menyarankan Anda memilih kunci partisi yang dapat memiliki sejumlah besar nilai berbeda yang berhubungan dengan jumlah item dalam tabel.

Distribusi data: Kunci partisi dan kunci urutan

Jika tabel memiliki kunci primer gabungan (kunci partisi dan kunci urutan), DynamoDB menghitung nilai hash kunci partisi dengan cara yang sama seperti yang dijelaskan dalamDistribusi data: Kunci partisi. Namun, ia cenderung menjaga item yang memiliki nilai kunci partisi yang sama berdekatan dan diurutkan berdasarkan nilai atribut kunci sortir. Kumpulan item yang memiliki nilai kunci partisi yang sama disebut koleksi item. Koleksi item dioptimalkan untuk pengambilan rentang item yang

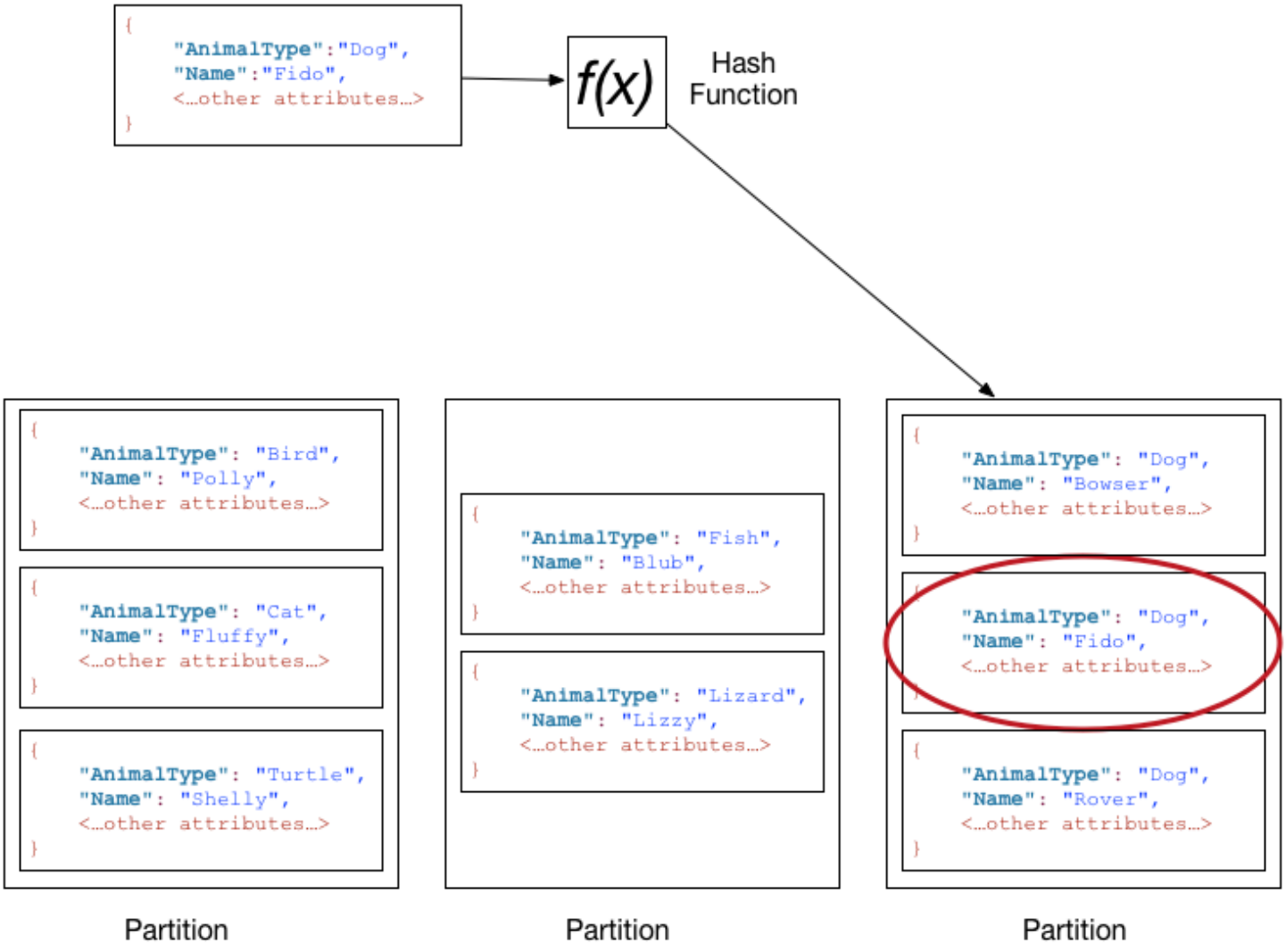
efisien dalam koleksi. Jika tabel Anda tidak memiliki indeks sekunder lokal, DynamoDB akan secara otomatis membagi koleksi item Anda menjadi partisi sebanyak yang diperlukan untuk menyimpan data dan menyajikan throughput baca dan tulis.

Untuk menulis item ke tabel, DynamoDB menghitung nilai hash kunci partisi untuk menentukan partisi mana yang harus berisi item tersebut. Di partisi itu, beberapa item mungkin memiliki nilai kunci partisi yang sama. Jadi DynamoDB menyimpan item tersebut di antara item lainnya dengan kunci partisi yang sama, dalam urutan menaik berdasarkan kunci urutan.

Untuk membaca item dari tabel, Anda harus menentukan nilai kunci partisi dan nilai kunci urutan. DynamoDB menghitung nilai hash kunci partisi, menghasilkan partisi tempat item dapat ditemukan.

Anda dapat membaca beberapa item dari tabel dalam satu operasi (`Query`) jika item yang Anda inginkan memiliki nilai kunci partisi yang sama. DynamoDB mengembalikan semua item dengan nilai kunci partisi tersebut. Secara opsional, Anda dapat menerapkan ketentuan pada kunci urutan sehingga hanya mengembalikan item dalam rentang nilai tertentu.

Misalkan tabel Pets memiliki kunci primer komposit yang terdiri dari AnimalType(kunci partisi) dan Nama (kunci sortir). Diagram berikut menunjukkan DynamoDB menulis item dengan nilai kunci partisi Dog dan nilai kunci urutan Fido.



Untuk membaca item yang sama dari tabel Pets, DynamoDB menghitung nilai hash dari Dog, menghasilkan partisi tempat item tersebut disimpan. DynamoDB kemudian memindai nilai atribut kunci urutan hingga menemukan Fido.

Untuk membaca semua item dengan Dog, Anda dapat mengeluarkan `Query` operasi tanpa menentukan kondisi kunci pengurutan. AnimalType Secara default, item dikembalikan dalam urutan penyimpanannya (yaitu, dalam urutan naik oleh kunci urutan). Atau, Anda dapat meminta urutan menurun.

Untuk mengkueri hanya beberapa item Dog, Anda dapat menerapkan syarat untuk kunci urutan (misalnya, hanya item Dog di mana Name dimulai dengan huruf yang berada dalam rentang A hingga K).

> **Note**
>
> Dalam tabel DynamoDB, tidak ada batasan atas jumlah nilai kunci urutan yang berbeda per nilai kunci partisi. Jika Anda perlu untuk menyimpan jutaan item Dog di tabel Pets, DynamoDB akan mengalokasikan cukup penyimpanan untuk menangani persyaratan ini secara otomatis.

Pelajari cara beralih dari SQL ke NoSQL

Jika Anda seorang developer aplikasi, Anda mungkin memiliki beberapa pengalaman menggunakan sistem manajemen basis data relasional (RDBMS) dan Structured Kueri Language (SQL). Ketika Anda mulai bekerja dengan Amazon DynamoDB, Anda akan menemukan banyak kesamaan, tetapi juga banyak hal yang berbeda. NoSQL adalah istilah yang digunakan untuk menggambarkan sistem basis data nonrelational yang sangat tersedia, dapat diskalakan, dan dioptimalkan untuk performa tinggi. Alih-alih model relasional, basis data NoSQL (seperti DynamoDB) menggunakan model alternatif untuk manajemen data, seperti pasangan nilai kunci atau penyimpanan dokumen. Untuk informasi selengkapnya, lihat Apa itu NoSQL?.

Amazon DynamoDB mendukung PartiQL, bahasa kueri open-source yang kompatibel dengan SQL yang memudahkan Anda untuk melakukan kueri data secara efisien, terlepas dari di mana atau dalam format apa itu disimpan. Dengan PartiQL, Anda dapat dengan mudah memproses data terstruktur dari basis data relasional, data semi-terstruktur dan bersarang dalam format data terbuka, dan bahkan data tanpa skema di NoSQL atau basis data dokumen yang memungkinkan atribut berbeda untuk baris yang berbeda. Untuk informasi selengkapnya, lihat bahasa kueri PartiQL.

Bagian berikut menjelaskan tugas basis data umum, membandingkan dan mengontraskan pernyataan SQL dengan operasi DynamoDB setara mereka.

> **Note**
>
> Contoh SQL di bagian ini kompatibel dengan MySQL RDBMS.
> Contoh DynamoDB di bagian ini menunjukkan nama operasi DynamoDB, bersama dengan parameter untuk operasi dalam format JSON.

Topik

- Memilih antara relasional (SQL) dan NoSQL
- Perbedaan dalam mengakses database relasional (SQL) dan DynamoDB
- Perbedaan antara database relasional (SQL) dan DynamoDB saat membuat tabel
- Perbedaan antara mendapatkan informasi tabel dari database relasional (SQL) dan DynamoDB
- Perbedaan antara database relasional (SQL) dan DynamoDB saat menulis data ke tabel
- Perbedaan antara database relasional (SQL) dan DynamoDB saat membaca data dari tabel
- Perbedaan antara database relasional (SQL) dan DynamoDB saat mengelola indeks
- Perbedaan antara database relasional (SQL) dan DynamoDB saat memodifikasi data dalam tabel
- Perbedaan antara database relasional (SQL) dan DynamoDB saat menghapus data dari tabel
- Perbedaan antara database relasional (SQL) dan DynamoDB saat menghapus tabel

Memilih antara relasional (SQL) dan NoSQL

Aplikasi masa kini memiliki persyaratan yang lebih menuntut daripada sebelumnya. Misalnya, game online mungkin dimulai dengan hanya beberapa pengguna dan jumlah data yang sangat kecil. Namun, jika game menjadi sukses, itu dapat dengan mudah melampaui sumber daya dari sistem manajemen basis data yang mendasarinya. Sudah biasa bagi aplikasi berbasis web untuk memiliki ratusan, ribuan, atau jutaan pengguna bersamaan, dengan data baru seukuran terabyte atau lebih yang dihasilkan per hari. Basis data untuk aplikasi tersebut harus menangani puluhan (atau ratusan) dari ribuan baca dan tulis per detik.

Amazon DynamoDB sangat cocok untuk jenis beban kerja tersebut. Sebagai developer, Anda dapat memulai dari kecil dan secara bertahap meningkatkan pemanfaatan Anda seiring aplikasi Anda menjadi lebih populer. DynamoDB menskalakan tanpa hambatan untuk menangani jumlah data yang sangat besar dan jumlah pengguna yang sangat besar.

Untuk informasi lebih lanjut tentang pemodelan basis data relasional tradisional dan cara menyesuaikannya untuk DynamoDB, lihat Praktik terbaik untuk memodelkan data relasional di DynamoDB.

Tabel berikut menunjukkan beberapa perbedaan tingkat tinggi antara sistem manajemen basis data relasional (RDBMS) dan DynamoDB.

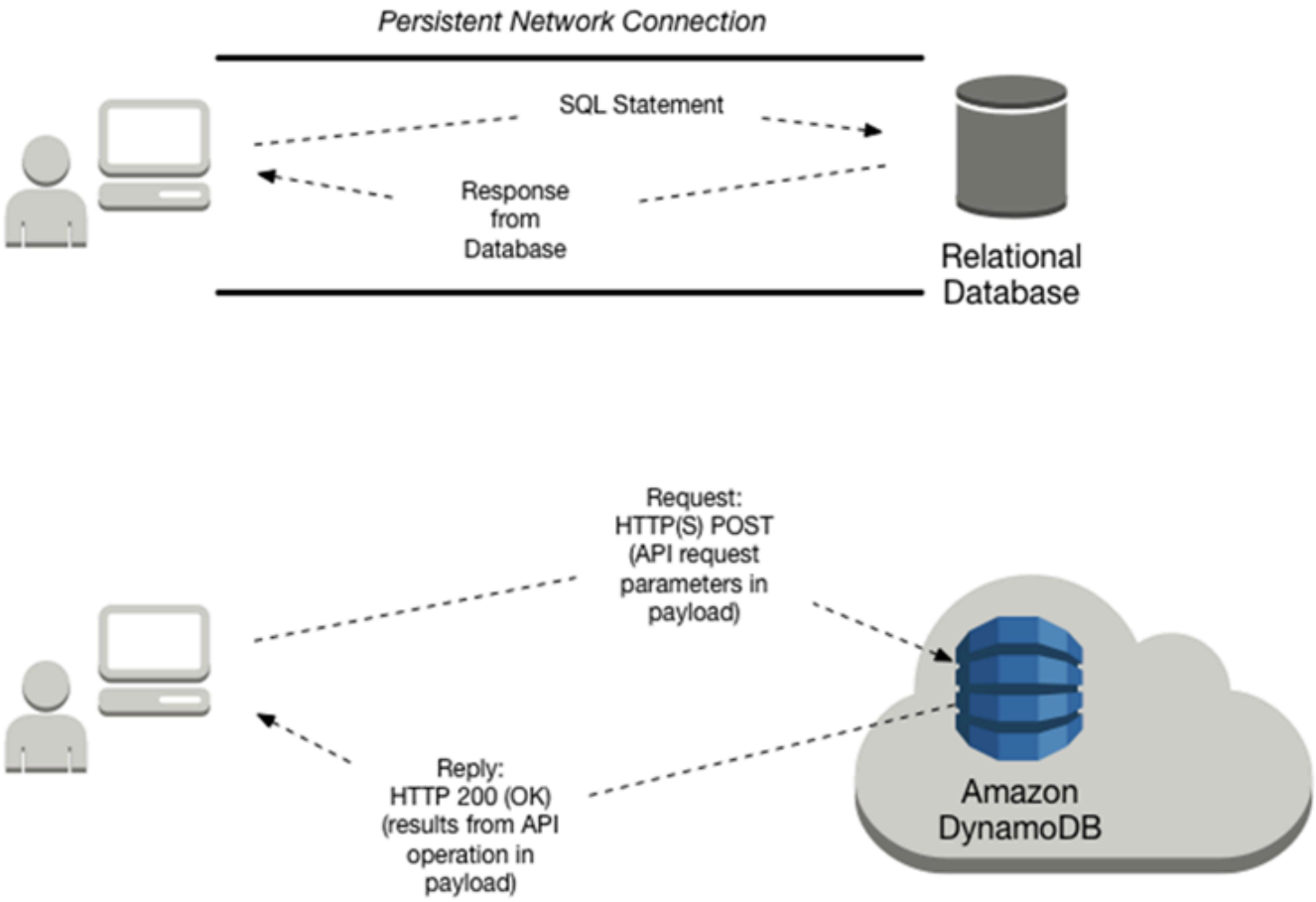
Karakteristik	Sistem manajemen basis data relasional (RDBMS)	Amazon DynamoDB
Beban Kerja Optimal	Kueri ad hoc; pergudangan data; OLAP (pemrosesan analitis online).	Aplikasi berskala web, termasuk jejaring sosial, game, berbagi media, dan Internet untuk Segala (IoT).
Model Data	Model relasional membutuhkan skema yang didefinisikan dengan baik, di mana data dinormalisasi ke dalam tabel, baris, dan kolom. Selain itu, semua hubungan didefinisikan antara tabel, kolom, indeks, dan elemen basis data lainnya.	DynamoDB tidak memiliki skema. Setiap tabel harus memiliki kunci primer untuk secara unik mengidentifikasi setiap item data, tetapi tidak ada kendala yang sama pada atribut non-kunci lainnya. DynamoDB dapat mengelola data terstruktur atau semi terstruktur, termasuk dokumen JSON.
Akses Data	SQL adalah standar untuk menyimpan dan mengambil data. Basis data relasional menawarkan seperangkat alat untuk menyederhanakan pengembangan aplikasi yang didorong basis data, tetapi semua alat ini menggunakan SQL.	Anda dapat menggunakan AWS Management Console, the AWS CLI, atau WorkBench NoSQL untuk bekerja dengan DynamoDB dan melakukan tugas ad hoc. PartiQL, bahasa kueri yang kompatibel dengan SQL, memungkinkan Anda memilih, menyisipkan, memperbarui, dan menghapus data di DynamoDB. Aplikasi dapat menggunakan kit pengembangan AWS perangkat lunak (SDKs) untuk bekerja dengan DynamoDB menggunakan antarmuka berbasis objek,

Karakteristik	Sistem manajemen basis data relasional (RDBMS)	Amazon DynamoDB
		berpusat pada dokumen, atau tingkat rendah.
Performa	Basis data relasional dioptimalkan untuk penyimpanan, sehingga performa umumnya tergantung pada subsistem disk. Developer dan administrator basis data harus mengoptimalkan permintaan, indeks, dan struktur tabel untuk mencapai puncak performa.	DynamoDB dioptimalkan untuk melakukan komputasi, sehingga performa utamanya adalah fungsi dari hardware yang mendasari dan latensi jaringan. Sebagai layanan terkelola, DynamoDB mengisolasi Anda dan aplikasi Anda dari detail implementasi ini, sehingga Anda dapat fokus pada merancang dan membangun aplikasi yang kuat dan berperforma tinggi.
Penskalaan	Paling mudah untuk menaikkan skala dengan perangkat keras yang lebih cepat. Mungkin juga bagi tabel basis data untuk menjangkau beberapa host dalam sistem terdistribusi, tetapi ini memerlukan investasi tambahan. basis data relasional memiliki ukuran maksimum untuk jumlah dan ukuran file, yang membebankan batas atas pada skalabilitas.	DynamoDB dirancang untuk diperluas menggunakan klaster perangkat keras terdistribusi. Desain ini memungkinkan peningkatan throughput tanpa peningkatan latensi. Pelanggan menentukan persyaratan throughput mereka, dan DynamoDB mengalokasikan sumber daya yang cukup untuk memenuhi persyaratan tersebut. Tidak ada batas atas pada jumlah item per tabel, atau ukuran total tabel tersebut.

Perbedaan dalam mengakses database relasional (SQL) dan DynamoDB

Sebelum aplikasi Anda dapat mengakses basis data, aplikasi harus diautentikasi untuk memastikan bahwa aplikasi diperbolehkan untuk menggunakan basis data. Aplikasi harus diotorisasi sehingga aplikasi hanya dapat melakukan tindakan yang memiliki izin.

Diagram berikut menunjukkan interaksi klien dengan basis data relasional dan dengan Amazon DynamoDB.



Tabel berikut memiliki lebih banyak detail tentang tugas interaksi klien.

Karakteristik	Sistem manajemen basis data relasional (RDBMS)	Amazon DynamoDB
Alat untuk Mengakses Basis Data	Kebanyakan basis data relasional menyediakan antarmuka baris perintah	Dalam sebagian besar kasus, Anda menulis kode aplikasi. Anda juga dapat

Karakteristik	Sistem manajemen basis data relasional (RDBMS)	Amazon DynamoDB
	(CLI) sehingga Anda dapat memasukkan pernyataan SQL ad hoc dan melihat hasilnya segera.	menggunakan Workbench AWS Management Console, the AWS Command Line Interface (AWS CLI), atau NoSQL untuk mengirim permintaan ad hoc ke DynamoDB dan melihat hasilnya. PartiQL, bahasa kueri yang kompatibel dengan SQL, memungkinkan Anda memilih, menyisipkan, memperbarui, dan menghapus data di DynamoDB.
Menghubungkan ke Basis Data	Program aplikasi menetapkan dan memelihara koneksi jaringan dengan basis data. Ketika aplikasi selesai, aplikasi mengakhiri koneksi.	DynamoDB adalah layanan web, dan interaksi dengannya adalah stateless. Aplikasi tidak perlu mempertahankan koneksi jaringan yang persisten. Sebaliknya, interaksi dengan DynamoDB terjadi menggunakan permintaan dan respons HTTP(S).

Karakteristik	Sistem manajemen basis data relasional (RDBMS)	Amazon DynamoDB
Autentikasi	Aplikasi tidak dapat terhubung ke basis data hingga diautentikasi. RDBMS dapat melakukan autentikasi itu sendiri, atau RDBMS dapat mengirimkan tugas ini ke sistem operasi host atau directory service.	Setiap permintaan untuk DynamoDB harus disertai dengan tanda tangan kriptografi, yang mengautentikasi permintaan tertentu. AWS SDKsMenyediakan semua logika yang diperlukan untuk membuat tanda tangan dan permintaan penandatanganan. Untuk informasi selengkapnya, lihat Menandatangani permintaan AWS API di bagian Referensi Umum AWS.
Otorisasi	Aplikasi hanya dapat melakukan tindakan yang telah mendapatkan otorisasi. Administrator basis data atau pemilik aplikasi dapat menggunakan SQL `GRANT` dan `REVOKE` pernyataan untuk mengontrol akses ke objek basis data (misalnya tabel), data (misalnya baris dalam tabel), atau kemampuan untuk menerbitkan pernyataan SQL tertentu.	Di DynamoDB, otorisasi ditangani AWS Identity and Access Management oleh (IAM). Anda dapat menulis kebijakan IAM untuk memberikan izin pada sumber daya DynamoDB (misalnya tabel), dan kemudian megizinkan pengguna dan peran IAM untuk menggunakan kebijakan tersebut. IAM juga dilengkapi kontrol akses terperinci untuk item data individual dalam tabel DynamoDB. Untuk informasi selengkapnya, lihat Manajemen Identitas dan Akses untuk Amazon DynamoDB.

Karakteristik	Sistem manajemen basis data relasional (RDBMS)	Amazon DynamoDB
Mengirim Permintaan	Aplikasi menerbitkan pernyataan SQL untuk setiap operasi basis data yang ingin dilakukan. Sebelum penerimaan pernyataan SQL, RDBMS memeriksa sintaks, membuat rencana untuk melakukan operasi, kemudian menjalankan rencana.	Aplikasi mengirimkan permintaan HTTP(S) untuk DynamoDB. Permintaa n berisi nama operasi DynamoDB untuk dilakukan , bersama dengan parameter . DynamoDB menjalankan permintaan segera.
Menerima Respons	RDBMS mengembalikan hasil dari pernyataan SQL. Jika ada kesalahan, RDBMS mengembalikan status dan pesan kesalahan.	DynamoDB mengembal ikan respons HTTP(S) yang berisi hasil operasi. Jika ada kesalahan, DynamoDB mengembalikan status dan pesan kesalahan HTTP.

Perbedaan antara database relasional (SQL) dan DynamoDB saat membuat tabel

Tabel adalah struktur data mendasar dalam basis data relasional dan di Amazon DynamoDB. Sistem manajemen basis data relasional (RDBMS) mengharuskan Anda untuk menentukan skema tabel ketika Anda membuatnya. Sebaliknya, tabel DynamoDB tidak memiliki skema-selain kunci primer, Anda tidak perlu mendefinisikan atribut tambahan atau jenis data ketika Anda membuat tabel.

Bagian berikut membandingkan bagaimana Anda akan membuat tabel dengan SQL dengan bagaimana Anda akan membuatnya dengan DynamoDB.

Topik

- Membuat tabel dengan SQL
- Membuat tabel dengan DynamoDB

Membuat tabel dengan SQL

Dengan SQL Anda akan menggunakan pernyataan `CREATE TABLE` untuk membuat tabel, seperti yang ditunjukkan pada contoh berikut.

```
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)
);
```

Kunci utama untuk tabel ini terdiri dari Artis dan SongTitle.

Anda harus menentukan semua kolom tabel dan jenis data, serta kunci primer tabel. (Anda dapat menggunakan pernyataan `ALTER TABLE` untuk mengubah definisi ini nanti, jika perlu.)

Banyak implementasi SQL memungkinkan Anda menentukan spesifikasi penyimpanan untuk tabel Anda, sebagai bagian dari pernyataan `CREATE TABLE`. Kecuali jika Anda mengindikasikan sebaliknya, tabel dibuat dengan pengaturan penyimpanan default. Dalam lingkungan produksi, administrator basis data dapat membantu menentukan parameter penyimpanan optimal.

Membuat tabel dengan DynamoDB

Gunakan operasi `CreateTable` untuk membuat tabel mode yang ditetapkan, menentukan parameter seperti yang ditunjukkan berikut:

```
{
    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
    }
}
```

Kunci utama untuk tabel ini terdiri dari Artist (partition key) dan SongTitle(sort key).

Anda harus memberikan parameter berikut untuk `CreateTable`:

- `TableName` — Nama tabel.
- `KeySchema` — Atribut yang digunakan untuk kunci primer. Untuk informasi selengkapnya, lihat Tabel, item, dan atribut dan Kunci primer.
- `AttributeDefinitions` — Jenis data untuk atribut skema kunci.
- `ProvisionedThroughput (for provisioned tables)` — Jumlah baca dan tulis per detik yang Anda butuhkan untuk tabel ini. DynamoDB mencadangkan cukup penyimpanan dan sumber daya sistem sehingga persyaratan throughput Anda selalu dipenuhi. Anda dapat menggunakan operasi `UpdateTable` untuk mengubah ini nanti, jika perlu. Anda tidak perlu menentukan persyaratan penyimpanan tabel karena alokasi penyimpanan terkelola sepenuhnya oleh DynamoDB.

Perbedaan antara mendapatkan informasi tabel dari database relasional (SQL) dan DynamoDB

Anda dapat memverifikasi bahwa tabel telah dibuat sesuai dengan spesifikasi Anda. Dalam basis data relasional, semua skema tabel ditampilkan. Tabel Amazon DynamoDB tidak memiliki skema, sehingga hanya atribut kunci primer yang ditampilkan.

Topik

- Mendapatkan informasi tentang tabel dengan SQL
- Dapatkan informasi tentang tabel DynamoDB

Mendapatkan informasi tentang tabel dengan SQL

Sebagian besar sistem manajemen basis data relasional (RDBMS) memungkinkan Anda untuk menggambarkan struktur tabel - kolom, jenis data, definisi kunci primer, dan sebagainya. Tidak ada cara standar untuk melakukan hal ini di SQL. Namun, banyak sistem basis data menyediakan perintah `DESCRIBE`. Berikut adalah contoh dari MySQL.

```
DESCRIBE Music;
```

Ini mengembalikan struktur tabel Anda, dengan semua nama kolom, jenis data, dan ukuran.

```
+------------+-------------+------+-----+---------+-------+
| 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    |       |
+------------+-------------+------+-----+---------+-------+
```

Kunci utama untuk tabel ini terdiri dari Artis dan SongTitle.

Dapatkan informasi tentang tabel DynamoDB

DynamoDB memiliki operasi, `DescribeTable` yang serupa. Satu-satunya parameter adalah nama tabel.

```
{
    TableName : "Music"
}
```

Balasan dari `DescribeTable` terlihat seperti berikut ini.

```
{
  "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` juga mengembalikan informasi tentang indeks pada tabel, pengaturan throughput yang ditetapkan, jumlah item perkiraan, dan metadata lainnya.

Perbedaan antara database relasional (SQL) dan DynamoDB saat menulis data ke tabel

Tabel basis data relasional berisi barisan data. Baris terdiri dari kolom. Tabel Amazon DynamoDB berisi item. Item terdiri dari atribut.

Bagian ini menjelaskan cara menulis satu baris (atau item) ke tabel.

Topik

- Menulis data ke tabel dengan SQL
- Menulis data ke tabel di DynamoDB

Menulis data ke tabel dengan SQL

Sebuah tabel dalam basis data relasional adalah struktur data dua dimensi yang terdiri dari baris dan kolom. Beberapa sistem manajemen basis data juga menyediakan dukungan untuk data semi terstruktur, biasanya dengan JSON asli atau jenis data XML. Namun, detail implementasi bervariasi antara vendor.

Dalam SQL, Anda akan menggunakan pernyataan `INSERT` untuk menambahkan baris ke tabel.

```
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}'
);
```

Kunci utama untuk tabel ini terdiri dari Artis dan SongTitle. Anda harus menentukan nilai untuk kolom ini.

> **Note**
>
> Contoh ini menggunakan kolom Tanda untuk menyimpan data semi terstruktur tentang lagu di tabel Music. Kolom Tanda didefinisikan sebagai jenis TEXT, yang dapat menyimpan hingga 65.535 karakter di MySQL.

Menulis data ke tabel di DynamoDB

Di Amazon DynamoDB, Anda dapat menggunakan DynamoDB API atau PartiQL (bahasa kueri yang kompatibel dengan SQL) untuk menambahkan item ke tabel.

DynamoDB API

Dengan DynamoDB API, Anda menggunakan operasi `PutItem` untuk menambahkan item ke tabel.

```
{
    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
        }
    }
}
```

Kunci utama untuk tabel ini terdiri dari Artis dan SongTitle. Anda harus menentukan nilai untuk atribut ini.

Berikut adalah beberapa hal penting untuk diketahui tentang contoh `PutItem`:

- DynamoDB menyediakan dukungan asli untuk dokumen, menggunakan JSON. Hal ini membuat DynamoDB ideal untuk menyimpan data semi terstruktur, seperti Tag. Anda juga dapat mengambil dan memanipulasi data dari dalam dokumen JSON.
- Tabel Musik tidak memiliki atribut yang telah ditentukan sebelumnya, selain kunci utama (Artis dan SongTitle).
- Sebagian besar basis data SQL berorientasi transaksi. Ketika Anda menerbitkan pernyataan `INSERT`, modifikasi data tidak permanen sampai Anda menerbitkan pernyataan `COMMIT`. Dengan Amazon DynamoDB, efek operasi `PutItem` bersifat permanen ketika DynamoDB membalas dengan kode status HTTP 200 (OK).

Berikut adalah beberapa contoh `PutItem` lainnya.

```
{
    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**
>
> Selain itu`PutItem`, DynamoDB mendukung operasi `BatchWriteItem` untuk menulis beberapa item pada saat yang sama.

PartiQL for DynamoDB

Dengan PartiQL, Anda menggunakan operasi `ExecuteStatement` untuk menambahkan item ke tabel, menggunakan pernyataan `Insert` PartiQL.

```
INSERT into Music value {
    'Artist': 'No One You Know',
    'SongTitle': 'Call Me Today',
    'AlbumTitle': 'Somewhat Famous',
    'Year' : '2015',
    'Genre' : 'Acme'
}
```

Kunci utama untuk tabel ini terdiri dari Artis dan SongTitle. Anda harus menentukan nilai untuk atribut ini.

> **Note**
>
> Untuk contoh kode menggunakan `Insert` dan `ExecuteStatement`, lihat Pernyataan sisipkan PartiQL untuk DynamoDB.

Perbedaan antara database relasional (SQL) dan DynamoDB saat membaca data dari tabel

Dengan SQL, Anda menggunakan pernyataan `SELECT` untuk mengambil satu baris atau lebih dari tabel. Anda menggunakan klausul `WHERE` untuk menentukan data yang dikembalikan kepada Anda.

Ini berbeda dengan menggunakan Amazon DynamoDB yang menyediakan operasi berikut untuk membaca data:

- `ExecuteStatement` mengambil satu atau beberapa item dari tabel. `BatchExecuteStatement`mengambil beberapa item dari tabel yang berbeda dalam satu operasi. Kedua operasi ini menggunakan PartiQL, bahasa kueri yang kompatibel dengan SQL.
- `GetItem` – Mengambil satu item dari tabel. Ini adalah cara yang paling efisien untuk membaca satu item karena ini menyediakan akses langsung ke lokasi fisik item. (DynamoDB juga menyediakan operasi `BatchGetItem`, memungkinkan Anda untuk melakukan hingga 100 panggilan `GetItem` dalam satu operasi.)
- `Query` – Mengambil semua item yang memiliki kunci partisi tertentu. Dalam item tersebut, Anda dapat menerapkan syarat untuk kunci urutan dan mengambil hanya subset dari data. `Query` menyediakan akses cepat dan efisien ke partisi tempat data disimpan. (Untuk informasi selengkapnya, lihat Partisi dan distribusi data di DynamoDB.)
- `Scan` – Mengambil semua item dalam tabel yang ditentukan. (Operasi ini tidak boleh digunakan dengan tabel besar karena dapat mengkonsumsi sejumlah besar sumber daya sistem.)

> **Note**
>
> Dengan basis data relasional, Anda dapat menggunakan pernyataan `SELECT` untuk menggabungkan data dari beberapa tabel dan mengembalikan hasil. Gabungan sangat penting untuk model relasional. Untuk memastikan bahwa gabungan berjalan efisien, basis data dan aplikasi harus disetel performanya secara berkelanjutan. DynamoDB adalah basis data NoSQL non-relasional yang tidak mendukung gabungan tabel. Sebaliknya, aplikasi membaca data dari satu tabel pada satu waktu.

Bagian berikut menjelaskan kasus penggunaan yang berbeda untuk membaca data, dan cara melakukan tugas-tugas ini dengan basis data relasional dan dengan DynamoDB.

Topik

- Perbedaan dalam membaca item menggunakan kunci utamanya
- Perbedaan dalam menanyakan tabel
- Perbedaan dalam memindai tabel

Perbedaan dalam membaca item menggunakan kunci utamanya

Salah satu pola akses umum untuk basis data adalah untuk membaca satu item dari tabel. Anda harus menentukan kunci primer dari item yang Anda inginkan.

Topik

- Membaca item menggunakan kunci primernya dengan SQL
- Membaca item menggunakan kunci primernya di DynamoDB

Membaca item menggunakan kunci primernya dengan SQL

Dalam SQL, Anda akan menggunakan pernyataan `SELECT` untuk mengambil data dari tabel. Anda dapat meminta satu kolom atau lebih dalam hasil (atau semuanya, jika Anda menggunakan operator `*`). Klausa `WHERE` menentukan baris mana yang dikembalikan.

Berikut ini adalah pernyataan `SELECT` untuk mengambil satu baris dari tabel Music. Klausa `WHERE` menentukan nilai-nilai kunci primer.

```
SELECT *
FROM Music
WHERE Artist='No One You Know' AND SongTitle = 'Call Me Today'
```

Anda dapat memodifikasi kueri ini untuk mengambil hanya subset dari kolom.

```
SELECT AlbumTitle, Year, Price
FROM Music
WHERE Artist='No One You Know' AND SongTitle = 'Call Me Today'
```

Perhatikan bahwa kunci utama untuk tabel ini terdiri dari Artis dan SongTitle.

Membaca item menggunakan kunci primernya di DynamoDB

Di Amazon DynamoDB, Anda dapat menggunakan DynamoDB API atau PartiQL (bahasa kueri yang kompatibel dengan SQL) untuk membaca item dari tabel.

DynamoDB API

Dengan DynamoDB API, Anda menggunakan operasi `PutItem` untuk menambahkan item ke tabel.

DynamoDB menyediakan operasi `GetItem` untuk mengambil item dengan kunci primernya. `GetItem` sangat efisien karena menyediakan akses langsung ke lokasi fisik item. (Untuk informasi selengkapnya, lihat Partisi dan distribusi data di DynamoDB.)

Secara default, `GetItem` mengembalikan seluruh item dengan semua atribut.

```
{
    TableName: "Music",
    Key: {
        "Artist": "No One You Know",
        "SongTitle": "Call Me Today"
    }
}
```

Anda dapat menambahkan parameter `ProjectionExpression` untuk mengembalikan hanya beberapa atribut.

```
{
    TableName: "Music",
    Key: {
        "Artist": "No One You Know",
        "SongTitle": "Call Me Today"
    },
    "ProjectionExpression": "AlbumTitle, Year, Price"
}
```

Perhatikan bahwa kunci utama untuk tabel ini terdiri dari Artis dan SongTitle.

Operasi `GetItem` DynamoDB sangat efisien. Ini menggunakan nilai-nilai kunci primer untuk menentukan lokasi penyimpanan yang tepat dari item yang bersangkutan, dan mengambilnya langsung dari sana. Pernyataan `SELECT` SQL sama efisiennya, dalam hal mengambil item dengan nilai-nilai kunci primer.

Pernyataan `SELECT` mendukung berbagai jenis tabel kueri dan pindai. DynamoDB menyediakan fungsionalitas serupa dengan operasi `Query` dan `Scan`, yang dijelaskan dalam Perbedaan dalam menanyakan tabel dan Perbedaan dalam memindai tabel.

Pernyataan SELECT SQL dapat melakukan gabungan tabel, memungkinkan Anda untuk mengambil data dari beberapa tabel pada waktu yang sama. Gabungan paling efektif di mana tabel basis data dinormalisasi dan hubungan antara tabel jelas. Namun, jika Anda menggabungkan terlalu banyak tabel dalam satu performa aplikasi pernyataan SELECT dapat terpengaruh. Anda dapat mengatasi masalah tersebut dengan menggunakan replikasi basis data, tampilan terwujud, atau penulisan ulang kueri.

DynamoDB adalah basis data non-relasional dan tidak mendukung gabungan tabel. Jika Anda memigrasikan aplikasi yang ada dari basis data relasional ke DynamoDB, Anda perlu mendenormalisasi model data Anda untuk menghapus kebutuhan gabungan.

PartiQL for DynamoDB

Dengan PartiQL, Anda menggunakan operasi `ExecuteStatement` untuk membaca item dari tabel, menggunakan pernyataan `Select` PartiQL.

```
SELECT AlbumTitle, Year, Price
FROM Music
WHERE Artist='No One You Know' AND SongTitle = 'Call Me Today'
```

Perhatikan bahwa kunci utama untuk tabel ini terdiri dari Artis dan SongTitle.

> **Note**
>
> Pernyataan pilih PartiQL dapat digunakan juga pada tabel Kueri atau Pindai DynamoDB

Untuk contoh kode menggunakan `Select` dan `ExecuteStatement`, lihat Pernyataan pemilihan PartiQL untuk DynamoDB.

Perbedaan dalam menanyakan tabel

Pola akses umum lainnya adalah membaca beberapa item dari tabel, berdasarkan kriteria kueri Anda.

Topik

- Mengkueri tabel dengan SQL
- Mengkueri tabel di DynamoDB

Mengkueri tabel dengan SQL

Saat menggunakan SQL, pernyataan `SELECT` memungkinkan Anda melakukan kueri pada kolom kunci, kolom non-kunci, atau kombinasi apa pun. Klausa `WHERE` menentukan baris mana yang dikembalikan, seperti yang ditunjukkan dalam contoh berikut.

```
/* 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, with a particular word in the title...
...but only if the price is less than 1.00 */

SELECT * FROM Music
WHERE Artist='No One You Know' AND SongTitle LIKE '%Today%'
AND Price < 1.00;
```

Perhatikan bahwa kunci utama untuk tabel ini terdiri dari Artis dan SongTitle.

Mengkueri tabel di DynamoDB

Di Amazon DynamoDB, Anda dapat menggunakan DynamoDB API atau PartiQL (bahasa kueri yang kompatibel dengan SQL) untuk menanyakan item dari tabel.

DynamoDB API

> Dengan Amazon DynamoDB, Anda dapat menggunakan operasi `Query` untuk mengambil data dengan cara yang serupa. Operasi `Query` ini menyediakan akses cepat dan efisien ke lokasi fisik di mana data disimpan. Untuk informasi selengkapnya, lihat Partisi dan distribusi data di DynamoDB.

Anda dapat menggunakan `Query` dengan tabel atau indeks sekunder apa pun. Anda harus menentukan syarat kesetaraan untuk nilai kunci partisi, dan secara opsional Anda dapat menyediakan syarat lain untuk atribut kunci urutan jika ditentukan.

Parameter `KeyConditionExpression` menentukan nilai kunci yang ingin Anda kueri. Anda dapat menggunakan opsi `FilterExpression` untuk menghapus item tertentu dari hasil sebelum item dikembalikan kepada Anda.

Di DynamoDB, Anda harus menggunakan `ExpressionAttributeValues` sebagai placeholder dalam parameter ekspresi (seperti `KeyConditionExpression` dan `FilterExpression`). Hal ini analog dengan penggunaan variabel pengikatan dalam basis data relasional, di mana Anda mengganti nilai-nilai aktual ke pernyataan `SELECT` pada saat runtime.

Perhatikan bahwa kunci utama untuk tabel ini terdiri dari Artis dan SongTitle.

Berikut adalah beberapa contoh `Query` DynamoDB.

```
// 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"
    }
}
```

PartiQL for DynamoDB

Dengan PartiQL, Anda dapat melakukan kueri dengan menggunakan operasi `ExecuteStatement` dan pernyataan `Select` pada kunci partisi.

```
SELECT AlbumTitle, Year, Price
FROM Music
WHERE Artist='No One You Know'
```

Menggunakan pernyataan `SELECT` dengan cara ini mengembalikan semua lagu yang terkait dengan `Artist` tertentu.

Untuk contoh kode menggunakan `Select` dan `ExecuteStatement`, lihat Pernyataan pemilihan PartiQL untuk DynamoDB.

Perbedaan dalam memindai tabel

Dalam SQL, pernyataan `SELECT` tanpa sebuah klausul `WHERE` akan mengembalikan setiap baris dalam sebuah tabel. Di Amazon DynamoDB, operasi `Scan` melakukan hal yang sama. Dalam kedua kasus, Anda dapat mengambil semua item atau hanya beberapa item.

Entah Anda menggunakan basis data SQL atau NoSQL, pindaian tidak boleh sering digunakan karena pindaian dapat mengonsumsi sejumlah besar sumber daya sistem. Terkadang pemindaian layak digunakan (seperti memindai tabel kecil) atau tidak dapat dihindari (seperti melakukan ekspor data secara massal). Namun, sebagai aturan umum, Anda harus merancang aplikasi Anda agar tidak melakukan pemindaian. Untuk informasi selengkapnya, lihat Menanyakan tabel di DynamoDB.

> **Note**
>
> Melakukan ekspor massal juga membuat setidaknya 1 file per partisi. Semua item di setiap file berasal dari keyspace hash partisi tertentu.

Topik

- Memindai tabel dengan SQL
- Memindai tabel di DynamoDB

Memindai tabel dengan SQL

Saat menggunakan SQL Anda dapat memindai tabel dan mengambil semua data dengan menggunakan pernyataan `SELECT` tanpa menentukan klausul `WHERE`. Anda dapat meminta satu kolom atau lebih dalam hasil. Atau, Anda dapat meminta semuanya jika Anda menggunakan karakter wildcard (*).

Berikut ini adalah contoh-contoh penggunaan pernyataan `SELECT`.

```
/* 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;
```

Memindai tabel di DynamoDB

Di Amazon DynamoDB, Anda dapat menggunakan DynamoDB API atau PartiQL (bahasa kueri yang kompatibel dengan SQL) untuk melakukan pemindaian pada tabel.

DynamoDB API

Dengan DynamoDB API, Anda menggunakan operasi `Scan` untuk mengembalikan satu atribut item dan item dengan mengakses setiap item dalam tabel atau indeks sekunder.

```
// Return all of the data in the table
{
    TableName:  "Music"
}
```

```
// Return all of the values for Artist and Title
{
    TableName:  "Music",
    ProjectionExpression: "Artist, Title"
}
```

Operasi `Scan` ini juga menyediakan parameter `FilterExpression`, yang dapat Anda gunakan untuk membuang item yang tidak ingin Anda tampilkan di hasil. Sebuah `FilterExpression` diterapkan setelah pemindaian dilakukan, namun sebelum hasilnya dikembalikan kepada Anda. (Ini tidak dianjurkan dengan tabel besar. Anda masih dikenakan biaya untuk seluruh `Scan`, bahkan jika hanya beberapa item yang cocok dikembalikan.)

PartiQL for DynamoDB

Dengan PartiQL, Anda melakukan pemindaian dengan menggunakan operasi `ExecuteStatement` untuk mengembalikan semua konten untuk tabel menggunakan pernyataan `Select`.

```
SELECT AlbumTitle, Year, Price
FROM Music
```

Ingat bahwa pernyataan ini akan mengembalikan semua item untuk dalam tabel Music.

Untuk contoh kode menggunakan `Select` dan `ExecuteStatement`, lihat Pernyataan pemilihan PartiQL untuk DynamoDB.

Perbedaan antara database relasional (SQL) dan DynamoDB saat mengelola indeks

Indeks memberikan Anda akses ke pola kueri alternatif, dan dapat mempercepat kueri. Bagian ini membandingkan dan mengontraskan pembuatan indeks dan penggunaan dalam SQL dan Amazon DynamoDB.

Entah Anda menggunakan basis data relasional atau DynamoDB, Anda harus bijaksana dengan pembuatan indeks. Setiap kali sebuah peristiwa tulis terjadi pada tabel, semua indeks tabel harus diperbarui. Dalam lingkungan yang banyak menulis dengan tabel besar, hal ini dapat menghabiskan banyak sumber daya sistem. Dalam lingkungan read-only atau read-mostly, hal ini tidak terlalu menjadi perhatian. Namun, Anda harus memastikan bahwa indeks tersebut benar-benar digunakan oleh aplikasi Anda, dan tidak hanya menghabiskan ruang.

Topik

- Perbedaan antara database relasional (SQL) dan DynamoDB saat membuat indeks
- Perbedaan antara database relasional (SQL) dan DynamoDB saat menanyakan dan memindai indeks

Perbedaan antara database relasional (SQL) dan DynamoDB saat membuat indeks

Bandingkan pernyataan `CREATE INDEX` di SQL dengan operasi `UpdateTable` di Amazon DynamoDB.

Topik

- Membuat indeks dengan SQL
- Membuat indeks di DynamoDB

Membuat indeks dengan SQL

Dalam basis data relasional, indeks adalah struktur data yang memungkinkan Anda melakukan kueri cepat pada kolom yang berbeda dalam sebuah tabel. Anda dapat menggunakan pernyataan SQL `CREATE INDEX` untuk menambahkan indeks ke tabel yang ada, menentukan kolom yang akan diindeks. Setelah indeks dibuat, Anda dapat melakukan kueri data di tabel seperti biasa, tetapi sekarang basis data dapat menggunakan indeks untuk menemukan baris yang ditentukan dengan cepat di tabel alih-alih memindai keseluruhan tabel.

Setelah Anda membuat indeks, basis data mengelolanya untuk Anda. Setiap kali Anda memodifikasi data dalam tabel, indeks secara otomatis dimodifikasi untuk mencerminkan perubahan dalam tabel.

Dalam MySQL, Anda akan membuat indeks seperti berikut.

```
CREATE INDEX GenreAndPriceIndex
ON Music (genre, price);
```

Membuat indeks di DynamoDB

Dalam DynamoDB, Anda dapat membuat dan menggunakan indeks sekunder untuk tujuan yang serupa.

Indeks di DynamoDB berbeda dari rekan-rekan relasional mereka. Ketika Anda membuat indeks sekunder, Anda harus menentukan atribut kunci—kunci partisi dan kunci urutan. Setelah membuat indeks sekunder, Anda dapat melakukan `Query` atau `Scan` padanya seperti yang Anda lakukan dengan tabel. DynamoDB tidak memiliki pengoptimal kueri, sehingga indeks sekunder hanya digunakan ketika Anda melakukan `Query` atau `Scan` padanya.

DynamoDB mendukung dua jenis indeks yang berbeda:

- Indeks sekunder global - Kunci primer dari indeks dapat berupa setiap dua atribut dari tabel.
- Indeks sekunder lokal - Kunci partisi indeks harus sama dengan kunci partisi dari tabel. Namun, kunci urutan dapat berupa atribut lainnya.

DynamoDB memastikan bahwa data dalam indeks sekunder akhirnya konsisten dengan tabel. Anda dapat meminta operasi `Query` atau `Scan` yang sangat konsisten pada tabel atau indeks sekunder lokal. Namun, indeks sekunder global hanya mendukung konsistensi akhir.

Anda dapat menambahkan indeks sekunder global ke tabel yang ada, menggunakan operasi `UpdateTable` dan menentukan `GlobalSecondaryIndexUpdates`.

```
{
    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
                }
            }
        }
    ]
}
```

Anda harus memberikan parameter berikut untuk `UpdateTable`:

- `TableName` – Tabel yang akan dikaitkan dengan indeks.
- `AttributeDefinitions` – Jenis data untuk atribut skema kunci dari indeks.

- `GlobalSecondaryIndexUpdates` – Detail tentang indeks yang ingin Anda buat:
  - `IndexName` – Nama untuk indeks.
  - `KeySchema` – Atribut yang digunakan untuk kunci primer indeks.
  - `Projection` – Atribut dari tabel yang disalin ke indeks. Dalam kasus ini, `ALL` berarti bahwa semua atribut disalin.
  - `ProvisionedThroughput (for provisioned tables)` – Jumlah baca dan tulis per detik yang Anda butuhkan untuk tabel ini. (Ini terpisah dari pengaturan throughput yang ditetapkan pada tabel.)

Bagian dari operasi ini melibatkan pengisian data dari tabel ke indeks baru. Selama pengisian, tabel tetap tersedia. Namun, indeks tidak siap hingga atribut `Backfilling` miliknya berubah dari true ke false. Anda dapat menggunakan operasi `DescribeTable` untuk melihat atribut ini.

Perbedaan antara database relasional (SQL) dan DynamoDB saat menanyakan dan memindai indeks

Bandingkan mengkueri dan memindai indeks menggunakan pernyataan SELECT dalam SQL dengan operasi `Query` dan `Scan` di Amazon DynamoDB.

Topik

- Mengkueri dan memindai indeks dengan SQL
- Mengkueri dan memindai indeks di DynamoDB

Mengkueri dan memindai indeks dengan SQL

Dalam basis data relasional, Anda tidak bekerja secara langsung dengan indeks. Sebaliknya, Anda membuat kueri tabel dengan menerbitkan pernyataan `SELECT`, dan pengoptimal kueri dapat menggunakan setiap indeks.

Pengoptimal kueri adalah komponen sistem manajemen basis data relasional (RDBMS) yang mengevaluasi indeks yang tersedia dan menentukan apakah komponen dapat digunakan untuk mempercepat kueri. Jika indeks dapat digunakan untuk mempercepat kueri, RDBMS mengakses indeks pertama kemudian menggunakannya untuk menemukan data dalam tabel.

Berikut adalah beberapa pernyataan SQL yang dapat digunakan GenreAndPriceIndexuntuk meningkatkan kinerja. Kami berasumsi bahwa tabel Music memiliki cukup data di dalamnya hingga pengoptimal kueri memutuskan untuk menggunakan indeks ini, alih-alih memindai seluruh tabel.

```
/* 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;
```

Mengkueri dan memindai indeks di DynamoDB

Dalam DynamoDB, Anda melakukan operasi `Query` dan `Scan` secara langsung pada indeks, dengan cara yang sama seperti yang Anda lakukan pada tabel. Anda dapat menggunakan DynamoDB API atau PartiQL (bahasa kueri yang kompatibel dengan SQL) untuk mengkueri atau memindai indeks. Anda harus menentukan baik `TableName` dan `IndexName`.

Berikut ini adalah beberapa pertanyaan GenreAndPriceIndexdi DynamoDB. (Skema kunci untuk indeks ini terdiri dari Genre dan Price.)

DynamoDB API

```
// All of the rock songs

{
    TableName: "Music",
    IndexName: "GenreAndPriceIndex",
    KeyConditionExpression: "Genre = :genre",
    ExpressionAttributeValues: {
        ":genre": "Rock"
    },
};
```

Contoh ini menggunakan `ProjectionExpression` untuk menunjukkan bahwa Anda hanya ingin beberapa atribut, bukan semuanya, untuk muncul dalam hasil.

```
// 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"
};
```

Berikut ini adalah pemindaian GenreAndPriceIndex.

```
// Return all of the data in the index

{
    TableName:  "Music",
    IndexName: "GenreAndPriceIndex"
}
```

PartiQL for DynamoDB

Dengan PartiQL, Anda menggunakan pernyataan PartiQL `Select` untuk melakukan kueri dan pemindaian pada indeks.

```
// 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
```

Berikut ini adalah pemindaian GenreAndPriceIndex.

```
// Return all of the data in the index

SELECT *
FROM Music.GenreAndPriceIndex
```

> **Note**
>
> Untuk contoh kode menggunakan `Select`, lihat Pernyataan pemilihan PartiQL untuk DynamoDB.

Perbedaan antara database relasional (SQL) dan DynamoDB saat memodifikasi data dalam tabel

Bahasa SQL menyediakan pernyataan `UPDATE` untuk memodifikasi data. Amazon DynamoDB menggunakan operasi `UpdateItem` untuk menyelesaikan tugas yang serupa.

Topik

- Memodifikasi data dalam tabel dengan SQL
- Memodifikasi data dalam tabel di DynamoDB

Memodifikasi data dalam tabel dengan SQL

Dalam SQL, Anda akan menggunakan pernyataan `UPDATE` untuk memodifikasi satu baris atau lebih. Klausa `SET` menentukan nilai-nilai baru untuk satu kolom atau lebih, dan klausul `WHERE` menentukan baris yang dimodifikasi. Berikut adalah contohnya.

```
UPDATE Music
SET RecordLabel = 'Global Records'
WHERE Artist = 'No One You Know' AND SongTitle = 'Call Me Today';
```

Jika tidak ada baris yang cocok dengan klausul `WHERE`, pernyataan `UPDATE` tidak berpengaruh.

Memodifikasi data dalam tabel di DynamoDB

Di DynamoDB, Anda dapat menggunakan DynamoDB API atau PartiQL (bahasa kueri yang kompatibel dengan SQL) untuk modifikasi satu item. Jika Anda ingin memodifikasi beberapa item, Anda harus menggunakan beberapa operasi.

DynamoDB API

Dengan DynamoDB API, Anda menggunakan operasi `UpdateItem` untuk memodifikasi satu item.

```
{
    TableName: "Music",
    Key: {
        "Artist":"No One You Know",
        "SongTitle":"Call Me Today"
    },
    UpdateExpression: "SET RecordLabel = :label",
    ExpressionAttributeValues: {
        ":label": "Global Records"
    }
}
```

Anda harus menentukan atribut `Key` item yang akan dimodifikasi dan `UpdateExpression` untuk menentukan nilai atribut. `UpdateItem` berperilaku seperti operasi "upsert". Item diperbarui jika ada di tabel, tetapi jika tidak, item baru ditambahkan (disisipkan).

`UpdateItem` mendukung tulis bersyarat, di mana operasi hanya berhasil jika `ConditionExpression` tertentu bernilai true. Misalnya, operasi `UpdateItem` berikut tidak melakukan pembaruan kecuali jika harga lagu lebih besar dari atau sama dengan 2,00.

```
{
    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` juga mendukung penghitung atom, atau atribut jenis `Number` yang dapat ditambahkan atau dikurangi. Penghitung atom serupa dalam banyak hal dengan generator urutan, kolom identitas, atau bidang penambahan otomatis dalam basis data SQL.

Berikut ini adalah contoh operasi `UpdateItem` untuk menginisialisasi atribut baru (Play) untuk melacak berapa kali lagu telah diputar.

```
{
    TableName: "Music",
    Key: {
        "Artist":"No One You Know",
        "SongTitle":"Call Me Today"
    },
    UpdateExpression: "SET Plays = :val",
    ExpressionAttributeValues: {
        ":val": 0
    },
    ReturnValues: "UPDATED_NEW"
}
```

Parameter `ReturnValues` diatur ke `UPDATED_NEW`, yang mengembalikan nilai-nilai baru dari setiap atribut yang diperbarui. Dalam hal ini, parameter mengembalikan 0 (nol).

Setiap kali seseorang memutar lagu ini, kita dapat menggunakan operasi `UpdateItem` berikut untuk menambahkan Play sebanyak satu.

```
{
    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

Dengan PartiQL, Anda menggunakan operasi `ExecuteStatement` untuk memodifikasi item pada tabel, menggunakan pernyataan `Update` PartiQL.

Kunci utama untuk tabel ini terdiri dari Artis dan SongTitle. Anda harus menentukan nilai untuk atribut ini.

```
UPDATE Music
SET RecordLabel ='Global Records'
```

```
WHERE Artist='No One You Know' AND SongTitle='Call Me Today'
```

Anda juga dapat memodifikasi beberapa bidang sekaligus, seperti pada contoh berikut.

```
UPDATE Music
SET RecordLabel = 'Global Records'
SET AwardsWon = 10
WHERE Artist ='No One You Know' AND SongTitle='Call Me Today'
```

`Update` juga mendukung penghitung atom, atau atribut jenis `Number` yang dapat ditambahkan atau dikurangi. Penghitung atom serupa dalam banyak hal dengan generator urutan, kolom identitas, atau bidang penambahan otomatis dalam basis data SQL.

Berikut ini adalah contoh pernyataan `Update` untuk menginisialisasi atribut baru (Play) untuk melacak berapa kali lagu telah diputar.

```
UPDATE Music
SET Plays = 0
WHERE Artist='No One You Know' AND SongTitle='Call Me Today'
```

Setiap kali seseorang memutar lagu ini, kita dapat menggunakan pernyataan `Update` berikut untuk menambahkan Play sebanyak satu.

```
UPDATE Music
SET Plays = Plays + 1
WHERE Artist='No One You Know' AND SongTitle='Call Me Today'
```

> **Note**
>
> Untuk contoh kode menggunakan `Update` dan `ExecuteStatement`, lihat Pernyataan pembaruan PartiQL untuk DynamoDB.

Perbedaan antara database relasional (SQL) dan DynamoDB saat menghapus data dari tabel

Di SQL, pernyataan `DELETE` menghapus satu baris atau lebih dari tabel. Amazon DynamoDB menggunakan operasi `DeleteItem` untuk menghapus satu item pada satu waktu.

Topik

- Menghapus data dari tabel dengan SQL
- Menghapus data dari tabel di DynamoDB

Menghapus data dari tabel dengan SQL

Dalam SQL, Anda menggunakan pernyataan `DELETE` untuk menghapus satu baris atau lebih. Klausa `WHERE` menentukan baris yang ingin Anda modifikasi. Berikut adalah contohnya.

```
DELETE FROM Music
WHERE Artist = 'The Acme Band' AND SongTitle = 'Look Out, World';
```

Anda dapat memodifikasi klausul `WHERE` untuk menghapus beberapa baris. Misalnya, Anda dapat menghapus semua lagu dari artis tertentu, seperti yang ditunjukkan dalam contoh berikut.

```
DELETE FROM Music WHERE Artist = 'The Acme Band'
```

Menghapus data dari tabel di DynamoDB

Di DynamoDB, Anda dapat menggunakan DynamoDB API atau PartiQL (bahasa kueri yang kompatibel dengan SQL) untuk menghapus satu item. Jika Anda ingin memodifikasi beberapa item, Anda harus menggunakan beberapa operasi.

DynamoDB API

Dengan DynamoDB API, Anda menggunakan operasi `DeleteItem` untuk menghapus data dari tabel, satu item pada satu waktu. Anda harus menentukan nilai kunci primer item.

```
{
    TableName: "Music",
    Key: {
        Artist: "The Acme Band",
        SongTitle: "Look Out, World"
    }
}
```

> **Note**
>
> Selain ituDeleteItem, Amazon DynamoDB mendukung operasi BatchWriteItem untuk menghapus beberapa item pada saat yang sama.

DeleteItem mendukung tulis bersyarat, di mana operasi hanya berhasil jika ConditionExpression tertentu bernilai true. Misalnya, DeleteItem operasi berikut menghapus item hanya jika memiliki RecordLabelatribut.

```
{
    TableName: "Music",
    Key: {
        Artist: "The Acme Band",
        SongTitle: "Look Out, World"
    },
   ConditionExpression: "attribute_exists(RecordLabel)"
}
```

PartiQL for DynamoDB

Dengan PartiQL, Anda menggunakan pernyataan Delete melalui operasi ExecuteStatement untuk menghapus data dari tabel, satu item pada satu waktu. Anda harus menentukan nilai kunci primer item.

Kunci utama untuk tabel ini terdiri dari Artis dan SongTitle. Anda harus menentukan nilai untuk atribut ini.

```
DELETE FROM Music
WHERE Artist = 'Acme Band' AND SongTitle = 'PartiQL Rocks'
```

Anda juga dapat menentukan kondisi tambahan untuk operasi. Operasi DELETE berikut hanya menghapus item jika memiliki lebih dari 11 Penghargaan.

```
DELETE FROM Music
WHERE Artist = 'Acme Band' AND SongTitle = 'PartiQL Rocks' AND Awards > 11
```

> **Note**
>
> Untuk contoh kode menggunakan `DELETE` dan `ExecuteStatement`, lihat Pernyataan hapus PartiQL untuk DynamoDB.

Perbedaan antara database relasional (SQL) dan DynamoDB saat menghapus tabel

Dalam SQL, Anda menggunakan pernyataan `DROP TABLE` untuk menghapus tabel. Di Amazon DynamoDB, Anda menggunakan operasi `DeleteTable`.

Topik

- Menghapus tabel dengan SQL
- Menghapus tabel di DynamoDB

Menghapus tabel dengan SQL

Ketika Anda tidak lagi membutuhkan tabel dan ingin membuangnya secara permanen, Anda akan menggunakan `DROP TABLE` pernyataan dalam SQL.

```
DROP TABLE Music;
```

Setelah tabel dilepaskan, tabel tidak dapat dipulihkan. (Beberapa basis data relasional memungkinkan Anda untuk membatalkan operasi `DROP TABLE`, tetapi ini adalah fungsionalitas khusus vendor dan tidak diterapkan secara luas.)

Menghapus tabel di DynamoDB

Pada DynamoDB, `DeleteTable` adalah operasi yang serupa. Pada contoh berikut, tabel dihapus secara permanen.

```
{
    TableName: "Music"
}
```

Sumber daya dan alat pembelajaran Amazon DynamoDB

Anda dapat menggunakan sumber daya tambahan berikut untuk memahami dan menggunakan DynamoDB.

Topik

- Alat untuk pengodean dan visualisasi
- Artikel Panduan Preskriptif
- Artikel Pusat Pengetahuan
- Posting blog, repositori, dan panduan
- Pemodelan data dan presentasi pola desain
- Kursus pelatihan

Alat untuk pengodean dan visualisasi

Anda dapat menggunakan alat pengodean dan visualisasi berikut ketika menggunakan DynamoDB:

- NoSQL Workbench untuk Amazon DynamoDB — Alat visual terpadu yang membantu Anda merancang, membuat, mengirim kueri, dan mengelola tabel DynamoDB. Alat ini menyediakan fitur pemodelan data, visualisasi data, dan pengembangan kueri.
- Dynobase – Alat desktop yang memudahkan Anda melihat tabel DynamoDB Anda dan menggunakannya, membuat kode aplikasi, dan mengedit catatan dengan validasi waktu nyata.
- DynamoDB Toolbox — Sebuah proyek dari Jeremy Daly yang menyediakan utilitas bermanfaat untuk bekerja dengan pemodelan data dan Node.js. andJavaScript
- DynamoDB Streams Processor – Alat sederhana yang dapat Anda gunakan untuk bekerja dengan aliran DynamoDB.

Artikel Panduan Preskriptif

AWS Panduan Preskriptif menyediakan strategi, panduan, dan pola yang telah teruji waktu untuk membantu mempercepat proyek Anda. Sumber daya ini dikembangkan oleh pakar AWS teknologi dan komunitas AWS Mitra global, berdasarkan pengalaman bertahun-tahun mereka membantu pelanggan mencapai tujuan bisnis mereka.

Pemodelan dan migrasi data

- Model data hierarkis di DynamoDB
- Pemodelan data dengan DynamoDB
- Migrasi database Oracle ke DynamoDB menggunakan AWS DMS

Tabel global

- Menggunakan tabel global Amazon DynamoDB

Nirserver

- Menerapkan pola saga tanpa server dengan AWS Step Functions

Arsitektur SaaS

- Mengelola penghuni di beberapa produk SaaS pada satu bidang kontrol
- Orientasi penghuni dalam arsitektur SaaS untuk model silo menggunakan C# dan AWS CDK

Perlindungan data dan pergerakan data

- Mengonfigurasikan akses lintas akun ke Amazon DynamoDB
- Opsi salinan tabel lengkap untuk DynamoDB
- Strategi pemulihan bencana untuk basis data di AWS

Lain-lain

- Membantu menegakkan penandaan di DynamoDB

Video panduan preskriptif

- Menggunakan Arsitektur Nirserver untuk Membuat Alur Data
- Novartis - Buying Engine: Portal Pengadaan Berkemampuan AI
- Veritiv: Aktifkan Wawasan untuk Memprediksi Permintaan Penjualan di Data Lakes AWS
- Mimik: Hybrid Edge Cloud Memanfaatkan untuk AWS Mendukung Edge Microservice Mesh
- Tangkapan data perubahan dengan Amazon DynamoDB

Artikel dan video Panduan Preskriptif lainnya untuk DynamoDB bisa dilihat di Panduan Preskriptif.

Artikel Pusat Pengetahuan

Artikel dan video Pusat AWS Pengetahuan mencakup pertanyaan dan permintaan yang paling sering kami terima dari AWS pelanggan. Berikut ini adalah beberapa artikel Pusat Pengetahuan saat ini tentang tugas-tugas tertentu yang berhubungan dengan DynamoDB:

Optimasi biaya

- Bagaimana cara mengoptimalkan biaya dengan Amazon DynamoDB?

Throttling dan latensi

- Mengapa metrik latensi maksimum DynamoDB saya tinggi ketika latensi rata-rata normal?
- Mengapa tabel DynamoDB saya dikenakan throttling?
- Mengapa tabel DynamoDB sesuai permintaan saya dikenakan throttling?

Paginasi

- Bagaimana cara menerapkan paginasi di DynamoDB

Transaksi

- Mengapa panggilan API `TransactWriteItems` saya gagal di DynamoDB

Pemecahan Masalah

- Bagaimana cara mengatasi masalah penskalaan otomatis DynamoDB?
- Bagaimana cara memecahkan masalah kesalahan HTTP 4XX di DynamoDB

Artikel dan video lainnya untuk DynamoDB bisa dilihat di artikel Pusat Pengetahuan.

Posting blog, repositori, dan panduan

Selain Panduan Developer DynamoDB, ada banyak sumber daya yang berguna terkait bekerja dengan DynamoDB. Berikut adalah beberapa posting blog, repositori, dan panduan pilihan untuk bekerja dengan DynamoDB:

- AWS repositori contoh kode DynamoDB dalam berbagai bahasa AWS SDK: Node.js, Java, Python, .Net,Go, dan Rust.
- Buku DynamoDB — Panduan komprehensif dari DeBrie Alex yang mengajarkan pendekatan berbasis strategi untuk pemodelan data dengan DynamoDB.
- Panduan DynamoDB — Panduan terbuka dari DeBrie Alex yang berjalan melalui konsep dasar dan fitur-fitur canggih dari database DynamoDB NoSQL.
- How to switch from RDBMS to DynamoDB in 20 easy steps – Daftar langkah-langkah yang berguna untuk mempelajari pemodelan data dari Jeremy Daly.
- Lembar contekan JavaScript DocumentClient DynamoDB - Lembar contekan untuk membantu Anda mulai membangun aplikasi dengan DynamoDB di Node.js atau lingkungan. JavaScript
- DynamoDB Core Concept Videos – Daftar putar ini membahas banyak konsep inti DynamoDB.

Pemodelan data dan presentasi pola desain

Anda dapat menggunakan sumber daya berikut tentang pemodelan data dan pola desain untuk membantu Anda mendapatkan hasil maksimal dari DynamoDB:

- AWS Re:invent 2019: Pemodelan data dengan DynamoDB
  - Pembicaraan oleh Alex DeBrie yang membantu Anda memulai dengan prinsip-prinsip pemodelan data DynamoDB.
- AWS RE: Invent 2020: Pemodelan data dengan DynamoDB - Bagian 1
- AWS RE: Invent 2020: Pemodelan data dengan DynamoDB - Bagian 2
- AWS re: invent 2017: pola desain canggih
- AWS re: invent 2018: pola desain canggih
- AWS re: invent 2019: pola desain canggih
  - Jeremy Daly membagikan 12 poin penting lewat sesi ini.
- AWS re:Invent 2020: DynamoDB advanced design patterns – Part 1
- AWS RE: invent 2020: pola desain lanjutan DynamoDB - Bagian 2

- DynamoDB Office Hours di Twitch

> **Note**
>
> Setiap sesi membahas contoh dan kasus penggunaan yang berbeda.

Kursus pelatihan

Ada banyak opsi kursus pelatihan dan edukasi yang berbeda untuk mempelajari DynamoDB lebih lanjut. Berikut ini contoh yang terbaru:

- Berkembang dengan Amazon DynamoDB - Dirancang AWS oleh untuk membawa Anda dari pemula hingga ahli dalam mengembangkan aplikasi dunia nyata dengan pemodelan data untuk Amazon DynamoDB.
- DynamoDB deep dive course — Kursus dari Pluralsight.
- Amazon DynamoDB: Membangun aplikasi berbasis database NoSQL — Kursus dari tim Pelatihan dan Sertifikasi yang diselenggarakan di edX. AWS

DynamoDB membaca dan menulis

DynamoDB membaca dan menulis mengacu pada operasi yang mengambil data dari tabel (membaca) dan menyisipkan, memperbarui, atau menghapus data dalam tabel (menulis). Ketika Anda bekerja dengan DynamoDB, penting untuk memahami konsep membaca dan menulis, karena mereka secara langsung memengaruhi kinerja dan biaya aplikasi Anda.

Topik ini memberikan rincian tentang berbagai jenis konsistensi baca yang berlaku untuk DynamoDB. Topik ini juga menjelaskan konsumsi unit untuk berbagai operasi baca dan tulis yang mungkin Anda lakukan.

Topik

- DynamoDB membaca konsistensi
- Operasi baca dan tulis DynamoDB

DynamoDB membaca konsistensi

Amazon DynamoDB membaca data dari tabel, indeks sekunder lokal (), indeks GSIs sekunder global LSIs (), dan aliran. Untuk informasi selengkapnya, lihat Komponen inti dari Amazon DynamoDB. Kedua tabel dan LSIs menyediakan dua opsi konsistensi baca: akhirnya konsisten (default) dan pembacaan yang sangat konsisten. Semua pembacaan dari GSIs dan aliran pada akhirnya konsisten.

Saat aplikasi Anda menulis data ke tabel DynamoDB dan menerima respons HTTP 200 (OK), itu berarti penulisan berhasil diselesaikan dan bertahan lama. DynamoDB menyediakan isolasi komitmen baca dan memastikan bahwa operasi baca selalu mengembalikan nilai komitmen untuk suatu item. Pembacaan tidak akan pernah menampilkan tampilan item dari penulisan yang pada akhirnya tidak berhasil. Isolasi komitmen baca tidak mencegah modifikasi item segera setelah operasi baca.

Akhirnya konsisten membaca

Akhir konsisten adalah model konsisten baca default untuk semua operasi baca. Saat mengeluarkan pembacaan yang konsisten pada tabel atau indeks DynamoDB, responsnya mungkin tidak mencerminkan hasil operasi penulisan yang baru saja diselesaikan. Jika Anda mengulangi permintaan baca setelah beberapa saat, responsnya pada akhirnya akan mengembalikan item yang lebih baru. Bacaan akhir konsisten didukung pada tabel, indeks sekunder lokal, dan indeks

sekunder global. Perhatikan juga bahwa semua pembacaan dari aliran DynamoDB pada akhirnya juga konsisten.

Bacaan akhir konsisten hanya menghabiskan separuh biaya dari bacaan sangat konsisten. Untuk informasi selengkapnya, lihat harga Amazon DynamoDB.

Bacaan yang sangat konsisten

Baca operasi seperti `GetItem`, `Query`, dan `Scan` menyediakan parameter `ConsistentRead` opsional. Jika Anda menyetel `ConsistentRead` ke true, DynamoDB mengembalikan respons dengan data up-to-date terbanyak, yang mencerminkan pembaruan dari semua operasi penulisan sebelumnya yang berhasil. Pembacaan yang sangat konsisten hanya didukung pada tabel dan indeks sekunder lokal. Pembacaan yang sangat konsisten dari indeks sekunder global atau aliran DynamoDB tidak didukung.

Tabel global membaca konsistensi

DynamoDB juga mendukung tabel global untuk replikasi multi-aktif dan Multi-wilayah. Tabel global terdiri dari beberapa tabel replika di AWS Wilayah yang berbeda. Setiap perubahan yang dilakukan pada item apa pun di tabel replika mana pun akan direplikasi ke semua replika lain dalam tabel global yang sama, biasanya dalam satu detik, dan pada akhirnya konsisten. Untuk informasi selengkapnya, lihat Konsistensi dan resolusi konflik.

Operasi baca dan tulis DynamoDB

Operasi baca DynamoDB memungkinkan Anda untuk mengambil satu atau beberapa item dari tabel dengan menentukan nilai kunci partisi dan, secara opsional, nilai kunci pengurutan. Menggunakan operasi penulisan DynamoDB, Anda dapat menyisipkan, memperbarui, atau menghapus item dalam tabel. Topik ini menjelaskan konsumsi unit kapasitas untuk dua operasi ini.

Topik

- Konsumsi unit kapasitas untuk operasi baca
- Konsumsi unit kapasitas untuk operasi tulis

Konsumsi unit kapasitas untuk operasi baca

Permintaan baca DynamoDB bisa sangat konsisten, pada akhirnya konsisten, atau bersifat transaksional.

- Permintaan baca yang sangat konsisten dari item hingga 4 KB memerlukan satu unit baca.
- Permintaan baca yang akhirnya konsisten dari item hingga 4 KB membutuhkan setengah unit baca.
- Permintaan baca transaksional dari item hingga 4 KB membutuhkan dua unit baca.

Untuk mempelajari selengkapnya tentang model konsistensi baca DynamoDB, lihat DynamoDB membaca konsistensi.

Ukuran item untuk pembacaan dibulatkan ke kelipatan 4 KB berikutnya. Misalnya, membaca item berukuran 3.500 byte menggunakan throughput yang sama dengan membaca item berukuran 4 KB.

Jika Anda perlu membaca item yang lebih besar dari 4 KB, DynamoDB membutuhkan unit baca tambahan. Jumlah total unit baca yang diperlukan tergantung pada ukuran item, dan apakah Anda menginginkan pembacaan yang konsisten atau sangat konsisten. Misalnya, jika ukuran item Anda adalah 8 KB, Anda memerlukan 2 unit baca untuk mempertahankan satu pembacaan yang sangat konsisten. Anda akan memerlukan 1 unit baca jika Anda memilih pembacaan yang konsisten atau 4 unit baca untuk permintaan baca transaksional.

Daftar berikut menjelaskan bagaimana operasi baca DynamoDB menggunakan unit baca:

- GetItem: Membaca satu item dari tabel. Untuk menentukan jumlah unit baca yang `GetItem` akan dikonsumsi, ambil ukuran item dan bulatkan ke batas 4 KB berikutnya. Ini adalah jumlah unit baca yang diperlukan jika Anda menentukan pembacaan yang sangat konsisten. Untuk pembacaan yang akhirnya konsisten, yang merupakan default, bagi angka ini dengan dua.

  Misalnya, jika Anda membaca item berukuran 3,5 KB, DynamoDB membulatkan ukuran item menjadi 4 KB. Jika Anda membaca item berukuran 10 KB, DynamoDB membulatkan ukuran item menjadi 12 KB.
- BatchGetItem: Membaca hingga 100 item dari satu atau lebih tabel. DynamoDB memproses setiap item dalam batch sebagai permintaan individual. `GetItem` DynamoDB pertama-tama membulatkan ukuran setiap item ke batas 4 KB berikutnya dan kemudian menghitung ukuran totalnya. Hasilnya belum tentu sama dengan ukuran total semua item. Misalnya, jika `BatchGetItem` membaca dua item ukuran 1,5 KB dan 6,5 KB, DynamoDB menghitung ukurannya sebagai 12 KB (4 KB + 8 KB). DynamoDB tidak menghitung ukuran sebagai 8 KB (1,5 KB +6,5 KB).
- Query: Membaca beberapa item yang memiliki nilai kunci partisi yang sama. Semua item yang dikembalikan diperlakukan sebagai operasi baca tunggal, di mana DynamoDB menghitung ukuran total semua item. DynamoDB kemudian membulatkan ukuran ke batas 4 KB berikutnya. Misalnya,

kueri Anda mengembalikan 10 item dengan ukuran gabungan 40,8 KB. DynamoDB membulatkan ukuran item untuk operasi menjadi 44 KB. Jika kueri mengembalikan 1500 item masing-masing 64 byte, ukuran kumulatifnya adalah 96 KB.

- Scan: Membaca semua item dalam tabel. DynamoDB mempertimbangkan ukuran item yang dievaluasi, bukan ukuran item yang dikembalikan melalui pemindaian. Untuk informasi selengkapnya tentang operasi Pemindaian, lihatMemindai tabel di DynamoDB.

> **Important**
>
> Jika Anda melakukan operasi baca pada item yang tidak ada, DynamoDB akan tetap menggunakan throughput baca seperti yang diuraikan di atas. Untuk `Scan` operasi`Query`/, Anda masih akan dikenakan biaya throughput baca tambahan berdasarkan konsistensi baca dan jumlah partisi yang dicari untuk melayani permintaan, bahkan jika tidak ada data.

Untuk setiap operasi yang mengembalikan item, Anda dapat meminta subset atribut untuk diambil. Namun, hal ini tidak berdampak pada penghitungan ukuran item. Selain itu, `Query` dan `Scan` dapat mengembalikan jumlah item, bukan nilai atribut. Mendapatkan jumlah item menggunakan jumlah unit baca yang sama dan tunduk pada perhitungan ukuran item yang sama. Hal ini karena DynamoDB harus membaca setiap item untuk menambah jumlah.

Konsumsi unit kapasitas untuk operasi tulis

Satu unit tulis mewakili satu tulis untuk item berukuran hingga 1 KB. Jika Anda perlu menulis item yang lebih besar dari 1 KB, DynamoDB perlu mengkonsumsi unit tulis tambahan. Permintaan penulisan transaksional membutuhkan 2 unit tulis untuk melakukan satu penulisan untuk item hingga 1 KB. Jumlah total unit permintaan tulis yang diperlukan bergantung pada ukuran item. Misalnya, jika ukuran item Anda adalah 2 KB, Anda memerlukan 2 unit tulis untuk mempertahankan satu permintaan tulis atau 4 unit tulis untuk permintaan tulis transaksional.

Ukuran item untuk penulisan dibulatkan ke kelipatan 1 KB berikutnya. Misalnya, menulis item berukuran 500 byte memerlukan throughput yang sama dengan menulis item berukuran 1 KB.

Daftar berikut menjelaskan bagaimana operasi penulisan DynamoDB menggunakan unit tulis:

- PutItem: Menulis satu item ke tabel. Jika item dengan kunci primer yang sama ada dalam tabel, operasi menggantikan item. Untuk menghitung konsumsi throughput yang disediakan, ukuran item yang penting lebih besar dari dua.

- UpdateItem: Memodifikasi satu item dalam tabel. DynamoDB menganggap ukuran item, seperti yang muncul sebelum dan setelah pembaruan. Throughput tersedia yang dikonsumsi mencerminkan ukuran item yang lebih besar. Bahkan jika Anda memperbarui subset atribut item, masih `UpdateItem` akan menggunakan jumlah penuh throughput yang disediakan (semakin besar ukuran item “sebelum” dan “setelah”).
- DeleteItem: Menghapus satu item dari tabel. Konsumsi throughput yang disediakan didasarkan pada ukuran item yang dihapus.
- BatchWriteItem: Menulis hingga 25 item ke satu atau lebih tabel. DynamoDB memproses setiap item dalam batch sebagai permintaan `PutItem` atau `DeleteItem` individu (pembaruan tidak didukung). DynamoDB pertama-tama membulatkan ukuran setiap item ke batas 1 KB berikutnya, dan kemudian menghitung ukuran totalnya. Hasilnya belum tentu sama dengan ukuran total semua item. Misalnya, jika `BatchWriteItem` menulis dua item ukuran 500-byte dan 3,5 KB, DynamoDB menghitung ukurannya sebagai 5 KB (1 KB+4 KB). DynamoDB tidak menghitung ukuran sebagai 4 KB (500 byte+3,5 KB).

Untuk operasi `PutItem`, `UpdateItem`, dan `DeleteItem`, DynamoDB membulatkan ukuran item hingga 1 KB berikutnya. Misalnya, jika Anda menempatkan atau menghapus item berukuran 1,6 KB, DynamoDB membulatkan ukuran item menjadi 2 KB.

`PutItem`,`UpdateItem`, dan `DeleteItem` operasi memungkinkan penulisan bersyarat, di mana Anda menentukan ekspresi yang harus mengevaluasi ke true agar operasi berhasil. Jika ekspresi bernilai salah, DynamoDB masih menggunakan unit kapasitas tulis dari tabel. Jumlah unit kapasitas tulis yang dikonsumsi tergantung pada ukuran item. Item ini dapat berupa item yang sudah ada di tabel atau yang baru Anda coba buat atau perbarui. Misalnya, katakan bahwa item yang ada adalah 300 KB. Item baru yang Anda coba buat atau perbarui adalah 310 KB. Unit kapasitas tulis yang dikonsumsi adalah 310 KB untuk item baru.

Kapasitas throughput DynamoDB

Bagian ini memberikan gambaran umum tentang dua mode throughput yang tersedia untuk tabel DynamoDB Anda dan pertimbangan dalam memilih mode kapasitas yang sesuai untuk aplikasi Anda. Mode throughput tabel menentukan bagaimana kapasitas tabel dikelola. Mode throughput juga menentukan bagaimana Anda dikenakan biaya untuk operasi baca dan tulis di tabel Anda. Di Amazon DynamoDB, Anda dapat memilih antara mode sesuai permintaan dan mode yang disediakan untuk tabel Anda guna mengakomodasi persyaratan beban kerja yang berbeda.

Topik

- Mode sesuai permintaan
- Mode yang disediakan
- Mode kapasitas sesuai permintaan DynamoDB
- Mode kapasitas yang disediakan DynamoDB
- Memahami throughput hangat DynamoDB
- DynamoDB burst dan kapasitas adaptif
- Pertimbangan saat mengganti mode kapasitas di DynamoDB

Mode sesuai permintaan

Mode on-demand Amazon DynamoDB adalah opsi throughput tanpa server yang menyederhanakan manajemen basis data dan secara otomatis menskalakan untuk mendukung aplikasi pelanggan yang paling menuntut. DynamoDB sesuai permintaan memungkinkan Anda membuat tabel tanpa mengkhawatirkan perencanaan kapasitas, memantau penggunaan, dan mengonfigurasi kebijakan penskalaan. DynamoDB on-demand pay-per-request menawarkan harga untuk permintaan baca dan tulis sehingga Anda hanya membayar untuk apa yang Anda gunakan. Untuk tabel mode sesuai permintaan, Anda tidak perlu menentukan jumlah throughput baca dan tulis yang Anda harapkan untuk dijalankan oleh aplikasi Anda.

Mode on-demand adalah opsi throughput default dan direkomendasikan untuk sebagian besar beban kerja DynamoDB. DynamoDB menangani semua aspek manajemen throughput dan penskalaan untuk mendukung beban kerja yang dapat dimulai dari yang kecil dan skala hingga jutaan permintaan per detik. Anda dapat membaca dan menulis ke tabel DynamoDB Anda sesuai kebutuhan tanpa mengelola kapasitas throughput di atas meja. Untuk informasi selengkapnya, lihat Mode kapasitas sesuai permintaan DynamoDB.

Mode yang disediakan

Dalam mode yang disediakan, Anda harus menentukan jumlah pembacaan dan penulisan per detik yang Anda perlukan untuk aplikasi Anda. Anda akan dikenakan biaya berdasarkan kapasitas baca dan tulis per jam yang telah Anda sediakan, bukan berapa banyak kapasitas yang disediakan yang sebenarnya Anda konsumsi. Hal ini membantu Anda mengatur penggunaan DynamoDB agar tetap pada atau di bawah tingkat permintaan yang ditentukan untuk memperoleh prediktabilitas biaya.

Anda dapat memilih untuk menggunakan kapasitas yang disediakan jika Anda memiliki beban kerja yang stabil dengan pertumbuhan yang dapat diprediksi, dan jika Anda dapat memperkirakan persyaratan kapasitas untuk aplikasi Anda dengan andal. Untuk informasi selengkapnya, lihat Mode kapasitas yang disediakan DynamoDB.

Mode kapasitas sesuai permintaan DynamoDB

Amazon DynamoDB on-demand menawarkan pengalaman database tanpa server yang secara otomatis menskalakan untuk mengakomodasi beban kerja yang paling menuntut tanpa perencanaan kapasitas. Sesuai permintaan menyederhanakan proses penyiapan, menghilangkan manajemen dan pemantauan kapasitas, dan menyediakan penskalaan otomatis yang cepat. Dengan pay-per-request harga, Anda tidak perlu khawatir tentang kapasitas idle karena Anda hanya membayar untuk throughput yang benar-benar Anda gunakan. Anda ditagih per permintaan baca atau tulis, sehingga biaya Anda secara langsung mencerminkan penggunaan Anda yang sebenarnya.

Saat Anda memilih mode sesuai permintaan, DynamoDB langsung mengakomodasi beban kerja Anda saat beban kerja tersebut meningkat atau menurun ke tingkat lalu lintas yang dicapai sebelumnya. Jika tingkat lalu lintas beban kerja mencapai puncak baru, DynamoDB secara otomatis menskalakan untuk mengakomodasi peningkatan persyaratan throughput. Mode on-demand adalah opsi throughput default dan direkomendasikan karena menyederhanakan pembuatan aplikasi modern tanpa server yang dapat dimulai dari yang kecil dan skala hingga jutaan permintaan per detik. Setelah tabel sesuai permintaan Anda ditingkatkan, Anda dapat langsung mencapai throughput yang sama lagi di masa depan tanpa pembatasan. Jika Anda mengarahkan nol lalu lintas ke meja Anda, maka dengan on-demand, Anda tidak dikenakan biaya untuk throughput apa pun. Untuk informasi selengkapnya tentang properti penskalaan mode sesuai permintaan, lihat. Throughput awal dan properti penskalaan

Tabel yang menggunakan mode sesuai permintaan memberikan latensi milidetik satu digit, perjanjian tingkat layanan (SLA), dan keamanan yang sama yang ditawarkan mode penyediaan DynamoDB.

> **Note**
>
> Secara default, DynamoDB melindungi Anda dari penggunaan yang tidak diinginkan dan tidak terkendali. Untuk menskalakan melampaui batas throughput baca dan tulis tingkat tabel 40.000 untuk semua tabel di akun Anda, Anda dapat meminta peningkatan untuk kuota ini. Permintaan throughput yang melebihi kuota throughput tabel default dibatasi.. Untuk informasi selengkapnya, lihat Kuota default throughput.

Secara opsional, Anda juga dapat mengonfigurasi throughput baca atau tulis maksimum (atau keduanya) per detik untuk tabel sesuai permintaan individu dan indeks sekunder global. Dengan mengonfigurasi throughput, Anda dapat menjaga penggunaan dan biaya tingkat tabel tetap terbatas, melindungi dari lonjakan sumber daya yang dikonsumsi secara tidak sengaja, dan mencegah penggunaan berlebihan untuk manajemen biaya yang dapat diprediksi. Permintaan throughput yang melebihi throughput tabel maksimum dibatasi. Anda dapat memodifikasi throughput maksimum khusus tabel kapan saja berdasarkan persyaratan aplikasi Anda. Untuk informasi selengkapnya, lihat DynamoDB throughput maksimum untuk tabel sesuai permintaan.

Untuk memulai, buat atau perbarui tabel untuk menggunakan mode sesuai permintaan. Untuk informasi selengkapnya, lihat Operasi dasar pada tabel DynamoDB.

Anda dapat mengganti tabel dari mode sesuai permintaan ke mode kapasitas yang disediakan kapan saja. Saat Anda melakukan beberapa sakelar di antara mode kapasitas, kondisi berikut berlaku:

- Anda dapat mengganti tabel yang baru dibuat dalam mode sesuai permintaan ke mode kapasitas yang disediakan kapan saja. Namun, Anda hanya dapat mengubahnya kembali ke mode sesuai permintaan 24 jam setelah stempel waktu pembuatan tabel.
- Anda dapat mengganti tabel yang ada dalam mode sesuai permintaan ke mode kapasitas yang disediakan kapan saja. Namun, Anda hanya dapat mengubahnya kembali ke mode sesuai permintaan 24 jam setelah stempel waktu terakhir yang menunjukkan peralihan ke sesuai permintaan.

Untuk informasi selengkapnya tentang beralih antara mode kapasitas baca dan tulis, lihatPertimbangan saat mengganti mode kapasitas di DynamoDB. Untuk kuota tabel sesuai permintaan, lihat. Membaca/menulis throughput

Topik

- Unit permintaan baca dan unit permintaan tulis
- Throughput awal dan properti penskalaan
- DynamoDB throughput maksimum untuk tabel sesuai permintaan

Unit permintaan baca dan unit permintaan tulis

DynamoDB menagih Anda untuk membaca dan menulis bahwa aplikasi Anda bekerja di tabel Anda dalam hal unit permintaan baca dan unit permintaan tulis.

Satu unit permintaan baca mewakili satu operasi baca yang sangat konsisten per detik, atau dua operasi baca yang akhirnya konsisten per detik, untuk item berukuran hingga 4 KB. Untuk informasi selengkapnya tentang DynamoDB, baca model konsistensi, lihat. DynamoDB membaca konsistensi

Satu unit permintaan tulis mewakili satu operasi tulis per detik, untuk item berukuran hingga 1 KB.

Untuk informasi lebih lanjut tentang bagaimana unit baca dan tulis dikonsumsi, lihatOperasi baca dan tulis DynamoDB.

Throughput awal dan properti penskalaan

Tabel DynamoDB yang menggunakan mode kapasitas sesuai permintaan secara otomatis beradaptasi dengan volume lalu lintas aplikasi Anda. Tabel on-demand baru akan dapat mempertahankan hingga 4.000 penulisan per detik dan 12.000 pembacaan per detik. Mode kapasitas sesuai permintaan secara instan mengakomodasi hingga dua kali lipat lalu lintas puncak sebelumnya pada sebuah tabel. Misalnya, katakan bahwa pola lalu lintas aplikasi Anda bervariasi antara 25.000 dan 50.000 pembacaan yang sangat konsisten per detik. 50.000 pembacaan per detik adalah puncak lalu lintas sebelumnya. Mode kapasitas sesuai permintaan langsung mengakomodasi lalu lintas berkelanjutan hingga 100.000 pembacaan per detik. Jika aplikasi Anda mempertahankan lalu lintas 100.000 pembacaan per detik, puncak itu menjadi puncak baru Anda sebelumnya. Puncak sebelumnya memungkinkan lalu lintas berikutnya mencapai hingga 200.000 pembacaan per detik.

Jika beban kerja Anda menghasilkan lebih dari dua kali lipat puncak sebelumnya di atas meja, DynamoDB secara otomatis mengalokasikan lebih banyak kapasitas saat volume lalu lintas Anda meningkat. Alokasi kapasitas ini membantu memastikan bahwa beban kerja Anda tidak mengalami pelambatan. Namun, throttling dapat terjadi jika Anda melebihi dua kali lipat puncak sebelumnya dalam waktu 30 menit. Misalnya, katakan bahwa pola lalu lintas aplikasi Anda bervariasi antara 25.000 dan 50.000 pembacaan yang sangat konsisten per detik. 50.000 pembacaan per detik adalah puncak lalu lintas yang dicapai sebelumnya. Kami menyarankan Anda melakukan pra-

pemanasan meja atau memberi jarak pertumbuhan lalu lintas Anda setidaknya selama 30 menit sebelum mengemudi lebih dari 100.000 pembacaan per detik. Untuk informasi lebih lanjut tentang pra-pemanasan, lihatMemahami throughput hangat DynamoDB.

DynamoDB tidak menempatkan pembatasan pembatasan 30 menit jika lalu lintas puncak beban kerja Anda tetap dalam dua kali lipat puncak sebelumnya. Jika lalu lintas puncak Anda melebihi dua kali lipat puncak, pastikan bahwa pertumbuhan ini terjadi 30 menit setelah Anda terakhir mencapai puncak.

DynamoDB throughput maksimum untuk tabel sesuai permintaan

Untuk tabel sesuai permintaan, Anda dapat menentukan throughput baca atau tulis maksimum (atau keduanya) per detik pada tabel individual dan indeks sekunder global terkait (). GSIs Menentukan throughput sesuai permintaan maksimum membantu menjaga penggunaan dan biaya tingkat tabel tetap terbatas. Secara default, pengaturan throughput maksimum tidak berlaku dan tingkat throughput sesuai permintaan Anda dibatasi oleh 40.000 kuota AWS layanan throughput baca dan tulis tingkat tabel untuk semua tabel di akun. Jika diperlukan, Anda dapat meminta peningkatan kuota layanan Anda.

Saat Anda mengonfigurasi throughput maksimum untuk tabel sesuai permintaan, permintaan throughput yang melebihi jumlah maksimum yang ditentukan akan dibatasi. Anda dapat mengubah pengaturan throughput tingkat tabel kapan saja berdasarkan persyaratan aplikasi Anda.

Berikut ini adalah beberapa kasus penggunaan umum yang dapat mengambil manfaat dari penggunaan throughput maksimum untuk tabel sesuai permintaan:

- Optimalisasi biaya throughput — Menggunakan throughput maksimum untuk tabel sesuai permintaan memberikan lapisan tambahan prediktabilitas biaya dan pengelolaan. Selain itu, ia menawarkan fleksibilitas yang lebih besar untuk menggunakan mode on-demand untuk mendukung beban kerja dengan pola lalu lintas dan anggaran yang berbeda.
- Perlindungan terhadap penggunaan berlebihan — Dengan menetapkan throughput maksimum, Anda dapat mencegah lonjakan konsumsi baca atau tulis yang tidak disengaja, yang mungkin timbul dari kode yang tidak dioptimalkan atau proses nakal, terhadap tabel sesuai permintaan. Pengaturan tingkat tabel ini dapat melindungi organisasi dari mengkonsumsi sumber daya yang berlebihan dalam jangka waktu tertentu.
- Melindungi layanan hilir — Aplikasi pelanggan dapat mencakup teknologi tanpa server dan non-server. Bagian arsitektur tanpa server dapat berskala cepat agar sesuai dengan permintaan. Tetapi komponen hilir dengan kapasitas tetap bisa kewalahan. Menerapkan pengaturan throughput

maksimum untuk tabel sesuai permintaan dapat mencegah volume besar peristiwa menyebar ke beberapa komponen hilir dengan efek samping yang tidak terduga.

Anda dapat mengonfigurasi throughput maksimum untuk mode sesuai permintaan untuk tabel Single-region baru dan yang sudah ada dan tabel global dan. GSIs Anda juga dapat mengonfigurasi throughput maksimum selama pemulihan tabel dan impor data dari alur kerja Amazon S3.

Anda dapat menentukan setelan throughput maksimum untuk tabel sesuai permintaan menggunakan konsol DynamoDB,, AWS CLIatau DynamoDB API. AWS CloudFormation

> **Note**
>
> Throughput maksimum untuk tabel sesuai permintaan diterapkan atas dasar upaya terbaik dan harus dianggap sebagai target alih-alih plafon permintaan yang dijamin. Beban kerja Anda mungkin untuk sementara melebihi throughput maksimum yang ditentukan karena kapasitas burst. Dalam beberapa kasus, DynamoDB menggunakan kapasitas burst untuk mengakomodasi pembacaan atau penulisan melebihi pengaturan throughput maksimum tabel Anda. Dengan kapasitas lonjakan, permintaan baca atau tulis yang tidak terduga dapat berhasil ketika mereka semestinya di-throttle.

Topik

- Pertimbangan saat menggunakan throughput maksimum untuk mode on-demand
- Minta pelambatan dan metrik CloudWatch

Pertimbangan saat menggunakan throughput maksimum untuk mode on-demand

Bila Anda menggunakan throughput maksimum untuk tabel dalam mode sesuai permintaan, pertimbangan berikut berlaku:

- Anda dapat secara mandiri mengatur throughput maksimum untuk membaca dan menulis untuk setiap tabel sesuai permintaan, atau indeks sekunder global individu dalam tabel tersebut untuk menyempurnakan pendekatan Anda berdasarkan persyaratan tertentu.
- Anda dapat menggunakan Amazon CloudWatch untuk memantau dan memahami metrik penggunaan tingkat tabel DynamoDB dan untuk menentukan pengaturan throughput maksimum yang sesuai untuk mode sesuai permintaan. Untuk informasi selengkapnya, lihat Dimensi dan Metrik DynamoDB.

- Saat Anda menentukan pengaturan throughput baca atau tulis (atau keduanya) maksimum pada satu replika tabel global, pengaturan throughput maksimum yang sama diterapkan secara otomatis ke semua tabel replika. Penting bahwa tabel replika dan indeks sekunder dalam tabel global memiliki pengaturan throughput tulis yang identik untuk memastikan replikasi data yang tepat. Untuk informasi selengkapnya, lihat Praktik dan persyaratan terbaik untuk mengelola tabel global DynamoDB.
- Throughput baca atau tulis maksimum terkecil yang dapat Anda tentukan adalah satu unit permintaan per detik.
- Throughput maksimum yang Anda tentukan harus lebih rendah dari kuota throughput default yang tersedia untuk setiap tabel sesuai permintaan, atau indeks sekunder global individual dalam tabel tersebut.

Minta pelambatan dan metrik CloudWatch

Jika aplikasi Anda melebihi throughput baca atau tulis maksimum yang telah Anda tetapkan pada tabel sesuai permintaan, DynamoDB mulai membatasi permintaan tersebut. Ketika DynamoDB melakukan throttle operasi baca atau tulis, ia mengembalikan `ThrottlingException` ke pemanggil. Anda kemudian dapat mengambil tindakan yang tepat, jika diperlukan. Misalnya, Anda dapat menambah atau menonaktifkan pengaturan throughput tabel maksimum, atau menunggu interval pendek sebelum mencoba kembali permintaan.

Untuk menyederhanakan pemantauan throughput maksimum yang dikonfigurasi untuk tabel atau indeks sekunder global, CloudWatch berikan metrik berikut: dan. OnDemandMaxReadRequestUnits OnDemandMaxWriteRequestUnits

Mode kapasitas yang disediakan DynamoDB

Saat Anda membuat tabel baru yang disediakan di DynamoDB, Anda harus menentukan kapasitas throughput yang disediakan. Ini adalah jumlah throughput baca dan tulis yang dapat didukung oleh tabel. Anda akan dikenakan biaya berdasarkan kapasitas baca dan tulis per jam yang telah Anda sediakan, bukan berapa banyak kapasitas yang disediakan yang sebenarnya Anda konsumsi.

Saat data aplikasi dan persyaratan akses berubah, Anda mungkin perlu menyesuaikan pengaturan throughput tabel Anda. Anda dapat menggunakan penskalaan otomatis untuk menyesuaikan kapasitas yang disediakan tabel Anda secara otomatis sebagai respons terhadap perubahan lalu lintas. DynamoDB auto scaling menggunakan kebijakan penskalaan di Application Auto Scaling.

Untuk mengonfigurasi penskalaan otomatis di DynamoDB, Anda menetapkan tingkat minimum dan maksimum kapasitas baca dan tulis selain persentase pemanfaatan target. Application Auto Scaling membuat dan mengelola CloudWatch alarm yang memicu peristiwa penskalaan saat metrik menyimpang dari target. Auto Scaling memantau aktivitas tabel Anda dan menyesuaikan pengaturan kapasitasnya naik atau turun berdasarkan ambang batas yang telah dikonfigurasi sebelumnya. Penskalaan otomatis terpicu ketika kapasitas yang Anda konsumsi melanggar pemanfaatan target yang dikonfigurasi selama dua menit berturut-turut. CloudWatch alarm mungkin memiliki penundaan singkat hingga beberapa menit sebelum memicu penskalaan otomatis. Untuk informasi selengkapnya, lihat Mengelola kapasitas throughput secara otomatis dengan penskalaan otomatis DynamoDB.

Jika Anda menggunakan penskalaan otomatis DynamoDB, pengaturan throughput secara otomatis disesuaikan sebagai respons terhadap beban kerja aktual. Anda juga dapat menggunakan operasi UpdateTable untuk menyesuaikan kapasitas throughput tabel Anda secara manual. Misalnya, Anda mungkin memutuskan untuk melakukan ini jika Anda perlu memuat data secara massal dari penyimpanan data yang ada ke dalam tabel DynamoDB baru Anda. Anda dapat membuat tabel dengan pengaturan throughput tulis yang besar, lalu mengurangi pengaturan ini setelah pemuatan data massal selesai.

> **Note**
>
> Secara default, DynamoDB melindungi Anda dari penggunaan yang tidak diinginkan dan tidak terkendali. Untuk menskalakan melampaui batas throughput baca dan tulis tingkat tabel 40.000 untuk semua tabel di akun Anda, Anda dapat meminta peningkatan untuk kuota ini. Permintaan throughput yang melebihi kuota throughput tabel default dibatasi.. Untuk informasi selengkapnya, lihat Kuota default throughput.

Anda dapat mengganti tabel dari mode sesuai permintaan ke mode kapasitas yang disediakan kapan saja. Saat Anda melakukan beberapa sakelar di antara mode kapasitas, kondisi berikut berlaku:

- Anda dapat mengganti tabel yang baru dibuat dalam mode sesuai permintaan ke mode kapasitas yang disediakan kapan saja. Namun, Anda hanya dapat mengubahnya kembali ke mode sesuai permintaan 24 jam setelah stempel waktu pembuatan tabel.
- Anda dapat mengganti tabel yang ada dalam mode sesuai permintaan ke mode kapasitas yang disediakan kapan saja. Namun, Anda hanya dapat mengubahnya kembali ke mode sesuai permintaan 24 jam setelah stempel waktu terakhir yang menunjukkan peralihan ke sesuai permintaan.

Untuk informasi selengkapnya tentang beralih antara mode kapasitas baca dan tulis, lihatPertimbangan saat mengganti mode kapasitas di DynamoDB.

Topik

- Unit kapasitas baca dan unit kapasitas tulis
- Memilih pengaturan throughput awal
- Penskalaan otomatis DynamoDB
- Mengelola kapasitas throughput secara otomatis dengan penskalaan otomatis DynamoDB
- Kapasitas cadangan DynamoDB

Unit kapasitas baca dan unit kapasitas tulis

Untuk tabel mode yang disediakan, Anda menentukan persyaratan throughput dalam hal unit kapasitas. Unit-unit ini mewakili jumlah data yang dibutuhkan aplikasi Anda untuk membaca atau menulis per detik. Anda dapat mengubah pengaturan ini nanti, jika diperlukan, atau mengaktifkan penskalaan otomatis DynamoDB untuk mengubahnya secara otomatis.

Untuk item hingga 4 KB, satu unit kapasitas baca (RCU) mewakili satu operasi baca yang sangat konsisten per detik, atau dua operasi baca yang akhirnya konsisten per detik. Untuk informasi selengkapnya tentang DynamoDB, baca model konsistensi, lihat. DynamoDB membaca konsistensi

Unit kapasitas tulis (WCU) mewakili satu penulisan per detik untuk item hingga 1 KB. Untuk informasi selengkapnya tentang operasi baca dan tulis yang berbeda, lihatOperasi baca dan tulis DynamoDB.

Memilih pengaturan throughput awal

Setiap aplikasi memiliki persyaratan yang berbeda untuk membaca dari dan menulis ke database. Saat Anda menentukan pengaturan throughput awal untuk tabel DynamoDB, pertimbangkan hal berikut:

- Tarif permintaan baca dan tulis yang diharapkan - Anda harus memperkirakan jumlah membaca dan menulis yang perlu Anda lakukan per detik.
- Ukuran barang — Beberapa item cukup kecil sehingga dapat dibaca atau ditulis menggunakan unit kapasitas tunggal. Item yang lebih besar memerlukan beberapa unit kapasitas. Dengan memperkirakan ukuran rata-rata item yang akan ada di tabel Anda, Anda dapat menentukan pengaturan akurat untuk throughput yang disediakan tabel Anda.

- Persyaratan konsistensi baca — Unit kapasitas Baca didasarkan pada operasi baca yang sangat konsisten, yang mengkonsumsi sumber daya basis data dua kali lebih banyak daripada pembacaan yang konsisten pada akhirnya. Anda harus menentukan apakah aplikasi Anda memerlukan bacaan akhir konsisten, atau apakah aplikasi dapat melonggarkan persyaratan ini dan pada akhirnya melakukan bacaan sangat konsisten. Operasi baca di DynamoDB pada akhirnya konsisten, secara default. Anda dapat meminta pembacaan yang sangat konsisten untuk operasi ini, jika perlu.

Misalnya, katakan bahwa Anda ingin membaca 80 item per detik dari sebuah tabel. Ukuran item ini adalah 3 KB, dan Anda ingin pembacaan yang sangat konsisten. Dalam hal ini, setiap pembacaan memerlukan satu unit kapasitas baca yang disediakan. Untuk menentukan nomor ini, bagi ukuran item operasi dengan 4 KB. Kemudian, bulatkan ke bilangan bulat terdekat, seperti yang ditunjukkan pada contoh berikut:

- 3 KB/ 4 KB = 0,75 atau 1 unit kapasitas baca

Oleh karena itu, untuk membaca 80 item per detik dari tabel, atur throughput baca yang disediakan tabel ke 80 unit kapasitas baca seperti yang ditunjukkan pada contoh berikut:

- 1 unit kapasitas baca setiap item × 80 bacaan per detik = 80 unit kapasitas baca

Sekarang anggaplah Anda ingin menulis 100 item per detik ke tabel Anda dan ukuran setiap item adalah 512 byte. Dalam hal ini, setiap penulisan memerlukan satu unit kapasitas tulis yang disediakan. Untuk menentukan nomor ini, bagi ukuran item operasi dengan 1 KB. Kemudian, bulatkan ke bilangan bulat terdekat, seperti yang ditunjukkan pada contoh berikut:

- 512 byte/ 1 KB = 0,5 atau 1 unit kapasitas tulis

Untuk menulis 100 item per detik ke tabel Anda, atur throughput tulis yang disediakan tabel menjadi 100 unit kapasitas tulis:

- 1 unit kapasitas tulis per item × 100 tulis per detik = 100 unit kapasitas tulis

Penskalaan otomatis DynamoDB

DynamoDB auto scaling secara aktif mengelola kapasitas throughput yang disediakan untuk tabel dan indeks sekunder global. Dengan penskalaan otomatis, Anda menentukan rentang (batas atas dan bawah) untuk unit kapasitas baca dan tulis. Anda juga menentukan persentase pemanfaatan target dalam rentang tersebut. Penskalaan otomatis DynamoDB berupaya mempertahankan pemanfaatan target Anda, bahkan saat beban kerja aplikasi Anda meningkat atau menurun.

Dengan penskalaan otomatis DynamoDB, tabel atau indeks sekunder global dapat meningkatkan kapasitas baca dan tulis yang disediakan untuk menangani peningkatan lalu lintas secara tiba-tiba, tanpa pembatasan permintaan. Ketika beban kerja berkurang, penskalaan otomatis DynamoDB dapat menurunkan throughput sehingga Anda tidak membayar kapasitas tersedia yang tidak terpakai.

> **Note**
>
> Jika Anda menggunakan AWS Management Console untuk membuat tabel atau indeks sekunder global, DynamoDB auto scaling diaktifkan secara default.
> Anda dapat mengelola pengaturan penskalaan otomatis kapan saja dengan menggunakan konsol, konsol AWS CLI, atau salah satu. AWS SDKs Untuk informasi selengkapnya, lihat Mengelola kapasitas throughput secara otomatis dengan penskalaan otomatis DynamoDB.

Tingkat pemanfaatan

Tingkat pemanfaatan dapat membantu Anda menentukan apakah Anda melebihi kapasitas penyediaan, dalam hal ini harus mengurangi kapasitas meja Anda untuk menghemat biaya. Sebaliknya, ini juga dapat membantu Anda menentukan apakah Anda berada di bawah kapasitas penyediaan. Dalam hal ini, Anda harus meningkatkan kapasitas tabel untuk mencegah potensi pembatasan permintaan selama kejadian lalu lintas tinggi yang tidak terduga. Untuk informasi selengkapnya, lihat Penskalaan otomatis Amazon DynamoDB: Pengoptimalan kinerja dan biaya pada skala apa pun.

Jika Anda menggunakan penskalaan otomatis DynamoDB, Anda juga harus menetapkan persentase pemanfaatan target. Penskalaan otomatis akan menggunakan persentase ini sebagai target untuk menyesuaikan kapasitas ke atas atau ke bawah. Kami merekomendasikan untuk menetapkan pemanfaatan target hingga 70%. Untuk informasi selengkapnya, lihat Mengelola kapasitas throughput secara otomatis dengan penskalaan otomatis DynamoDB.

Mengelola kapasitas throughput secara otomatis dengan penskalaan otomatis DynamoDB

Banyak beban kerja basis data yang bersifat siklik, sementara yang lainnya sulit diprediksi sebelumnya. Sebagai salah satu contoh, perhatikan aplikasi jejaring sosial yang sebagian besar penggunanya aktif pada siang hari. Basis data harus mampu menangani aktivitas siang hari, tetapi tingkat throughput yang sama tidak diperlukan pada malam hari. Sebagai contoh lainnya, pertimbangkan aplikasi game seluler baru yang tidak disangka mengalami adopsi yang begitu cepat. Jika game menjadi terlalu populer, game tersebut bisa melampaui sumber daya basis data yang tersedia, sehingga performanya lambat dan pelanggan menjadi tidak puas. Beban kerja semacam ini kerap perlu dinaikkan atau dikurangi skala sumber daya basis datanya secara manual guna merespons berbagai tingkat penggunaan.

Penskalaan otomatis Amazon DynamoDB menggunakan layanan Application AWS Auto Scaling untuk menyesuaikan kapasitas throughput yang disediakan secara dinamis atas nama Anda, sebagai respons terhadap pola lalu lintas yang sebenarnya. Hal ini memungkinkan tabel atau indeks sekunder global (GSI) untuk meningkatkan kapasitas baca dan tulis yang disediakan untuk menangani peningkatan lalu lintas yang tiba-tiba, tanpa pembatasan. Ketika beban kerja berkurang, Application Auto Scaling dapat menurunkan throughput agar Anda tidak membayar kapasitas tersedia yang tidak terpakai.

> **Note**
>
> Jika Anda menggunakan AWS Management Console untuk membuat tabel atau indeks sekunder global, DynamoDB auto scaling diaktifkan secara default. Anda dapat memodifikasi pengaturan penskalaan otomatis kapan saja. Untuk informasi selengkapnya, lihat Menggunakan penskalaan AWS Management Console otomatis with DynamoDB.
> Saat Anda menghapus tabel atau replika tabel global, target terukur, kebijakan penskalaan, atau CloudWatch alarm yang terkait tidak akan dihapus secara otomatis.

Dengan Application Auto Scaling, Anda membuat kebijakan penskalaan untuk tabel atau indeks sekunder global. Kebijakan penskalaan menentukan apakah Anda ingin menskalakan kapasitas baca atau tulis (atau keduanya), serta pengaturan unit kapasitas minimum dan maksimum yang tersedia untuk tabel atau indeks.

Kebijakan penskalaan juga berisi pemanfaatan target—persentase throughput tersedia yang dikonsumsi pada suatu titik waktu. Application Auto Scaling menggunakan algoritma pelacakan target

untuk menyesuaikan throughput tersedia dari tabel (atau indeks) ke atas atau ke bawah sebagai respons atas beban kerja aktual, sehingga pemanfaatan kapasitas aktual tetap atau mendekati pemanfaatan target Anda.

Output DynamoDB menggunakan throughput yang disediakan untuk periode satu menit. Penskalaan otomatis terpicu ketika kapasitas yang Anda konsumsi melanggar pemanfaatan target yang dikonfigurasi selama dua menit berturut-turut. CloudWatch alarm mungkin memiliki penundaan singkat hingga beberapa menit sebelum memicu penskalaan otomatis. Penundaan ini memastikan evaluasi CloudWatch metrik yang akurat. Jika lonjakan throughput yang dikonsumsi terpisah lebih dari satu menit, penskalaan otomatis mungkin tidak terpicu. Demikian pula, peristiwa penurunan skala dapat terjadi ketika 15 titik data berturut-turut lebih rendah dari pemanfaatan target. Dalam kedua kasus tersebut, setelah pemicu penskalaan otomatis, UpdateTableAPI dipanggil. Kemudian dibutuhkan beberapa menit untuk memperbarui kapasitas yang disediakan untuk tabel atau indeks. Selama periode ini, setiap permintaan yang melebihi kapasitas tabel yang disediakan sebelumnya akan dibatasi.

> **Important**
>
> Anda tidak dapat menyesuaikan jumlah titik data yang akan dilanggar untuk memicu alarm yang mendasarinya (meskipun nomor saat ini dapat berubah di masa mendatang).

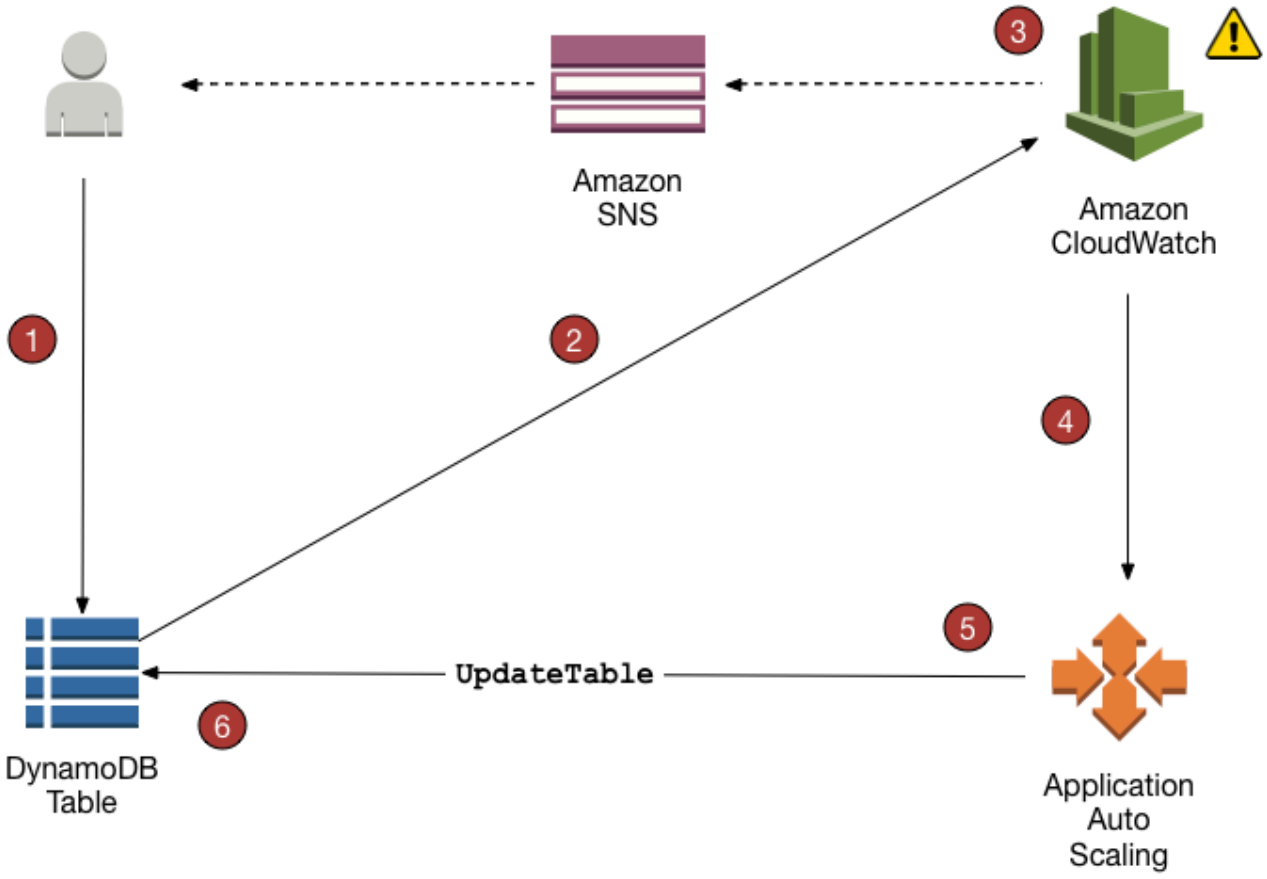
Anda dapat menetapkan nilai pemanfaatan target penskalaan otomatis antara 20 dan 90 persen untuk kapasitas baca dan tulis Anda.

> **Note**
>
> Selain tabel, penskalaan otomatis DynamoDB juga mendukung indeks sekunder global. Setiap indeks sekunder global memiliki kapasitas throughput tersedia miliknya sendiri, terpisah dari tabel dasarnya. Ketika Anda membuat kebijakan penskalaan untuk indeks sekunder global, Application Auto Scaling menyesuaikan pengaturan throughput tersedia untuk indeks guna memastikan bahwa pemanfaatan aktual tetap atau mendekati rasio pemanfaatan yang Anda inginkan.

Cara kerja penskalaan otomatis DynamoDB

> **Note**
>
> Untuk memulai penggunaan penskalaan otomatis DynamoDB dengan cepat, lihat Menggunakan penskalaan AWS Management Console otomatis with DynamoDB.

Diagram berikut memberikan gambaran umum tingkat tinggi tentang bagaimana penskalaan otomatis DynamoDB mengelola kapasitas throughput untuk tabel.



Langkah-langkah berikut meringkas proses penskalaan otomatis seperti yang ditunjukkan pada diagram sebelumnya:

1. Anda membuat kebijakan Application Auto Scaling untuk tabel DynamoDB Anda.
2. DynamoDB menerbitkan metrik kapasitas yang dikonsumsi ke Amazon. CloudWatch

3. Jika kapasitas konsumsi tabel melebihi pemanfaatan target Anda (atau jatuh di bawah target) untuk jangka waktu tertentu, Amazon CloudWatch memicu alarm. Anda dapat melihat alarm di konsol dan menerima pemberitahuan menggunakan Amazon Simple Notification Service (Amazon SNS).
4. CloudWatch Alarm memanggil Application Auto Scaling untuk mengevaluasi kebijakan penskalaan Anda.
5. Application Auto Scaling mengeluarkan permintaan `UpdateTable` untuk menyesuaikan throughput tersedia pada tabel Anda.
6. DynamoDB memproses permintaan `UpdateTable`, secara dinamis menaikkan (atau menurunkan) kapasitas throughput tersedia pada tabel agar mendekati pemanfaatan target Anda.

Untuk memahami cara kerja penskalaan otomatis DynamoDB, misalnya Anda memiliki tabel bernama `ProductCatalog`. Tabel ini dimuat secara massal dengan data secara jarang, sehingga tidak mengalami aktivitas tulis yang sangat banyak. Namun, tabel tersebut memang mengalami aktivitas baca yang tinggi dan bervariasi dari waktu ke waktu. Dengan memantau CloudWatch metrik Amazon`ProductCatalog`, Anda menentukan bahwa tabel memerlukan 1.200 unit kapasitas baca (untuk menghindari DynamoDB membatasi permintaan baca saat aktivitas mencapai puncaknya). Anda juga menentukan bahwa `ProductCatalog` memerlukan minimum 150 unit kapasitas baca, ketika lalu lintas baca berada pada titik terendah. Untuk informasi selengkapnya tentang mencegah throttling, lihat Masalah pelambatan untuk DynamoDB.

Dalam kisaran 150 sampai 1.200 unit kapasitas baca, Anda memutuskan bahwa pemanfaatan target sebesar 70 persen akan sesuai untuk tabel `ProductCatalog`. Pemanfaatan target adalah rasio unit kapasitas yang dikonsumsi terhadap unit kapasitas yang disediakan, dinyatakan dengan persentase. Application Auto Scaling menggunakan algoritma pelacakan targetnya untuk memastikan bahwa kapasitas baca yang disediakan dari `ProductCatalog` disesuaikan sesuai kebutuhan sehingga pemanfaatan tetap atau mendekati 70 persen.

> **Note**
>
> Penskalaan otomatis DynamoDB memodifikasi pengaturan throughput yang disediakan hanya ketika beban kerja sebenarnya tetap dinaikkan atau ditekan untuk jangka waktu berkelanjutan selama beberapa menit. Algoritma pelacakan target Application Auto Scaling berusaha menjaga pemanfaatan target ini pada atau mendekati nilai yang Anda pilih dalam jangka panjang.

Lonjakan aktivitas yang mendadak dan berdurasi singkat diakomodasi oleh kapasitas lonjakan bawaan tabel. Untuk informasi selengkapnya, lihat Kapasitas lonjakan.

Untuk mengaktifkan penskalaan otomatis DynamoDB untuk tabel `ProductCatalog`, Anda membuat kebijakan penskalaan. Kebijakan ini menentukan hal berikut:

- Tabel atau indeks sekunder global yang ingin Anda kelola
- Jenis kapasitas mana yang akan dikelola (kapasitas baca atau kapasitas tulis)
- Batas atas dan bawah untuk pengaturan throughput yang tersedia
- Pemanfaatan target Anda

Saat Anda membuat kebijakan penskalaan, Application Auto Scaling akan membuat sepasang alarm CloudWatch Amazon atas nama Anda. Setiap pasangan merepresentasikan batas atas dan bawah untuk pengaturan throughput tersedia Anda. CloudWatch Alarm ini dipicu ketika penggunaan tabel yang sebenarnya menyimpang dari penggunaan target Anda untuk jangka waktu yang berkelanjutan.

Ketika salah satu CloudWatch alarm dipicu, Amazon SNS mengirimi Anda pemberitahuan (jika Anda telah mengaktifkannya). CloudWatch Alarm kemudian memanggil Application Auto Scaling, yang pada gilirannya memberi tahu DynamoDB untuk menyesuaikan kapasitas yang disediakan tabel ke atas atau ke bawah `ProductCatalog` sebagaimana mestinya.

Selama acara penskalaan, AWS Config dibebankan per item konfigurasi yang direkam. Saat peristiwa penskalaan terjadi, empat CloudWatch alarm dibuat untuk setiap peristiwa auto-scaling baca dan tulis: alarm:, dan ProvisionedCapacity alarm:,. ProvisionedCapacityLow ProvisionedCapacityHigh ConsumedCapacity AlarmHigh AlarmLow Ini menghasilkan total delapan alarm. Oleh karena itu, AWS Config catat delapan item konfigurasi untuk setiap peristiwa penskalaan.

> **Note**
>
> Anda juga dapat menjadwalkan penskalaan DynamoDB Anda sehingga terjadi pada waktu-waktu tertentu. Pelajari langkah-langkah dasar di sini.

Catatan penggunaan

Sebelum Anda mulai menggunakan penskalaan otomatis DynamoDB, ketahui hal berikut:

- Penskalaan otomatis DynamoDB dapat meningkatkan kapasitas baca atau kapasitas tulis sesering yang diperlukan, sesuai dengan kebijakan penskalaan otomatis Anda. Semua kuota DynamoDB tetap berlaku, seperti yang dijelaskan dalam Kuota di Amazon DynamoDB.
- Penskalaan otomatis DynamoDB tidak mencegah Anda untuk memodifikasi sendiri pengaturan throughput yang tersedia. Penyesuaian manual ini tidak memengaruhi CloudWatch alarm yang ada yang terkait dengan penskalaan otomatis DynamoDB.
- Jika Anda mengaktifkan penskalaan otomatis DynamoDB untuk tabel yang memiliki satu atau beberapa indeks global sekunder, kami sangat menyarankan Anda agar juga menerapkan penskalaan otomatis yang seragam untuk indeks tersebut. Hal ini akan membantu memastikan performa yang lebih baik untuk penulisan dan pembacaan tabel, serta membantu menghindari throttling. Anda dapat mengaktifkan penskalaan otomatis dengan memilih Terapkan pengaturan yang sama untuk indeks sekunder global di AWS Management Console. Untuk informasi selengkapnya, lihat Mengaktifkan penskalaan otomatis DynamoDB pada tabel yang ada.
- Saat Anda menghapus tabel atau replika tabel global, target yang dapat diskalakan terkait, kebijakan penskalaan, atau CloudWatch alarm tidak akan dihapus secara otomatis dengannya.
- Saat membuat GSI untuk tabel yang ada, penskalaan otomatis tidak diaktifkan untuk GSI. Anda harus mengelola kapasitas secara manual saat GSI sedang dibangun. Setelah backfill pada GSI selesai dan mencapai status aktif, penskalaan otomatis akan beroperasi seperti biasa.

Menggunakan penskalaan AWS Management Console otomatis with DynamoDB

Saat Anda menggunakan AWS Management Console untuk membuat tabel baru, penskalaan otomatis Amazon DynamoDB diaktifkan untuk tabel tersebut secara default. Anda juga dapat menggunakan konsol untuk mengaktifkan penskalaan otomatis untuk tabel yang ada, memodifikasi pengaturan penskalaan otomatis, atau menonaktifkan penskalaan otomatis.

> **Note**
>
> Untuk fitur yang lebih canggih seperti menyetel waktu cooldown scale-in dan scale-out, gunakan AWS Command Line Interface () untuk AWS CLI mengelola penskalaan otomatis DynamoDB. Untuk informasi selengkapnya, lihat Menggunakan AWS CLI untuk mengelola penskalaan otomatis DynamoDB.

Topik

- Sebelum memulai: Memberikan izin pengguna untuk penskalaan otomatis DynamoDB

- Membuat tabel baru dengan penskalaan otomatis diaktifkan
- Mengaktifkan penskalaan otomatis DynamoDB pada tabel yang ada
- Melihat aktivitas penskalaan otomatis di konsol
- Memodifikasi atau menonaktifkan pengaturan penskalaan otomatis DynamoDB

Sebelum memulai: Memberikan izin pengguna untuk penskalaan otomatis DynamoDB

Di AWS Identity and Access Management (IAM), kebijakan AWS terkelola `DynamoDBFullAccess` memberikan izin yang diperlukan untuk menggunakan konsol DynamoDB. Namun, untuk penskalaan otomatis DynamoDB, pengguna memerlukan izin tambahan.

> **Important**
>
> Untuk menghapus tabel yang diaktifkan untuk penskalaan otomatis, diperlukan izin `application-autoscaling:*`. Kebijakan AWS terkelola `DynamoDBFullAccess` mencakup izin tersebut.

Untuk menyiapkan pengguna untuk akses konsol DynamoDB dan penskalaan otomatis DynamoDB, buat peran dan tambahkan kebijakan Access ke peran tersebut. AmazonDynamo DBFull Kemudian tetapkan peran ke seorang pengguna.

Membuat tabel baru dengan penskalaan otomatis diaktifkan

> **Note**
>
> Penskalaan otomatis DynamoDB memerlukan kehadiran peran yang ditautkan dengan layanan (`AWSServiceRoleForApplicationAutoScaling_DynamoDBTable`) yang melakukan tindakan penskalaan otomatis atas nama Anda. Peran ini dibuat secara otomatis untuk Anda. Untuk informasi selengkapnya, lihat Peran terkait layanan untuk Application Auto Scaling di Panduan Pengguna Application Auto Scaling.

Cara membuat tabel baru dengan penskalaan otomatis diaktifkan

1. Buka konsol DynamoDB di. https://console.aws.amazon.com/dynamodb/
2. Pilih Buat tabel.
3. Pada halaman Buat tabel, masukkan nama Tabel dan detail kunci utama.

4. Jika Anda memilih Pengaturan default, penskalaan otomatis diaktifkan di tabel baru.

   Jika tidak, pilih Sesuaikan pengaturan dan lakukan hal berikut untuk menentukan pengaturan khusus untuk tabel:

   a. Untuk kelas Table, pertahankan pilihan default DynamoDB Standard.

   b. Untuk pengaturan kapasitas baca/tulis, pertahankan pilihan default Provisioned, lalu lakukan hal berikut:

      i. Untuk kapasitas Baca, pastikan penskalaan Otomatis diatur ke Aktif.

      ii. Untuk kapasitas Tulis, pastikan penskalaan Otomatis diatur ke Aktif.

      iii. Untuk kapasitas Baca dan kapasitas Tulis, tetapkan kebijakan penskalaan yang Anda inginkan untuk tabel dan, secara opsional, semua indeks sekunder global tabel.

      - Unit kapasitas minimum – Masukkan batas bawah Anda untuk rentang penskalaan otomatis.
      - Unit kapasitas maksimum – Masukkan batas atas Anda untuk rentang penskalaan otomatis.
      - Pemanfaatan target – Masukkan persentase pemanfaatan target Anda untuk tabel.

      > **Note**
      >
      > Jika Anda membuat indeks sekunder global untuk tabel baru, kapasitas indeks pada saat pembuatan akan sama dengan kapasitas tabel dasar Anda. Anda dapat mengubah kapasitas indeks dalam pengaturan tabel setelah membuat tabel.

5. Pilih Buat tabel. Ini membuat tabel Anda dengan parameter penskalaan otomatis yang Anda tentukan.

Mengaktifkan penskalaan otomatis DynamoDB pada tabel yang ada

> **Note**
>
> Penskalaan otomatis DynamoDB memerlukan kehadiran peran yang ditautkan dengan layanan (`AWSServiceRoleForApplicationAutoScaling_DynamoDBTable`) yang melakukan tindakan penskalaan otomatis atas nama Anda. Peran ini dibuat secara otomatis

untuk Anda. Untuk informasi selengkapnya. lihat Peran tertaut layanan untuk Application Auto Scaling.

Cara mengaktifkan penskalaan otomatis DynamoDB untuk tabel yang ada

1. Buka konsol DynamoDB di. https://console.aws.amazon.com/dynamodb/
2. Di panel navigasi di sisi kiri konsol, pilih Tabel.
3. Pilih tabel tempat Anda ingin mengaktifkan penskalaan otomatis, lalu lakukan hal berikut:
   a. Pilih tab Pengaturan tambahan.
   b. Di bagian Kapasitas baca/tulis, pilih Edit.
   c. Di bagian Mode kapasitas, pilih Disediakan.
   d. Di bagian Kapasitas tabel, atur Penskalaan otomatis ke Aktif untuk Kapasitas baca, Kapasitas tulis, atau keduanya. Untuk setiap pengaturan tersebut, tetapkan kebijakan penskalaan yang Anda inginkan untuk tabel, dan semua indeks sekunder global tabel (opsional).
      - Unit kapasitas minimum – Masukkan batas bawah Anda untuk rentang penskalaan otomatis.
      - Unit kapasitas maksimum – Masukkan batas atas Anda untuk rentang penskalaan otomatis.
      - Pemanfaatan target – Masukkan persentase pemanfaatan target Anda untuk tabel.
      - Gunakan pengaturan kapasitas read/write kapasitas yang sama untuk semua indeks sekunder global — Pilih apakah indeks sekunder global harus menggunakan kebijakan penskalaan otomatis yang sama dengan tabel dasar.

        Note

        Untuk kinerja terbaik, kami sarankan Anda mengaktifkan Gunakan pengaturan read/write kapasitas yang sama untuk semua indeks sekunder global. Opsi ini memungkinkan penskalaan otomatis DynamoDB untuk menskalakan semua indeks sekunder global pada tabel dasar secara seragam. Ini termasuk indeks sekunder global yang ada, dan setiap indeks lain yang Anda buat untuk tabel ini nantinya.

Dengan mengaktifkan opsi ini, Anda tidak dapat menetapkan kebijakan penskalaan pada indeks sekunder global individual.

4. Jika pengaturan sudah sesuai keinginan Anda, pilih Simpan.

Melihat aktivitas penskalaan otomatis di konsol

Saat aplikasi Anda mendorong lalu lintas baca dan tulis ke tabel Anda, penskalaan otomatis DynamoDB secara dinamis memodifikasi pengaturan throughput tabel. Amazon CloudWatch melacak kapasitas yang disediakan dan dikonsumsi, peristiwa yang dibatasi, latensi, dan metrik lainnya untuk semua tabel DynamoDB dan indeks sekunder Anda.

Untuk melihat metrik ini di konsol DynamoDB, pilih tabel yang ingin Anda kerjakan, lalu pilih tab Monitor. Untuk membuat tampilan metrik tabel yang dapat disesuaikan, pilih Lihat semua dalam. CloudWatch

Memodifikasi atau menonaktifkan pengaturan penskalaan otomatis DynamoDB

Anda dapat menggunakan AWS Management Console untuk memodifikasi pengaturan penskalaan otomatis DynamoDB Anda. Untuk melakukan hal ini, buka tab Pengaturan tambahan untuk tabel Anda, lalu pilih Edit di bagian Kapasitas baca/tulis. Untuk informasi selengkapnya tentang pengaturan ini, lihat Mengaktifkan penskalaan otomatis DynamoDB pada tabel yang ada.

Menggunakan AWS CLI untuk mengelola penskalaan otomatis DynamoDB

Alih-alih menggunakan AWS Management Console, Anda dapat menggunakan AWS Command Line Interface (AWS CLI) untuk mengelola penskalaan otomatis Amazon DynamoDB. Tutorial di bagian ini menunjukkan cara menginstal dan mengonfigurasi AWS CLI untuk mengelola penskalaan otomatis DynamoDB. Dalam tutorial ini, Anda melakukan hal-hal berikut:

- Buat tabel DynamoDB bernama `TestTable`. Pengaturan throughput awal adalah 5 unit kapasitas baca dan 5 unit kapasitas tulis.
- Buat kebijakan Application Auto Scaling untuk `TestTable`. Kebijakan ini bertujuan untuk mempertahankan rasio target sebesar 50 persen antara kapasitas tulis yang dikonsumsi dan kapasitas tulis yang disediakan. Kisaran untuk metrik ini adalah antara 5 dan 10 unit kapasitas tulis. (Application Auto Scaling tidak diizinkan untuk menyesuaikan throughput di luar kisaran ini.)
- Jalankan program Python untuk mendorong lalu lintas tulis ke `TestTable`. Ketika rasio target melebihi 50 persen untuk jangka waktu yang berkelanjutan, Application Auto Scaling memberi

tahu DynamoDB untuk menyesuaikan throughput `TestTable` ke atas guna mempertahankan pemanfaatan target sebesar 50 persen.

- Verifikasi bahwa DynamoDB telah berhasil menyesuaikan kapasitas tulis yang disediakan untuk `TestTable`.

> **Note**
>
> Anda juga dapat menjadwalkan penskalaan DynamoDB Anda sehingga terjadi pada waktu-waktu tertentu. Pelajari langkah-langkah dasar di sini.

Topik

- Sebelum Anda mulai
- Langkah 1: Buat tabel DynamoDB
- Langkah 2: Daftarkan target yang dapat diskalakan
- Langkah 3: Buat kebijakan penskalaan
- Langkah 4: Dorong lalu lintas tulis ke TestTable
- Langkah 5: Lihat tindakan Application Auto Scaling
- (Opsional) Langkah 6: Hapus

Sebelum Anda mulai

Selesaikan tugas berikut sebelum memulai tutorial.

Instal AWS CLI

Jika belum, Anda harus menginstal dan mengonfigurasi AWS CLI. Untuk melakukannya, ikuti petunjuk berikut di Panduan Pengguna AWS Command Line Interface :

- Menginstal AWS CLI
- Mengonfigurasi AWS CLI

Instal Python

Anda perlu menjalankan program Python di bagian tutorial ini (lihat Langkah 4: Dorong lalu lintas tulis ke TestTable). Jika belum menginstalnya, Anda dapat mengunduh Python.

Langkah 1: Buat tabel DynamoDB

Pada langkah ini, Anda menggunakan AWS CLI to create`TestTable`. Kunci primer terdiri dari `pk` (kunci partisi) dan `sk` (kunci urutan). Kedua atribut ini berjenis `Number`. Pengaturan throughput awal adalah 5 unit kapasitas baca dan 5 unit kapasitas tulis.

1. Gunakan AWS CLI perintah berikut untuk membuat tabel.

```
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. Untuk memeriksa status tabel, gunakan perintah berikut.

```
aws dynamodb describe-table \
    --table-name TestTable \
    --query "Table.[TableName,TableStatus,ProvisionedThroughput]"
```

   Tabel siap digunakan saat statusnya `ACTIVE`.

Langkah 2: Daftarkan target yang dapat diskalakan

Selanjutnya Anda mendaftar kapasitas tulis tabel sebagai target yang dapat diskalakan dengan Application Auto Scaling. Hal ini memungkinkan Application Auto Scaling untuk menyesuaikan kapasitas tulis yang disediakan untuk TestTable, tetapi hanya dalam kisaran 5-10 unit kapasitas.

> **Note**
>
> Penskalaan otomatis DynamoDB memerlukan kehadiran peran tertaut layanan (`AWSServiceRoleForApplicationAutoScaling_DynamoDBTable`) yang melakukan tindakan penskalaan otomatis atas nama Anda. Peran ini dibuat secara otomatis untuk Anda. Untuk informasi selengkapnya, lihat Peran tertaut layanan untuk Application Auto Scaling dalam Panduan Pengguna Application Auto Scaling.

1. Gunakan perintah berikut untuk mendaftarkan target yang dapat diskalakan.

```
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. Untuk memverifikasi pendaftaran, gunakan perintah berikut.

```
aws application-autoscaling describe-scalable-targets \
    --service-namespace dynamodb \
    --resource-id "table/TestTable"
```

> **Note**
>
> Anda juga dapat mendaftarkan target yang dapat diskalakan terhadap indeks sekunder global. Sebagai contoh, untuk indeks sekunder global ("test-index"), ID sumber daya dan argumen dimensi yang dapat diskalakan diperbarui dengan sesuai.
>
> ```
> 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
> ```

Langkah 3: Buat kebijakan penskalaan

Pada langkah ini, Anda membuat kebijakan penskalaan untuk `TestTable`. Kebijakan ini mendefinisikan detail di mana Application Auto Scaling dapat menyesuaikan throughput yang disediakan pada tabel, dan tindakan yang harus diambil ketika melakukannya. Anda mengaitkan kebijakan ini dengan target yang dapat diskalakan yang Anda tetapkan pada langkah sebelumnya (unit kapasitas tulis untuk tabel `TestTable`).

Kebijakan tersebut berisi elemen berikut:

- `PredefinedMetricSpecification`—Metrik yang dapat disesuaikan oleh Application Auto Scaling. Untuk DynamoDB, nilai berikut adalah nilai yang valid untuk `PredefinedMetricType`:
  - `DynamoDBReadCapacityUtilization`
  - `DynamoDBWriteCapacityUtilization`
- `ScaleOutCooldown`—Jumlah waktu minimum (dalam detik) antara setiap peristiwa Application Auto Scaling yang meningkatkan throughput yang disediakan. Parameter ini memungkinkan Application Auto Scaling untuk secara terus-menerus, tetapi tanpa agresivitas, meningkatkan throughput sebagai respons terhadap beban kerja dunia nyata. Pengaturan default untuk `ScaleOutCooldown` adalah 0.
- `ScaleInCooldown`—Jumlah waktu minimum (dalam detik) antara setiap peristiwa Application Auto Scaling yang mengurangi throughput yang disediakan. Parameter ini memungkinkan Application Auto Scaling untuk mengurangi throughput secara bertahap dan terprediksi. Pengaturan default untuk `ScaleInCooldown` adalah 0.
- `TargetValue`—Application Auto Scaling memastikan bahwa rasio kapasitas yang dikonsumsi terhadap kapasitas yang disediakan tetap atau mendekati nilai ini. Anda mendefinisikan `TargetValue` sebagai persentase.

> **Note**
>
> Untuk lebih memahami cara kerja `TargetValue`, misalkan Anda memiliki sebuah tabel dengan pengaturan throughput yang disediakan sebanyak 200 unit kapasitas tulis. Anda memutuskan untuk membuat kebijakan penskalaan untuk tabel ini, dengan `TargetValue` sebesar 70 persen.
>
> Sekarang anggaplah Anda mulai mendorong lalu lintas tulis ke tabel sehingga throughput tulis aktual adalah sebesar 150 unit kapasitas. consumed-to-provisionedRasionya sekarang (150/200), atau 75 persen. Rasio ini melebihi target Anda, sehingga Application Auto Scaling meningkatkan kapasitas tulis yang disediakan menjadi 215, sehingga rasionya (150 / 215), atau 69,77 persen—sedekat mungkin dengan `TargetValue`, tetapi tidak melebihinya.

Untuk `TestTable`, Anda mengatur `TargetValue` ke 50 persen. Application Auto Scaling menyesuaikan throughput yang disediakan tabel dalam kisaran 5-10 unit kapasitas (lihatLangkah 2: Daftarkan target yang dapat diskalakan) sehingga rasio tetap pada atau mendekati 50 persen. consumed-to-provisioned Anda mengatur nilai untuk `ScaleOutCooldown` dan `ScaleInCooldown` ke 60 detik.

1. Buat file bernama `scaling-policy.json` dengan isi berikut ini.

```
{
    "PredefinedMetricSpecification": {
        "PredefinedMetricType": "DynamoDBWriteCapacityUtilization"
    },
    "ScaleOutCooldown": 60,
    "ScaleInCooldown": 60,
    "TargetValue": 50.0
}
```

2. Gunakan AWS CLI perintah berikut untuk membuat kebijakan.

```
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. Dalam output, perhatikan bahwa Application Auto Scaling telah menciptakan dua CloudWatch alarm Amazon — masing-masing untuk batas atas dan bawah rentang target penskalaan.
4. Gunakan AWS CLI perintah berikut untuk melihat detail selengkapnya tentang kebijakan penskalaan.

```
aws application-autoscaling describe-scaling-policies \
    --service-namespace dynamodb \
    --resource-id "table/TestTable" \
    --policy-name "MyScalingPolicy"
```

5. Dalam output, verifikasi bahwa pengaturan kebijakan sesuai dengan spesifikasi Anda dari Langkah 2: Daftarkan target yang dapat diskalakan dan Langkah 3: Buat kebijakan penskalaan.

Langkah 4: Dorong lalu lintas tulis ke TestTable

Sekarang Anda dapat menguji kebijakan penskalaan Anda dengan menulis data ke `TestTable`. Untuk melakukan ini, jalankan program Python.

1. Buat file bernama `bulk-load-test-table.py` dengan isi berikut ini.

```
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. Masukkan perintah berikut untuk menjalankan program.

   ```
   python bulk-load-test-table.py
   ```

   Kapasitas tulis yang disediakan untuk `TestTable` sangat rendah (5 unit kapasitas tulis), sehingga terkadang mengalami kegagalan karena throttling tulis. Ini adalah perilaku yang diharapkan.

   Biarkan program terus berjalan sementara Anda melanjutkan ke langkah berikutnya.

Langkah 5: Lihat tindakan Application Auto Scaling

Pada langkah ini, Anda melihat tindakan Application Auto Scaling yang dimulai atas nama Anda. Anda juga memverifikasi bahwa Application Auto Scaling telah memperbarui kapasitas tulis yang disediakan untuk `TestTable`.

1. Masukkan perintah berikut untuk melihat tindakan Application Auto Scaling.

```
aws application-autoscaling describe-scaling-activities \
    --service-namespace dynamodb
```

Jalankan kembali perintah ini sesekali, sementara program Python sedang berjalan. (Perlu beberapa menit hingga kebijakan penskalaan Anda diinvokasi.) Anda akan melihat output berikut.

```
...
{
    "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"
},
...
```

Hal ini menunjukkan bahwa Application Auto Scaling telah mengeluarkan permintaan `UpdateTable` untuk DynamoDB.

2. Masukkan perintah berikut untuk memverifikasi bahwa DynamoDB meningkatkan kapasitas tulis tabel.

```
aws dynamodb describe-table \
    --table-name TestTable \
    --query "Table.[TableName,TableStatus,ProvisionedThroughput]"
```

`WriteCapacityUnits` semestinya telah diskalakan dari `5` ke `10`.

(Opsional) Langkah 6: Hapus

Dalam tutorial ini, Anda telah membuat beberapa sumber daya. Anda dapat menghapus sumber daya ini jika sudah tidak membutuhkannya lagi.

1. Hapus kebijakan penskalaan untuk `TestTable`.

```
aws application-autoscaling delete-scaling-policy \
    --service-namespace dynamodb \
    --resource-id "table/TestTable" \
    --scalable-dimension "dynamodb:table:WriteCapacityUnits" \
    --policy-name "MyScalingPolicy"
```

2. Batalkan pendaftaran target yang dapat diskalakan.

```
aws application-autoscaling deregister-scalable-target \
    --service-namespace dynamodb \
    --resource-id "table/TestTable" \
    --scalable-dimension "dynamodb:table:WriteCapacityUnits"
```

3. Hapus tabel `TestTable`.

```
aws dynamodb delete-table --table-name TestTable
```

Menggunakan AWS SDK untuk mengonfigurasi penskalaan otomatis pada tabel Amazon DynamoDB

Selain menggunakan AWS Management Console dan AWS Command Line Interface (AWS CLI), Anda dapat menulis aplikasi yang berinteraksi dengan penskalaan otomatis Amazon DynamoDB. Bagian ini berisi dua program Java yang dapat Anda gunakan untuk menguji fungsi ini:

- `EnableDynamoDBAutoscaling.java`
- `DisableDynamoDBAutoscaling.java`

Mengaktifkan Application Auto Scaling untuk suatu tabel

Program berikut menunjukkan contoh penyiapan kebijakan penskalaan otomatis untuk tabel DynamoDB (`TestTable`). Hal ini berjalan sebagai berikut:

- Program mendaftarkan unit kapasitas tulis sebagai target yang dapat diskalakan untuk `TestTable`. Kisaran untuk metrik ini adalah antara 5 dan 10 unit kapasitas tulis.
- Setelah target yang dapat diskalakan dibuat, program ini membangun konfigurasi pelacakan target. Kebijakan ini bertujuan untuk mempertahankan rasio target sebesar 50 persen antara kapasitas tulis yang dikonsumsi dan kapasitas tulis yang disediakan.
- Program ini kemudian membuat kebijakan penskalaan berdasarkan konfigurasi pelacakan target.

> **Note**
>
> Saat menghapus tabel atau replika tabel global secara manual, Anda tidak secara otomatis menghapus target, kebijakan penskalaan, atau alarm terkait yang dapat diskalakan. 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

Program ini meminta Anda untuk menyediakan Amazon Resource Name (ARN) untuk peran yang tertaut dengan layanan Application Auto Scaling. (Misalnya: `arn:aws:iam::122517410325:role/AWSServiceRoleForApplicationAutoScaling_DynamoDBTable`.) Pada program berikut, ganti `SERVICE_ROLE_ARN_GOES_HERE` dengan ARN yang sebenarnya.

```
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());
  }

 }

}
```

Menonaktifkan Application Auto Scaling untuk suatu tabel

Program berikut membalikkan proses sebelumnya. Program akan menghapus kebijakan penskalaan otomatis dan kemudian membatalkan pendaftaran target yang dapat diskalakan.

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());
   }

  }

}
```

Kapasitas cadangan DynamoDB

Untuk tabel kapasitas yang disediakan yang menggunakan kelas tabel Standar, DynamoDB menawarkan kemampuan untuk membeli kapasitas cadangan untuk kapasitas baca dan tulis Anda. Pembelian kapasitas cadangan adalah perjanjian untuk membayar jumlah minimum kapasitas throughput yang disediakan, selama jangka waktu perjanjian, dengan imbalan harga diskon.

> **Note**
>
> Anda tidak dapat membeli kapasitas cadangan untuk unit kapasitas tulis yang direplikasi (rWCUs). Kapasitas cadangan hanya diterapkan ke Wilayah di mana ia dibeli. Kapasitas terpesan juga tidak tersedia untuk tabel yang menggunakan kelas tabel DynamoDB Standard-IA atau mode kapasitas sesuai permintaan.

Kapasitas cadangan dibeli dalam alokasi 100 WCUs atau 100 RCUs. Penawaran kapasitas cadangan terkecil adalah 100 unit kapasitas (baca atau tulis). Kapasitas cadangan DynamoDB ditawarkan sebagai komitmen satu tahun atau di Wilayah tertentu sebagai komitmen tiga tahun. Anda dapat menghemat hingga 54% dari tarif standar untuk jangka waktu satu tahun dan diskon 77% dari tarif standar untuk jangka waktu tiga tahun. Untuk informasi selengkapnya tentang cara dan kapan Anda harus membeli, lihat Kapasitas Cadangan Amazon DynamoDB.

Saat Anda membeli kapasitas cadangan DynamoDB, Anda membayar pembayaran di muka sebagian satu kali dan menerima tarif per jam diskon untuk penggunaan yang disediakan. Anda membayar seluruh penggunaan yang disediakan, terlepas dari penggunaan aktual, sehingga penghematan biaya Anda terkait erat dengan penggunaan. Setiap kapasitas yang Anda berikan melebihi kapasitas cadangan yang dibeli ditagih dengan tarif kapasitas standar yang disediakan.

Dengan memesan unit kapasitas baca dan tulis terlebih dahulu, Anda mewujudkan penghematan biaya yang signifikan pada biaya kapasitas yang disediakan.

Anda tidak dapat menjual, membatalkan, atau mentransfer kapasitas cadangan ke Wilayah atau akun lain.

Memahami throughput hangat DynamoDB

Throughput hangat mengacu pada jumlah operasi baca dan tulis yang dapat didukung oleh tabel DynamoDB Anda secara instan. Nilai-nilai ini tersedia secara default untuk semua tabel dan indeks sekunder global (GSI) dan mewakili seberapa besar skalanya berdasarkan penggunaan historis. Jika Anda menggunakan mode sesuai permintaan, atau jika Anda memperbarui throughput yang disediakan ke nilai-nilai ini, aplikasi Anda akan dapat mengeluarkan permintaan hingga nilai tersebut secara instan.

DynamoDB akan secara otomatis menyesuaikan nilai throughput hangat saat penggunaan Anda meningkat. Namun, Anda juga dapat meningkatkan nilai-nilai ini secara proaktif saat diperlukan, yang sangat berguna untuk acara puncak mendatang seperti peluncuran produk atau penjualan. Untuk acara puncak yang direncanakan, di mana tingkat permintaan ke tabel DynamoDB Anda mungkin meningkat 10x, 100x, atau lebih, Anda sekarang dapat menilai apakah throughput hangat saat ini cukup untuk menangani lalu lintas yang diharapkan. Jika tidak, Anda dapat meningkatkan nilai throughput hangat tanpa mengubah pengaturan throughput atau mode penagihan. Proses ini disebut sebagai pra-pemanasan tabel, memungkinkan Anda untuk menetapkan garis dasar yang dapat langsung didukung oleh tabel Anda. Ini memastikan aplikasi Anda dapat menangani tingkat permintaan yang lebih tinggi dari saat terjadi.

Anda dapat meningkatkan nilai throughput hangat untuk operasi baca, operasi tulis, atau keduanya. Anda dapat meningkatkan nilai ini untuk tabel Single-region baru dan yang sudah ada, tabel global, dan GSIs tabel. Untuk tabel global, fitur ini tersedia untuk versi 2019.11.21 (Saat ini), dan pengaturan throughput hangat yang Anda atur akan secara otomatis berlaku untuk semua tabel replika di tabel global. Tidak ada batasan jumlah tabel DynamoDB yang dapat Anda lakukan sebelum pemanasan kapan saja. Waktu untuk menyelesaikan pra-pemanasan tergantung pada nilai yang Anda tetapkan dan ukuran tabel atau indeks. Anda dapat mengirimkan permintaan pra-hangat secara simultan dan permintaan ini tidak akan mengganggu operasi tabel apa pun. Anda dapat melakukan pra-pemanasan tabel hingga batas kuota tabel atau indeks untuk akun Anda di Wilayah tersebut. Gunakan konsol Service Quotas untuk memeriksa batas Anda saat ini dan meningkatkannya jika diperlukan.

Nilai throughput hangat tersedia secara default untuk semua tabel dan indeks sekunder tanpa biaya. Namun, jika Anda secara proaktif meningkatkan nilai throughput hangat default ini untuk pra-pemanasan tabel, Anda akan dikenakan biaya untuk permintaan tersebut. Untuk informasi selengkapnya, lihat harga Amazon DynamoDB.

Untuk informasi lebih lanjut tentang throughput hangat, lihat topik di bawah ini:

Topik

- Periksa throughput hangat tabel DynamoDB Anda saat ini
- Tingkatkan throughput hangat tabel DynamoDB Anda yang ada
- Buat tabel DynamoDB baru dengan throughput hangat yang lebih tinggi
- Memahami throughput hangat DynamoDB dalam skenario yang berbeda

Periksa throughput hangat tabel DynamoDB Anda saat ini

Gunakan petunjuk berikut AWS CLI dan AWS Konsol untuk memeriksa tabel atau nilai throughput hangat indeks Anda saat ini.

AWS Management Console

Untuk memeriksa throughput hangat tabel DynamoDB Anda menggunakan konsol DynamoDB:

1. Masuk ke AWS Management Console dan buka konsol DynamoDB di. https://console.aws.amazon.com/dynamodb/
2. Di panel navigasi kiri, pilih Tabel.
3. Pada halaman Tabel, pilih tabel yang Anda inginkan.
4. Pilih Pengaturan tambahan untuk melihat nilai throughput hangat Anda saat ini. Nilai ini ditampilkan sebagai satuan baca per detik dan satuan tulis per detik.

AWS CLI

AWS CLI Contoh berikut menunjukkan cara memeriksa throughput hangat tabel DynamoDB Anda.

1. Jalankan `describe-table` operasi pada tabel DynamoDB Anda.

```
aws dynamodb describe-table --table-name GameScores
```

2. Anda akan menerima respons yang mirip dengan yang di bawah ini. `WarmThroughput`Pengaturan Anda akan ditampilkan sebagai `ReadUnitsPerSecond` dan`WriteUnitsPerSecond`. Itu `Status` akan terjadi `UPDATING` ketika nilai throughput hangat sedang diperbarui, dan `ACTIVE` ketika nilai throughput hangat baru ditetapkan.


```
{
    "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"
        }
    }
}
```

Tingkatkan throughput hangat tabel DynamoDB Anda yang ada

Setelah Anda memeriksa nilai throughput hangat tabel DynamoDB saat ini, Anda dapat memperbaruinya dengan langkah-langkah berikut:

AWS Management Console

Untuk memeriksa nilai throughput hangat tabel DynamoDB Anda menggunakan konsol DynamoDB:

1. Masuk ke AWS Management Console dan buka konsol DynamoDB di. https://console.aws.amazon.com/dynamodb/
2. Di panel navigasi kiri, pilih Tabel.
3. Pada halaman Tabel, pilih tabel yang Anda inginkan.
4. Di bidang Hangat throughput, pilih Edit.
5. Pada halaman Edit throughput hangat, pilih Tingkatkan throughput hangat.
6. Sesuaikan unit baca per detik dan tulis satuan per detik. Kedua pengaturan ini menentukan throughput yang dapat langsung ditangani tabel Anda.
7. Pilih Simpan.
8. Unit baca Anda per detik dan unit tulis per detik akan diperbarui di bidang throughput hangat saat permintaan selesai diproses.

> **Note**
>
> Memperbarui nilai throughput hangat Anda adalah tugas asinkron. `Status`Akan berubah dari `UPDATING` `ACTIVE` saat pembaruan selesai.

AWS CLI

AWS CLI Contoh berikut menunjukkan cara memperbarui nilai throughput hangat tabel DynamoDB Anda.

1. Jalankan `update-table` operasi pada tabel DynamoDB Anda.

```
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. Anda akan menerima respons yang mirip dengan yang di bawah ini. `WarmThroughput`Pengaturan Anda akan ditampilkan sebagai `ReadUnitsPerSecond` dan`WriteUnitsPerSecond`. Itu `Status` akan terjadi `UPDATING` ketika nilai throughput hangat sedang diperbarui, dan `ACTIVE` ketika nilai throughput hangat baru ditetapkan.

```
{
    "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

Contoh SDK berikut menunjukkan cara memperbarui nilai throughput hangat tabel DynamoDB Anda.

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;
  }
}
```

Buat tabel DynamoDB baru dengan throughput hangat yang lebih tinggi

Anda dapat menyesuaikan nilai throughput hangat saat membuat tabel DynamoDB dengan mengikuti langkah-langkah di bawah ini. Langkah-langkah ini juga berlaku saat membuat tabel global atau indeks sekunder.

AWS Management Console

Untuk membuat tabel DynamoDB dan menyesuaikan nilai throughput hangat melalui konsol:

1. Masuk ke AWS Management Console dan buka konsol DynamoDB di. https://console.aws.amazon.com/dynamodb/
2. Pilih Buat tabel.
3. Pilih nama Table, Partition key, dan Sort key (opsional).
4. Untuk pengaturan Tabel, pilih Sesuaikan pengaturan.
5. Di bidang throughput hangat, pilih Tingkatkan throughput hangat.
6. Sesuaikan unit baca per detik dan tulis satuan per detik. Kedua pengaturan ini menentukan throughput maksimum yang dapat ditangani oleh tabel Anda secara instan.
7. Lanjutkan menambahkan detail tabel yang tersisa dan kemudian pilih Buat tabel.

AWS CLI

AWS CLI Contoh berikut menunjukkan cara membuat tabel DynamoDB dengan nilai throughput hangat yang disesuaikan.

1. Jalankan `create-table` operasi untuk membuat tabel DynamoDB berikut.

```
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. Anda akan menerima respons yang mirip dengan yang di bawah ini. `WarmThroughput`Pengaturan Anda akan ditampilkan sebagai `ReadUnitsPerSecond` dan`WriteUnitsPerSecond`. Itu `Status` akan terjadi `UPDATING` ketika nilai throughput hangat sedang diperbarui, dan `ACTIVE` ketika nilai throughput hangat baru ditetapkan.

```
{
    "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

Contoh SDK berikut menunjukkan cara membuat tabel DynamoDB dengan nilai throughput hangat yang disesuaikan.

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;
  }
}
```

Memahami throughput hangat DynamoDB dalam skenario yang berbeda

Berikut adalah beberapa skenario berbeda yang mungkin Anda temui saat bekerja dengan throughput hangat DynamoDB.

Topik

- Throughput hangat dan pola akses yang tidak merata
- Throughput hangat untuk tabel yang disediakan
- Throughput hangat untuk meja sesuai permintaan
- Throughput hangat untuk tabel sesuai permintaan dengan throughput maksimum yang dikonfigurasi

Throughput hangat dan pola akses yang tidak merata

Sebuah tabel mungkin memiliki throughput hangat 30.000 unit baca per detik dan 10.000 unit tulis per detik, tetapi Anda masih bisa mengalami pelambatan pada membaca atau menulis sebelum mencapai nilai tersebut. Ini kemungkinan karena partisi panas. Sementara DynamoDB dapat mempertahankan penskalaan untuk mendukung throughput yang hampir tidak terbatas, setiap partisi individu dibatasi hingga 1.000 unit tulis per detik dan 3.000 unit baca per detik. Jika aplikasi Anda mendorong terlalu banyak lalu lintas ke sebagian kecil partisi tabel, pelambatan dapat terjadi bahkan sebelum Anda mencapai nilai throughput hangat tabel. Kami merekomendasikan mengikuti praktik terbaik DynamoDB untuk memastikan skalabilitas yang mulus dan menghindari partisi panas.

Throughput hangat untuk tabel yang disediakan

Pertimbangkan tabel yang disediakan yang memiliki throughput hangat 30.000 unit baca per detik dan 10.000 unit tulis per detik tetapi saat ini memiliki throughput yang disediakan sebesar 4.000 RCU dan 8.000 WCU. Anda dapat langsung menskalakan throughput yang disediakan tabel hingga 30.000 RCU atau 10.000 WCU dengan memperbarui pengaturan throughput yang disediakan.

Saat Anda meningkatkan throughput yang disediakan di luar nilai-nilai ini, throughput hangat akan secara otomatis menyesuaikan dengan nilai baru yang lebih tinggi, karena Anda telah menetapkan throughput puncak baru. Misalnya, jika Anda mengatur throughput yang disediakan menjadi 50.000 RCU, throughput hangat akan meningkat menjadi 50.000 unit baca per detik.

```
"ProvisionedThroughput":
    {
        "ReadCapacityUnits": 4000,
        "WriteCapacityUnits": 8000
    }
"WarmThroughput":
    {
        "ReadUnitsPerSecond": 30000,
        "WriteUnitsPerSecond": 10000
    }
```

Throughput hangat untuk meja sesuai permintaan

Tabel on-demand baru dimulai dengan throughput hangat 12.000 unit baca per detik dan 4.000 unit tulis per detik. Meja Anda dapat langsung mengakomodasi lalu lintas berkelanjutan hingga level ini. Jika permintaan Anda melebihi 12.000 unit baca per detik atau 4.000 unit tulis per detik, throughput hangat akan secara otomatis menyesuaikan ke nilai yang lebih tinggi.

```
"WarmThroughput":
    {
        "ReadUnitsPerSecond": 12000,
        "WriteUnitsPerSecond": 4000
    }
```

Throughput hangat untuk tabel sesuai permintaan dengan throughput maksimum yang dikonfigurasi

Pertimbangkan tabel sesuai permintaan dengan throughput hangat 30.000 unit baca per detik tetapi dengan throughput maksimum yang dikonfigurasi pada 5.000 unit permintaan baca (RRU). Dalam skenario ini, throughput tabel akan dibatasi hingga maksimum 5.000 RRU yang Anda tetapkan. Setiap permintaan throughput yang melebihi maksimum ini akan dibatasi. Namun, Anda dapat memodifikasi throughput maksimum khusus tabel kapan saja berdasarkan kebutuhan aplikasi Anda.

```
"OnDemandThroughput":
    {
```

```
        "MaxReadRequestUnits": 5000,
        "MaxWriteRequestUnits": 4000
    }
"WarmThroughput":
    {
        "ReadUnitsPerSecond": 30000,
        "WriteUnitsPerSecond": 10000
    }
```

DynamoDB burst dan kapasitas adaptif

Untuk meminimalkan throttling karena pengecualian throughput, DynamoDB menggunakan kapasitas burst untuk menangani lonjakan penggunaan. DynamoDB menggunakan kapasitas adaptif untuk membantu mengakomodasi pola akses yang tidak merata.

Kapasitas lonjakan

DynamoDB memberikan beberapa fleksibilitas untuk penyediaan throughput Anda dengan kapasitas lonjakan. Setiap kali Anda tidak sepenuhnya menggunakan throughput yang tersedia, DynamoDB menyimpan sebagian dari kapasitas yang tidak terpakai itu untuk semburan throughput nanti untuk menangani lonjakan penggunaan. Dengan kapasitas lonjakan, permintaan baca atau tulis yang tidak terduga dapat berhasil ketika mereka semestinya di-throttle.

DynamoDB saat ini mempertahankan hingga lima menit (300 detik) kapasitas baca dan tulis yang tidak terpakai. Selama ledakan aktivitas baca atau tulis sesekali, unit kapasitas ekstra ini dapat dikonsumsi dengan cepat — bahkan lebih cepat daripada kapasitas throughput yang disediakan per detik yang telah Anda tetapkan untuk tabel Anda.

DynamoDB juga dapat menggunakan kapasitas lonjakan untuk pemeliharaan di latar belakang dan tugas lain tanpa pemberitahuan sebelumnya.

Perhatikan bahwa detail kapasitas lonjakan mungkin berubah di masa mendatang.

Kapasitas adaptif

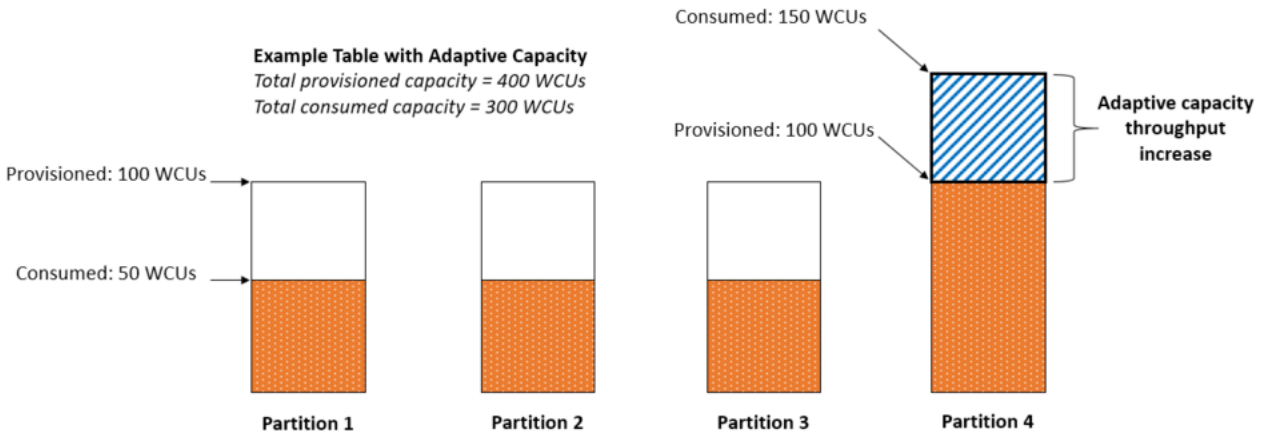
DynamoDB secara otomatis mendistribusikan data Anda di seluruh partisi, yang disimpan di beberapa server di file. AWS Cloud Tidak selalu mungkin untuk mendistribusikan aktivitas membaca dan menulis secara merata sepanjang waktu. Ketika akses data tidak seimbang, partisi "panas" dapat menerima volume lalu lintas baca dan tulis yang lebih tinggi daripada partisi lain. Karena operasi

baca dan tulis pada partisi dikelola secara independen, pelambatan akan terjadi jika satu partisi menerima lebih dari 3000 operasi baca atau lebih dari 1000 operasi tulis. Kapasitas adaptif bekerja dengan meningkatkan kapasitas throughput secara otomatis untuk partisi yang menerima lebih banyak lalu lintas.

Untuk lebih mengakomodasi pola akses yang tidak merata, kapasitas adaptif DynamoDB memungkinkan aplikasi Anda untuk terus membaca dan menulis ke partisi panas tanpa mengalami throttling, selama lalu lintas tidak melebihi total kapasitas yang disediakan tabel atau kapasitas maksimum partisi. Kapasitas adaptif meningkatkan kapasitas throughput secara otomatis dan langsung untuk partisi yang menerima lebih banyak lalu lintas.

Diagram berikut menggambarkan cara kerja kapasitas adaptif. Tabel contoh disediakan dengan 400 dibagi WCUs secara merata di empat partisi, memungkinkan setiap partisi mempertahankan hingga 100 per detik. WCUs Partisi 1, 2, dan 3 masing-masing menerima lalu lintas tulis 50WCU/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.

Kapasitas adaptif DynamoDB merespons dengan meningkatkan kapasitas partisi 4 sehingga dapat mempertahankan beban kerja yang lebih tinggi yaitu 150 tanpa dibatasi. WCU/sec



Kapasitas adaptif diaktifkan secara otomatis untuk setiap tabel DynamoDB, tanpa biaya tambahan. Anda tidak perlu mengaktifkan atau menonaktifkannya secara eksplisit.

Mengisolasi item yang sering diakses

Jika aplikasi Anda mendorong lalu lintas tinggi secara tidak proporsional ke satu atau beberapa item, kapasitas adaptif akan menyeimbangkan kembali partisi Anda sehingga item yang sering diakses tidak berada di partisi yang sama. Isolasi item yang sering diakses ini mengurangi kemungkinan

throttling permintaan karena beban kerja Anda melebihi kuota throughput pada satu partisi. Anda juga dapat memecah koleksi item menjadi beberapa segmen berdasarkan kunci urutan, selama koleksi item tersebut bukan lalu lintas yang dilacak oleh peningkatan atau penurunan kunci urutan yang monoton.

Jika aplikasi Anda terus-menerus mendorong lalu lintas yang tinggi ke satu item, kapasitas adaptif mungkin menyeimbangkan kembali data Anda sehingga partisi hanya berisi item tunggal yang sering diakses tersebut. Dalam hal ini, DynamoDB dapat mengirimkan throughput hingga maksimum partisi RCUs 3.000 dan WCUs 1.000 ke kunci utama item tunggal itu. Kapasitas adaptif tidak akan membagi koleksi item di beberapa partisi tabel ketika ada indeks sekunder lokal di tabel.

Pertimbangan saat mengganti mode kapasitas di DynamoDB

Saat membuat tabel DynamoDB, Anda harus memilih mode kapasitas sesuai permintaan atau yang disediakan.

Anda dapat mengganti tabel dari mode sesuai permintaan ke mode kapasitas yang disediakan kapan saja. Saat Anda melakukan beberapa sakelar di antara mode kapasitas, kondisi berikut berlaku:

- Anda dapat mengganti tabel yang baru dibuat dalam mode sesuai permintaan ke mode kapasitas yang disediakan kapan saja. Namun, Anda hanya dapat mengubahnya kembali ke mode sesuai permintaan 24 jam setelah stempel waktu pembuatan tabel.
- Anda dapat mengganti tabel yang ada dalam mode sesuai permintaan ke mode kapasitas yang disediakan kapan saja. Namun, Anda hanya dapat mengubahnya kembali ke mode sesuai permintaan 24 jam setelah stempel waktu terakhir yang menunjukkan peralihan ke sesuai permintaan.

Topik

- Beralih dari mode kapasitas yang disediakan ke mode kapasitas sesuai permintaan
- Beralih dari mode kapasitas sesuai permintaan ke mode kapasitas yang disediakan

Beralih dari mode kapasitas yang disediakan ke mode kapasitas sesuai permintaan

Dalam mode yang disediakan, Anda mengatur kapasitas baca dan tulis berdasarkan kebutuhan aplikasi yang Anda harapkan. Saat Anda memperbarui tabel dari mode yang disediakan ke mode

sesuai permintaan, Anda tidak perlu menentukan berapa banyak throughput baca dan tulis yang Anda harapkan untuk dijalankan aplikasi. DynamoDB on-demand menawarkan harga pay-per-request sederhana untuk permintaan baca dan tulis sehingga Anda hanya membayar untuk apa yang Anda gunakan, sehingga mudah untuk menyeimbangkan biaya dan kinerja. Anda dapat mengonfigurasi throughput baca atau tulis maksimum (atau keduanya) untuk tabel sesuai permintaan individu dan indeks sekunder global terkait untuk membantu menjaga biaya dan penggunaan tetap terbatas. Untuk informasi selengkapnya tentang menyetel throughput maksimum untuk tabel atau indeks tertentu, lihatDynamoDB throughput maksimum untuk tabel sesuai permintaan.

Saat Anda beralih dari mode kapasitas yang disediakan ke mode kapasitas sesuai permintaan, DynamoDB membuat beberapa perubahan pada struktur tabel dan partisi Anda. Proses ini dapat memakan waktu beberapa menit. Selama periode peralihan, tabel Anda memberikan throughput yang konsisten dengan jumlah unit kapasitas tulis dan kapasitas baca yang disediakan sebelumnya.

Throughput awal untuk mode kapasitas sesuai permintaan

Jika Anda baru saja mengalihkan tabel yang ada ke mode kapasitas sesuai permintaan untuk pertama kalinya, tabel memiliki pengaturan puncak sebelumnya berikut, meskipun tabel sebelumnya tidak melayani lalu lintas menggunakan mode kapasitas sesuai permintaan.

Berikut ini adalah contoh skenario yang mungkin:

- Setiap tabel yang disediakan dikonfigurasi di bawah 4000 WCU dan 12.000 RCU, yang belum pernah disediakan sebelumnya untuk lebih. Saat Anda mengganti tabel ini ke on-demand untuk pertama kalinya, DynamoDB akan memastikannya ditingkatkan untuk secara instan mempertahankan setidaknya 4.000 unit tulis dan 12.000 unit baca/detik. units/sec
- Tabel yang disediakan dikonfigurasi sebagai 8.000 WCU dan 24.000 RCU. Ketika Anda mengganti tabel ini ke on-demand, itu akan terus dapat mempertahankan setidaknya 8.000 tulis units/sec dan 24.000 membaca units/sec setiap saat.
- Tabel yang disediakan dikonfigurasi dengan 8.000 WCU dan 24.000 RCU, yang mengkonsumsi 6.000 tulis units/sec dan 18.000 bacaan untuk periode yang berkelanjutan. units/sec Ketika Anda mengganti tabel ini ke on-demand, itu akan terus dapat mempertahankan setidaknya 8.000 unit tulis units/sec dan 24.000 unit baca/detik. Lalu lintas sebelumnya memungkinkan tabel untuk mempertahankan tingkat lalu lintas yang jauh lebih tinggi tanpa throttling.
- Tabel yang sebelumnya tersedia 10.000 WCU dan 10.000 RCU, namun saat ini tersedia 10 RCU dan 10 WCU. Ketika Anda mengganti tabel ini ke on-demand, itu akan dapat mempertahankan setidaknya 10.000 unit tulis units/sec dan 10.000 unit baca/detik.

Pengaturan penskalaan otomatis

Saat Anda memperbarui tabel dari mode yang ditetapkan ke mode sesuai permintaan:

- Jika Anda menggunakan konsol, semua pengaturan penskalaan otomatis (jika ada) akan dihapus.
- Jika Anda menggunakan AWS CLI atau AWS SDK, semua pengaturan penskalaan otomatis Anda akan dipertahankan. Pengaturan ini dapat diterapkan ketika Anda memperbarui tabel lagi ke mode penagihan yang ditetapkan.

Beralih dari mode kapasitas sesuai permintaan ke mode kapasitas yang disediakan

Saat beralih dari mode kapasitas sesuai permintaan kembali ke mode kapasitas yang disediakan, tabel Anda memberikan throughput yang konsisten dengan puncak sebelumnya yang dicapai ketika tabel diatur ke mode kapasitas sesuai permintaan.

Mengelola kapasitas

Saat memperbarui tabel dari mode sesuai permintaan ke mode yang ditetapkan, pertimbangkan hal berikut:

- Jika Anda menggunakan AWS CLI atau AWS SDK, pilih pengaturan kapasitas yang disediakan yang tepat untuk tabel dan indeks sekunder global dengan menggunakan Amazon CloudWatch untuk melihat konsumsi historis (`ConsumedWriteCapacityUnits`dan `ConsumedReadCapacityUnits` metrik) Anda guna menentukan setelan throughput baru.

  > **Note**
  >
  > Jika Anda memindahkan tabel global ke mode yang ditetapkan, amati konsumsi maksimum di seluruh replika regional untuk tabel dasar global dan indeks sekunder saat membuat pengaturan throughput baru.

- Jika Anda beralih dari mode sesuai permintaan kembali ke mode yang disediakan, pastikan untuk mengatur unit awal yang disediakan cukup tinggi untuk menangani tabel atau kapasitas indeks Anda selama transisi.

Mengelola penskalaan otomatis

Saat Anda memperbarui tabel dari mode sesuai permintaan ke mode yang ditetapkan:

- Jika Anda menggunakan konsol, sebaiknya aktifkan penskalaan otomatis dengan default berikut:
  - Pemanfaatan target: 70%
  - Kapasitas minimum yang disediakan: 5 unit
  - Kapasitas maksimum yang disediakan: Wilayah maksimum
- Jika Anda menggunakan AWS CLI atau SDK, pengaturan penskalaan otomatis Anda sebelumnya (jika ada) dipertahankan.

Pemrograman dengan DynamoDB dan AWS SDKs

Bagian ini mencakup topik terkait developer. Jika Anda ingin menjalankan contoh kode, lihat Menjalankan contoh kode dalam Panduan Developer ini.

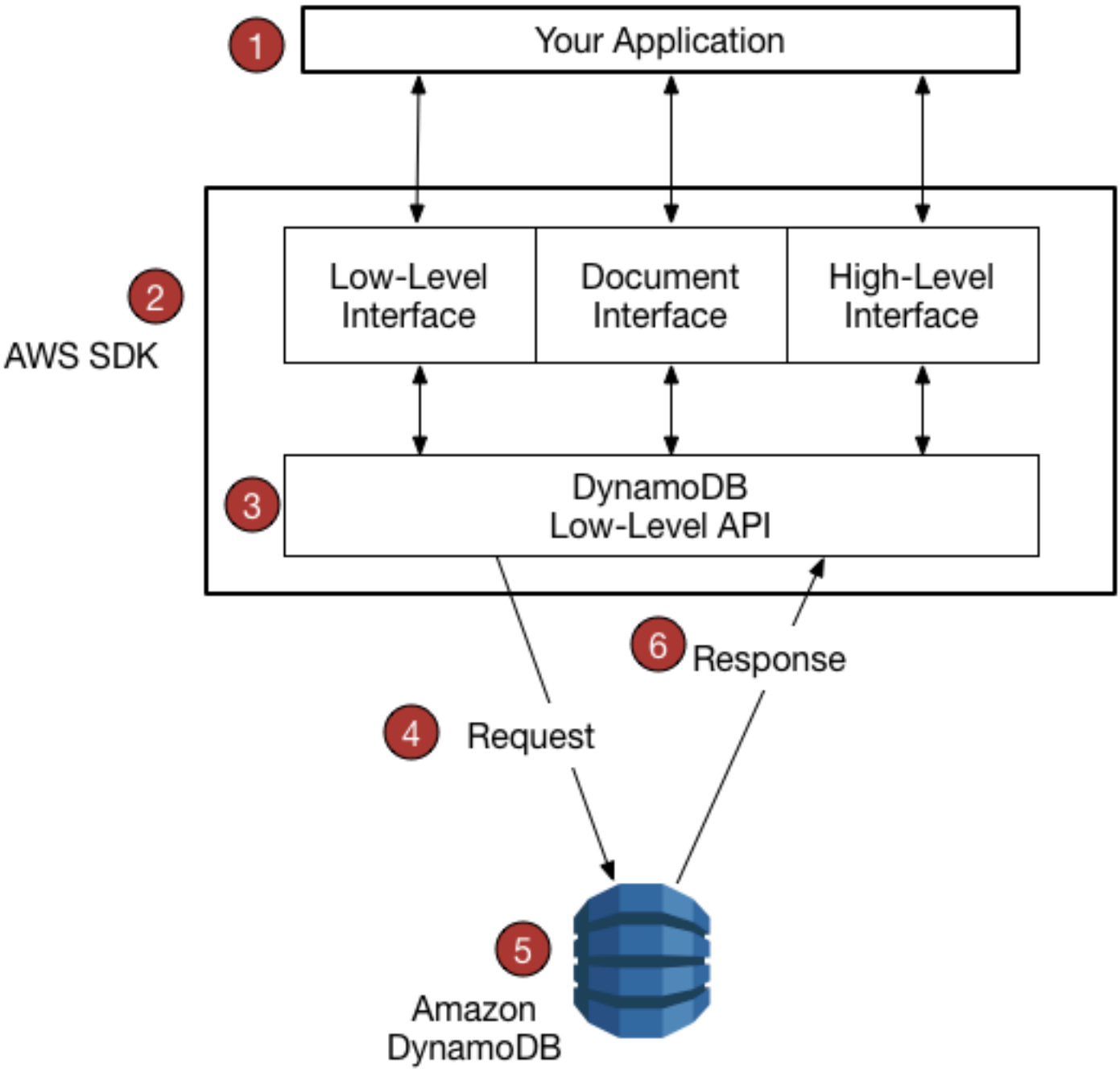
> **Note**
>
> Pada bulan Desember 2017, AWS memulai proses migrasi semua titik akhir Amazon DynamoDB untuk menggunakan sertifikat aman yang dikeluarkan oleh Amazon Trust Services (ATS). Untuk informasi selengkapnya, lihat Memecahkan masalah pembuatan koneksi SSL/TLS dengan DynamoDB.

Topik

- Ikhtisar dukungan AWS SDK untuk DynamoDB
- Pemrograman Amazon DynamoDB dengan Python dan Boto3
- Pemrograman Amazon DynamoDB dengan JavaScript
- Pemrograman DynamoDB dengan AWS SDK for Java 2.x
- Penanganan kesalahan dengan DynamoDB
- Menggunakan DynamoDB dengan SDK AWS

Ikhtisar dukungan AWS SDK untuk DynamoDB

Diagram berikut memberikan gambaran tingkat tinggi pemrograman aplikasi Amazon DynamoDB menggunakan file. AWS SDKs



1. Anda menulis aplikasi menggunakan AWS SDK untuk bahasa pemrograman Anda.
2. Setiap AWS SDK menyediakan satu atau lebih antarmuka terprogram untuk bekerja dengan DynamoDB. Antarmuka spesifik yang tersedia bergantung pada bahasa pemrograman dan AWS SDK yang Anda gunakan. Pilihannya meliputi:
   - Antarmuka tingkat rendah yang bekerja dengan DynamoDB
   - Antarmuka dokumen yang bekerja dengan DynamoDB
   - Antarmuka persistensi objek yang bekerja dengan DynamoDB

- Antarmuka Tingkat Tinggi

3. AWS SDK membuat permintaan HTTP (S) untuk digunakan dengan API DynamoDB tingkat rendah.
4. AWS SDK mengirimkan permintaan ke titik akhir DynamoDB.
5. DynamoDB menjalankan permintaan. Jika permintaan berhasil, DynamoDB mengembalikan kode respons HTTP 200 (OK). Jika permintaan tidak berhasil, DynamoDB mengembalikan kode galat dan pesan kesalahan HTTP.
6. AWS SDK memproses respons dan menyebarkannya kembali ke aplikasi Anda.

Masing-masing AWS SDKs menyediakan layanan penting untuk aplikasi Anda, termasuk yang berikut:

- Memformat permintaan HTTP(S) dan menyerialisasi parameter permintaan.
- Menghasilkan tanda tangan kriptografi untuk setiap permintaan.
- Meneruskan permintaan ke titik akhir DynamoDB dan menerima respons dari DynamoDB.
- Mengekstraksi hasil dari respons tersebut.
- Menerapkan logika coba lagi dasar dalam kasus kesalahan.

Anda tidak perlu menulis kode untuk semua tugas ini.

> **Note**
>
> Untuk informasi selengkapnya AWS SDKs, termasuk petunjuk penginstalan dan dokumentasi, lihat Alat untuk Amazon Web Services.

Dukungan SDK dari endpoint berbasis AWS akun

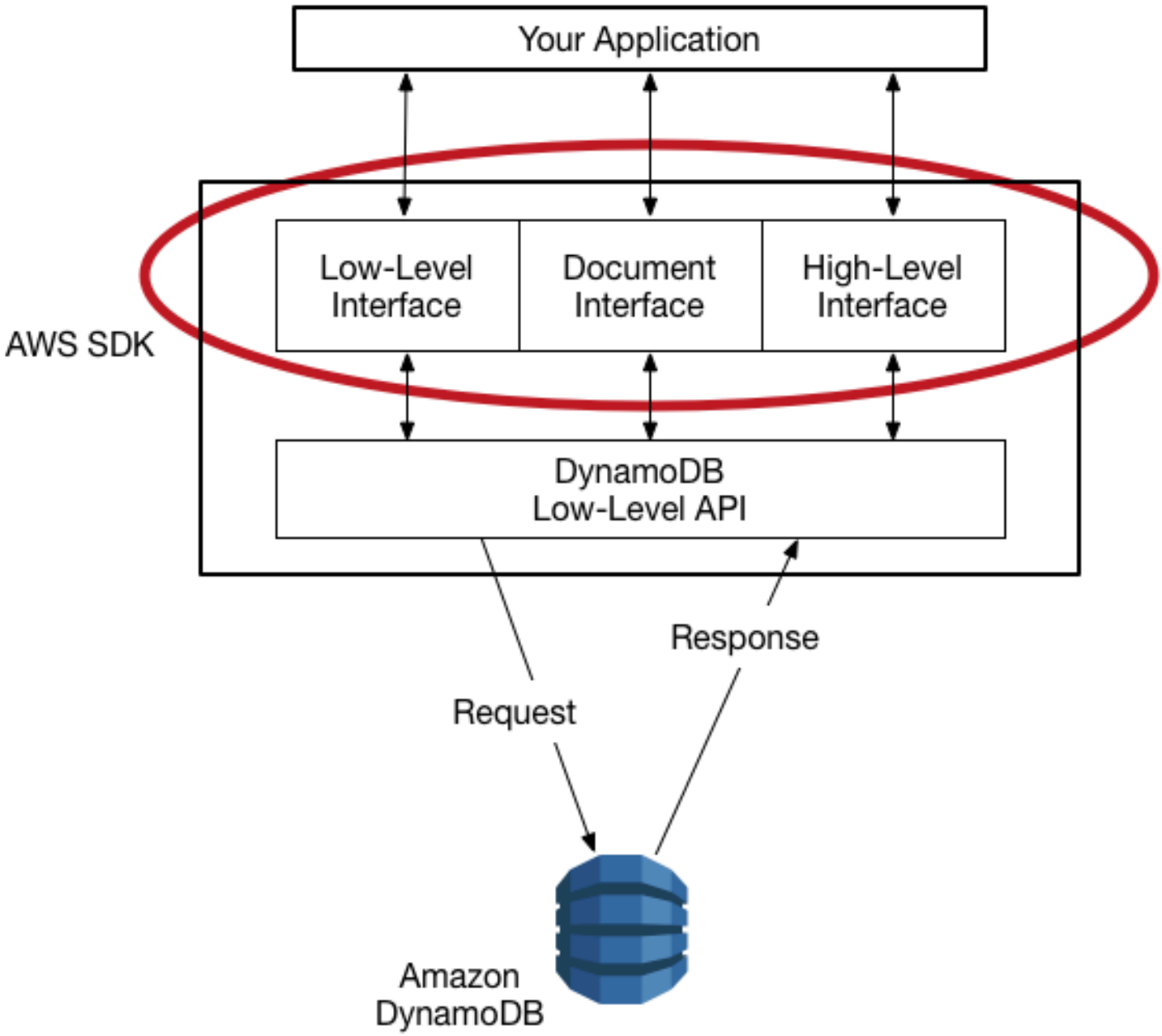
AWS meluncurkan dukungan SDK untuk AWS titik akhir berbasis akun untuk DynamoDB, dimulai dengan SDK for AWS Java V1 pada 4 September 2024. Titik akhir baru ini membantu AWS memastikan kinerja dan skalabilitas tinggi. Yang diperbarui SDKs akan secara otomatis menggunakan titik akhir baru, yang memiliki format`https://(account-id).ddb.(region).amazonaws.com`.

Jika Anda menggunakan satu instance klien SDK untuk membuat permintaan ke beberapa akun, aplikasi Anda akan memiliki lebih sedikit kesempatan untuk menggunakan kembali koneksi. AWS

merekomendasikan untuk memodifikasi aplikasi Anda untuk terhubung ke lebih sedikit akun per instans klien SDK. Alternatifnya adalah menyetel klien SDK Anda untuk terus menggunakan titik akhir Regional menggunakan `ACCOUNT_ID_ENDPOINT_MODE` pengaturan, seperti yang didokumentasikan dalam Panduan Referensi Alat AWS SDKs dan Alat.

Antarmuka terprogram yang bekerja dengan DynamoDB

Setiap AWS SDK menyediakan satu atau lebih antarmuka program untuk bekerja dengan Amazon DynamoDB. Antarmuka ini memiliki rentang dari pembungkus DynamoDB level rendah sederhana hingga lapisan persistensi berorientasi objek. Antarmuka yang tersedia bervariasi tergantung pada AWS SDK dan bahasa pemrograman yang Anda gunakan.



Bagian berikut menyoroti beberapa antarmuka yang tersedia, menggunakan AWS SDK untuk Java sebagai contoh. (Tidak semua antarmuka tersedia di semua AWS SDKs.)

Topik

- Antarmuka tingkat rendah yang bekerja dengan DynamoDB
- Antarmuka dokumen yang bekerja dengan DynamoDB
- Antarmuka persistensi objek yang bekerja dengan DynamoDB

Antarmuka tingkat rendah yang bekerja dengan DynamoDB

Setiap AWS SDK khusus bahasa menyediakan antarmuka tingkat rendah untuk Amazon DynamoDB, dengan metode yang sangat mirip dengan permintaan API DynamoDB tingkat rendah.

Dalam beberapa kasus, Anda akan perlu untuk mengidentifikasi jenis data dari atribut-atribut tersebut menggunakan Deskriptor jenis data, seperti `S` untuk string, atau `N` untuk angka.

> **Note**
>
> Antarmuka tingkat rendah tersedia dalam setiap AWS SDK khusus bahasa.

Program Java berikut menggunakan antarmuka AWS SDK untuk Java tingkat rendah.

Contoh antarmuka tingkat rendah

```
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);
        }
    }
}
```

Antarmuka dokumen yang bekerja dengan DynamoDB

Banyak yang AWS SDKs menyediakan antarmuka dokumen, memungkinkan Anda untuk melakukan operasi bidang data (membuat, membaca, memperbarui, menghapus) pada tabel dan indeks. Dengan antarmuka dokumen, Anda tidak perlu menentukan Deskriptor jenis data. Jenis data yang tersirat oleh semantik data itu sendiri. Ini AWS SDKs juga menyediakan metode untuk dengan mudah mengonversi dokumen JSON ke dan dari tipe data Amazon DynamoDB asli.

> **Note**
>
> Antarmuka dokumen tersedia di AWS SDKs untuk Java, .NET, Node.js, dan JavaScriptSDK.

Program Java berikut menggunakan antarmuka dokumen AWS SDK untuk Java. Program ini menciptakan objek `Table` yang merepresentasikan tabel `Music`, lalu meminta objek tersebut untuk menggunakan `GetItem` guna mengambil lagu. Program ini selanjutnya mencetak tahun lagu tersebut dirilis.

Kelas `com.amazonaws.services.dynamodbv2.document.DynamoDB` mengimplementasikan antarmuka dokumen DynamoDB. Perhatikan bagaimana `DynamoDB` berfungsi sebagai pembungkus di sekitar klien tingkat rendah (`AmazonDynamoDB`).

Contoh antarmuka dokumen

```
package com.amazonaws.codesamples.gsg;

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.GetItemOutcome;
import com.amazonaws.services.dynamodbv2.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);

    }
}
```

Antarmuka persistensi objek yang bekerja dengan DynamoDB

Beberapa AWS SDKs menyediakan antarmuka persistensi objek di mana Anda tidak secara langsung melakukan operasi bidang data. Sebagai gantinya, Anda akan membuat objek yang mewakili item dalam tabel dan indeks Amazon DynamoDB, dan hanya berinteraksi dengan objek tersebut. Ini memungkinkan Anda menulis kode yang berfokus pada objek, bukan kode yang berfokus pada basis data.

> **Note**
>
> Antarmuka persistensi objek tersedia di AWS SDKs untuk Java dan .NET. Untuk informasi selengkapnya, lihat Antarmuka pemrograman tingkat tinggi untuk DynamoDB DynamoDB.

Contoh antarmuka ketekunan objek

```
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();
    }
}
```

Antarmuka pemrograman tingkat tinggi untuk DynamoDB

AWS SDKs Menyediakan aplikasi dengan antarmuka tingkat rendah untuk bekerja dengan Amazon DynamoDB. Metode dan kelas sisi klien ini berkaitan langsung dengan DynamoDB API tingkat rendah. Namun, banyak developer mengalami putusnya koneksi, atau ketidakcocokan impedansi,

ketika mereka perlu memetakan jenis data yang kompleks untuk item dalam tabel basis data. Dengan antarmuka basis data tingkat rendah, developer harus menulis metode untuk membaca atau menulis data objek ke tabel basis data, dan sebaliknya. Jumlah kode tambahan yang diperlukan untuk setiap kombinasi jenis objek dan tabel basis data dapat terlihat sangat banyak.

Untuk menyederhanakan pengembangan, AWS SDKs untuk Java dan .NET menyediakan antarmuka tambahan dengan tingkat abstraksi yang lebih tinggi. Antarmuka tingkat yang lebih tinggi untuk DynamoDB memungkinkan Anda menentukan hubungan di antara objek dalam program Anda dan tabel basis data yang menyimpan data objek tersebut. Setelah menentukan pemetaan ini, Anda memanggil metode objek sederhana seperti `save`, `load`, atau `delete`, dan operasi DynamoDB tingkat rendah yang mendasarinya akan dipanggil secara otomatis atas nama Anda. Ini memungkinkan Anda menulis kode yang berfokus pada objek, bukan kode yang berfokus pada basis data.

Antarmuka pemrograman tingkat tinggi untuk DynamoDB tersedia di untuk Java dan .NET. AWS SDKs

Java

- Java 1.x: Dinamo DBMapper
- Java 2.x: DynamoDB Enhanced Client

.NET

- Bekerja dengan model dokumen.NET di DynamoDB
- Bekerja dengan model persistensi objek.NET dan DynamoDB

Java 1.x: Dinamo DBMapper

> **Note**
>
> SDK for Java memiliki dua versi: 1.x dan 2.x. end-of-supportUntuk 1.x diumumkan pada 12 Januari 2024. Itu akan dan akan jatuh tempo pada 31 Desember 2025. end-of-support Untuk pengembangan baru, kami sangat menyarankan Anda menggunakan 2.x.

AWS SDK untuk Java Ini menyediakan `DynamoDBMapper` kelas, memungkinkan Anda untuk memetakan kelas sisi klien Anda ke tabel Amazon DynamoDB. Untuk menggunakan

`DynamoDBMapper`, tentukan hubungan antara item dalam tabel DynamoDB dan instans objek yang sesuai dalam kode Anda. Kelas `DynamoDBMapper` memungkinkan Anda melakukan berbagai operasi buat, baca, perbarui, dan hapus (CRUD) pada item, dan menjalankan kueri serta memindai tabel.

Topik

- Kelas Dynamo DBMapper
- Tipe data yang didukung untuk Dynamo DBMapper untuk Java
- Anotasi Java untuk DynamoDB
- Pengaturan konfigurasi opsional untuk Dynamo DBMapper
- DynamoDB dan penguncian optimis dengan nomor versi
- Memetakan data arbitrer di DynamoDB
- Contoh Dynamo DBMapper

> **Note**
>
> Kelas `DynamoDBMapper` tidak memungkinkan Anda membuat, memperbarui, atau menghapus tabel. Untuk melakukan tugas tersebut, gunakan antarmuka SDK untuk Java tingkat rendah.

SDK untuk Java menyediakan serangkaian jenis anotasi agar Anda dapat memetakan kelas ke tabel. Misalnya, pertimbangkan tabel `ProductCatalog` yang memiliki `Id` sebagai kunci partisi.

```
ProductCatalog(Id, ...)
```

Anda dapat memetakan kelas di aplikasi klien ke tabel `ProductCatalog` seperti yang ditunjukkan dalam kode Java berikut. Kode ini menentukan plain old Java object (POJO) bernama `CatalogItem`, yang menggunakan anotasi untuk memetakan bidang objek ke nama atribut DynamoDB.

Example

```
package com.amazonaws.codesamples;

import java.util.Set;
```

```
import com.amazonaws.services.dynamodbv2.datamodeling.DynamoDBAttribute;
import com.amazonaws.services.dynamodbv2.datamodeling.DynamoDBHashKey;
import com.amazonaws.services.dynamodbv2.datamodeling.DynamoDBIgnore;
import com.amazonaws.services.dynamodbv2.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; }
}
```

Dalam kode sebelumnya, anotasi `@DynamoDBTable` memetakan kelas `CatalogItem` ke tabel `ProductCatalog`. Anda dapat menyimpan masing-masing instans kelas sebagai item dalam tabel. Dalam definisi kelas, anotasi `@DynamoDBHashKey` memetakan properti `Id` ke kunci primer.

Secara default, properti kelas memetakan atribut nama yang sama dalam tabel. Properti `Title` dan `ISBN` memetakan atribut nama yang sama di tabel.

Anotasi `@DynamoDBAttribute` bersifat opsional saat nama atribut DynamoDB cocok dengan nama properti yang dinyatakan di kelas. Jika namanya berbeda, gunakan anotasi ini dengan parameter `attributeName` untuk menentukan atribut DynamoDB yang sesuai dengan properti ini.

Dalam contoh sebelumnya, `@DynamoDBAttribute` anotasi ditambahkan ke setiap properti untuk memastikan bahwa nama properti cocok persis dengan tabel yang dibuat pada langkah sebelumnya, dan agar konsisten dengan nama atribut yang digunakan dalam contoh kode lain dalam panduan ini.

Definisi kelas Anda dapat memiliki properti yang tidak dipetakan ke setiap atribut dalam tabel. Anda mengidentifikasi properti ini dengan menambahkan anotasi `@DynamoDBIgnore`. Dalam contoh sebelumnya, properti `SomeProp` ditandai dengan anotasi `@DynamoDBIgnore`. Saat Anda mengunggah instans `CatalogItem` ke tabel, instans `DynamoDBMapper` tidak menyertakan properti `SomeProp`. Selain itu, pemeta tidak mengembalikan atribut ini ketika Anda mengambil item dari tabel.

Setelah menentukan kelas pemetaan, Anda dapat menggunakan metode `DynamoDBMapper` untuk menulis instans kelas tersebut untuk item yang sesuai dalam tabel `Catalog`. Contoh kode berikut mendemonstrasikan teknik ini.

```
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);
```

Contoh kode berikut menunjukkan cara mengambil item dan mengakses beberapa atributnya.

```
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` menawarkan cara natural yang intuitif untuk menggunakan data DynamoDB dalam Java. Ini juga menyediakan beberapa fitur bawaan, seperti penguncian optimis, transaksi ACID, nilai kunci urutan dan kunci partisi yang dibuat secara otomatis, serta objek penentuan versi.

Kelas Dynamo DBMapper

Kelas `DynamoDBMapper` adalah titik masuk ke Amazon DynamoDB. Kelas ini menyediakan akses ke titik akhir DynamoDB dan memungkinkan Anda mengakses data Anda dalam berbagai tabel. Kelas ini juga memungkinkan Anda melakukan berbagai operasi buat, baca, perbarui, dan hapus (CRUD) pada item, dan menjalankan kueri serta memindai tabel. Kelas ini menyediakan metode berikut agar berfungsi dengan DynamoDB.

Untuk dokumentasi Javadoc yang sesuai, lihat Dynamo DBMapper di Referensi API.AWS SDK untuk Java

Topik

- save
- muat
- hapus
- kueri
- queryPage
- scan
- scanPage
- parallelScan
- batchSave
- batchLoad
- batchDelete
- batchWrite
- transactionWrite

- transactionLoad
- count
- generateCreateTablePermintaan
- createS3Link
- GetS3 ClientCache

save

Menyimpan objek tertentu ke tabel. Objek yang ingin Anda simpan adalah satu-satunya parameter yang diperlukan untuk metode ini. Anda dapat memberikan parameter konfigurasi opsional menggunakan objek `DynamoDBMapperConfig`.

Jika item yang memiliki kunci primer yang sama tidak ada, metode ini akan membuat item baru dalam tabel. Jika item yang memiliki kunci primer yang sama ada, item yang ada akan diperbarui. Jika kunci partisi dan kunci urutan berupa String jenis dan dianotasikan dengan `@DynamoDBAutoGeneratedKey`, kunci tersebut diberi pengidentifikasi unik universal acak (UUID) jika tidak diinisialisasi. Bidang versi yang diberi anotasi dengan `@DynamoDBVersionAttribute` bertambah satu. Selain itu, jika bidang versi diperbarui atau kunci dibuat, objek yang diteruskan akan diperbarui sebagai hasil operasi.

Secara default, hanya atribut yang sesuai dengan properti kelas yang dipetakan yang akan diperbarui. Atribut tambahan yang ada pada item tidak akan terpengaruh. Namun, jika menentukan `SaveBehavior.CLOBBER`, Anda dapat memaksa item untuk ditimpa sepenuhnya.

```
DynamoDBMapperConfig config = DynamoDBMapperConfig.builder()
    .withSaveBehavior(DynamoDBMapperConfig.SaveBehavior.CLOBBER).build();

mapper.save(item, config);
```

Jika penentuan versi diaktifkan, versi item sisi klien dan sisi server harus cocok. Namun, versi tidak harus cocok jika opsi `SaveBehavior.CLOBBER` digunakan. Untuk informasi selengkapnya tentang penentuan versi, lihat DynamoDB dan penguncian optimis dengan nomor versi.

muat

Mengambil item dari tabel. Anda harus menyediakan kunci primer item yang ingin Anda ambil. Anda dapat memberikan parameter konfigurasi opsional menggunakan objek `DynamoDBMapperConfig`. Misalnya, Anda secara opsional dapat meminta bacaan sangat konsisten untuk memastikan bahwa

metode ini hanya mengambil nilai item terbaru seperti yang ditunjukkan dalam pernyataan Java berikut.

```
DynamoDBMapperConfig config = DynamoDBMapperConfig.builder()
    .withConsistentReads(DynamoDBMapperConfig.ConsistentReads.CONSISTENT).build();

CatalogItem item = mapper.load(CatalogItem.class, item.getId(), config);
```

Secara default, DynamoDB akan mengembalikan item yang memiliki nilai yang akhirnya konsisten. Untuk informasi tentang model konsistensi akhir DynamoDB, lihat DynamoDB membaca konsistensi.

hapus

Menghapus item dari tabel. Anda harus meneruskan instans objek dari kelas yang dipetakan.

Jika penentuan versi diaktifkan, versi item sisi klien dan sisi server harus cocok. Namun, versi tidak harus cocok jika opsi `SaveBehavior.CLOBBER` digunakan. Untuk informasi selengkapnya tentang penentuan versi, lihat DynamoDB dan penguncian optimis dengan nomor versi.

kueri

Mengkueri tabel atau indeks sekunder.

Misalnya, Anda memiliki tabel, `Reply`, yang menyimpan balasan utas forum. Setiap subjek utas dapat berisi nol atau beberapa balasan. Kunci primer tabel `Reply` terdiri dari bidang `Id` dan `ReplyDateTime`, dengan `Id` sebagai kunci partisi dan `ReplyDateTime` sebagai kunci urutan kunci primer.

```
Reply ( Id, ReplyDateTime, ... )
```

Misalnya, Anda membuat pemetaan antara kelas `Reply` dan tabel `Reply` yang sesuai di DynamoDB. Kode Java berikut menggunakan `DynamoDBMapper` untuk menemukan semua balasan dalam dua minggu terakhir untuk subjek utas tertentu.

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);
```

Kueri ini mengembalikan koleksi objek `Reply`.

Secara default, metode `query` mengembalikan koleksi "lazy-loaded". Metode ini awalnya hanya mengembalikan satu halaman hasil, lalu membuat panggilan layanan untuk halaman berikutnya jika diperlukan. Untuk mendapatkan semua item yang cocok, iterasikan ke koleksi `latestReplies`.

Perhatikan bahwa memanggil metode `size()` pada koleksi akan memuat setiap hasil untuk memberikan jumlah yang akurat. Hal ini dapat mengakibatkan penggunaan banyak throughput yang disediakan, dan bahkan bisa menghabiskan semua memori di JVM Anda pada tabel yang sangat besar.

Untuk mencari indeks, Anda harus membuat model indeks sebagai kelas pemeta terlebih dahulu. Misalkan `Reply` tabel memiliki indeks sekunder global bernama PostedBy-Message-Index. Kunci partisi untuk indeks ini adalah `PostedBy`, dan kunci urutannya adalah `Message`. Definisi kelas untuk item dalam indeks akan tampak seperti berikut ini.

```
@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.
}
```

Anotasi `@DynamoDBTable` menunjukkan bahwa indeks ini terkait dengan tabel `Reply`. `@DynamoDBIndexHashKey`Anotasi menunjukkan kunci partisi (PostedBy) dari indeks, dan `@DynamoDBIndexRangeKey` menunjukkan kunci sortir (Pesan) indeks.

Sekarang, Anda dapat menggunakan `DynamoDBMapper` untuk mengkueri indeks, mengambil subset pesan yang diposting oleh pengguna tertentu. Anda tidak perlu menentukan nama indeks jika Anda tidak memiliki pemetaan yang bertentangan di seluruh tabel dan indeks serta pemetaan sudah dibuat di mapper. Pemeta akan menyimpulkan berdasarkan kunci primer dan kunci urutan. Kode berikut mengkueri indeks sekunder global. Karena indeks sekunder global mendukung bacaan akhir konsisten tetapi bukan bacaan sangat konsisten, Anda harus menentukan `withConsistentRead(false)`.

```
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);
```

Kueri ini mengembalikan koleksi objek `PostedByMessage`.

queryPage

Mengkueri tabel atau indeks sekunder dan mengembalikan satu halaman hasil yang cocok. Seperti metode `query`, Anda harus menentukan nilai kunci partisi dan filter kueri yang diterapkan pada atribut kunci urutan. Namun, `queryPage` hanya mengembalikan “halaman” pertama data, yaitu jumlah data yang sesuai dalam ukuran 1 MB

scan

Memindai seluruh tabel atau indeks sekunder. Anda dapat menentukan `FilterExpression` untuk memfilter kumpulan hasil secara opsional.

Misalnya, Anda memiliki tabel, `Reply`, yang menyimpan balasan utas forum. Setiap subjek utas dapat berisi nol atau beberapa balasan. Kunci primer tabel `Reply` terdiri dari bidang `Id` dan `ReplyDateTime`, dengan `Id` sebagai kunci partisi dan `ReplyDateTime` sebagai kunci urutan kunci primer.

```
Reply ( Id, ReplyDateTime, ... )
```

Jika memetakan kelas Java ke tabel `Reply`, Anda dapat menggunakan `DynamoDBMapper` untuk memindai tabel. Misalnya, kode Java berikut memindai seluruh tabel `Reply`, sehingga hanya mengembalikan balasan untuk tahun tertentu.

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);
```

Secara default, metode `scan` mengembalikan koleksi "lazy-loaded". Metode ini awalnya hanya mengembalikan satu halaman hasil, lalu membuat panggilan layanan untuk halaman berikutnya jika diperlukan. Untuk mendapatkan semua item yang cocok, iterasikan ke koleksi `replies`.

Perhatikan bahwa memanggil metode `size()` pada koleksi akan memuat setiap hasil untuk memberikan jumlah yang akurat. Hal ini dapat mengakibatkan penggunaan banyak throughput yang disediakan, dan bahkan bisa menghabiskan semua memori di JVM Anda pada tabel yang sangat besar.

Untuk memindai indeks, Anda harus membuat model indeks sebagai kelas pemeta terlebih dahulu. Misalkan tabel `Reply` memiliki indeks sekunder global bernama `PostedBy-Message-Index`. Kunci partisi untuk indeks ini adalah `PostedBy`, dan kunci urutannya adalah `Message`. Kelas pemeta untuk

indeks ini ditampilkan di bagian kueri. Kelas ini menggunakan anotasi `@DynamoDBIndexHashKey` dan `@DynamoDBIndexRangeKey` untuk menentukan kunci partisi indeks dan kunci urutan.

Contoh kode berikut memindai `PostedBy-Message-Index`. Contoh kode ini tidak menggunakan filter pemindaian, sehingga semua item dalam indeks dikembalikan kepada Anda.

```
DynamoDBScanExpression scanExpression = new DynamoDBScanExpression()
    .withIndexName("PostedBy-Message-Index")
    .withConsistentRead(false);

    List<PostedByMessage> iList =  mapper.scan(PostedByMessage.class, scanExpression);
    Iterator<PostedByMessage> indexItems = iList.iterator();
```

scanPage

Memindai tabel atau indeks sekunder dan mengembalikan satu halaman hasil yang cocok. Seperti halnya metode `scan`, Anda dapat menentukan `FilterExpression` untuk memfilter kumpulan hasil secara opsional. Namun, `scanPage` hanya mengembalikan “halaman” pertama data, yaitu jumlah data yang sesuai dalam ukuran 1 MB.

parallelScan

Melakukan pemindaian paralel dari seluruh tabel atau indeks sekunder. Anda menentukan sejumlah segmen logis untuk tabel, beserta ekspresi pemindaian untuk memfilter hasil. `parallelScan` membagi tugas pemindaian antara beberapa pekerja, satu tugas untuk setiap segmen logis; pekerja akan memproses data secara paralel dan mengembalikan hasilnya.

Contoh kode Java berikut melakukan pemindaian paralel pada tabel `Product`.

```
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

Menyimpan objek ke satu atau beberapa tabel menggunakan satu atau beberapa panggilan ke metode `AmazonDynamoDB.batchWriteItem`. Metode ini tidak memberikan jaminan transaksi.

Kode Java berikut menyimpan dua item (buku) ke tabel `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

Mengambil beberapa item dari satu atau beberapa tabel menggunakan kunci primernya.

Kode Java berikut akan mengambil dua item dari dua tabel yang berbeda.

```
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

Menghapus objek dari satu atau beberapa tabel menggunakan satu atau beberapa panggilan ke metode `AmazonDynamoDB.batchWriteItem`. Metode ini tidak memberikan jaminan transaksi.

Kode Java berikut akan menghapus dua item (buku) dari tabel `ProductCatalog`.

```
Book book1 = mapper.load(Book.class, 901);
Book book2 = mapper.load(Book.class, 902);
mapper.batchDelete(Arrays.asList(book1, book2));
```

batchWrite

Menyimpan objek ke dan menghapus objek dari satu atau beberapa tabel menggunakan satu atau beberapa panggilan ke metode `AmazonDynamoDB.batchWriteItem`. Metode ini tidak memberikan jaminan transaksi atau penentuan versi dukungan (penempatan atau penghapusan bersyarat).

Kode Java berikut menulis item baru ke tabel `Forum`, menulis item baru ke tabel `Thread`, dan menghapus item dari tabel `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

Menyimpan objek ke dan menghapus objek dari satu atau beberapa tabel menggunakan satu panggilan ke metode `AmazonDynamoDB.transactWriteItems`.

Untuk daftar pengecualian khusus transaksi, lihat kesalahan. TransactWriteItems

Untuk informasi selengkapnya tentang transaksi DynamoDB dan jaminan atomisitas, konsistensi, isolasi, dan durabilitas atau atomicity, consistency, isolation, and durability (ACID), lihat Amazon DynamoDB Transactions.

> **Note**
>
> Metode ini tidak mendukung hal berikut:
>
> - Konfigurasi DynamoDBMapper. SaveBehavior.

Kode Java berikut menulis item baru ke setiap tabel `Forum` dan `Thread`, secara transaksional.

```
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

Memuat objek dari satu atau beberapa tabel menggunakan satu panggilan ke metode `AmazonDynamoDB.transactGetItems`.

Untuk daftar pengecualian khusus transaksi, lihat kesalahan. TransactGetItems

Untuk informasi selengkapnya tentang transaksi DynamoDB dan jaminan atomisitas, konsistensi, isolasi, dan durabilitas atau atomicity, consistency, isolation, and durability (ACID), lihat Amazon DynamoDB Transactions.

Kode Java berikut akan memuat satu item dari setiap tabel `Forum` dan `Thread`, secara transaksional.

```
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

Mengevaluasi ekspresi pemindaian yang ditentukan dan mengembalikan jumlah item yang cocok. Tidak ada data item yang dikembalikan.

generateCreateTablePermintaan

Mengurai kelas POJO yang mewakili tabel DynamoDB, dan mengembalikan `CreateTableRequest` untuk tabel tersebut.

createS3Link

Membuat tautan ke objek di Amazon S3. Anda harus menentukan nama bucket dan nama kunci, yang secara unik mengidentifikasi objek di bucket.

Untuk menggunakan `createS3Link`, kelas pemeta Anda harus menentukan metode getter dan setter. Contoh kode berikut mengilustrasikan hal ini dengan menambahkan atribut baru dan metode getter/setter untuk kelas `CatalogItem`.

```
@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;
```

```
    }

...
}
```

Kode Java berikut mendefinisikan item baru yang akan ditulis ke tabel `Product`. Item tersebut termasuk link ke gambar produk; data gambar diunggah ke Amazon S3.

```
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);
```

Kelas `S3Link` menyediakan banyak metode lain untuk memanipulasi objek di Amazon S3. Untuk informasi selengkapnya, lihat Javadocs untuk `S3Link`.

GetS3 ClientCache

Mengembalikan `S3ClientCache` yang mendasarinya untuk mengakses Amazon S3. `S3ClientCache` adalah Peta cerdas untuk objek `AmazonS3Client`. Jika Anda memiliki banyak klien, an `S3ClientCache` dapat membantu Anda menjaga klien tetap terorganisir berdasarkan AWS Wilayah, dan dapat membuat klien Amazon S3 baru sesuai permintaan.

Tipe data yang didukung untuk Dynamo DBMapper untuk Java

Bagian ini menjelaskan jenis data Java primitif, koleksi, dan jenis data arbitrer yang didukung di Amazon DynamoDB.

Amazon DynamoDB mendukung jenis data Java primitif dan kelas pembungkus primitif berikut.

- `String`
- `Boolean`, `boolean`
- `Byte`, `byte`

- `Date` (sebagai string ISO_8601 millisecond-precision, yang beralih ke UTC)
- `Calendar` (sebagai string ISO_8601 millisecond-precision, yang beralih ke UTC)
- `Long`, `long`
- `Integer`, `int`
- `Double`, `double`
- `Float`, `float`
- `BigDecimal`
- `BigInteger`

> **Note**
>
> - Untuk informasi selengkapnya tentang aturan penamaan DynamoDB dan berbagai jenis data yang didukung, lihat Jenis data dan aturan penamaan yang didukung di Amazon DynamoDB.
> - Nilai biner kosong didukung oleh DynamoDBMapper.
> - Nilai String kosong didukung oleh AWS SDK for Java 2.x.
>
> Dalam AWS SDK for Java 1.x, DBMapper Dynamo mendukung pembacaan nilai atribut String kosong, namun, itu tidak akan menulis nilai atribut String kosong karena atribut ini dijatuhkan dari permintaan.

DynamoDB mendukung jenis koleksi Kumpulan, Daftar, dan Peta Java. Tabel berikut merangkum bagaimana jenis Java ini dipetakan untuk jenis DynamoDB.

Jenis Java	Jenis DynamoDB
Semua jenis angka	`N` (jenis angka)
String	`S` (jenis string)
Boolean	`BOOL` (jenis Boolean), 0 atau 1.
ByteBuffer	`B` (jenis biner)

Jenis Java	Jenis DynamoDB
Tanggal	`S` (jenis string). Nilai data disimpan sebagai string berformat ISO-8601.
Jenis koleksi Kumpulan	Jenis `SS` (kumpulan biner), jenis `NS` (kumpulan angka), atau jenis `BS` (kumpulan biner).

Antarmuka `DynamoDBTypeConverter` memungkinkan Anda memetakan jenis data arbitrer Anda sendiri ke jenis data yang secara asli didukung oleh DynamoDB. Untuk informasi selengkapnya, lihat Memetakan data arbitrer di DynamoDB.

Anotasi Java untuk DynamoDB

Bagian ini menjelaskan anotasi yang tersedia untuk memetakan kelas dan properti ke tabel dan atribut di Amazon DynamoDB.

Untuk dokumentasi Javadoc yang sesuai, lihat Ringkasan Jenis Anotasi di Referensi API AWS SDK untuk Java.

> **Note**
>
> Dalam anotasi berikut, hanya `DynamoDBTable` dan `DynamoDBHashKey` yang diperlukan.

Topik

- Dinamo DBAttribute
- Dinamo DBAuto GeneratedKey
- Dinamo DBAuto GeneratedTimestamp
- Dinamo DBDocument
- Kunci Dynamo DBHash
- Dinamo DBIgnore
- Dinamo DBIndex HashKey
- Dinamo DBIndex RangeKey
- Kunci Dynamo DBRange

- Dinamo DBTable
- Dynamo Dikonversi DBType
- Dinamo DBTyped
- Atribut Dynamo DBVersion

Dinamo DBAttribute

Memetakan properti ke atribut tabel. Secara default, setiap properti kelas dipetakan ke atribut item dengan nama yang sama. Namun, jika nama tidak sama, Anda dapat menggunakan anotasi ini untuk memetakan properti ke atribut. Dalam cuplikan Java berikut, `DynamoDBAttribute` memetakan properti `BookAuthors` ke nama atribut `Authors` dalam tabel.

```
@DynamoDBAttribute(attributeName = "Authors")
public List<String> getBookAuthors() { return BookAuthors; }
public void setBookAuthors(List<String> BookAuthors) { this.BookAuthors =
 BookAuthors; }
```

`DynamoDBMapper` menggunakan `Authors` sebagai nama atribut saat menyimpan objek ke tabel.

Dinamo DBAuto GeneratedKey

Menandai kunci partisi atau properti kunci urutan sebagai dibuat otomatis. `DynamoDBMapper` membuat UUID acak saat menyimpan atribut ini. Hanya properti String yang dapat ditandai sebagai kunci yang dibuat otomatis.

Contoh berikut mendemonstrasikan menggunakan kunci yang dibuat otomatis.

```
@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());
    }
}
```

Dinamo DBAuto GeneratedTimestamp

Membuat stempel waktu secara otomatis.

```
@DynamoDBAutoGeneratedTimestamp(strategy=DynamoDBAutoGenerateStrategy.ALWAYS)
public Date getLastUpdatedDate() { return lastUpdatedDate; }
public void setLastUpdatedDate(Date lastUpdatedDate) { this.lastUpdatedDate =
 lastUpdatedDate; }
```

Secara opsional, strategi pembuatan otomatis dapat didefinisikan dengan menyediakan atribut strategi. Default-nya adalah `ALWAYS`.

Dinamo DBDocument

Menunjukkan bahwa kelas dapat diserialisasi sebagai dokumen Amazon DynamoDB.

Misalnya, katakanlah Anda ingin memetakan dokumen JSON ke atribut DynamoDB untuk jenis Peta (`M`). Contoh kode berikut mendefinisikan item yang berisi atribut bersarang (Gambar) untuk jenis Peta.

```
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; }

     }
}
```

Kemudian, Anda dapat menyimpan item `ProductCatalog`, dengan `Pictures`, seperti yang ditunjukkan dalam contoh berikut.

```
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);
```

Item `ProductCatalog` yang dihasilkan akan tampak seperti berikut (dalam format JSON).

```
{
  "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"
  }
}
```

Kunci Dynamo DBHash

Memetakan properti kelas ke kunci partisi tabel. Properti harus berupa salah satu string skalar, angka, atau jenis biner. Properti tidak dapat berupa jenis koleksi.

Asumsikan bahwa Anda memiliki tabel, `ProductCatalog`, yang memiliki `Id` sebagai kunci primer. Kode Java berikut mendefinisikan kelas `CatalogItem` dan memetakan properti `Id`-nya ke kunci primer dari tabel `ProductCatalog` menggunakan tanda `@DynamoDBHashKey`.

```
@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.
}
```

Dinamo DBIgnore

Menunjukkan ke instans `DynamoDBMapper` bahwa properti terkait harus diabaikan. Saat menyimpan data ke tabel, `DynamoDBMapper` tidak menyimpan properti ini ke tabel.

Diterapkan pada metode getter atau bidang kelas untuk properti yang tidak dimodelkan. Jika anotasi diterapkan langsung ke bidang kelas, getter dan setter yang sesuai harus dinyatakan di kelas yang sama.

Dinamo DBIndex HashKey

Memetakan properti kelas ke kunci partisi untuk indeks sekunder global. Properti harus berupa salah satu string skalar, angka, atau jenis biner. Properti tidak dapat berupa jenis koleksi.

Gunakan anotasi ini jika Anda perlu `Query` indeks sekunder global. Anda harus menentukan nama indeks (`globalSecondaryIndexName`). Jika nama properti kelas berbeda dari kunci partisi indeks, Anda juga harus menentukan nama atribut indeks (`attributeName`).

Dinamo DBIndex RangeKey

Memetakan properti kelas untuk kunci urutan indeks sekunder global atau indeks sekunder lokal. Properti harus berupa salah satu string skalar, angka, atau jenis biner. Properti tidak dapat berupa jenis koleksi.

Gunakan anotasi ini jika anda perlu `Query` indeks sekunder lokal atau indeks sekunder global dan ingin menyempurnakan hasil menggunakan kunci urutan indeks. Anda harus menentukan nama indeks (baik `globalSecondaryIndexName` maupun `localSecondaryIndexName`). Jika nama properti kelas berbeda dari kunci urutan indeks, Anda juga harus menentukan nama atribut indeks (`attributeName`).

Kunci Dynamo DBRange

Memetakan properti kelas untuk kunci urutan tabel. Properti harus berupa salah satu string skalar, angka, atau jenis biner. Properti ini tidak boleh berupa jenis koleksi.

Jika kunci primer adalah komposit (kunci partisi dan kunci urutan), Anda dapat menggunakan tanda ini untuk memetakan bidang kelas Anda ke kunci urutan. Misalnya, Anda memiliki tabel `Reply` yang menyimpan balasan untuk utas forum. Setiap utas dapat berisi banyak balasan. Jadi, kunci primer tabel ini adalah `ThreadId` dan `ReplyDateTime`. `ThreadId` adalah kunci partisinya, dan `ReplyDateTime` adalah kunci urutannya.

Kode Java berikut akan mendefinisikan `Reply` dan memetakannya ke tabel `Reply`. Kode tersebut menggunakan tanda `@DynamoDBHashKey` dan `@DynamoDBRangeKey` untuk mengidentifikasi properti kelas yang dipetakan ke kunci primer.

```
@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.
}
```

Dinamo DBTable

Mengidentifikasi tabel target di DynamoDB. Misalnya, kode Java berikut menentukan kelas `Developer` dan memetakannya ke tabel `People` di DynamoDB.

```
@DynamoDBTable(tableName="People")
public class Developer { ...}
```

Anotasi `@DynamoDBTable` dapat diwariskan. Setiap kelas baru yang diwariskan dari kelas `Developer` juga dipetakan ke tabel `People` yang sama. Misalnya, Anda membuat kelas `Lead` yang diwariskan dari kelas `Developer`. Karena Anda memetakan kelas `Developer` ke tabel `People`, objek kelas `Lead` juga disimpan dalam tabel yang sama.

`@DynamoDBTable` juga dapat diganti. Setiap kelas baru yang diwariskan dari kelas `Developer` secara default dipetakan ke tabel `People` yang sama. Namun, Anda dapat mengganti pemetaan default ini. Misalnya, jika Anda membuat kelas yang diwariskan dari kelas `Developer`, Anda dapat secara eksplisit memetakannya ke tabel lain dengan menambahkan anotasi `@DynamoDBTable` seperti yang ditunjukkan dalam contoh kode Java berikut.

```
@DynamoDBTable(tableName="Managers")
public class Manager extends Developer { ...}
```

Dynamo Dikonversi DBType

Anotasi untuk menandai properti sebagai menggunakan konverter jenis khusus. Dapat dianotasikan pada anotasi yang ditetapkan pengguna untuk meneruskan properti tambahan ke `DynamoDBTypeConverter`.

Antarmuka `DynamoDBTypeConverter` memungkinkan Anda memetakan jenis data arbitrer Anda sendiri ke jenis data yang secara asli didukung oleh DynamoDB. Untuk informasi selengkapnya, lihat Memetakan data arbitrer di DynamoDB.

Dinamo DBTyped

Anotasi untuk menimpa pengikatan jenis atribut standar. Jenis standar tidak memerlukan anotasi jika menerapkan pengikatan atribut default untuk jenis tersebut.

Atribut Dynamo DBVersion

Mengidentifikasi properti kelas untuk menyimpan nomor versi penguncian optimis. `DynamoDBMapper` menetapkan nomor versi untuk properti ini ketika menyimpan item baru, dan menambahkannya setiap kali Anda memperbarui item. Hanya jenis skalar nomor yang didukung. Untuk informasi selengkapnya tentang jenis data, lihat Jenis Data. Untuk informasi selengkapnya tentang penentuan versi, lihat DynamoDB dan penguncian optimis dengan nomor versi.

Pengaturan konfigurasi opsional untuk Dynamo DBMapper

Saat Anda membuat instans `DynamoDBMapper`, instans tersebut memiliki perilaku default tertentu; Anda dapat menimpa pengaturan default ini menggunakan kelas `DynamoDBMapperConfig`.

Cuplikan kode berikut akan membuat `DynamoDBMapper` dengan pengaturan kustom:

```
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);
```

Untuk informasi selengkapnya, lihat Dynamo DBMapper Config di Referensi API AWS SDK untuk Java .

Anda dapat menggunakan argumen berikut untuk instans `DynamoDBMapperConfig`:

- Nilai enumerasi `DynamoDBMapperConfig.ConsistentReads`:

- `EVENTUAL`—instans pemeta menggunakan permintaan bacaan akhir konsisten.
- `CONSISTENT`—instans pemeta menggunakan permintaan bacaan sangat konsisten. Anda dapat menggunakan pengaturan opsional ini dengan operasi `load`, `query`, atau `scan`. Bacaan sangat konsisten memiliki implikasi terhadap performa dan penagihan; lihat halaman detail produk DynamoDB untuk informasi selengkapnya.

Jika Anda tidak menentukan pengaturan konsistensi baca untuk instans pemeta Anda, pengaturan defaultnya adalah `EVENTUAL`.

> **Note**
>
> Nilai ini diterapkan dalam`query`,, `querypageload`, dan `batch load` operasi DynamoDBMapper.

- Nilai enumerasi `DynamoDBMapperConfig.PaginationLoadingStrategy`—Mengontrol cara instans pemeta memproses daftar data yang diberi nomor halaman, seperti hasil dari `query` atau `scan`:
  - `LAZY_LOADING`—instans pemeta memuat data jika memungkinkan, dan menyimpan semua hasil yang dimuat dalam memori.
  - `EAGER_LOADING`—instans pemeta memuat data segera setelah daftar diinisialisasi.
  - `ITERATION_ONLY`—Anda hanya dapat menggunakan Iterator untuk membaca dari daftar. Selama iterasi, daftar akan menghapus semua hasil lama sebelum memuat halaman berikutnya, sehingga daftar akan menyimpan maksimal satu halaman dari hasil yang dimuat dalam memori. Hal ini juga berarti bahwa daftar hanya dapat diiterasi satu kali. Strategi ini direkomendasikan saat menangani item besar, guna mengurangi overhead memori.

  Jika Anda tidak menentukan strategi pemuatan pemberian nomor halaman untuk instans pemeta Anda, pengaturan defaultnya adalah `LAZY_LOADING`.
- Nilai enumerasi `DynamoDBMapperConfig.SaveBehavior` - Menentukan cara instans pemeta akan menangani atribut selama operasi simpan:
  - `UPDATE`—selama operasi simpan, semua atribut model diperbarui, dan atribut yang tidak dibuat modelnya tidak terpengaruh. Jenis angka primitif (byte, int, long) diatur ke 0. Jenis objek diatur ke null.
  - `CLOBBER`—menghapus dan mengganti semua atribut, termasuk yang tidak dibuat modelnya, selama operasi simpan. Hal ini dilakukan dengan menghapus item lalu membuatnya kembali. Batasan bidang dengan versi juga diabaikan.

Jika Anda tidak menentukan perilaku simpan untuk instans pemeta, pengaturan defaultnya adalah `UPDATE`.

> **Note**
>
> Operasi DBMapper transaksional Dynamo tidak mendukung pencacahan. `DynamoDBMapperConfig.SaveBehavior`

- Objek `DynamoDBMapperConfig.TableNameOverride`—Menginstruksikan instans pemeta untuk mengabaikan nama tabel yang ditentukan oleh anotasi `DynamoDBTable` kelas, dan menggunakan nama tabel lain yang Anda berikan. Ini berguna saat mempartisi data Anda menjadi beberapa tabel saat runtime.

Anda dapat mengganti objek konfigurasi default untuk `DynamoDBMapper` per operasi, jika diperlukan.

DynamoDB dan penguncian optimis dengan nomor versi

Penguncian optimis adalah strategi untuk memastikan bahwa item sisi klien yang Anda perbarui (atau hapus) sama dengan item di Amazon DynamoDB. Jika Anda menggunakan strategi ini, penulisan basis data Anda dilindungi agar tidak ditimpa oleh penulisan orang lain, dan sebaliknya.

Dengan penguncian optimis, setiap item memiliki atribut yang berfungsi sebagai nomor versi. Jika Anda mengambil item dari tabel, aplikasi akan mencatat nomor versi item tersebut. Anda dapat memperbarui item, tetapi hanya jika nomor versi di sisi server tidak berubah. Jika ada ketidakcocokan versi, artinya orang lain telah mengubah item sebelum Anda melakukannya. Upaya pembaruan gagal, karena versi item Anda sudah usang. Jika ini terjadi, coba lagi dengan mengambil item dan kemudian mencoba memperbaruinya. Penguncian optimis mencegah Anda menimpa perubahan secara tidak sengaja yang dibuat oleh orang lain. Penguncian optimis juga mencegah orang lain menimpa perubahan Anda secara tidak sengaja.

Meskipun Anda dapat menerapkan strategi penguncian optimis Anda sendiri, AWS SDK untuk Java memberikan `@DynamoDBVersionAttribute` anotasi. Di kelas pemetaan untuk tabel, Anda menetapkan satu properti untuk menyimpan nomor versi, dan menandainya menggunakan anotasi ini. Saat Anda menyimpan objek, item terkait dalam tabel DynamoDB akan memiliki atribut yang menyimpan nomor versi. `DynamoDBMapper` menetapkan nomor versi saat Anda pertama kali menyimpan objek, dan otomatis menambahkan nomor versi setiap kali Anda memperbarui item.

Permintaan perbarui atau hapus Anda hanya akan berhasil jika versi objek sisi klien cocok dengan nomor versi item yang sesuai dalam tabel DynamoDB.

`ConditionalCheckFailedException` ditampilkan jika:

- Anda menggunakan penguncian optimis dengan `@DynamoDBVersionAttribute` dan nilai versi pada server yang berbeda dari nilai pada sisi klien.
- Anda menentukan batasan bersyarat Anda sendiri saat menyimpan data menggunakan `DynamoDBMapper` dengan `DynamoDBSaveExpression` dan batasan ini gagal.

> **Note**
>
> - Tabel global DynamoDB menggunakan rekonsiliasi “penulis terakhir menang” antara pembaruan bersamaan. Jika Anda menggunakan tabel global, kebijakan penulis terakhir akan menang. Jadi dalam hal ini, strategi penguncian tidak berfungsi seperti yang diharapkan.
> - Operasi tulis transaksional `DynamoDBMapper` tidak mendukung ekspresi syarat dan anotasi `@DynamoDBVersionAttribute` pada objek yang sama. Jika sebuah objek dalam penulisan transaksional dianotasi dengan `@DynamoDBVersionAttribute` dan juga memiliki ekspresi kondisi, maka akan dilemparkan. SdkClientException

Misalnya, kode Java berikut mendefinisikan kelas `CatalogItem` yang memiliki beberapa properti. Properti `Version` ditandai dengan anotasi `@DynamoDBVersionAttribute`.

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;}
}
```

Anda dapat menerapkan anotasi `@DynamoDBVersionAttribute` untuk jenis yang dapat dinihilkan yang diberikan oleh kelas pembungkus primitif yang menyediakan jenis yang dapat di-null-kan, seperti `Long` dan `Integer`.

Penguncian optimis memiliki dampak sebagai berikut pada metode `DynamoDBMapper` ini:

- `save` — Untuk item baru, `DynamoDBMapper` menetapkan nomor versi awal 1. Jika Anda mengambil item, memperbarui satu atau beberapa propertinya, dan mencoba untuk menyimpan perubahan, operasi simpan hanya akan berhasil jika nomor versi di sisi klien dan sisi server cocok. `DynamoDBMapper` menambah nomor versi secara otomatis.
- `delete` — Metode `delete` mengambil sebuah objek sebagai parameter, dan `DynamoDBMapper` melakukan pemeriksaan versi sebelum menghapus item. Pemeriksaan versi dapat dinonaktifkan jika `DynamoDBMapperConfig.SaveBehavior.CLOBBER` ditentukan dalam permintaan.

  Implementasi internal penguncian optimis dalam `DynamoDBMapper` menggunakan pembaruan bersyarat dan dukungan penghapusan bersyarat yang disediakan oleh DynamoDB.
- `transactionWrite` —

- `Put` — Untuk item baru, `DynamoDBMapper` menetapkan nomor versi awal 1. Jika Anda mengambil item, memperbarui satu atau beberapa propertinya, dan mencoba untuk menyimpan perubahan, operasi masukkan hanya akan berhasil jika nomor versi di sisi klien dan sisi server cocok. `DynamoDBMapper` menambah nomor versi secara otomatis.
- `Update` — Untuk item baru, `DynamoDBMapper` menetapkan nomor versi awal 1. Jika Anda mengambil item, memperbarui satu atau beberapa propertinya, dan mencoba untuk menyimpan perubahan, operasi perbarui hanya akan berhasil jika nomor versi di sisi klien dan sisi server cocok. `DynamoDBMapper` menambah nomor versi secara otomatis.
- `Delete` — `DynamoDBMapper` melakukan pemeriksaan versi sebelum menghapus item. Operasi hapus hanya akan berhasil jika nomor versi pada sisi klien dan sisi server cocok.
- `ConditionCheck` — Anotasi `@DynamoDBVersionAttribute` tidak didukung untuk operasi `ConditionCheck`. An SdkClientException akan dilemparkan ketika `ConditionCheck` item dianotasi dengan. `@DynamoDBVersionAttribute`

Menonaktifkan penguncian optimis

Untuk menonaktifkan penguncian optimis, Anda dapat mengubah nilai enumerasi `DynamoDBMapperConfig.SaveBehavior` dari `UPDATE` menjadi `CLOBBER`. Anda dapat melakukannya dengan membuat instans `DynamoDBMapperConfig` yang melewatkan pemeriksaan versi dan menggunakan instans ini untuk semua permintaan Anda. Untuk informasi tentang `DynamoDBMapperConfig.SaveBehavior` dan parameter `DynamoDBMapper` opsional lainnya, lihat Pengaturan konfigurasi opsional untuk Dynamo DBMapper .

Anda juga dapat mengatur perilaku penguncian untuk operasi tertentu saja. Misalnya, cuplikan Java berikut menggunakan `DynamoDBMapper` untuk menyimpan item katalog. Cuplikan ini menentukan `DynamoDBMapperConfig.SaveBehavior` dengan menambahkan parameter `DynamoDBMapperConfig` opsional untuk metode `save`.

> **Note**
>
> Metode TransactionWrite tidak mendukung Dynamo Config. DBMapper SaveBehaviorkonfigurasi. Menonaktifkan penguncian optimis untuk transactionWrite tidak didukung.

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));
```

Memetakan data arbitrer di DynamoDB

Selain jenis Java yang didukung (lihat Tipe data yang didukung untuk Dynamo DBMapper untuk Java), Anda dapat menggunakan jenis di aplikasi Anda yang tidak memiliki pemetaan langsung ke jenis Amazon DynamoDB. Untuk memetakan jenis ini, Anda harus menyediakan implementasi yang mengonversi jenis kompleks Anda ke jenis yang didukung DynamoDB dan sebaliknya, serta menganotasi metode penilai jenis kompleks menggunakan anotasi `@DynamoDBTypeConverted`. Kode konverter mengubah data ketika objek disimpan atau dimuat. Kode ini juga digunakan untuk semua operasi yang mengonsumsi jenis kompleks. Perhatikan bahwa saat membandingkan data selama operasi kueri dan pindai, perbandingan pada data yang disimpan dalam DynamoDB akan dilakukan.

Misalnya, pertimbangkan kelas `CatalogItem` berikut yang mendefinisikan properti, `Dimension`, yaitu `DimensionType`. Properti ini menyimpan dimensi item sebagai tinggi, lebar, dan ketebalan. Misalnya Anda memutuskan untuk menyimpan dimensi item ini sebagai string (seperti 8.5x11x.05) dalam DynamoDB. Contoh berikut memberikan kode konverter yang mengonversi objek `DimensionType` menjadi string dan string menjadi `DimensionType`.

> **Note**
>
> Contoh kode ini mengasumsikan bahwa Anda telah memuat data ke DynamoDB untuk akun Anda dengan mengikuti petunjuk di bagian Membuat tabel dan memuat data untuk contoh kode di DynamoDB.
> Untuk step-by-step instruksi untuk menjalankan contoh berikut, lihatContoh kode Java.

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;
        }
    }
```

```
}
```

Contoh Dynamo DBMapper

AWS SDK for Java menyediakan `DynamoDBMapper` kelas, memungkinkan Anda untuk memetakan kelas sisi klien Anda ke tabel DynamoDB. Untuk menggunakan `DynamoDBMapper`, tentukan hubungan antara item dalam tabel DynamoDB dan instans objek yang sesuai dalam kode Anda. Kelas `DynamoDBMapper` memungkinkan Anda melakukan berbagai operasi buat, baca, perbarui, dan hapus (CRUD) pada item, dan menjalankan kueri serta memindai tabel.

Untuk mempelajari lebih lanjut tentang cara menggunakan*`DynamoDBMapper`*, lihat Contoh DynamoDB Menggunakan SDK AWS for Java di SDK for Java 1.x Panduan AWS Pengembang.

Java 2.x: DynamoDB Enhanced Client

Klien yang disempurnakan DynamoDB adalah pustaka tingkat tinggi yang merupakan bagian dari versi 2 (AWS SDK untuk Java v2). Ini menawarkan cara mudah untuk memetakan kelas sisi klien ke tabel DynamoDB. Anda menentukan hubungan antara tabel dan kelas model yang sesuai dalam kode Anda. Setelah menentukan hubungan tersebut, Anda dapat melakukan berbagai operasi buat, baca, perbarui, atau hapus (CRUD) pada tabel atau item di DynamoDB.

Untuk informasi selengkapnya tentang bagaimana Anda dapat menggunakan klien yang disempurnakan dengan DynamoDB, lihat Menggunakan DynamoDB Enhanced Client di 2.x. AWS SDK untuk Java

Bekerja dengan model dokumen.NET di DynamoDB

AWS SDK untuk .NET Ini menyediakan kelas model dokumen yang membungkus beberapa operasi Amazon DynamoDB tingkat rendah, yang selanjutnya menyederhanakan pengkodean Anda. Dalam model dokumen, kelas utamanya adalah `Table` dan `Document`. Kelas `Table` menyediakan metode operasi data seperti `PutItem`, `GetItem`, dan `DeleteItem`. Kelas ini juga menyediakan metode `Query` dan `Scan`. Kelas `Document` mewakili satu item dalam tabel.

Kelas model dokumen sebelumnya tersedia di namespace `Amazon.DynamoDBv2.DocumentModel`.

> **Note**
>
> Anda tidak dapat menggunakan kelas model dokumen untuk membuat, memperbarui, dan menghapus tabel. Namun, model dokumen mendukung operasi data yang paling umum.

Topik

- Jenis data yang didukung

Jenis data yang didukung

Model dokumen mendukung serangkaian jenis data .NET primitif dan jenis data koleksi. Model ini mendukung jenis data primitif berikut.

- `bool`
- `byte`
- `char`
- `DateTime`
- `decimal`
- `double`
- `float`
- `Guid`
- `Int16`
- `Int32`
- `Int64`
- `SByte`
- `string`
- `UInt16`
- `UInt32`
- `UInt64`

Tabel berikut merangkum pemetaan jenis .NET sebelumnya untuk jenis DynamoDB.

Jenis .NET primitif	Jenis DynamoDB
Semua jenis angka	`N` (jenis angka)
Semua jenis string	`S` (jenis string)

Jenis .NET primitif	Jenis DynamoDB
MemoryStream, byte []	`B` (jenis biner)
bool	`N` (jenis angka). 0 mewakili false dan 1 mewakili true.
DateTime	`S` (jenis string). Nilai `DateTime` disimpan sebagai string berformat ISO-8601.
Guid	`S` (jenis string).
Jenis koleksi (Daftar, HashSet, dan array)	Jenis `BS` (set biner), jenis `SS` (set string), dan jenis `NS` (set angka)

AWS SDK untuk .NET mendefinisikan tipe untuk memetakan tipe Boolean, null, list, dan map DynamoDB ke API model dokumen.NET:

- Gunakan `DynamoDBBool` untuk jenis Boolean.
- Gunakan `DynamoDBNull` untuk jenis null.
- Gunakan `DynamoDBList` untuk jenis daftar.
- Gunakan `Document` untuk jenis peta.

> **Note**
>
> - Nilai biner kosong didukung.
> - Pembacaan nilai string kosong didukung. Nilai atribut string kosong didukung dalam nilai atribut string dari jenis Set saat menulis ke DynamoDB. Nilai atribut string kosong dari jenis string dan nilai string kosong dalam jenis Daftar atau Peta dihilangkan dari permintaan tulis

Bekerja dengan model persistensi objek.NET dan DynamoDB

AWS SDK untuk .NET Ini menyediakan model persistensi objek yang memungkinkan Anda memetakan kelas sisi klien Anda ke tabel Amazon DynamoDB. Kemudian, setiap instans objek dipetakan ke item dalam tabel yang sesuai. Untuk menyimpan objek sisi klien ke tabel, model

persistensi objek menyediakan kelas `DynamoDBContext`, yaitu titik masuk ke DynamoDB. Kelas ini menyediakan koneksi ke DynamoDB dan memungkinkan Anda mengakses tabel, melakukan berbagai operasi CRUD, dan menjalankan kueri.

Model persistensi objek menyediakan sekumpulan atribut untuk memetakan kelas sisi klien ke tabel, dan properti/bidang ke atribut tabel.

> **Note**
>
> Model persistensi objek tidak menyediakan API untuk membuat, memperbarui, atau menghapus tabel. Model ini hanya menyediakan operasi data. Anda hanya dapat menggunakan API AWS SDK untuk .NET tingkat rendah untuk membuat, memperbarui, dan menghapus tabel.

Contoh berikut menunjukkan cara kerja model persistensi objek. Contoh ini dimulai dengan tabel `ProductCatalog`. Contoh ini memiliki `Id` sebagai kunci primernya.

```
ProductCatalog(Id, ...)
```

Katakanlah Anda memiliki kelas Book yang berisi properti `Title`, `ISBN`, dan `Authors`. Anda dapat memetakan kelas Book ke tabel `ProductCatalog` dengan menambahkan atribut yang didefinisikan oleh model persistensi objek, seperti yang ditunjukkan dalam kode contoh C# berikut.

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; }
```

```
}
```

Dalam contoh sebelumnya, atribut `DynamoDBTable` memetakan kelas `Book` ke tabel `ProductCatalog`.

Model persistensi objek mendukung pemetaan eksplisit dan default antara properti kelas dan atribut tabel.

- Pemetaan eksplisit—Untuk memetakan properti ke kunci primer, Anda harus menggunakan atribut model persistensi objek `DynamoDBHashKey` dan `DynamoDBRangeKey`. Selain itu, untuk atribut kunci nonprimer, jika nama properti di kelas Anda dan atribut tabel terkait yang ingin Anda petakan tidak sama, Anda harus mendefinisikan pemetaan dengan menambahkan atribut `DynamoDBProperty` secara eksplisit.

  Dalam contoh sebelumnya, properti `Id` dipetakan ke kunci primer dengan nama yang sama, dan properti `BookAuthors` dipetakan ke atribut `Authors` dalam tabel `ProductCatalog`.
- Pemetaan default—Secara default, model persistensi objek memetakan properti kelas ke atribut dengan nama yang sama dalam tabel.

  Dalam contoh sebelumnya, properti `Title` dan `ISBN` dipetakan ke atribut dengan nama yang sama dalam tabel `ProductCatalog`.

Anda tidak perlu memetakan setiap properti kelas tunggal. Anda mengidentifikasi properti ini dengan menambahkan atribut `DynamoDBIgnore`. Ketika Anda menyimpan instans `Book` ke tabel, `DynamoDBContext` tidak menyertakan properti `CoverPage`. Instans tersebut juga tidak mengembalikan properti ini saat Anda mengambil instans buku.

Anda dapat memetakan properti jenis .NET primitif seperti int dan string. Anda juga dapat memetakan setiap jenis data arbitrer selama Anda memberikan konverter yang sesuai untuk memetakan data arbitrer ke salah satu jenis DynamoDB. Untuk mempelajari cara memetakan jenis arbitrer, lihat Memetakan data arbitrer dengan DynamoDB menggunakan model persistensi objek AWS SDK untuk .NET.

Model persistensi objek mendukung penguncian optimis. Selama operasi pembaruan, model ini memastikan bahwa Anda memiliki salinan terbaru dari item yang akan Anda perbarui. Untuk informasi selengkapnya, lihat Penguncian optimis menggunakan DynamoDB dan model persistensi objek AWS SDK untuk .NET.

Untuk informasi selengkapnya, lihat topik di bawah.

Topik

- Jenis data yang didukung
- Atribut DynamoDB dari model persistensi objek.NET
- DBContext Kelas Dynamo dari model persistensi objek.NET
- Penguncian optimis menggunakan DynamoDB dan model persistensi objek AWS SDK untuk .NET
- Memetakan data arbitrer dengan DynamoDB menggunakan model persistensi objek AWS SDK untuk .NET

Jenis data yang didukung

Model persistensi objek mendukung sekumpulan jenis data .NET primitif, koleksi, dan jenis data arbitrer. Model ini mendukung jenis data primitif berikut.

- `bool`
- `byte`
- `char`
- `DateTime`
- `decimal`
- `double`
- `float`
- `Int16`
- `Int32`
- `Int64`
- `SByte`
- `string`
- `UInt16`
- `UInt32`
- `UInt64`

Model persistensi objek juga mendukung jenis koleksi.NET. `DynamoDBContext`mampu mengkonversi jenis koleksi beton dan sederhana Plain Old CLR Objects (POCOs).

Tabel berikut merangkum pemetaan jenis .NET sebelumnya untuk jenis DynamoDB.

Jenis .NET primitif	Jenis DynamoDB
Semua jenis angka	`N` (jenis angka)
Semua jenis string	`S` (jenis string)
MemoryStream, byte []	`B` (jenis biner)
bool	`N` (jenis angka). 0 mewakili false dan 1 mewakili true.
Jenis koleksi	Jenis `BS` (kumpulan biner), jenis `SS` (kumpulan string), dan jenis `NS` (kumpulan angka).
DateTime	`S` (jenis string). Nilai `DateTime` disimpan sebagai string berformat ISO-8601.

Model persistensi objek juga mendukung jenis data arbitrer. Namun, Anda harus menyediakan kode konverter untuk memetakan jenis kompleks ke jenis DynamoDB.

> **Note**
>
> - Nilai biner kosong didukung.
> - Pembacaan nilai string kosong didukung. Nilai atribut string kosong didukung dalam nilai atribut string dari jenis Set saat menulis ke DynamoDB. Nilai atribut string kosong dari jenis string dan nilai string kosong dalam jenis Daftar atau Peta dihilangkan dari permintaan tulis

Atribut DynamoDB dari model persistensi objek.NET

Bagian ini menjelaskan atribut yang ditawarkan model persistensi objek agar Anda dapat memetakan kelas dan properti ke tabel dan atribut DynamoDB.

> **Note**
>
> Dalam atribut berikut, hanya `DynamoDBTable` dan `DynamoDBHashKey` yang diperlukan.

Dinamo DBGlobal SecondaryIndexHashKey

Memetakan properti kelas ke kunci partisi untuk indeks sekunder global. Gunakan atribut ini jika Anda perlu `Query` indeks sekunder global.

Dinamo DBGlobal SecondaryIndexRangeKey

Memetakan properti kelas ke kunci urutan indeks sekunder global. Gunakan atribut ini jika Anda perlu `Query` indeks sekunder global dan ingin menyempurnakan hasil menggunakan kunci urutan indeks.

Kunci Dynamo DBHash

Memetakan properti kelas ke kunci partisi untuk kunci primer tabel. Atribut kunci primer tidak boleh berupa jenis koleksi.

Contoh kode C# berikut memetakan kelas `Book` ke tabel `ProductCatalog`, dan properti `Id` ke kunci partisi kunci kunci primer tabel.

```
[DynamoDBTable("ProductCatalog")]
public class Book
{
    [DynamoDBHashKey]
    public int Id { get; set; }

    // Additional properties go here.
}
```

Dinamo DBIgnore

Menunjukkan bahwa properti terkait harus diabaikan. Jika tidak ingin menyimpan properti kelas, Anda dapat menambahkan atribut ini untuk meminta agar `DynamoDBContext` tidak menyertakan properti ini saat menyimpan objek ke tabel.

Dinamo DBLocal SecondaryIndexRangeKey

Memetakan properti kelas ke kunci urutan indeks sekunder lokal. Gunakan atribut ini jika Anda perlu `Query` indeks sekunder lokal dan ingin menyempurnakan hasil menggunakan kunci urutan indeks.

Dinamo DBProperty

Memetakan properti kelas ke atribut tabel. Jika properti kelas dipetakan ke atribut tabel dengan nama yang sama, Anda tidak perlu menentukan atribut ini. Namun, jika nama tidak sama, Anda

dapat menggunakan tanda ini untuk memberikan pemetaan. Dalam pernyataan C# berikut, `DynamoDBProperty` memetakan properti `BookAuthors` ke atribut `Authors` dalam tabel.

```
[DynamoDBProperty("Authors")]
public List<string> BookAuthors { get; set; }
```

`DynamoDBContext` menggunakan informasi pemetaan ini untuk membuat atribut `Authors` saat menyimpan data objek ke tabel yang sesuai.

Dinamo DBRenamable

Menentukan nama alternatif untuk properti kelas. Langkah ini berguna jika Anda menulis konverter khusus untuk memetakan data arbitrer ke tabel DynamoDB yang nama properti kelasnya berbeda dengan atribut tabel.

Kunci Dynamo DBRange

Memetakan properti kelas ke kunci urutan untuk kunci primer tabel. Jika tabel memiliki kunci primer komposit (kunci partisi dan kunci urutan), Anda harus menentukan atribut `DynamoDBHashKey` dan `DynamoDBRangeKey` dalam pemetaan kelas Anda.

Misalnya, tabel sampel `Reply` memiliki kunci primer yang dibuat dari kunci partisi `Id` dan kunci urutan `Replenishment`. Contoh kode C# berikut memetakan kelas `Reply` ke tabel `Reply`. Definisi kelas juga menunjukkan bahwa dua propertinya dipetakan ke kunci primer.

```
[DynamoDBTable("Reply")]
public class Reply
{
   [DynamoDBHashKey]
   public int ThreadId { get; set; }
   [DynamoDBRangeKey]
   public string Replenishment { get; set; }

   // Additional properties go here.
}
```

Dinamo DBTable

Mengidentifikasi tabel target di DynamoDB tempat kelas dipetakan. Misalnya, contoh kode C# berikut memetakan kelas `Developer` ke tabel `People` di DynamoDB.

```
[DynamoDBTable("People")]
public class Developer { ...}
```

Atribut ini dapat diwariskan atau diganti.

- Atribut `DynamoDBTable` dapat diwariskan. Dalam contoh sebelumnya, jika Anda menambahkan kelas baru, yaitu `Lead`, yang diwariskan dari kelas `Developer`, kelas tersebut juga dipetakan ke tabel `People`. Objek `Developer` dan `Lead` disimpan dalam tabel `People`.
- Atribut `DynamoDBTable` juga dapat diganti. Dalam contoh kode C# berikut, kelas `Manager` diwariskan dari kelas `Developer`. Namun, penambahan eksplisit atribut `DynamoDBTable` memetakan kelas ke tabel lainnya (`Managers`).

  ```
  [DynamoDBTable("Managers")]
  public class Manager : Developer { ...}
  ```

Anda dapat menambahkan parameter opsional, yaitu `LowerCamelCaseProperties`, untuk meminta DynamoDB menulis huruf pertama dari nama properti dalam huruf kecil ketika menyimpan objek ke tabel, seperti yang ditunjukkan dalam contoh C# berikut.

```
[DynamoDBTable("People", LowerCamelCaseProperties=true)]
public class Developer
{
    string DeveloperName;
    ...
}
```

Ketika menyimpan instans kelas `Developer`, `DynamoDBContext` menyimpan properti `DeveloperName` sebagai `developerName`.

Dinamo DBVersion

Mengidentifikasi properti kelas untuk menyimpan nomor versi item. Untuk informasi selengkapnya tentang penentuan versi, lihat Penguncian optimis menggunakan DynamoDB dan model persistensi objek AWS SDK untuk .NET.

DBContext Kelas Dynamo dari model persistensi objek.NET

Kelas `DynamoDBContext` adalah titik masuk ke basis data Amazon DynamoDB. Kelas ini menyediakan koneksi ke DynamoDB dan memungkinkan Anda mengakses data dalam berbagai

tabel, melakukan berbagai operasi CRUD, dan menjalankan kueri. Kelas `DynamoDBContext` menyediakan metode berikut.

Topik

- Buat MultiTable BatchGet
- Buat MultiTable BatchWrite
- CreateBatchGet
- CreateBatchWrite
- Delete
- Dispose
- ExecuteBatchGet
- ExecuteBatchWrite
- FromDocument
- FromQuery
- FromScan
- GetTargetTable
- Muat
- Kueri
- Save
- Pindai
- ToDocument
- Menentukan parameter opsional untuk Dynamo DBContext

Buat MultiTable BatchGet

Membuat objek `MultiTableBatchGet`, yang terdiri dari beberapa objek `BatchGet` individual. Setiap objek `BatchGet` dapat digunakan untuk mengambil item dari tabel DynamoDB tunggal.

Untuk mengambil item dari tabel, gunakan metode `ExecuteBatchGet`, dengan meneruskan objek `MultiTableBatchGet` sebagai parameter.

Buat MultiTable BatchWrite

Membuat objek `MultiTableBatchWrite`, yang terdiri dari beberapa objek `BatchWrite` individual. Setiap objek `BatchWrite` dapat digunakan untuk menulis atau menghapus item dalam tabel DynamoDB tunggal.

Untuk menulis ke tabel, gunakan metode `ExecuteBatchWrite`, dengan meneruskan objek `MultiTableBatchWrite` sebagai parameter.

CreateBatchGet

Membuat objek `BatchGet` yang dapat Anda gunakan untuk mengambil beberapa item dari tabel.

CreateBatchWrite

Membuat objek `BatchWrite` yang dapat Anda gunakan untuk menempatkan beberapa item ke dalam tabel, atau untuk menghapus beberapa item dari tabel.

Delete

Menghapus item dari tabel. Metode ini memerlukan kunci primer dari item yang ingin Anda hapus. Anda dapat memberikan nilai kunci primer maupun objek sisi klien yang berisi nilai kunci primer sebagai parameter untuk metode ini.

- Jika Anda menentukan objek sisi klien sebagai parameter dan Anda telah mengaktifkan penguncian optimis, penghapusan hanya akan berhasil jika versi sisi klien dan sisi server objek cocok.
- Jika Anda hanya menetapkan nilai kunci primer sebagai parameter, penghapusan akan berhasil terlepas dari apakah Anda telah mengaktifkan penguncian optimis atau tidak.

> **Note**
>
> Untuk melakukan operasi ini di latar belakang, gunakan metode `DeleteAsync`.

Dispose

Membuang semua sumber daya yang dikelola dan tidak dikelola.

ExecuteBatchGet

Membaca data dari satu atau beberapa tabel, yang memproses semua objek `BatchGet` dalam `MultiTableBatchGet`.

> **Note**
>
> Untuk melakukan operasi ini di latar belakang, gunakan metode `ExecuteBatchGetAsync`.

ExecuteBatchWrite

Menulis atau menghapus data dalam satu atau beberapa tabel, yang memproses semua objek `BatchWrite` dalam `MultiTableBatchWrite`.

> **Note**
>
> Untuk melakukan operasi ini di latar belakang, gunakan metode `ExecuteBatchWriteAsync`.

FromDocument

Mengingat instans `Document`, metode `FromDocument` akan mengembalikan instans dari kelas sisi klien.

Langkah ini membantu jika Anda ingin menggunakan kelas model dokumen bersama dengan model persistensi objek untuk melakukan operasi data. Untuk informasi selengkapnya tentang kelas model dokumen yang disediakan oleh AWS SDK untuk .NET, lihatBekerja dengan model dokumen.NET di DynamoDB.

Misalkan Anda memiliki objek `Document` bernama `doc`, yang berisi representasi item `Forum`. (Untuk mengetahui cara membuat objek ini, lihat deskripsi untuk metode `ToDocument` nanti dalam topik ini.) Anda dapat menggunakan `FromDocument` untuk mengambil item `Forum` dari `Document`, seperti yang ditunjukkan dalam contoh kode C# berikut.

Example

```
forum101 = context.FromDocument<Forum>(101);
```

> **Note**
>
> Jika objek `Document` mengimplementasikan antarmuka `IEnumerable`, Anda dapat menggunakan metode `FromDocuments` sebagai gantinya. Metode ini memungkinkan Anda melakukan iterasi pada semua instans kelas dalam `Document`.

FromQuery

Menjalankan operasi `Query`, dengan parameter kueri yang didefinisikan dalam objek `QueryOperationConfig`.

> **Note**
>
> Untuk melakukan operasi ini di latar belakang, gunakan metode `FromQueryAsync`.

FromScan

Menjalankan operasi `Scan`, dengan parameter pemindaian yang didefinisikan dalam objek `ScanOperationConfig`.

> **Note**
>
> Untuk melakukan operasi ini di latar belakang, gunakan metode `FromScanAsync`.

GetTargetTable

Mengambil tabel target untuk jenis yang ditentukan. Langkah ini berguna jika Anda menulis konverter khusus untuk memetakan data arbitrer ke tabel DynamoDB, dan Anda perlu menentukan tabel yang terkait dengan jenis data khusus.

Muat

Mengambil item dari tabel. Metode ini hanya memerlukan kunci primer dari item yang ingin Anda ambil.

Secara default, DynamoDB akan mengembalikan item dengan nilai yang akhirnya konsisten. Untuk informasi tentang model konsistensi akhir, lihat DynamoDB membaca konsistensi.

`Load`atau `LoadAsync` metode memanggil GetItemoperasi, yang mengharuskan Anda untuk menentukan kunci utama untuk tabel. Karena `GetItem` mengabaikan `IndexName` parameter, Anda tidak dapat memuat item menggunakan partisi indeks atau kunci sortir. Oleh karena itu, Anda harus menggunakan kunci utama tabel untuk memuat item.

> **Note**
>
> Untuk melakukan operasi ini di latar belakang, gunakan metode `LoadAsync`. Untuk melihat contoh penggunaan `LoadAsync` metode untuk melakukan operasi CRUD tingkat tinggi pada tabel DynamoDB, lihat contoh berikut.

```
/// <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");
            }
        }
    }
```

Kueri

Mengkueri tabel berdasarkan parameter kueri yang Anda berikan.

Anda dapat mengkueri tabel hanya jika memiliki kunci primer komposit (kunci partisi dan kunci urutan). Saat mengkueri, Anda harus menentukan kunci partisi dan syarat yang berlaku untuk kunci urutan.

Katakanlah Anda memiliki kelas `Reply` sisi klien yang dipetakan ke tabel `Reply` di DynamoDB. Contoh kode C# berikut mengkueri tabel `Reply` untuk menemukan balasan utas dalam forum yang

diposting dalam 15 hari terakhir. Tabel `Reply` memiliki kunci primer yang berisi kunci partisi `Id` dan kunci urutan `ReplyDateTime`.

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);
```

Tabel ini mengembalikan koleksi objek `Reply`.

Metode `Query` mengembalikan koleksi `IEnumerable` “lazy-loaded”. Metode ini awalnya hanya mengembalikan satu halaman hasil, lalu membuat panggilan layanan untuk halaman berikutnya jika diperlukan. Untuk mendapatkan semua item yang cocok, Anda perlu melakukan iterasi hanya pada `IEnumerable`.

Jika tabel Anda memiliki kunci primer sederhana (kunci partisi), Anda tidak dapat menggunakan metode `Query`. Sebaliknya, Anda dapat menggunakan metode `Load` dan menyediakan kunci partisi untuk mengambil item.

> **Note**
>
> Untuk melakukan operasi ini di latar belakang, gunakan metode `QueryAsync`.

Save

Menyimpan objek tertentu ke tabel. Jika kunci primer yang ditentukan dalam objek input tidak ada dalam tabel, metode akan menambahkan item baru ke tabel. Jika kunci primer ada, metode akan memperbarui item yang ada.

Jika Anda mengonfigurasi penguncian optimis, pembaruan hanya akan berhasil jika versi klien dan sisi server item cocok. Untuk informasi selengkapnya, lihat Penguncian optimis menggunakan DynamoDB dan model persistensi objek AWS SDK untuk .NET.

> **Note**
>
> Untuk melakukan operasi ini di latar belakang, gunakan metode `SaveAsync`.

Pindai

Melakukan pemindaian pada seluruh tabel.

Anda dapat memfilter hasil pemindaian dengan menentukan syarat pemindaian. Syarat ini dapat dievaluasi pada setiap atribut dalam tabel. Katakanlah Anda memiliki kelas sisi klien `Book` yang dipetakan ke tabel `ProductCatalog` di DynamoDB. Contoh C# berikut memindai tabel dan hanya mengembalikan item buku dengan harga kurang dari 0.

Example

```
IEnumerable<Book> itemsWithWrongPrice = context.Scan<Book>(
                    new ScanCondition("Price", ScanOperator.LessThan, price),
                    new ScanCondition("ProductCategory", ScanOperator.Equal, "Book")
      );
```

Metode `Scan` mengembalikan koleksi `IEnumerable` “lazy-loaded”. Metode ini awalnya hanya mengembalikan satu halaman hasil, lalu membuat panggilan layanan untuk halaman berikutnya jika diperlukan. Untuk mendapatkan semua item yang cocok, Anda hanya perlu melakukan iterasi pada `IEnumerable`.

Untuk alasan performa, Anda harus mencari tabel dan menghindari pemindaian tabel.

> **Note**
>
> Untuk melakukan operasi ini di latar belakang, gunakan metode `ScanAsync`.

ToDocument

Mengembalikan instans untuk kelas model dokumen `Document` dari instans kelas Anda.

Langkah ini membantu jika Anda ingin menggunakan kelas model dokumen bersama dengan model persistensi objek untuk melakukan operasi data. Untuk informasi selengkapnya tentang kelas model dokumen yang disediakan oleh AWS SDK untuk .NET, lihatBekerja dengan model dokumen.NET di DynamoDB.

Misalkan Anda memiliki kelas sisi klien yang dipetakan ke tabel `Forum` sampel. Anda kemudian dapat menggunakan `DynamoDBContext` untuk mendapatkan item sebagai objek `Document` dari tabel `Forum`, seperti yang ditunjukkan dalam contoh kode C# berikut.

Example

```
DynamoDBContext context = new DynamoDBContext(client);

Forum forum101 = context.Load<Forum>(101); // Retrieve a forum by primary key.
Document doc = context.ToDocument<Forum>(forum101);
```

Menentukan parameter opsional untuk Dynamo DBContext

Saat menggunakan model persistensi objek, Anda dapat menentukan parameter opsional berikut untuk `DynamoDBContext`.

- **`ConsistentRead`**—Saat mengambil data menggunakan operasi `Load`, `Query`, atau `Scan`, Anda dapat menambahkan parameter opsional ini guna meminta nilai data terbaru.
- **`IgnoreNullValues`**—Parameter ini memberi tahu `DynamoDBContext` untuk mengabaikan nilai null pada atribut selama operasi `Save`. Jika parameter ini adalah false (atau jika tidak diatur), nilai null ditafsirkan sebagai direktif untuk menghapus atribut tertentu.
- **`SkipVersionCheck`**— Parameter ini memberi tahu `DynamoDBContext` untuk tidak membandingkan versi saat menyimpan atau menghapus item. Untuk informasi selengkapnya tentang penentuan versi, lihat Penguncian optimis menggunakan DynamoDB dan model persistensi objek AWS SDK untuk .NET.
- **`TableNamePrefix`**— Memberikan prefiks pada semua nama tabel dengan string tertentu. Jika parameter ini adalah null (atau jika tidak diatur), tidak ada prefiks yang digunakan.
- `DynamoDBEntryConversion`- Menentukan skema konversi yang digunakan oleh klien. Anda dapat mengatur parameter ini ke versi V1 atau V2. V1 adalah versi default.

  Berdasarkan versi yang Anda tetapkan, perilaku parameter ini berubah. Sebagai contoh:

  - Di V1, tipe `bool` data dikonversi ke tipe `N` angka, di mana 0 mewakili false dan 1 mewakili true. Dalam V2, `bool` dikonversi ke`B00L`.
  - Di V2, daftar dan array tidak dikelompokkan bersama. HashSets Daftar dan array numerik, tipe berbasis string, dan tipe berbasis biner dikonversi ke tipe `L` (Daftar), yang dapat dikirim kosong untuk memperbarui daftar. Ini tidak seperti V1, di mana daftar kosong tidak dikirim melalui kabel.

Di V1, jenis koleksi, seperti Daftar, HashSet, dan array diperlakukan sama. Daftar, HashSet, dan array numerik dikonversi ke tipe NS (jumlah set).

Contoh berikut menetapkan versi skema konversi ke V2, yang mengubah perilaku konversi antara tipe.NET dan tipe data DynamoDB.

```
var config = new DynamoDBContextConfig
{
    Conversion = DynamoDBEntryConversion.V2
};
var contextV2 = new DynamoDBContext(client, config);
```

Contoh C # berikut membuat yang baru `DynamoDBContext` dengan menentukan dua parameter opsional sebelumnya, dan. `ConsistentRead` `SkipVersionCheck`

Example

```
AmazonDynamoDBClient client = new AmazonDynamoDBClient();
...
DynamoDBContext context =
       new DynamoDBContext(client, new DynamoDBContextConfig { ConsistentRead = true,
 SkipVersionCheck = true});
```

`DynamoDBContext` menyertakan parameter opsional ini dengan setiap permintaan yang Anda kirimkan menggunakan konteks ini.

Alih-alih mengatur parameter ini di tingkat `DynamoDBContext`, Anda dapat menentukannya untuk setiap operasi yang Anda jalankan menggunakan `DynamoDBContext`, seperti yang ditunjukkan dalam contoh kode C# berikut. Contoh ini memuat item buku tertentu. `Load`Metode `DynamoDBContext` menentukan `ConsistentRead` dan parameter `SkipVersionCheck` opsional.

Example

```
AmazonDynamoDBClient client = new AmazonDynamoDBClient();
...
DynamoDBContext context = new DynamoDBContext(client);
Book bookItem = context.Load<Book>(productId,new DynamoDBContextConfig{ ConsistentRead
 = true, SkipVersionCheck = true });
```

Dalam kasus ini, `DynamoDBContext` menyertakan parameter tersebut hanya ketika mengirimkan permintaan `Get`.

Penguncian optimis menggunakan DynamoDB dan model persistensi objek AWS SDK untuk .NET

Dukungan penguncian optimis dalam model persistensi objek memastikan bahwa versi item untuk aplikasi Anda sama dengan versi item di sisi server sebelum memperbarui atau menghapus item. Misalkan Anda mengambil item untuk pembaruan. Namun, sebelum Anda mengirimkan pembaruan Anda kembali, beberapa aplikasi lain memperbarui item yang sama. Sekarang aplikasi Anda memiliki salinan item yang sudah usang. Tanpa penguncian optimis, setiap pembaruan yang Anda lakukan akan menimpa pembaruan yang dibuat oleh aplikasi lain.

Fitur penguncian optimis model persistensi objek menyediakan tanda `DynamoDBVersion` yang dapat Anda gunakan untuk mengaktifkan penguncian optimis. Untuk menggunakan fitur ini, Anda menambahkan properti ke kelas Anda untuk menyimpan nomor versi. Anda menambahkan atribut `DynamoDBVersion` untuk properti. Ketika Anda pertama kali menyimpan objek, `DynamoDBContext` menetapkan nomor versi dan menambah nilai ini setiap kali Anda memperbarui item.

Permintaan perbarui atau hapus Anda akan berhasil hanya jika versi objek sisi kliennya cocok dengan nomor versi item yang sesuai di sisi server. Jika aplikasi Anda memiliki salinan yang sudah usang, aplikasi tersebut harus mendapatkan versi terbaru dari server sebelum dapat memperbarui atau menghapus item.

Contoh kode C# berikut mendefinisikan kelas `Book` dengan atribut persistensi objek yang memetakannya ke tabel `ProductCatalog`. Properti `VersionNumber` di kelas yang dilengkapi dengan atribut `DynamoDBVersion` menyimpan nilai nomor versi.

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**
>
> Anda dapat menerapkan atribut `DynamoDBVersion` hanya untuk jenis primitif numerik yang dapat di-null-kan (seperti `int?`).

Penguncian optimis memiliki dampak berikut terhadap operasi `DynamoDBContext`:

- Untuk item baru, `DynamoDBContext` menetapkan nomor versi awal 0. Jika Anda mengambil item yang ada, memperbarui satu atau beberapa properti, dan mencoba untuk menyimpan perubahan, operasi simpan hanya akan berhasil jika nomor versi di sisi klien dan sisi server cocok. `DynamoDBContext` menambahkan nomor versi. Anda tidak perlu mengatur nomor versi.
- Metode `Delete` menyediakan beban berlebih yang dapat mengambil nilai kunci primer atau objek sebagai parameter, seperti yang ditunjukkan dalam contoh kode C# berikut.

  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);
  ```

  Jika Anda menyediakan objek sebagai parameter, penghapusan berhasil hanya jika versi objek cocok dengan versi item sisi server yang sesuai. Namun, jika Anda memberikan nilai kunci primer sebagai parameter, `DynamoDBContext` tidak mengenali nomor versi apa pun, dan menghapus item tanpa melakukan pemeriksaan versi.

  Perhatikan bahwa implementasi internal penguncian optimis dalam kode model persistensi objek menggunakan tindakan API pembaruan bersyarat dan penghapusan bersyarat di DynamoDB.

Menonaktifkan penguncian optimis

Untuk menonaktifkan penguncian optimis, gunakan properti konfigurasi `SkipVersionCheck`. Anda dapat mengatur properti ini saat membuat `DynamoDBContext`. Dalam kasus ini, penguncian optimis dinonaktifkan untuk permintaan apa pun yang Anda buat menggunakan konteks. Untuk informasi selengkapnya, lihat Menentukan parameter opsional untuk Dynamo DBContext .

Alih-alih menetapkan properti pada tingkat konteks, Anda dapat menonaktifkan penguncian optimis untuk operasi tertentu, seperti yang ditunjukkan dalam contoh kode C# berikut. Contoh ini menggunakan konteks untuk menghapus item buku. Metode `Delete` menetapkan properti `SkipVersionCheck` opsional ke true, menonaktifkan pemeriksaan penentuan versi.

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 });
```

Memetakan data arbitrer dengan DynamoDB menggunakan model persistensi objek AWS SDK untuk .NET

Selain jenis .NET yang didukung (lihat Jenis data yang didukung), Anda dapat menggunakan jenis di aplikasi Anda yang tidak memiliki pemetaan langsung ke jenis Amazon DynamoDB. Model persistensi objek mendukung jenis penyimpanan data arbitrer selama Anda memberikan konverter untuk mengonversi data dari jenis arbitrer untuk jenis DynamoDB dan sebaliknya. Kode konverter mengubah data saat menyimpan dan memuat objek.

Anda dapat membuat jenis apa pun di sisi klien. Namun, data yang disimpan di tabel adalah salah satu jenis DynamoDB, dan selama kueri dan pemindaian, perbandingan data apa pun yang dibuat didasarkan pada data yang disimpan di DynamoDB.

Contoh kode C# berikut mendefinisikan kelas `Book` dengan properti `Id`, `Title`, `ISBN`, dan `Dimension`. Properti `Dimension` adalah bagian dari `DimensionType` yang mendeskripsikan properti `Height`, `Width`, dan `Thickness`. Contoh kode ini menyediakan metode konverter `ToEntry` dan `FromEntry` untuk mengonversi data antara jenis string `DimensionType` dan DynamoDB. Misalnya, saat menyimpan instans `Book`, konverter membuat sebuah string `Dimension` buku seperti

“8.5x11x.05". Saat Anda mengambil buku, konverter akan mengonversi string tersebut menjadi instans `DimensionType`.

Contoh ini memetakan jenis `Book` ke tabel `ProductCatalog`. Ini menyimpan instans `Book` sampel, mengambilnya, memperbarui dimensinya, dan menyimpan `Book` yang terbaru lagi.

Untuk step-by-step instruksi untuk menguji contoh berikut, lihatContoh kode .NET.

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;
        }
    }
}
```

Menjalankan contoh kode dalam Panduan Developer ini

AWS SDKs Memberikan dukungan luas untuk Amazon DynamoDB dalam bahasa berikut:

- Java
- JavaScript di browser
- .NET
- Node.js
- PHP
- Python
- Ruby
- C++
- Go

- Android
- iOS

Contoh kode dalam panduan developer ini menyediakan cakupan yang lebih mendalam tentang operasi DynamoDB, menggunakan bahasa pemrograman berikut:

- Contoh kode Java
- Contoh kode .NET

Sebelum Anda dapat memulai dengan latihan ini, Anda perlu membuat AWS akun, mendapatkan kunci akses dan kunci rahasia, dan mengatur AWS Command Line Interface (AWS CLI) di komputer Anda. Untuk informasi selengkapnya, lihat Menyiapkan DynamoDB (layanan web) .

> **Note**
>
> Jika Anda menggunakan versi DynamoDB yang dapat diunduh, Anda perlu menggunakan untuk membuat tabel AWS CLI dan data sampel. Anda juga perlu menentukan `--endpoint-url` parameter dengan setiap AWS CLI perintah. Untuk informasi selengkapnya, lihat Mengatur Titik Akhir Lokal .

Membuat tabel dan memuat data untuk contoh kode di DynamoDB

Lihat di bawah untuk dasar-dasar membuat tabel di DynamoDB, memuat sampel set data, membuat kueri data, dan memperbarui data.

- Langkah 1: Buat tabel di DynamoDB
- Langkah 2: Menulis data ke tabel DynamoDB
- Langkah 3: Baca data dari tabel DynamoDB
- Langkah 4: Perbarui data dalam tabel DynamoDB

Contoh kode Java

Topik

- Java: Mengatur kredensial AWS Anda
- Java: Mengatur AWS Wilayah dan titik akhir

Panduan Pengembang ini berisi cuplikan kode Java dan ready-to-run program. Anda dapat menemukan contoh kode ini di bagian berikut:

- Bekerja dengan item dan atribut di DynamoDB
- Bekerja dengan tabel dan data di DynamoDB
- Menanyakan tabel di DynamoDB
- Memindai tabel di DynamoDB
- Meningkatkan akses data dengan indeks sekunder di DynamoDB
- Java 1.x: Dinamo DBMapper
- Tangkapan data perubahan DynamoDB Streams

Anda dapat memulai dengan cepat menggunakan Eclipse dengan AWS Toolkit for Eclipse. Selain IDE berfitur lengkap, Anda juga mendapatkan pembaruan otomatis, dan templat AWS SDK untuk Java yang telah dikonfigurasi sebelumnya untuk membangun AWS aplikasi.

Untuk menjalankan contoh kode Java (menggunakan Eclipse)

1. Unduh dan instal IDE Eclipse.
2. Unduh dan instal AWS Toolkit for Eclipse.
3. Mulai Eclipse, dan pada menu Eclipse, pilih File, Baru, lalu Lainnya.
4. Di Pilih wizard, pilih AWS, pilih Proyek Java AWS , lalu pilih Selanjutnya.
5. Di Create an AWS Java, lakukan hal berikut:

    a. Dalam Nama proyek, masukkan nama untuk proyek Anda.

    b. Di Pilih Akun, pilih profil kredensial Anda dari daftar.

    Jika ini adalah pertama kalinya Anda menggunakan AWS Toolkit for Eclipse, pilih Konfigurasi AWS Akun untuk menyiapkan AWS kredensil Anda.
6. Pilih Selesai untuk membuat proyek.
7. Dari menu Eclipse, pilih File, Baru, kemudian Kelas.
8. Dalam Kelas Java, masukkan nama untuk kelas Anda di Nama (menggunakan nama yang sama sebagai contoh kode yang ingin Anda jalankan), lalu pilih Selesai untuk membuat kelas.
9. Salin contoh kode dari halaman dokumentasi ke editor Eclipse.
10. Untuk menjalankan kode, pilih Jalankan pada menu Eclipse.

SDK untuk Java menyediakan klien thread-safe untuk bekerja dengan DynamoDB. Sebagai praktik terbaik, aplikasi Anda harus membuat satu klien dan menggunakan kembali klien di antara thread.

Untuk informasi selengkapnya, lihat AWS SDK untuk Java.

> **Note**
>
> Contoh kode dalam panduan ini dimaksudkan untuk digunakan dengan versi terbaru dari AWS SDK untuk Java.
> Jika Anda menggunakan AWS Toolkit for Eclipse, Anda dapat mengonfigurasi pembaruan otomatis untuk SDK for Java. Untuk melakukan ini di Eclipse, buka Preferensi dan pilih AWS Toolkit, AWS SDK untuk Java, Unduh baru SDKs secara otomatis.

Java: Mengatur kredensial AWS Anda

SDK for Java mengharuskan Anda AWS memberikan kredensi ke aplikasi Anda saat runtime. Contoh kode dalam panduan ini mengasumsikan bahwa Anda menggunakan file AWS kredensial, seperti yang dijelaskan dalam Mengatur AWS kredensil Anda di Panduan Pengembang.AWS SDK untuk Java

Berikut ini adalah contoh file AWS kredensial bernama`~/.aws/credentials`, di mana karakter tilde (~) mewakili direktori home Anda.

```
[default]
aws_access_key_id = AWS access key ID goes here
aws_secret_access_key = Secret key goes here
```

Java: Mengatur AWS Wilayah dan titik akhir

Secara default, contoh kode mengakses DynamoDB di Wilayah AS Barat (Oregon). Anda dapat mengubah Wilayah dengan memodifikasi properti `AmazonDynamoDB`.

Contoh kode berikut menginstansiasi `AmazonDynamoDB` baru.

```
import com.amazonaws.services.dynamodbv2.AmazonDynamoDBClientBuilder;
import com.amazonaws.regions.Regions;
...
// This client will default to US West (Oregon)
AmazonDynamoDB client = AmazonDynamoDBClientBuilder.standard()
```

```
.withRegion(Regions.US_WEST_2)
.build();
```

Anda dapat menggunakan metode `withRegion` untuk menjalankan kode Anda terhadap DynamoDB di Wilayah ketersediaan mana pun. Untuk daftar lengkap, lihat wilayah dan titik akhir AWS di Referensi Umum Amazon Web Services.

Jika Anda ingin menjalankan contoh kode menggunakan DynamoDB secara lokal pada komputer Anda, atur titik akhir sebagai berikut.

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();
```

Contoh kode .NET

Topik

- .NET: Mengatur kredensial AWS Anda
- .NET: Mengatur Wilayah dan titik akhir AWS

Panduan ini berisi potongan kode .NET dan ready-to-run program. Anda dapat menemukan contoh kode ini di bagian berikut:

- Bekerja dengan item dan atribut di DynamoDB
- Bekerja dengan tabel dan data di DynamoDB

- Menanyakan tabel di DynamoDB
- Memindai tabel di DynamoDB
- Meningkatkan akses data dengan indeks sekunder di DynamoDB
- Bekerja dengan model dokumen.NET di DynamoDB
- Bekerja dengan model persistensi objek.NET dan DynamoDB
- Tangkapan data perubahan DynamoDB Streams

Anda dapat memulai dengan cepat dengan menggunakan Toolkit for Visual Studio. AWS SDK untuk .NET

Untuk menjalankan contoh kode .NET (menggunakan Visual Studio)

1. Unduh dan instal Microsoft Visual Studio.
2. Unduh dan instal Toolkit for Visual Studio.
3. Mulai Visual Studio. Pilih File, Baru, Proyek.
4. Dalam Proyek Baru, pilih Proyek Kosong AWS , lalu pilih OK.
5. Dalam Kredensial Akses AWS , pilih Gunakan profil yang ada, pilih kredensial profil Anda dari daftar, lalu pilih OK.

   Jika ini adalah pertama kalinya Anda menggunakan Toolkit for Visual Studio, pilih Gunakan profil baru untuk mengatur AWS kredensialnya.
6. Dalam proyek Visual Studio, pilih tab untuk kode sumber program Anda (`Program.cs`). Salin contoh kode dari halaman dokumentasi ke editor Visual Studio, menggantikan kode lain yang Anda lihat di editor.
7. Jika Anda melihat pesan kesalahan formulir Jenis atau nama namespace... tidak dapat ditemukan, Anda perlu menginstal rakitan AWS SDK untuk DynamoDB sebagai berikut:

   a. Di Solution Explorer, buka menu konteks (klik kanan) untuk proyek Anda, lalu pilih Kelola NuGet Paket.

   b. Di NuGet Package Manager, pilih Browse.

   c. Di kotak pencarian, masukkan **`AWSSDK.DynamoDBv2`**, lalu tunggu hingga pencarian selesai.

   d. Pilih AWSSDK.Dynamo DBv2, dan kemudian pilih Install.

   e. Saat penginstalan selesai, pilih opsi tab Program.cs untuk kembali ke program Anda.
8. Untuk menjalankan kode, pilih Mulai di toolbar Visual Studio.

SDK untuk .NET Ini menyediakan klien thread-safe untuk bekerja dengan DynamoDB. Sebagai praktik terbaik, aplikasi Anda harus membuat satu klien dan menggunakan kembali klien di antara thread.

Untuk informasi selengkapnya, lihat AWS SDK untuk .NET.

> **Note**
>
> Contoh kode dalam panduan ini dimaksudkan untuk digunakan dengan versi terbaru dari AWS SDK untuk .NET.

.NET: Mengatur kredensial AWS Anda

Ini SDK untuk .NET mengharuskan Anda memberikan AWS kredensil ke aplikasi Anda saat runtime. Contoh kode dalam panduan ini mengasumsikan bahwa Anda menggunakan SDK Store untuk mengelola file AWS kredensial Anda, seperti yang dijelaskan dalam Menggunakan penyimpanan SDK di Panduan Pengembang.AWS SDK untuk .NET

Toolkit for Visual Studio mendukung beberapa set kredensial dari sejumlah akun. Setiap set disebut sebagai profil. Visual Studio menambahkan entri ke `App.config` file proyek sehingga aplikasi Anda dapat menemukan AWS kredensialnya saat runtime.

Contoh berikut menunjukkan file `App.config` default yang dihasilkan ketika Anda membuat proyek baru menggunakan Toolkit for Visual Studio.

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
    <appSettings>
    <add key="AWSProfileName" value="default"/>
    <add key="AWSRegion" value="us-west-2" />
 </appSettings>
</configuration>
```

Saat runtime, program menggunakan `default` set AWS kredensil, seperti yang ditentukan oleh entri. `AWSProfileName` AWS Kredensialnya sendiri disimpan di SDK Store dalam bentuk terenkripsi. Toolkit for Visual Studio menyediakan antarmuka pengguna grafis untuk mengelola kredensial Anda, semua dari dalam Visual Studio. Untuk informasi selengkapnya, lihat Menentukan kredensial dalam Panduan Pengguna AWS Toolkit for Visual Studio .

> **Note**
>
> Secara default, contoh kode mengakses DynamoDB di Wilayah AS Barat (Oregon). Anda dapat mengubah Wilayah dengan memodifikasi entri `AWSRegion` dalam file App.config. Anda dapat mengatur `AWSRegion` ke Wilayah mana pun tempat DynamoDB tersedia. Untuk daftar lengkap, lihat wilayah dan titik akhir AWS di Referensi Umum Amazon Web Services.

.NET: Mengatur Wilayah dan titik akhir AWS

Secara default, contoh kode mengakses DynamoDB di Wilayah AS Barat (Oregon). Anda dapat mengubah Wilayah dengan memodifikasi entri `AWSRegion` dalam file `App.config`. Atau, Anda dapat mengubah Wilayah dengan memodifikasi properti `AmazonDynamoDBClient`.

Contoh kode berikut menginstansiasi `AmazonDynamoDBClient` baru. Klien dimodifikasi sehingga kode berjalan terhadap DynamoDB di Wilayah yang berbeda.

```
AmazonDynamoDBConfig clientConfig = new AmazonDynamoDBConfig();
// This client will access the US East 1 region.
clientConfig.RegionEndpoint = RegionEndpoint.USEast1;
AmazonDynamoDBClient client = new AmazonDynamoDBClient(clientConfig);
```

Untuk daftar lengkap Wilayah, lihat wilayah dan titik akhir AWS di Referensi Umum Amazon Web Services.

Jika Anda ingin menjalankan contoh kode menggunakan DynamoDB secara lokal pada komputer Anda, atur titik akhir sebagai berikut.

```
AmazonDynamoDBConfig clientConfig = new AmazonDynamoDBConfig();
// Set the endpoint URL
clientConfig.ServiceURL = "http://localhost:8000";
AmazonDynamoDBClient client = new AmazonDynamoDBClient(clientConfig);
```

API tingkat rendah DynamoDB

API tingkat rendah Amazon DynamoDB adalah antarmuka tingkat protokol untuk DynamoDB. Pada tingkat ini, setiap permintaan HTTP(S) harus diformat dengan benar dan membawa tanda tangan digital yang valid.

AWS SDKs Membangun permintaan API DynamoDB tingkat rendah atas nama Anda dan memproses tanggapan dari DynamoDB. Ini memungkinkan Anda berfokus pada logika aplikasi Anda, bukan detail tingkat rendah. Namun, Anda masih bisa mendapatkan keuntungan dari pengetahuan dasar tentang bagaimana DynamoDB API tingkat rendah bekerja.

Untuk informasi selengkapnya tentang DynamoDB API, lihat Referensi Amazon DynamoDB API.

> **Note**
>
> DynamoDB Streams memiliki API tingkat rendah sendiri, yang terpisah dari DynamoDB dan sepenuhnya didukung oleh. AWS SDKs
> Untuk informasi selengkapnya, lihat Tangkapan data perubahan DynamoDB Streams. Untuk DynamoDB Streams API tingkat rendah, lihat Referensi Amazon DynamoDB Streams API.

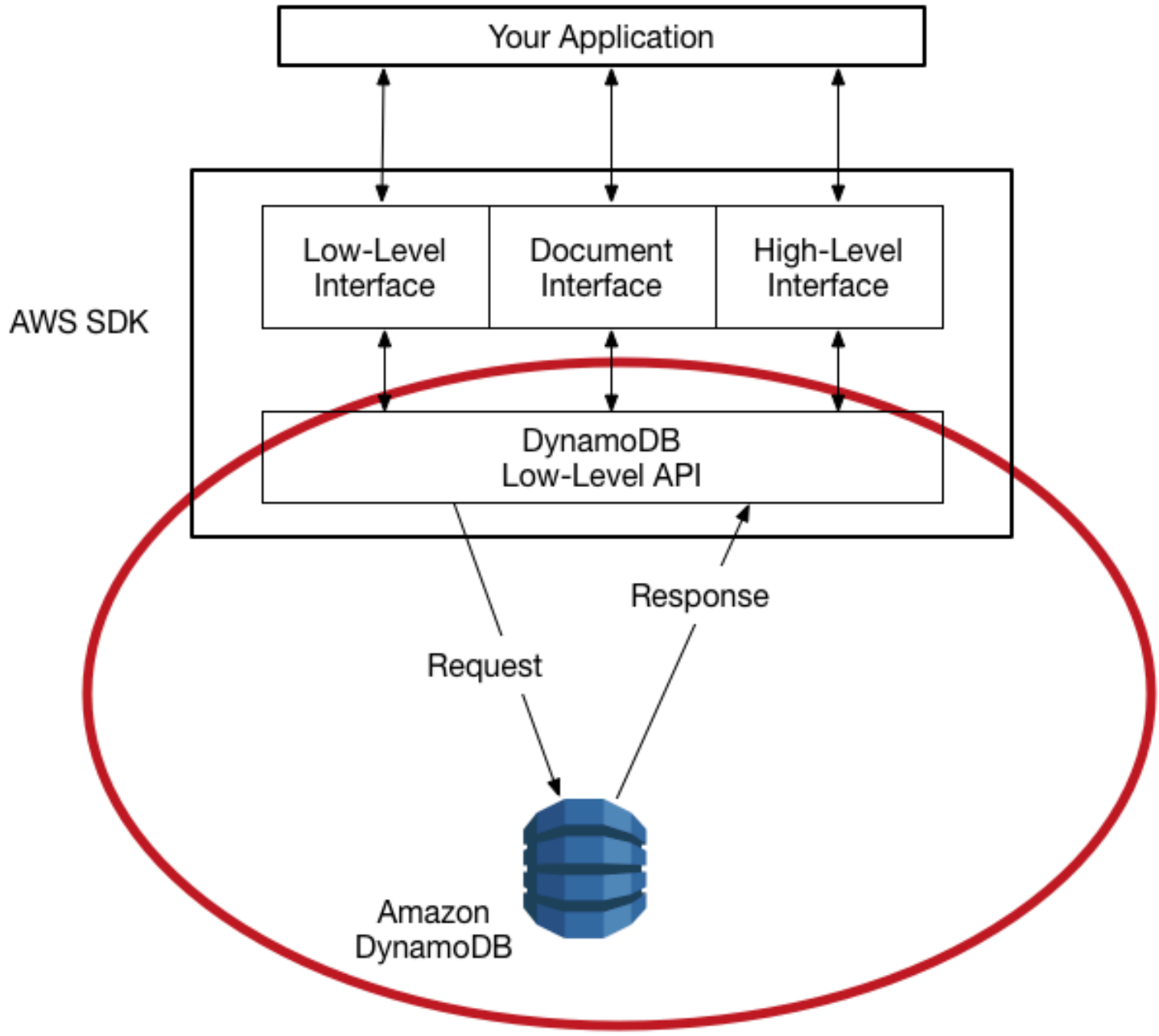
DynamoDB API tingkat rendah JavaScript menggunakan Object Notation (JSON) sebagai format protokol kawat. JSON menyajikan data dalam hierarki sehingga nilai data dan struktur data disampaikan secara bersamaan. Pasangan nama-nilai ditentukan dalam format `name:value`. Hierarki data ditentukan oleh tanda kurung berisi pasangan nama-nilai.

DynamoDB menggunakan JSON hanya sebagai protokol transportasi, bukan sebagai format penyimpanan. AWS SDKs Penggunaan JSON untuk mengirim data ke DynamoDB, dan DynamoDB merespons dengan JSON. DynamoDB tidak menyimpan data terus-menerus dalam format JSON.

> **Note**
>
> Untuk informasi selengkapnya tentang JSON, lihat Memperkenalkan JSON pada situs web `JSON.org`.

Topik

- Format permintaan
- Format respons
- Deskriptor jenis data
- Data numerik
- Data biner



Format permintaan

API tingkat rendah DynamoDB menerima permintaan `POST` HTTP(S) sebagai masukan. AWS SDKs Bangun permintaan ini untuk Anda.

Misalkan Anda memiliki tabel bernama `Pets`, dengan skema kunci yang terdiri dari `AnimalType` (kunci partisi) dan `Name` (kunci urutan). Kedua atribut ini berjenis `string`. Untuk mengambil item dari`Pets`, AWS SDK membuat permintaan berikut.

```
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"}
    }
}
```

Perhatikan hal berikut tentang permintaan ini:

- Header `Authorization` berisi informasi yang diperlukan untuk DynamoDB guna mengautentikasi permintaan. Untuk informasi selengkapnya, lihat Menandatangani permintaan AWS API dan proses penandatanganan Versi Tanda Tangan 4 di bagian Referensi Umum Amazon Web Services.
- Header `X-Amz-Target` berisi nama operasi DynamoDB: `GetItem`. (Ini juga disertai dengan versi API tingkat rendah, dalam hal ini `20120810`.)
- Muatan (isi) permintaan mengandungi parameter untuk operasi, dalam format JSON. Untuk operasi `GetItem`, parameternya adalah `TableName` dan `Key`.

Format respons

Setelah menerima permintaan, DynamoDB memprosesnya dan mengembalikan respons. Untuk permintaan yang ditunjukkan sebelumnya, muatan respons HTTP(S) berisi hasil dari operasi, seperti yang ditunjukkan pada contoh berikut.

```
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"}
    }
}
```

Pada titik ini, AWS SDK mengembalikan data respons ke aplikasi Anda untuk diproses lebih lanjut.

> **Note**
>
> Jika DynamoDB tidak dapat memproses permintaan, DynamoDB akan mengembalikan kode kesalahan HTTP dan pesan. AWS SDK menyebarkannya untuk aplikasi Anda dalam bentuk pengecualian. Untuk informasi selengkapnya, lihat Penanganan kesalahan dengan DynamoDB.

Deskriptor jenis data

Protokol DynamoDB API tingkat rendah membutuhkan setiap atribut untuk disertai dengan deskriptor jenis data. Deskriptor jenis data adalah token yang memberi tahu DynamoDB cara menafsirkan setiap atribut.

Contoh di Format permintaan dan Format respons menunjukkan contoh bagaimana deskriptor jenis data digunakan. Permintaan `GetItem` menentukan `S` untuk atribut skema kunci `Pets` (`AnimalType` dan `Name`), yang berjenis `string`. Respons `GetItem` berisi item Pets dengan atribut jenis `string` (`S`), `number` (`N`), `map` (`M`), dan `list` (`L`).

Berikut ini adalah daftar lengkap deskriptor jenis data DynamoDB:

- **`S`** – String
- **`N`** – Nomor
- **`B`** – Biner
- **`BOOL`** – Boolean
- **`NULL`** – Null
- **`M`** – Peta
- **`L`** – Daftar
- **`SS`** – Set String
- **`NS`** – Set Nomor
- **`BS`** – Set Biner

> **Note**
>
> Untuk deskripsi mendetail tentang jenis data DynamoDB, lihat Jenis Data.

Data numerik

Bahasa pemrograman yang berbeda menawarkan tingkat dukungan yang berbeda untuk JSON. Dalam beberapa kasus, Anda mungkin memutuskan untuk menggunakan pustaka pihak ketiga untuk memvalidasi dan menguraikan dokumen JSON.

Beberapa pustaka pihak ketiga membangun jenis angka JSON, menyediakan jenisnya sendiri seperti `int`, `long`, atau `double`. Namun, jenis data angka asli di DynamoDB tidak memetakan persis untuk

jenis data lainnya, sehingga perbedaan jenis ini dapat menyebabkan konflik. Selain itu, banyak pustaka JSON tidak menangani nilai numerik presisi tetap, dan secara otomatis menyimpulkan jenis data ganda untuk urutan digit yang berisi titik desimal.

Untuk mengatasi masalah ini, DynamoDB menyediakan jenis numerik tunggal tanpa kehilangan data. Untuk menghindari konversi implisit yang tidak diinginkan ke nilai ganda, DynamoDB menggunakan string untuk transfer data nilai numerik. Pendekatan ini memberikan fleksibilitas untuk memperbarui nilai atribut sekaligus mempertahankan semantik pengurutan yang tepat, seperti menempatkan nilai "01", "2", dan "03" dalam urutan yang tepat.

Jika presisi angka penting untuk aplikasi, Anda harus mengonversi nilai numerik untuk string tersebut sebelum meneruskannya ke DynamoDB.

Data biner

DynamoDB mendukung atribut binari. Namun, JSON tidak secara native mendukung data biner pengodean. Untuk mengirim data biner dalam permintaan, Anda perlu untuk mengodekannya dalam format base64. Setelah menerima permintaan, DynamoDB mengodekan data base64 kembali ke biner.

Skema pengodean base64 yang digunakan oleh DynamoDB dijelaskan di RFC 4648 di situs web Internet Engineering Task Force (IETF).

Pemrograman Amazon DynamoDB dengan Python dan Boto3

Panduan ini memberikan orientasi kepada programmer yang ingin menggunakan Amazon DynamoDB dengan Python. Pelajari tentang berbagai lapisan abstraksi, manajemen konfigurasi, penanganan kesalahan, pengendalian kebijakan coba lagi, pengelolaan keep-alive, dan banyak lagi.

Topik

- Tentang Boto
- Menggunakan dokumentasi Boto
- Memahami lapisan abstraksi klien dan sumber daya
- Menggunakan sumber daya tabel batch_writer
- Contoh kode tambahan yang mengeksplorasi lapisan klien dan sumber daya
- Memahami bagaimana objek Klien dan Sumber Daya berinteraksi dengan sesi dan utas
- Menyesuaikan objek Config
- Penanganan kesalahan

- Pencatatan log
- Kait acara
- Pagination dan Paginator
- Pelayan

Tentang Boto

Anda dapat mengakses DynamoDB dari Python dengan menggunakan SDK AWS resmi untuk Python, yang biasa disebut sebagai Boto3. Nama Boto (diucapkan boh-toh) berasal dari lumba-lumba air tawar asli Sungai Amazon. Perpustakaan Boto3 adalah versi utama ketiga perpustakaan, pertama kali dirilis pada tahun 2015. Perpustakaan Boto3 cukup besar, karena mendukung semua AWS layanan, bukan hanya DynamoDB. Orientasi ini hanya menargetkan bagian Boto3 yang relevan dengan DynamoDB.

Boto dikelola dan diterbitkan oleh AWS sebagai proyek sumber terbuka yang dihosting di. GitHub Ini dibagi menjadi dua paket: Botocore dan Boto3.

- Botocore menyediakan fungsionalitas tingkat rendah. Di Botocore Anda akan menemukan kelas klien, sesi, kredensyal, konfigurasi, dan pengecualian.
- Boto3 dibangun di atas Botocore. Ini menawarkan antarmuka Pythonic tingkat yang lebih tinggi dan lebih banyak. Secara khusus, ini mengekspos tabel DynamoDB sebagai Sumber Daya dan menawarkan antarmuka yang lebih sederhana dan lebih elegan dibandingkan dengan antarmuka klien berorientasi layanan tingkat rendah.

Karena proyek ini di-host GitHub, Anda dapat melihat kode sumber, melacak masalah terbuka, atau mengirimkan masalah Anda sendiri.

Menggunakan dokumentasi Boto

Mulailah dengan dokumentasi Boto dengan sumber daya berikut:

- Mulailah dengan bagian Quickstart yang menyediakan titik awal yang solid untuk instalasi paket. Pergi ke sana untuk petunjuk tentang menginstal Boto3 jika belum (Boto3 sering tersedia secara otomatis dalam AWS layanan seperti). AWS Lambda
- Setelah itu, fokuslah pada panduan DynamoDB dokumentasi. Ini menunjukkan cara melakukan aktivitas DynamoDB dasar: membuat dan menghapus tabel, memanipulasi item, menjalankan operasi batch, menjalankan kueri, dan melakukan pemindaian. Contohnya menggunakan

antarmuka sumber daya. Ketika Anda melihat `boto3.resource('dynamodb')` itu menunjukkan Anda menggunakan antarmuka sumber daya tingkat yang lebih tinggi.

- Setelah panduan, Anda dapat meninjau referensi DynamoDB. Halaman arahan ini menyediakan daftar lengkap kelas dan metode yang tersedia untuk Anda. Di bagian atas, Anda akan melihat `DynamoDB.Client` kelas. Ini memberikan akses tingkat rendah ke semua operasi bidang kontrol dan bidang data. Di bagian bawah, lihat `DynamoDB.ServiceResource` kelasnya. Ini adalah antarmuka Pythonic tingkat yang lebih tinggi. Dengan itu Anda dapat membuat tabel, melakukan operasi batch di seluruh tabel, atau mendapatkan `DynamoDB.ServiceResource.Table` instance untuk tindakan khusus tabel.

Memahami lapisan abstraksi klien dan sumber daya

Dua antarmuka yang akan Anda gunakan adalah antarmuka klien dan antarmuka sumber daya.

- Antarmuka klien tingkat rendah menyediakan pemetaan 1-ke-1 ke API layanan yang mendasarinya. Setiap API yang ditawarkan oleh DynamoDB tersedia melalui klien. Ini berarti antarmuka klien dapat menyediakan fungsionalitas yang lengkap, tetapi seringkali lebih bertele-tele dan kompleks untuk digunakan.
- Antarmuka sumber daya tingkat yang lebih tinggi tidak menyediakan pemetaan 1-ke-1 dari API layanan yang mendasarinya. Namun, ini menyediakan metode yang membuatnya lebih nyaman bagi Anda untuk mengakses layanan seperti`batch_writer`.

Berikut adalah contoh menyisipkan item menggunakan antarmuka klien. Perhatikan bagaimana semua nilai dilewatkan sebagai peta dengan kunci yang menunjukkan tipenya ('S' untuk string, 'N' untuk nomor) dan nilainya sebagai string. Ini dikenal sebagai format DynamoDB JSON.

```
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 ...
```

```
    }
)
```

Berikut adalah `PutItem` operasi yang sama menggunakan antarmuka sumber daya. Pengetikan data tersirat:

```
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 ...
    }
)
```

Jika diperlukan, Anda dapat mengonversi antara JSON biasa dan DynamoDB JSON menggunakan `TypeSerializer` kelas dan yang disediakan dengan boto3: `TypeDeserializer`

```
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()
    }
```

Berikut adalah cara melakukan query menggunakan antarmuka klien. Ini mengekspresikan query sebagai konstruksi JSON. Ini menggunakan `KeyConditionExpression` string yang membutuhkan substitusi variabel untuk menangani konflik kata kunci potensial:

```
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',
    }
)
```

Operasi kueri yang sama menggunakan antarmuka sumber daya dapat dipersingkat dan disederhanakan:

```
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')
)
```

Sebagai contoh terakhir, bayangkan Anda ingin mendapatkan perkiraan ukuran tabel (yang merupakan metadata yang disimpan di atas meja yang diperbarui setiap 6 jam). Dengan antarmuka klien, Anda melakukan `describe_table()` operasi dan menarik jawaban dari struktur JSON yang dikembalikan:

```
import boto3

dynamodb = boto3.client('dynamodb')

response = dynamodb.describe_table(TableName='YourTableName')
size = response['Table']['TableSizeBytes']
```

Dengan antarmuka sumber daya, tabel melakukan operasi deskripsi secara implisit dan menyajikan data secara langsung sebagai atribut:

```
import boto3

dynamodb = boto3.resource('dynamodb')

table = dynamodb.Table('YourTableName')
size = table.table_size_bytes
```

> **Note**
>
> Saat mempertimbangkan apakah akan mengembangkan menggunakan antarmuka klien atau sumber daya, ketahuilah bahwa fitur baru tidak akan ditambahkan ke antarmuka sumber daya per dokumentasi sumber daya: “Tim SDK AWS Python tidak bermaksud menambahkan fitur baru ke antarmuka sumber daya di boto3. Antarmuka yang ada akan terus beroperasi selama siklus hidup boto3. Pelanggan dapat menemukan akses ke fitur layanan yang lebih baru melalui antarmuka klien.

Menggunakan sumber daya tabel batch_writer

Satu kenyamanan yang hanya tersedia dengan sumber daya tabel tingkat yang lebih tinggi adalah. `batch_writer` DynamoDB mendukung operasi batch write yang memungkinkan hingga 25 operasi put atau delete dalam satu permintaan jaringan. Batching seperti ini meningkatkan efisiensi dengan meminimalkan perjalanan pulang-pergi jaringan.

Dengan pustaka klien tingkat rendah, Anda menggunakan `client.batch_write_item()` operasi untuk menjalankan batch. Anda harus membagi pekerjaan Anda secara manual menjadi 25 batch. Setelah setiap operasi, Anda juga harus meminta untuk menerima daftar item yang belum diproses

(beberapa operasi penulisan mungkin berhasil sementara yang lain bisa gagal). Anda kemudian harus meneruskan item yang belum diproses itu lagi ke `batch_write_item()` operasi selanjutnya. Ada sejumlah besar kode boilerplate.

Metode Table.batch_Writer menciptakan manajer konteks untuk menulis objek dalam batch. Ini menyajikan antarmuka di mana tampaknya seolah-olah Anda sedang menulis item satu per satu, tetapi secara internal itu buffering dan mengirim item dalam batch. Ini juga menangani percobaan ulang item yang belum diproses secara implisit.

```
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)
```

Contoh kode tambahan yang mengeksplorasi lapisan klien dan sumber daya

Anda juga dapat merujuk ke repositori contoh kode berikut yang mengeksplorasi penggunaan berbagai fungsi, menggunakan klien dan sumber daya:

- Contoh kode AWS aksi tunggal resmi.
- Contoh kode AWS berorientasi skenario resmi.
- Contoh kode aksi tunggal yang dikelola komunitas.

Memahami bagaimana objek Klien dan Sumber Daya berinteraksi dengan sesi dan utas

Objek Resource tidak aman untuk utas dan tidak boleh dibagikan di seluruh utas atau proses. Lihat panduan tentang Sumber Daya untuk lebih jelasnya.

Objek Klien, sebaliknya, umumnya aman untuk utas, kecuali untuk fitur lanjutan tertentu. Lihat panduan tentang Klien untuk lebih jelasnya.

Objek Session tidak aman untuk utas. Jadi, setiap kali Anda membuat Klien atau Sumber Daya di lingkungan multi-utas, Anda harus membuat Sesi baru terlebih dahulu dan kemudian membuat Klien atau Sumber Daya dari Sesi. Lihat panduan tentang Sesi untuk lebih jelasnya.

Saat Anda memanggil`boto3.resource()`, Anda secara implisit menggunakan Sesi default. Ini nyaman untuk menulis kode single-threaded. Saat menulis kode multi-utas, Anda harus terlebih dahulu membuat Sesi baru untuk setiap utas dan kemudian mengambil sumber daya dari Sesi itu:

```
# Explicitly create a new Session for this thread
session = boto3.Session()
dynamodb = session.resource('dynamodb')
```

Menyesuaikan objek Config

Saat membuat objek Klien atau Sumber Daya, Anda dapat meneruskan parameter bernama opsional untuk menyesuaikan perilaku. Parameter bernama `config` membuka berbagai fungsi. Ini adalah instance dari `botocore.client.Config` dan dokumentasi referensi untuk Config menunjukkan semua yang diekspos untuk Anda kendalikan. Panduan untuk Konfigurasi memberikan gambaran yang baik.

> **Note**
>
> Anda dapat memodifikasi banyak pengaturan perilaku ini di tingkat Sesi, dalam file AWS konfigurasi, atau sebagai variabel lingkungan.

Config untuk batas waktu

Salah satu penggunaan konfigurasi khusus adalah untuk menyesuaikan perilaku jaringan:

- connect_timeout (float atau int) - Waktu dalam hitungan detik hingga pengecualian batas waktu dilemparkan ketika mencoba membuat koneksi. Bawaannya adalah 60 detik.
- read_timeout (float atau int) - Waktu dalam hitungan detik hingga pengecualian batas waktu dilemparkan ketika mencoba membaca dari koneksi. Bawaannya adalah 60 detik.

Batas waktu 60 detik berlebihan untuk DynamoDB. Ini berarti kesalahan jaringan sementara akan menyebabkan penundaan satu menit untuk klien sebelum dapat mencoba lagi. Kode berikut mempersingkat batas waktu menjadi satu detik:

```
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)
```

Untuk diskusi selengkapnya tentang batas waktu, lihat Menyetel setelan permintaan HTTP SDK AWS Java untuk aplikasi DynamoDB yang sadar latensi. Perhatikan bahwa Java SDK memiliki konfigurasi batas waktu lebih banyak daripada Python.

Config untuk keep-alive

Jika Anda menggunakan botocore 1.27.84 atau yang lebih baru, Anda juga dapat mengontrol TCP Keep-Alive:

- tcp_keepalive (bool) - Mengaktifkan opsi soket TCP Keep-Alive yang digunakan saat membuat koneksi baru jika disetel ke (default ke). `True` `False` Ini hanya tersedia mulai dengan botocore 1.27.84.

Mengatur TCP Keep-Alive untuk `True` dapat mengurangi latensi rata-rata. Berikut contoh kode yang secara kondisional menetapkan TCP Keep-Alive ke true ketika Anda memiliki versi botocore yang tepat:

```
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 Keep-Alive berbeda dari HTTP Keep-Alive. Dengan TCP Keep-Alive, paket-paket kecil dikirim oleh sistem operasi yang mendasarinya melalui koneksi soket untuk menjaga koneksi tetap hidup dan segera mendeteksi tetesan apa pun. Dengan HTTP Keep-Alive, koneksi web yang dibangun di atas soket yang mendasarinya akan digunakan kembali. HTTP Keep-Alive selalu diaktifkan dengan boto3.

Ada batasan berapa lama koneksi idle dapat tetap hidup. Pertimbangkan untuk mengirim permintaan berkala (katakanlah setiap menit) jika Anda memiliki koneksi idle tetapi ingin permintaan berikutnya menggunakan koneksi yang sudah dibuat.

Config untuk percobaan ulang

Konfigurasi juga menerima kamus yang disebut percobaan ulang di mana Anda dapat menentukan perilaku coba lagi yang Anda inginkan. Percobaan ulang terjadi dalam SDK ketika SDK menerima kesalahan dan kesalahan adalah tipe sementara. Jika kesalahan dicoba ulang secara internal (dan percobaan ulang akhirnya menghasilkan respons yang berhasil), tidak ada kesalahan yang terlihat dari perspektif kode panggilan, hanya latensi yang sedikit meningkat. Berikut adalah nilai yang dapat Anda tentukan:

- max_attempts — Bilangan bulat yang mewakili jumlah maksimum percobaan ulang yang akan dilakukan pada satu permintaan. Misalnya, menyetel nilai ini ke 2 akan mengakibatkan permintaan dicoba ulang paling banyak dua kali setelah permintaan awal. Menyetel nilai ini ke 0 akan menghasilkan tidak ada percobaan ulang yang pernah dicoba setelah permintaan awal.
- total_max_attempts — Bilangan bulat yang mewakili jumlah maksimum upaya total yang akan dilakukan pada satu permintaan. Ini termasuk permintaan awal, jadi nilai 1 menunjukkan bahwa tidak ada permintaan yang akan dicoba ulang. Jika `total_max_attempts` dan `max_attempts` keduanya disediakan, `total_max_attempts` diutamakan. `total_max_attempts`lebih disukai daripada `max_attempts` karena memetakan ke variabel `AWS_MAX_ATTEMPTS` lingkungan dan nilai file `max_attempts` konfigurasi.
- mode — String yang mewakili jenis mode coba lagi yang harus digunakan botocore. Nilai yang valid adalah:

- legacy — Mode default. Menunggu 50 ms percobaan ulang pertama, kemudian menggunakan backoff eksponensial dengan faktor dasar 2. Untuk DynamoDB, ia melakukan hingga 10 upaya maksimal total (kecuali diganti dengan yang di atas).

  > **Note**
  >
  > Dengan backoff eksponensial, upaya terakhir akan menunggu hampir 13 detik.

- standard — Dinamakan standar karena lebih konsisten dengan yang lain AWS SDKs. Menunggu waktu acak dari 0ms hingga 1.000 ms untuk percobaan ulang pertama. Jika percobaan lagi diperlukan, ia mengambil waktu acak lain dari 0ms menjadi 1.000 ms dan mengalikannya dengan 2. Jika percobaan ulang tambahan diperlukan, ia melakukan pengambilan acak yang sama dikalikan dengan 4, dan seterusnya. Setiap penantian dibatasi pada 20 detik. Mode ini akan melakukan percobaan ulang pada kondisi kegagalan yang lebih terdeteksi daripada `legacy` mode. Untuk DynamoDB, ia melakukan hingga 3 upaya maksimal total (kecuali diganti dengan yang di atas).
- adaptif - Mode coba lagi eksperimental yang mencakup semua fungsionalitas mode standar tetapi menambahkan pelambatan sisi klien otomatis. Dengan pembatasan tingkat adaptif, SDKs dapat memperlambat tingkat di mana permintaan dikirim untuk lebih mengakomodasi kapasitas AWS layanan. Ini adalah mode sementara yang perilakunya mungkin berubah.

Definisi yang diperluas dari mode coba lagi ini dapat ditemukan di panduan untuk mencoba ulang serta dalam topik perilaku Coba lagi dalam referensi SDK.

Berikut adalah contoh yang secara eksplisit menggunakan kebijakan `legacy` coba lagi dengan maksimal 3 total permintaan (2 percobaan ulang):

```
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)
```

Karena DynamoDB adalah sistem latensi rendah yang sangat tersedia, Anda mungkin ingin lebih agresif dengan kecepatan percobaan ulang daripada yang diizinkan oleh kebijakan coba ulang bawaan. Anda dapat menerapkan kebijakan coba ulang Anda sendiri dengan menyetel upaya maksimal ke 0, menangkap pengecualian sendiri, dan mencoba lagi sebagaimana mestinya dari kode Anda sendiri alih-alih mengandalkan boto3 untuk melakukan percobaan ulang implisit.

Jika Anda mengelola kebijakan coba ulang Anda sendiri, Anda akan ingin membedakan antara throttle dan error:

- Throttle (ditunjukkan oleh `ProvisionedThroughputExceededException` atau`ThrottlingException`) menunjukkan layanan sehat yang memberi tahu Anda bahwa Anda telah melebihi kapasitas baca atau tulis pada tabel atau partisi DynamoDB. Setiap milidetik yang berlalu, sedikit lebih banyak kapasitas baca atau tulis tersedia, sehingga Anda dapat mencoba lagi dengan cepat (seperti setiap 50 ms) untuk mencoba mengakses kapasitas yang baru dirilis. Dengan throttle, Anda tidak memerlukan backoff eksponensial karena throttle ringan untuk DynamoDB untuk kembali dan tidak dikenakan biaya per permintaan kepada Anda. Backoff eksponensial memberikan penundaan yang lebih lama ke utas klien yang telah menunggu paling lama, yang secara statistik memperluas p50 dan p99 ke luar.
- Kesalahan (ditunjukkan oleh `InternalServerError` atau a`ServiceUnavailable`, antara lain) menunjukkan masalah sementara dengan layanan. Ini bisa untuk seluruh tabel atau mungkin hanya partisi yang Anda baca atau tulis. Dengan kesalahan, Anda dapat berhenti lebih lama sebelum mencoba ulang (seperti 250ms atau 500ms) dan menggunakan jitter untuk membuat percobaan ulang terhuyung-huyung.

Config untuk koneksi kolam maksimal

Terakhir, konfigurasi memungkinkan Anda mengontrol ukuran kumpulan koneksi:

- max_pool_connections (int) - Jumlah maksimum koneksi untuk disimpan dalam kumpulan koneksi. Jika nilai ini tidak diatur, nilai default 10 digunakan.

Opsi ini mengontrol jumlah maksimum koneksi HTTP agar tetap dikumpulkan untuk digunakan kembali. Kolam yang berbeda disimpan per Sesi. Jika Anda mengantisipasi lebih dari 10 utas yang bertentangan dengan klien atau sumber daya yang dibangun dari Sesi yang sama, Anda harus

mempertimbangkan untuk meningkatkan ini, jadi utas tidak harus menunggu utas lain menggunakan koneksi gabungan.

```
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
```

Penanganan kesalahan

AWS pengecualian layanan tidak semuanya didefinisikan secara statis di Boto3. Ini karena kesalahan dan pengecualian dari AWS layanan sangat bervariasi dan dapat berubah. Boto3 membungkus semua pengecualian layanan sebagai a `ClientError` dan mengekspos detail sebagai JSON terstruktur. Misalnya, respons kesalahan mungkin terstruktur seperti ini:

```
{
    '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
    }
}
```

Kode berikut menangkap `ClientError` pengecualian dan melihat nilai string di `Code` dalam `Error` untuk menentukan tindakan apa yang harus diambil:

```
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
```

Beberapa (tetapi tidak semua) kode pengecualian telah diwujudkan sebagai kelas tingkat atas. Anda dapat memilih untuk menangani ini secara langsung. Saat menggunakan antarmuka Klien, pengecualian ini diisi secara dinamis di klien Anda dan Anda menangkap pengecualian ini menggunakan instance klien Anda, seperti ini:

```
except ddb_client.exceptions.ProvisionedThroughputExceededException:
```

Saat menggunakan antarmuka Resource, Anda harus menggunakan `.meta.client` untuk melintasi dari sumber daya ke Klien yang mendasarinya untuk mengakses pengecualian, seperti ini:

```
except ddb_resource.meta.client.exceptions.ProvisionedThroughputExceededException:
```

Untuk meninjau daftar jenis pengecualian terwujud, Anda dapat membuat daftar secara dinamis:

```
ddb = boto3.client("dynamodb")
```

```
print([e for e in dir(ddb.exceptions) if e.endswith('Exception') or
 e.endswith('Error')])
```

Saat melakukan operasi tulis dengan ekspresi kondisi, Anda dapat meminta bahwa jika ekspresi gagal, nilai item harus dikembalikan dalam respons kesalahan.

```
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'])
```

Untuk bacaan lebih lanjut tentang penanganan kesalahan dan pengecualian:

- Panduan boto3 tentang penanganan kesalahan memiliki informasi lebih lanjut tentang teknik penanganan kesalahan.
- Bagian panduan pengembang DynamoDB tentang kesalahan pemrograman mencantumkan kesalahan apa yang mungkin Anda temui.
- Bagian Kesalahan Umum dalam referensi API.
- Dokumentasi pada setiap operasi API mencantumkan kesalahan apa yang mungkin dihasilkan oleh panggilan (misalnya BatchWriteItem).

Pencatatan log

Pustaka boto3 terintegrasi dengan modul logging bawaan Python untuk melacak apa yang terjadi selama sesi. Untuk mengontrol level logging, Anda dapat mengonfigurasi modul logging:

```
import logging

logging.basicConfig(level=logging.INFO)
```

Ini mengonfigurasi root logger untuk mencatat `INFO` dan pesan tingkat di atas. Pesan logging yang kurang parah dari level akan diabaikan. Tingkat logging termasuk`DEBUG`,`INFO`,`WARNING`,`ERROR`, dan`CRITICAL`. Nilai default-nya `WARNING`.

Logger di boto3 bersifat hierarkis. Pustaka menggunakan beberapa logger yang berbeda, masing-masing sesuai dengan bagian perpustakaan yang berbeda. Anda dapat mengontrol perilaku masing-masing secara terpisah:

- boto3: Logger utama untuk modul boto3.
- botocore: Logger utama untuk paket botocore.
- botocore.auth: Digunakan untuk mencatat pembuatan tanda tangan untuk permintaan. AWS
- botocore.credentials: Digunakan untuk mencatat proses pengambilan dan penyegaran kredenal.
- botocore.endpoint: Digunakan untuk pembuatan permintaan logging sebelum dikirim melalui jaringan.
- botocore.hooks: Digunakan untuk peristiwa logging yang dipicu di perpustakaan.
- botocore.loaders: Digunakan untuk logging ketika bagian-bagian dari model layanan dimuat. AWS
- botocore.parsers: Digunakan untuk mencatat respons AWS layanan sebelum diurai.
- botocore.retryhandler: Digunakan untuk mencatat pemrosesan percobaan ulang permintaan layanan (mode lama). AWS
- botocore.retries.standard: Digunakan untuk mencatat pemrosesan percobaan ulang permintaan AWS layanan (mode standar atau adaptif).
- botocore.utils: Digunakan untuk mencatat aktivitas lain-lain di perpustakaan.
- botocore.waiter: Digunakan untuk mencatat fungsionalitas pelayan, yang melakukan polling layanan sampai keadaan tertentu AWS tercapai.

Perpustakaan lain juga mencatat. Secara internal, boto3 menggunakan urllib3 pihak ketiga untuk penanganan koneksi HTTP. Ketika latensi penting, Anda dapat menonton lognya untuk memastikan kolam Anda dimanfaatkan dengan baik dengan melihat kapan urllib3 membuat koneksi baru atau menutup yang tidak aktif.

- urllib3.connectionpool: Digunakan untuk mencatat peristiwa penanganan kumpulan koneksi.

Cuplikan kode berikut menyetel sebagian besar logging `INFO` dengan `DEBUG` logging untuk aktivitas titik akhir dan kumpulan koneksi:

```
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)
```

Kait acara

Botocore memancarkan peristiwa selama berbagai bagian pelaksanaannya. Anda dapat mendaftarkan penangan untuk acara ini sehingga setiap kali suatu peristiwa dipancarkan, handler Anda akan dipanggil. Ini memungkinkan Anda memperluas perilaku botocore tanpa harus memodifikasi internal.

Misalnya, katakanlah Anda ingin melacak setiap kali `PutItem` operasi dipanggil pada setiap tabel DynamoDB dalam aplikasi Anda. Anda dapat mendaftar pada `'provide-client-params.dynamodb.PutItem'` acara untuk menangkap dan mencatat setiap kali `PutItem` operasi dipanggil pada Sesi terkait. Inilah contohnya:

```
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 ...
    }
```

```
)
```

Di dalam handler, Anda bahkan dapat memanipulasi params secara terprogram untuk mengubah perilaku:

```
params['TableName'] = "NewTableName"
```

Untuk informasi lebih lanjut tentang acara, lihat dokumentasi botocore tentang acara dan dokumentasi boto3 tentang acara.

Pagination dan Paginator

Beberapa permintaan, seperti Query dan Scan, membatasi ukuran data yang dikembalikan pada satu permintaan dan mengharuskan Anda untuk membuat permintaan berulang untuk menarik halaman berikutnya.

Anda dapat mengontrol jumlah maksimum item yang akan dibaca untuk setiap halaman dengan `limit` parameter. Misalnya, jika Anda menginginkan 10 item terakhir, Anda dapat menggunakan `limit` untuk mengambil hanya 10 item terakhir. Perhatikan batasnya adalah berapa banyak yang harus dibaca dari tabel sebelum penyaringan apa pun diterapkan. Tidak ada cara untuk menentukan yang Anda inginkan tepat 10 setelah pemfilteran; Anda hanya dapat mengontrol jumlah yang telah difilter sebelumnya dan memeriksa sisi klien ketika Anda benar-benar mengambil 10. Terlepas dari batasnya, setiap respons selalu memiliki ukuran maksimum 1 MB.

Jika respons menyertakan a`LastEvaluatedKey`, ini menunjukkan respons berakhir karena mencapai batas hitungan atau ukuran. Kuncinya adalah kunci terakhir yang dievaluasi untuk respons. Anda dapat mengambil ini `LastEvaluatedKey` dan meneruskannya ke panggilan tindak lanjut `ExclusiveStartKey` untuk membaca potongan berikutnya dari titik awal itu. Ketika tidak ada yang `LastEvaluatedKey` dikembalikan, berarti tidak ada lagi item yang cocok dengan Kueri atau Pindai.

Berikut adalah contoh sederhana (menggunakan antarmuka Sumber Daya, tetapi antarmuka Klien memiliki pola yang sama) yang membaca paling banyak 100 item per halaman dan loop hingga semua item telah dibaca.

```
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']
```

Untuk kenyamanan, boto3 dapat melakukan ini untuk Anda dengan Paginator. Namun, ini hanya berfungsi dengan antarmuka Klien. Berikut kode yang ditulis ulang untuk menggunakan Paginator:

```
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)
```

Untuk informasi selengkapnya, lihat Panduan tentang Paginator dan referensi API untuk DynamoDb.paginator.Query.

> **Note**
>
> Paginator juga memiliki pengaturan konfigurasi mereka sendiri bernama`MaxItems`,`StartingToken`, dan. `PageSize` Untuk paginating dengan DynamoDB, Anda harus mengabaikan pengaturan ini.

Pelayan

Pelayan memberikan kemampuan untuk menunggu sesuatu selesai sebelum melanjutkan. Saat ini, mereka hanya mendukung menunggu tabel dibuat atau dihapus. Di latar belakang, operasi pelayan melakukan pemeriksaan untuk Anda setiap 20 detik hingga 25 kali. Anda bisa melakukannya sendiri, tetapi menggunakan pelayan itu elegan saat menulis otomatisasi.

Kode ini menunjukkan cara menunggu tabel tertentu dibuat:

```
# 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']
```

Untuk informasi lebih lanjut, lihat Panduan untuk Pelayan dan Referensi tentang Pelayan.

Pemrograman Amazon DynamoDB dengan JavaScript

Panduan ini memberikan orientasi kepada programmer yang ingin menggunakan Amazon DynamoDB dengan. JavaScript Pelajari tentang AWS SDK untuk JavaScript, lapisan abstraksi yang tersedia, mengonfigurasi koneksi, menangani kesalahan, menentukan kebijakan coba lagi, mengelola keep-alive, dan banyak lagi.

Topik

- Tentang AWS SDK untuk JavaScript
- Menggunakan AWS SDK untuk JavaScript V3

- Mengakses dokumentasi JavaScript
- Lapisan abstraksi
- Menggunakan fungsi utilitas marshall
- Membaca item
- Penulisan bersyarat
- Paginasi
- Menentukan konfigurasi
- Pelayan
- Penanganan kesalahan
- Pencatatan log
- Pertimbangan

Tentang AWS SDK untuk JavaScript

AWS SDK untuk JavaScript Ini menyediakan akses untuk Layanan AWS menggunakan skrip browser atau Node.js. Dokumentasi ini berfokus pada versi terbaru SDK (V3). AWS SDK untuk JavaScript V3 dikelola oleh AWS sebagai proyek sumber terbuka yang dihosting di. GitHub Masalah dan permintaan fitur bersifat publik dan Anda dapat mengaksesnya di halaman masalah untuk GitHub repositori.

JavaScript V2 mirip dengan V3, tetapi berisi perbedaan sintaks. V3 lebih modular, membuatnya lebih mudah untuk mengirimkan dependensi yang lebih kecil, dan memiliki dukungan kelas satu. TypeScript Sebaiknya gunakan SDK versi terbaru.

Menggunakan AWS SDK untuk JavaScript V3

Anda dapat menambahkan SDK ke aplikasi Node.js Anda menggunakan Node Package Manager. Contoh di bawah ini menunjukkan cara menambahkan paket SDK paling umum untuk bekerja dengan DynamoDB.

- `npm install @aws-sdk/client-dynamodb`
- `npm install @aws-sdk/lib-dynamodb`
- `npm install @aws-sdk/util-dynamodb`

Menginstal paket menambahkan referensi ke bagian ketergantungan file proyek package.json Anda. Anda memiliki opsi untuk menggunakan sintaks ECMAScript modul yang lebih baru. Untuk detail lebih lanjut tentang dua pendekatan ini, lihat bagian Pertimbangan.

Mengakses dokumentasi JavaScript

Mulailah dengan JavaScript dokumentasi dengan sumber daya berikut:

- Akses panduan Pengembang untuk JavaScript dokumentasi inti. Petunjuk instalasi terletak di bagian Pengaturan.
- Akses dokumentasi referensi API untuk menjelajahi semua kelas dan metode yang tersedia.
- SDK untuk JavaScript mendukung banyak Layanan AWS selain DynamoDB. Gunakan prosedur berikut untuk menemukan cakupan API tertentu untuk DynamoDB:

  1. Dari Services, pilih DynamoDB dan Libraries. Ini mendokumentasikan klien tingkat rendah.
  2. Pilih lib-dynamodb. Ini mendokumentasikan klien tingkat tinggi. Kedua klien mewakili dua lapisan abstraksi berbeda yang Anda memiliki pilihan untuk digunakan. Lihat bagian di bawah ini untuk informasi lebih lanjut tentang lapisan abstraksi.

Lapisan abstraksi

SDK untuk JavaScript V3 memiliki klien tingkat rendah (`DynamoDBClient`) dan klien tingkat tinggi (). `DynamoDBDocumentClient`

Topik

- Klien tingkat rendah () DynamoDBClient
- Klien tingkat tinggi () DynamoDBDocumentClient

Klien tingkat rendah () **`DynamoDBClient`**

Klien tingkat rendah tidak memberikan abstraksi tambahan atas protokol kawat yang mendasarinya. Ini memberi Anda kontrol penuh atas semua aspek komunikasi, tetapi karena tidak ada abstraksi, Anda harus melakukan hal-hal seperti memberikan definisi item menggunakan format DynamoDB JSON.

Seperti contoh di bawah ini menunjukkan, dengan format ini tipe data harus dinyatakan secara eksplisit. S menunjukkan nilai string dan N menunjukkan nilai angka. Angka pada kawat selalu

dikirim sebagai string yang ditandai sebagai jenis angka untuk memastikan tidak ada kehilangan presisi. Panggilan API tingkat rendah memiliki pola penamaan seperti `PutItemCommand` dan`GetItemCommand`.

Contoh berikut adalah menggunakan klien tingkat rendah dengan `Item` didefinisikan menggunakan DynamoDB JSON:

```
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();
```

Klien tingkat tinggi () **`DynamoDBDocumentClient`**

Klien dokumen DynamoDB tingkat tinggi menawarkan fitur kenyamanan bawaan, seperti menghilangkan kebutuhan untuk mengumpulkan data secara manual dan memungkinkan untuk membaca dan menulis langsung menggunakan objek standar. JavaScript Dokumentasi untuk `lib-dynamodb` menyediakan daftar keuntungan.

Untuk membuat instance`DynamoDBDocumentClient`, buat level rendah `DynamoDBClient` dan kemudian bungkus dengan a. `DynamoDBDocumentClient` Konvensi penamaan fungsi sedikit

berbeda antara kedua paket. Misalnya, penggunaan tingkat rendah `PutItemCommand` sementara penggunaan tingkat tinggi. `PutCommand` Nama yang berbeda memungkinkan kedua set fungsi untuk hidup berdampingan dalam konteks yang sama, memungkinkan Anda untuk mencampur keduanya dalam skrip yang sama.

```
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();
```

Pola penggunaan konsisten saat Anda membaca item menggunakan operasi API seperti`GetItem`,`Query`, atau`Scan`.

Menggunakan fungsi utilitas marshall

Anda dapat menggunakan klien tingkat rendah dan marshall atau unmarshall tipe data Anda sendiri. Paket utilitas, util-dynamodb, memiliki fungsi `marshall()` utilitas yang menerima JSON dan menghasilkan DynamoDB JSON, serta fungsi, yang melakukan sebaliknya. `unmarshall()` Contoh

berikut menggunakan klien tingkat rendah dengan data marshalling ditangani oleh panggilan. `marshall()`

```
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();
```

Membaca item

Untuk membaca satu item dari DynamoDB, Anda menggunakan `GetItem` operasi API. Mirip dengan `PutItem` perintah, Anda memiliki pilihan untuk menggunakan klien tingkat rendah atau klien Dokumen tingkat tinggi. Contoh di bawah ini menunjukkan menggunakan klien Dokumen tingkat tinggi untuk mengambil item.

```
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();
```

Gunakan operasi `Query` API untuk membaca beberapa item. Anda dapat menggunakan klien tingkat rendah atau klien Dokumen. Contoh di bawah ini menggunakan klien Dokumen tingkat tinggi.

```
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();
```

Penulisan bersyarat

Operasi penulisan DynamoDB dapat menentukan ekspresi kondisi logis yang harus mengevaluasi ke true agar penulisan dapat dilanjutkan. Jika kondisi tidak mengevaluasi ke true, operasi tulis menghasilkan pengecualian. Ekspresi kondisi dapat memeriksa apakah item sudah ada atau apakah atributnya cocok dengan batasan tertentu.

`ConditionExpression = "version = :ver AND size(VideoClip) < :maxsize"`

Ketika ekspresi kondisional gagal, Anda dapat menggunakan `ReturnValuesOnConditionCheckFailure` untuk meminta agar respons kesalahan menyertakan item yang tidak memenuhi kondisi untuk menyimpulkan apa masalahnya. Untuk detail selengkapnya, lihat Menangani kesalahan penulisan bersyarat dalam skenario konkurensi tinggi dengan 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;
      }
```

```
  }
```

Contoh kode tambahan yang menunjukkan aspek lain dari penggunaan JavsScript SDK V3 tersedia di Dokumentasi JavaScript SDK V3 dan di bawah repositori. DynamoDB-SDK-Examples GitHub

Paginasi

Topik

- Menggunakan metode paginateScan kenyamanan

Permintaan baca seperti `Scan` atau kemungkinan `Query` akan mengembalikan beberapa item dalam kumpulan data. Jika Anda melakukan `Scan` atau `Query` dengan `Limit` parameter, maka setelah sistem membaca banyak item, sebagian respons akan dikirim, dan Anda harus membuat paginasi untuk mengambil item tambahan.

Sistem hanya akan membaca maksimal 1 megabyte data per permintaan. Jika Anda menyertakan `Filter` ekspresi, sistem akan tetap membaca megabyte, maksimum, data dari disk, tetapi akan mengembalikan item megabyte yang cocok dengan filter. Operasi filter dapat mengembalikan 0 item untuk halaman, tetapi masih memerlukan pagination lebih lanjut sebelum pencarian habis.

Anda harus mencari `LastEvaluatedKey` dalam respons dan menggunakannya sebagai `ExclusiveStartKey` parameter dalam permintaan berikutnya untuk melanjutkan pengambilan data. Ini berfungsi sebagai bookmark seperti yang tercantum dalam contoh berikut.

> **Note**
>
> Sampel melewati nol `lastEvaluatedKey` sebagai `ExclusiveStartKey` pada iterasi pertama dan ini diperbolehkan.

Contoh menggunakan`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);
});
```

Menggunakan metode **`paginateScan`** kenyamanan

SDK menyediakan metode kenyamanan yang dipanggil `paginateScan` dan `paginateQuery` yang melakukan ini bekerja untuk Anda dan membuat permintaan berulang di belakang layar. Tentukan jumlah maksimum item yang akan dibaca per permintaan menggunakan `Limit` parameter standar.

```
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**
>
> Melakukan pemindaian tabel penuh secara teratur bukanlah pola akses yang disarankan kecuali tabelnya kecil.

Menentukan konfigurasi

Topik

- Config untuk batas waktu
- Config untuk keep-alive
- Config untuk percobaan ulang

Saat mengaturDynamoDBClient, Anda dapat menentukan berbagai penggantian konfigurasi dengan meneruskan objek konfigurasi ke konstruktor. Misalnya, Anda dapat menentukan Wilayah yang akan disambungkan jika belum diketahui konteks panggilan atau URL titik akhir yang akan digunakan. Ini berguna jika Anda ingin menargetkan instance DynamoDB Local untuk tujuan pengembangan.

```
const client = new DynamoDBClient({
  region: "eu-west-1",
  endpoint: "http://localhost:8000",
});
```

Config untuk batas waktu

DynamoDB menggunakan HTTPS untuk komunikasi client-server. Anda dapat mengontrol beberapa aspek dari lapisan HTTP dengan menyediakan NodeHttpHandler objek. Misalnya, Anda dapat menyesuaikan nilai batas waktu kunci connectionTimeout danrequestTimeout. connectionTimeoutIni adalah durasi maksimum, dalam milidetik, bahwa klien akan menunggu sambil mencoba membuat koneksi sebelum menyerah.

Ini `requestTimeout` menentukan berapa lama klien akan menunggu respons setelah permintaan dikirim, juga dalam milidetik. Default untuk keduanya adalah nol, artinya batas waktu dinonaktifkan dan tidak ada batasan berapa lama klien akan menunggu jika respons tidak tiba. Anda harus mengatur batas waktu ke sesuatu yang masuk akal sehingga jika terjadi masalah jaringan, permintaan akan error dan permintaan baru dapat dimulai. Misalnya:

```
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**
>
> Contoh yang diberikan menggunakan impor Smithy. Smithy adalah bahasa untuk mendefinisikan layanan dan SDKs, open-source dan dikelola oleh. AWS

Selain mengonfigurasi nilai batas waktu, Anda dapat mengatur jumlah soket maksimum, yang memungkinkan peningkatan jumlah koneksi bersamaan per asal. Panduan pengembang mencakup detail tentang mengonfigurasi `maxSockets` parameter.

Config untuk keep-alive

Saat menggunakan HTTPS, permintaan pertama selalu membutuhkan back-and-forth komunikasi untuk membuat koneksi yang aman. HTTP Keep-Alive memungkinkan permintaan berikutnya untuk menggunakan kembali koneksi yang sudah dibuat, membuat permintaan lebih efisien dan menurunkan latensi. HTTP Keep-Alive diaktifkan secara default dengan V3. JavaScript

Ada batasan berapa lama koneksi idle dapat tetap hidup. Pertimbangkan untuk mengirim permintaan berkala, mungkin setiap menit, jika Anda memiliki koneksi idle tetapi ingin permintaan berikutnya menggunakan koneksi yang sudah dibuat.

> **Note**
>
> Perhatikan bahwa di V2 SDK yang lebih lama, keep-alive dimatikan secara default, yang berarti setiap koneksi akan segera ditutup setelah digunakan. Jika menggunakan V2, Anda dapat mengganti pengaturan ini.

Config untuk percobaan ulang

Ketika SDK menerima respons kesalahan dan kesalahan dapat dilanjutkan seperti yang ditentukan oleh SDK, seperti pengecualian pelambatan atau pengecualian layanan sementara, SDK akan mencoba lagi. Ini terjadi tanpa terlihat bagi Anda sebagai penelepon, kecuali bahwa Anda mungkin memperhatikan permintaan membutuhkan waktu lebih lama untuk berhasil.

SDK untuk JavaScript V3 akan membuat 3 permintaan total, secara default, sebelum menyerah dan meneruskan kesalahan ke dalam konteks panggilan. Anda dapat menyesuaikan jumlah dan frekuensi percobaan ulang ini.

`DynamoDBClient`Konstruktor menerima `maxAttempts` pengaturan yang membatasi berapa banyak upaya yang akan terjadi. Contoh di bawah ini menaikkan nilai dari default 3 menjadi total 5. Jika Anda menyetelnya ke 0 atau 1, itu menunjukkan Anda tidak ingin mencoba ulang otomatis dan ingin menangani kesalahan yang dapat dilanjutkan sendiri dalam blok catch Anda.

```
const client = new DynamoDBClient({
  maxAttempts: 5,
});
```

Anda juga dapat mengontrol waktu percobaan ulang dengan strategi coba lagi khusus. Untuk melakukan ini, impor paket `util-retry` utilitas dan buat fungsi backoff khusus yang menghitung waktu tunggu antara percobaan ulang berdasarkan jumlah coba lagi saat ini.

Contoh di bawah ini mengatakan untuk melakukan maksimal 5 upaya dengan penundaan 15, 30, 90, dan 360 milidetik jika upaya pertama gagal. Fungsi backoff kustom, `calculateRetryBackoff`, menghitung penundaan dengan menerima nomor percobaan ulang (dimulai dengan 1 untuk percobaan ulang pertama) dan mengembalikan berapa milidetik untuk menunggu permintaan itu.

```
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.
  ),
});
```

Pelayan

Klien DynamoDB mencakup dua fungsi pelayan berguna yang dapat digunakan saat membuat, memodifikasi, atau menghapus tabel saat Anda ingin kode Anda menunggu untuk melanjutkan hingga modifikasi tabel selesai. Misalnya, Anda dapat menyebarkan tabel, memanggil `waitUntilTableExists` fungsi, dan kode akan memblokir sampai tabel telah dibuat AKTIF. Pelayan secara internal melakukan polling layanan DynamoDB dengan setiap 20 detik. `describe-table`

```
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}`);
```

`waitUntilTableExists`Fitur mengembalikan kontrol hanya ketika dapat melakukan `describe-table` perintah yang menunjukkan status tabel ACTIVE. Ini memastikan bahwa Anda dapat menggunakan `waitUntilTableExists` untuk menunggu penyelesaian pembuatan, serta modifikasi seperti menambahkan indeks GSI, yang mungkin memerlukan beberapa waktu untuk diterapkan sebelum tabel kembali ke status AKTIF.

Penanganan kesalahan

Dalam contoh awal di sini, kami telah menangkap semua kesalahan secara luas. Namun, dalam aplikasi praktis, penting untuk membedakan antara berbagai jenis kesalahan dan menerapkan penanganan kesalahan yang lebih tepat.

Respons kesalahan DynamoDB berisi metadata, termasuk nama kesalahan. Anda dapat menangkap kesalahan kemudian mencocokkan dengan kemungkinan nama string dari kondisi kesalahan untuk menentukan cara melanjutkan. Untuk kesalahan sisi server, Anda dapat memanfaatkan `instanceof` operator dengan jenis kesalahan yang diekspor oleh `@aws-sdk/client-dynamodb` paket untuk mengelola penanganan kesalahan secara efisien.

Penting untuk dicatat bahwa kesalahan ini hanya terwujud setelah semua percobaan ulang telah habis. Jika kesalahan dicoba lagi dan akhirnya diikuti oleh panggilan yang berhasil, dari perspektif kode, tidak ada kesalahan hanya latensi yang sedikit meningkat. Percobaan ulang akan muncul di CloudWatch bagan Amazon sebagai permintaan yang tidak berhasil, seperti permintaan throttle atau error. Jika klien mencapai jumlah percobaan ulang maksimum, itu akan menyerah dan menghasilkan pengecualian. Ini adalah cara klien mengatakan itu tidak akan mencoba lagi.

Di bawah ini adalah cuplikan untuk menangkap kesalahan dan mengambil tindakan berdasarkan jenis kesalahan yang dikembalikan.

```
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.
      }
    }
}
```

Lihat the section called “Penanganan kesalahan” string kesalahan umum di DynamoDB Developer Guide. Kesalahan persis yang mungkin terjadi dengan panggilan API tertentu dapat ditemukan dalam dokumentasi untuk panggilan API tersebut, seperti dokumen Query API.

Metadata kesalahan mencakup properti tambahan, tergantung pada kesalahannya. Untuk a `TimeoutError`, metadata mencakup jumlah upaya yang dilakukan dan`totalRetryDelay`, seperti yang ditunjukkan di bawah ini.

```
{
  "name": "TimeoutError",
  "$metadata": {
    "attempts": 3,
    "totalRetryDelay": 199
  }
}
```

Jika Anda mengelola kebijakan coba ulang Anda sendiri, Anda akan ingin membedakan antara throttle dan error:

- Throttle (ditunjukkan oleh `ProvisionedThroughputExceededException` atau`ThrottlingException`) menunjukkan layanan sehat yang memberi tahu Anda bahwa Anda telah melebihi kapasitas baca atau tulis pada tabel atau partisi DynamoDB. Setiap milidetik yang berlalu, sedikit lebih banyak kapasitas baca atau tulis tersedia, sehingga Anda dapat mencoba lagi dengan cepat, seperti setiap 50 ms, untuk mencoba mengakses kapasitas yang baru dirilis.

  Dengan throttle Anda tidak terlalu membutuhkan backoff eksponensial karena throttle ringan untuk DynamoDB untuk kembali dan tidak dikenakan biaya per permintaan kepada Anda. Backoff eksponensial memberikan penundaan yang lebih lama ke utas klien yang telah menunggu paling lama, yang secara statistik memperluas p50 dan p99 ke luar.
- Kesalahan (ditunjukkan oleh `InternalServerError` atau a`ServiceUnavailable`, antara lain) menunjukkan masalah sementara dengan layanan, mungkin seluruh tabel atau hanya partisi yang Anda baca atau tulis. Dengan kesalahan, Anda dapat berhenti lebih lama sebelum mencoba

ulang, seperti 250ms atau 500ms, dan menggunakan jitter untuk membuat percobaan ulang terhuyung-huyung.

Pencatatan log

Aktifkan logging untuk mendapatkan detail lebih lanjut tentang apa yang dilakukan SDK. Anda dapat mengatur parameter pada `DynamoDBClient` seperti yang ditunjukkan pada contoh di bawah ini. Informasi log lainnya akan muncul di konsol dan menyertakan metadata seperti kode status dan kapasitas yang dikonsumsi. Jika Anda menjalankan kode secara lokal di jendela terminal, log akan muncul di sana. Jika Anda menjalankan kode di AWS Lambda, dan Anda memiliki CloudWatch log Amazon yang disiapkan, maka output konsol akan ditulis di sana.

```
const client = new DynamoDBClient({
  logger: console
});
```

Anda juga dapat menghubungkan ke aktivitas SDK internal dan melakukan pencatatan khusus saat peristiwa tertentu terjadi. Contoh di bawah ini menggunakan klien `middlewareStack` untuk mencegat setiap permintaan saat dikirim dari SDK dan mencatatnya saat terjadi.

```
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",
  }
);
```

`MiddlewareStack`Ini menyediakan pengait yang kuat untuk mengamati dan mengendalikan perilaku SDK. Lihat blog Memperkenalkan Middleware Stack in Modular AWS SDK untuk JavaScript, untuk informasi lebih lanjut.

Pertimbangan

Saat menerapkan AWS SDK untuk JavaScript dalam proyek Anda, berikut adalah beberapa faktor lebih lanjut yang perlu dipertimbangkan.

Sistem modul

SDK mendukung dua sistem modul, CommonJS dan ES ()ECMAScript. CommonJS menggunakan `require` fungsi, sedangkan ES menggunakan `import` kata kunci.

1. JS umum - `const { DynamoDBClient, PutItemCommand } = require("@aws-sdk/client-dynamodb");`
2. ES (ECMAScript— `import { DynamoDBClient, PutItemCommand } from "@aws-sdk/client-dynamodb";`

Jenis proyek menentukan sistem modul yang akan digunakan dan ditentukan di bagian tipe file package.json Anda. Defaultnya adalah CommonJS. Gunakan `"type": "module"` untuk menunjukkan proyek ES. Jika Anda memiliki proyek Node.JS yang sudah ada yang menggunakan format paket CommonJS, Anda masih dapat menambahkan fungsi dengan sintaks Impor SDK V3 yang lebih modern dengan menamai file fungsi Anda dengan ekstensi.mjs. Ini akan memungkinkan file kode diperlakukan sebagai ES (ECMAScript).

Operasi asinkron

Anda akan melihat banyak contoh kode menggunakan callback dan janji untuk menangani hasil operasi DynamoDB. Dengan modern, kompleksitas JavaScript ini tidak lagi diperlukan dan pengembang dapat memanfaatkan sintaks async/await yang lebih ringkas dan mudah dibaca untuk operasi asinkron.

Waktu proses peramban web

Pengembang web dan seluler yang membangun dengan React atau React Native dapat menggunakan SDK untuk JavaScript proyek mereka. Dengan V2 SDK sebelumnya, pengembang web harus memuat SDK lengkap ke browser, merujuk gambar SDK yang dihosting di. https://sdk.amazonaws.com/js/

Dengan V3, dimungkinkan untuk menggabungkan hanya modul klien V3 yang diperlukan dan semua JavaScript fungsi yang diperlukan ke dalam satu JavaScript file menggunakan Webpack, dan menambahkannya dalam tag skrip di halaman HTML Anda, seperti yang dijelaskan di bagian Memulai di skrip browser dari dokumentasi SDK. `<head>`

Operasi pesawat data DAX

Operasi bidang data Amazon DynamoDB Streams Accelerator (DAX) didukung oleh SDK untuk V3. JavaScript

Pemrograman DynamoDB dengan AWS SDK for Java 2.x

Panduan pemrograman ini memberikan orientasi bagi programmer yang ingin menggunakan Amazon DynamoDB dengan Java. Panduan ini mencakup konsep yang berbeda termasuk lapisan abstraksi, manajemen konfigurasi, penanganan kesalahan, pengendalian kebijakan coba lagi, dan pengelolaan keep-alive.

Topik

- Tentang AWS SDK for Java 2.x
- Memulai dengan AWS SDK for Java 2.x
- Meninjau dokumentasi AWS SDK for Java 2.x
- Antarmuka yang didukung
- Contoh kode tambahan
- Pemrograman sinkron dan asinkron
- Klien HTTP
- Mengkonfigurasi klien HTTP
- Penanganan kesalahan
- AWS ID permintaan
- Pencatatan log
- Paginasi
- Anotasi kelas data

Tentang AWS SDK for Java 2.x

Anda dapat mengakses DynamoDB dari Java menggunakan resmi. AWS SDK untuk Java SDK for Java memiliki dua versi: 1.x dan 2.x. end-of-supportUntuk 1.x diumumkan pada 12 Januari 2024. Ini akan memasuki mode pemeliharaan pada 31 Juli 2024 dan akan jatuh tempo pada 31 Desember 2025. end-of-support Untuk pengembangan baru, kami sangat menyarankan Anda menggunakan

2.x, yang pertama kali dirilis pada tahun 2018. Panduan ini secara eksklusif menargetkan 2.x dan hanya berfokus pada bagian SDK yang relevan dengan DynamoDB.

Untuk informasi tentang pemeliharaan dan dukungan AWS SDKs, lihat Kebijakan pemeliharaan AWS SDK dan Tools serta matriks dukungan versi Tools di Panduan Referensi Alat AWS SDKs dan.AWS SDKs

AWS SDK for Java 2.x Ini adalah penulisan ulang utama dari basis kode 1.x. SDK for Java 2.x mendukung fitur Java modern, seperti I/O non-blocking yang diperkenalkan di Java 8. SDK for Java 2.x juga menambahkan dukungan untuk implementasi klien HTTP pluggable untuk memberikan lebih banyak fleksibilitas koneksi jaringan dan opsi konfigurasi.

Perubahan nyata antara SDK for Java 1.x dan SDK for Java 2.x adalah penggunaan nama paket baru. Java 1.x SDK menggunakan nama `com.amazonaws` paket, sedangkan Java 2.x SDK menggunakan. `software.amazon.awssdk` Demikian pula, artefak Maven untuk Java 1.x SDK menggunakan `com.amazonawsgroupId`, sedangkan artefak Java 2.x SDK menggunakan file. `software.amazon.awssdk` `groupId`

> **Important**
>
> AWS SDK untuk Java 1.x memiliki paket DynamoDB bernama. `com.amazonaws.dynamodbv2` “v2" dalam nama paket tidak menunjukkan bahwa itu untuk Java 2 (J2SE). Sebaliknya, “v2" menunjukkan bahwa paket mendukung versi kedua API tingkat rendah DynamoDB alih-alih versi asli API tingkat rendah.

Support untuk versi Java

AWS SDK for Java 2.x Ini memberikan dukungan penuh untuk rilis Java dukungan jangka panjang (LTS).

Memulai dengan AWS SDK for Java 2.x

Tutorial berikut menunjukkan cara menggunakan Apache Maven untuk mendefinisikan dependensi untuk SDK for Java 2.x. Tutorial ini juga menunjukkan cara menulis kode yang menghubungkan ke DynamoDB untuk daftar tabel DynamoDB yang tersedia. Tutorial dalam panduan ini didasarkan pada tutorial Memulai dengan AWS SDK for Java 2.x di Panduan AWS SDK for Java 2.x Pengembang. Kami telah mengedit tutorial ini untuk melakukan panggilan ke DynamoDB alih-alih Amazon S3.

Untuk menyelesaikan tutorial ini, lakukan hal berikut:

- Langkah 1: Siapkan tutorial ini
- Langkah 2: Buat proyek
- Langkah 3: Tulis kodenya
- Langkah 4: Bangun dan jalankan aplikasi

Langkah 1: Siapkan tutorial ini

Sebelum Anda memulai tutorial ini, Anda memerlukan yang berikut:

- Izin untuk mengakses DynamoDB.
- Lingkungan pengembangan Java yang dikonfigurasi dengan akses masuk tunggal untuk Layanan AWS menggunakan file. Portal akses AWS

Untuk mengatur tutorial ini, ikuti petunjuk di Ikhtisar pengaturan di Panduan AWS SDK for Java 2.x Pengembang. Setelah Anda mengkonfigurasi lingkungan pengembangan Anda dengan akses masuk tunggal untuk Java SDK dan Anda memiliki sesi portal AWS akses aktif, kemudian lanjutkan ke Langkah 2 dari tutorial ini.

Langkah 2: Buat proyek

Untuk membuat proyek untuk tutorial ini, Anda menjalankan perintah Maven yang meminta Anda untuk masukan tentang cara mengkonfigurasi proyek. Setelah semua input dimasukkan dan dikonfirmasi, Maven selesai membangun proyek dengan membuat `pom.xml` file dan membuat file Java rintisan.

1. Buka terminal atau jendela prompt perintah dan arahkan ke direktori pilihan Anda, misalnya, `Home` folder Anda `Desktop` atau.
2. Masukkan perintah berikut di terminal, lalu tekan Enter.

```
mvn archetype:generate \
   -DarchetypeGroupId=software.amazon.awssdk \
   -DarchetypeArtifactId=archetype-app-quickstart \
   -DarchetypeVersion=2.22.0
```

3. Untuk setiap prompt, masukkan nilai yang tercantum di kolom kedua.

Prompt	Nilai untuk masuk
`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. Setelah Anda memasukkan nilai terakhir, Maven mencantumkan pilihan yang Anda buat. Untuk mengonfirmasi, masukkan Y. Atau, masukkan N, lalu masukkan kembali pilihan Anda.

Maven membuat folder proyek bernama `getstarted` berdasarkan `artifactId` nilai yang Anda masukkan. Di dalam `getstarted` folder, temukan file bernama `README.md` yang dapat Anda tinjau, `pom.xml` file, dan `src` direktori.

Maven membangun pohon direktori berikut.

```
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
```

Berikut ini menunjukkan isi dari file `pom.xml` proyek.

`pom.xml`

`dependencyManagement`Bagian ini berisi ketergantungan ke AWS SDK for Java 2.x, dan `dependencies` bagian memiliki ketergantungan untuk DynamoDB. Menentukan dependensi ini memaksa Maven untuk menyertakan `.jar` file yang relevan di jalur kelas Java Anda. Secara default, AWS SDK tidak menyertakan semua kelas untuk semua Layanan AWS. Untuk DynamoDB, jika Anda menggunakan antarmuka tingkat rendah, maka Anda harus memiliki ketergantungan pada artefak. `dynamodb` Atau, jika Anda menggunakan antarmuka tingkat tinggi, pada `dynamodb-enhanced` artefak. Jika Anda tidak menyertakan dependensi yang relevan, maka kode Anda tidak dapat dikompilasi. Proyek ini menggunakan Java 1.8 karena `1.8` nilai dalam `maven.compiler.source` dan `maven.compiler.target` properti.

```
<?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>
```

Langkah 3: Tulis kodenya

Kode berikut menunjukkan App kelas yang dibuat Maven. `main`Metode ini adalah titik masuk ke dalam aplikasi, yang menciptakan sebuah instance dari `Handler` kelas dan kemudian memanggil `sendRequest` metodenya.

`App` kelas

```
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");
     }
 }
```

`DependencyFactory`Kelas yang dibuat Maven berisi metode `dynamoDbClient` pabrik yang membangun dan mengembalikan sebuah instance. `DynamoDbClient` `DynamoDbClient`Instance menggunakan instance dari klien HTTP berbasis Apache. Ini karena Anda menentukan `apache-client` kapan Maven meminta Anda untuk klien HTTP mana yang akan digunakan.

Kode berikut menunjukkan `DependencyFactory` kelas.

DependencyFactory kelas

```
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();
     }
 }
```

`Handler`Kelas berisi logika utama program Anda. Ketika sebuah instance `Handler` dibuat di App kelas, `DependencyFactory` melengkapi klien `DynamoDbClient` layanan. Kode Anda menggunakan `DynamoDbClient` instance untuk memanggil DynamoDB.

Maven menghasilkan `Handler` kelas berikut dengan komentar. *`TODO`* Langkah selanjutnya dalam tutorial menggantikan *`TODO`*komentar dengan kode.

`Handler`kelas, yang dihasilkan oleh Maven

```
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.
    }
}
```

Untuk mengisi logika, ganti seluruh isi `Handler` kelas dengan kode berikut. `sendRequest`Metode diisi dan impor yang diperlukan ditambahkan.

`Handler`kelas, diimplementasikan

Kode berikut menggunakan `DynamoDbClient`contoh untuk mengambil daftar tabel yang ada. Jika tabel ada untuk akun tertentu dan Wilayah AWS, maka kode menggunakan `Logger` instance untuk mencatat nama-nama tabel ini.

```
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));
        }
    }
}
```

Langkah 4: Bangun dan jalankan aplikasi

Setelah Anda membuat proyek dan berisi `Handler` kelas lengkap, membangun dan menjalankan aplikasi.

1. Pastikan Anda memiliki AWS IAM Identity Center sesi aktif. Untuk mengonfirmasi, jalankan perintah AWS Command Line Interface (AWS CLI) `aws sts get-caller-identity` dan periksa responsnya. Jika Anda tidak memiliki sesi aktif, lihat Masuk menggunakan AWS CLI petunjuk.
2. Buka terminal atau jendela prompt perintah dan arahkan ke direktori proyek Anda`getstarted`.
3. Untuk membangun proyek Anda, jalankan perintah berikut:

   ```
   mvn clean package
   ```

4. Untuk menjalankan aplikasi, jalankan perintah berikut:

   ```
   mvn exec:java -Dexec.mainClass="org.example.App"
   ```

Setelah Anda melihat file, hapus objek, dan kemudian hapus ember.

Berhasil

Jika proyek Maven Anda dibangun dan berjalan tanpa kesalahan, maka selamat! Anda telah berhasil membangun aplikasi Java pertama Anda menggunakan SDK for Java 2.x.

Pembersihan

Untuk membersihkan sumber daya yang Anda buat selama tutorial ini, hapus folder proyek`getstarted`.

Meninjau dokumentasi AWS SDK for Java 2.x

Panduan AWS SDK for Java 2.x Pengembang mencakup semua aspek SDK di semua Layanan AWS aspek. Kami menyarankan Anda meninjau topik-topik berikut:

- Migrasi dari versi 1.x ke 2.x - Termasuk penjelasan rinci tentang perbedaan antara 1.x dan 2.x. Topik ini juga berisi petunjuk tentang cara menggunakan kedua versi utama side-by-side.
- Panduan DynamoDB untuk Java 2.x SDK - Menunjukkan cara melakukan operasi DynamoDB dasar: membuat tabel, memanipulasi item, dan mengambil item. Contoh-contoh ini menggunakan antarmuka tingkat rendah. Java memiliki beberapa antarmuka, seperti yang dijelaskan di bagian berikut: Antarmuka yang didukung

> **Tip**
>
> Setelah Anda meninjau topik ini, tandai Referensi AWS SDK for Java 2.x API. Ini mencakup semua Layanan AWS, dan kami menyarankan Anda menggunakannya sebagai referensi API utama Anda.

Antarmuka yang didukung

AWS SDK for Java 2.x Mendukung antarmuka berikut, tergantung pada tingkat abstraksi yang Anda inginkan.

Topik di bagian ini

- Antarmuka tingkat rendah
- Antarmuka tingkat tinggi
- Antarmuka dokumen
- Membandingkan antarmuka dengan contoh Query

Antarmuka tingkat rendah

Antarmuka tingkat rendah menyediakan one-to-one pemetaan ke API layanan yang mendasarinya. Setiap DynamoDB API tersedia melalui antarmuka ini. Ini berarti bahwa antarmuka tingkat rendah dapat menyediakan fungsionalitas yang lengkap, tetapi seringkali lebih bertele-tele dan kompleks untuk digunakan. Misalnya, Anda harus menggunakan `.s()` fungsi untuk menahan string dan `.n()` fungsi untuk menahan angka. Contoh berikut PutItemmenyisipkan item menggunakan antarmuka tingkat rendah.

```
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)");
    }
}
```

Antarmuka tingkat tinggi

Antarmuka tingkat tinggi di dalam disebut DynamoDB Enhanced Client. AWS SDK for Java 2.x Antarmuka ini memberikan pengalaman penulisan kode yang lebih idiomatis.

Klien yang disempurnakan menawarkan cara untuk memetakan antara kelas data sisi klien dan tabel DynamoDB yang dirancang untuk menyimpan data tersebut. Anda menentukan hubungan antara tabel dan kelas model yang sesuai dalam kode Anda. Kemudian, Anda dapat mengandalkan SDK untuk mengelola manipulasi tipe data. Untuk informasi selengkapnya tentang klien yang disempurnakan, lihat DynamoDB API klien yang disempurnakan di AWS SDK for Java 2.x Panduan Pengembang.

Contoh berikut PutItemmenggunakan antarmuka tingkat tinggi. Dalam contoh ini, yang `DynamoDbBean` bernama `YourItem` menciptakan a `TableSchema` yang memungkinkan penggunaan langsungnya sebagai input untuk `putItem()` panggilan.

```
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;
        }
    }
}
```

AWS SDK untuk Java 1.x memiliki antarmuka tingkat tinggi sendiri, yang sering disebut oleh kelas utamanya. `DynamoDBMapper` AWS SDK for Java 2.x Ini diterbitkan dalam paket terpisah (dan artefak Maven) bernama. `software.amazon.awssdk.enhanced.dynamodb` Java 2.x SDK sering disebut oleh kelas utamanya. `DynamoDbEnhancedClient`

Antarmuka tingkat tinggi menggunakan kelas data yang tidak dapat diubah

Fitur pemetaan API klien yang disempurnakan DynamoDB juga berfungsi dengan kelas data yang tidak dapat diubah. Kelas yang tidak dapat diubah hanya memiliki getter dan memerlukan kelas pembangun yang digunakan SDK untuk membuat instance kelas. Kekekalan di Jawa adalah gaya yang umum digunakan yang dapat digunakan pengembang untuk membuat kelas yang tidak memiliki efek samping. Kelas-kelas ini lebih dapat diprediksi dalam perilaku mereka dalam aplikasi multi-threaded yang kompleks. Alih-alih menggunakan `@DynamoDbBean` anotasi seperti yang ditunjukkan dalamHigh-level interface example, kelas yang tidak dapat diubah menggunakan `@DynamoDbImmutable` anotasi, yang mengambil kelas pembangun sebagai inputnya.

Contoh berikut mengambil kelas builder `DynamoDbEnhancedClientImmutablePutItem` sebagai masukan untuk membuat skema tabel. Contoh kemudian memberikan skema sebagai input untuk panggilan PutItemAPI.

```
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)");
    }
}
```

Contoh berikut menunjukkan kelas data yang tidak dapat diubah.

```
@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); }
    }
}
```

Antarmuka tingkat tinggi menggunakan kelas data yang tidak dapat diubah dan pustaka generasi boilerplate pihak ketiga

Kelas data yang tidak dapat diubah (ditunjukkan pada contoh sebelumnya) memerlukan beberapa kode boilerplate. Misalnya, logika getter dan setter pada kelas data, selain kelas. `Builder` Perpustakaan pihak ketiga, seperti Project Lombok, dapat membantu Anda menghasilkan jenis kode boilerplate tersebut. Mengurangi sebagian besar kode boilerplate membantu Anda membatasi jumlah kode yang diperlukan untuk bekerja dengan kelas data yang tidak dapat diubah dan SDK. AWS Ini selanjutnya menghasilkan peningkatan produktivitas dan keterbacaan kode Anda. Untuk informasi selengkapnya, lihat Menggunakan pustaka pihak ketiga, seperti Lombok di Panduan AWS SDK for Java 2.x Pengembang.

Contoh berikut menunjukkan bagaimana Project Lombok menyederhanakan kode yang diperlukan untuk menggunakan DynamoDB ditingkatkan client API.

```
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)");
    }
}
```

Contoh berikut menunjukkan objek data yang tidak berubah dari kelas data yang tidak dapat diubah.

```
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;
}
```

`YourImmutableLombokItem`Kelas menggunakan anotasi berikut yang Project Lombok dan AWS SDK menyediakan:

- @Builder — Menghasilkan pembangun kompleks APIs untuk kelas data yang disediakan Project Lombok.
- @ DynamoDbImmutable — Mengidentifikasi `DynamoDbImmutable` kelas sebagai anotasi entitas DynamoDB yang dapat dipetakan yang disediakan SDK. AWS
- @Value — Varian yang tidak dapat diubah dari. `@Data` Secara default, semua bidang dibuat pribadi dan final, dan setter tidak dibuat. Proyek Lombok memberikan anotasi ini.

Antarmuka dokumen

Antarmuka AWS SDK for Java 2.x Dokumen menghindari kebutuhan untuk menentukan deskriptor tipe data. Jenis data yang tersirat oleh semantik data itu sendiri. Antarmuka Dokumen ini mirip dengan antarmuka AWS SDK untuk Java 1.x, Dokumen, tetapi dengan antarmuka yang didesain ulang.

Berikut ini Document interface example menunjukkan `PutItem` panggilan yang dinyatakan menggunakan antarmuka Dokumen. Contoh ini juga menggunakan EnhancedDocument. Untuk menjalankan perintah terhadap tabel DynamoDB menggunakan API dokumen yang disempurnakan, Anda harus terlebih dahulu mengaitkan tabel dengan skema tabel dokumen Anda untuk membuat objek sumber daya. `DynamoDBTable` Pembuat skema tabel Dokumen memerlukan kunci indeks utama dan penyedia konverter atribut.

Anda dapat menggunakan `AttributeConverterProvider.defaultProvider()` untuk mengonversi atribut dokumen dari tipe default. Anda dapat mengubah keseluruhan perilaku default dengan `AttributeConverterProvider` implementasi kustom. Anda juga dapat mengubah konverter untuk satu atribut. Panduan Referensi AWS SDKs and Tools memberikan rincian dan contoh lebih lanjut tentang cara menggunakan konverter khusus. Penggunaan utamanya adalah untuk atribut kelas domain Anda yang tidak memiliki konverter default yang tersedia. Menggunakan

konverter khusus, Anda dapat memberikan SDK dengan informasi yang diperlukan untuk menulis atau membaca ke DynamoDB.

```
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)");
```

```
    }

}
```

Untuk mengonversi dokumen JSON ke dan dari tipe data Amazon DynamoDB asli, Anda dapat menggunakan metode utilitas berikut:

- `EnhancedDocument.fromJson(String json)`— Membuat EnhancedDocument instance baru dari string JSON.
- `EnhancedDocument.toJson()`— Membuat representasi string JSON dari dokumen yang dapat Anda gunakan dalam aplikasi Anda seperti objek JSON lainnya.

Membandingkan antarmuka dengan contoh **`Query`**

Bagian ini menunjukkan `Query`panggilan yang sama yang diekspresikan menggunakan berbagai antarmuka. Untuk menyempurnakan hasil kueri ini, perhatikan hal berikut:

- DynamoDB menargetkan satu nilai kunci partisi tertentu, jadi Anda harus menentukan kunci partisi sepenuhnya.
- Untuk memiliki item keranjang target kueri saja, kunci pengurutan memiliki ekspresi kondisi kunci yang menggunakan`begins_with`.
- Kami gunakan `limit()` untuk membatasi kueri hingga maksimum 100 item yang dikembalikan.
- Kami mengatur `scanIndexForward` ke false. Hasilnya dikembalikan dalam urutan UTF-8 byte, yang biasanya berarti item keranjang dengan jumlah terendah dikembalikan terlebih dahulu. Dengan menyetel `scanIndexForward` ke false, kami membalikkan pesanan dan item keranjang dengan angka tertinggi dikembalikan terlebih dahulu.
- Kami menerapkan filter untuk menghapus hasil apa pun yang tidak sesuai dengan kriteria. Data yang difilter menghabiskan kapasitas baca apakah item tersebut cocok dengan filter.

Example **`Query`**menggunakan antarmuka tingkat rendah

Contoh berikut query tabel bernama `YourTableName` menggunakan. `keyConditionExpression` Ini membatasi kueri ke nilai kunci partisi tertentu dan mengurutkan nilai kunci yang dimulai dengan nilai awalan tertentu. Kondisi utama ini membatasi jumlah data yang dibaca dari DynamoDB. Akhirnya, kueri menerapkan filter pada data yang diambil dari DynamoDB menggunakan file. `filterExpression`

```
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`**menggunakan antarmuka Dokumen

Contoh berikut query tabel bernama `YourTableName` menggunakan antarmuka Dokumen.

```
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`**menggunakan antarmuka tingkat tinggi

Contoh berikut menanyakan tabel bernama `YourTableName` menggunakan DynamoDB API klien yang disempurnakan.

```
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;
        }
    }
}
```

Antarmuka tingkat tinggi menggunakan kelas data yang tidak dapat diubah

Saat Anda melakukan a `Query` dengan kelas data abadi tingkat tinggi, kodenya sama dengan contoh antarmuka tingkat tinggi kecuali untuk konstruksi kelas entitas atau. `YourItem` `YourImmutableItem` Untuk informasi lebih lanjut, lihat PutItemcontoh.

Antarmuka tingkat tinggi menggunakan kelas data yang tidak dapat diubah dan pustaka generasi boilerplate pihak ketiga

Saat Anda melakukan a `Query` dengan kelas data abadi tingkat tinggi, kodenya sama dengan contoh antarmuka tingkat tinggi kecuali untuk konstruksi kelas entitas atau. `YourItem` `YourImmutableLombokItem` Untuk informasi lebih lanjut, lihat PutItemcontoh.

Contoh kode tambahan

Untuk contoh tambahan tentang cara menggunakan DynamoDB dengan SDK for Java 2.x, lihat repositori contoh kode berikut:

- Contoh kode AWS aksi tunggal resmi
- Contoh kode aksi tunggal yang dikelola komunitas
- Contoh kode AWS berorientasi skenario resmi

Pemrograman sinkron dan asinkron

AWS SDK for Java 2.x Ini menyediakan klien sinkron dan asinkron untuk, Layanan AWS seperti DynamoDB.

`DynamoDbEnhancedClient`Kelas `DynamoDbClient` dan menyediakan metode sinkron yang memblokir eksekusi thread Anda hingga klien menerima respons dari layanan. Klien ini adalah cara paling mudah untuk berinteraksi dengan DynamoDB jika Anda tidak memerlukan operasi asinkron.

`DynamoDbEnhancedAsyncClient`Kelas `DynamoDbAsyncClient` and menyediakan metode asinkron yang segera kembali, dan memberikan kontrol kembali ke utas panggilan tanpa menunggu respons. Klien non-pemblokiran memiliki keuntungan yang digunakannya untuk konkurensi tinggi di

beberapa utas, yang menyediakan penanganan permintaan I/O yang efisien dengan sumber daya komputasi minimal. Ini meningkatkan throughput dan daya tanggap.

AWS SDK for Java 2.x Menggunakan dukungan asli untuk non-blocking I/O. AWS SDK untuk Java 1.x harus mensimulasikan non-blocking I/O.

Metode sinkron kembali sebelum respons tersedia, jadi Anda memerlukan cara untuk mendapatkan respons saat sudah siap. Metode asinkron dalam AWS SDK untuk Java mengembalikan `CompletableFuture`objek yang berisi hasil operasi asinkron di masa depan. Ketika Anda memanggil `get()` atau `join()` pada `CompletableFuture` objek-objek ini, kode Anda memblokir sampai hasilnya tersedia. Jika Anda memanggil ini pada saat yang sama saat Anda membuat permintaan, maka perilakunya mirip dengan panggilan sinkron biasa.

Untuk informasi selengkapnya tentang pemrograman asinkron, lihat Menggunakan pemrograman asinkron di Panduan Pengembang.AWS SDK for Java 2.x

Klien HTTP

Untuk mendukung setiap klien, ada klien HTTP yang menangani komunikasi dengan klien Layanan AWS. Anda dapat mencolokkan klien HTTP alternatif, memilih salah satu yang memiliki karakteristik yang paling sesuai dengan aplikasi Anda. Beberapa lebih ringan; beberapa memiliki lebih banyak opsi konfigurasi.

Beberapa klien HTTP hanya mendukung penggunaan sinkron, sementara yang lain hanya mendukung penggunaan asinkron. Untuk diagram alur yang dapat membantu Anda memilih klien HTTP optimal untuk beban kerja Anda, lihat rekomendasi klien HTTP di Panduan AWS SDK for Java 2.x Pengembang.

Daftar berikut menyajikan beberapa kemungkinan klien HTTP:

Topik

- Klien HTTP berbasis Apache
- URLConnectionberbasis HTTP klien
- Klien HTTP berbasis Netty
- AWS Klien HTTP berbasis CRT

Klien HTTP berbasis Apache

`ApacheHttpClient`Kelas mendukung klien layanan sinkron. Ini adalah klien HTTP default untuk penggunaan sinkron. Untuk informasi tentang mengonfigurasi *`ApacheHttpClient`* kelas, lihat Mengkonfigurasi klien HTTP berbasis Apache di Panduan Pengembang.AWS SDK for Java 2.x

`URLConnection`berbasis HTTP klien

`UrlConnectionHttpClient`Kelas adalah pilihan lain untuk klien sinkron. Ini memuat lebih cepat daripada klien HTTP berbasis Apache, tetapi memiliki lebih sedikit fitur. Untuk informasi tentang mengonfigurasi `UrlConnectionHttpClient` kelas, lihat Mengkonfigurasi klien HTTP URLConnection berbasis di Panduan AWS SDK for Java 2.x Pengembang.

Klien HTTP berbasis Netty

`NettyNioAsyncHttpClient`Kelas mendukung klien async. Ini adalah pilihan default untuk penggunaan async. Untuk informasi tentang mengonfigurasi *`NettyNioAsyncHttpClient`* kelas, lihat Mengkonfigurasi klien HTTP berbasis Netty di Panduan Pengembang.AWS SDK for Java 2.x

AWS Klien HTTP berbasis CRT

Yang lebih baru `AwsCrtHttpClient` dan `AwsCrtAsyncHttpClient` kelas dari pustaka AWS Common Runtime (CRT) adalah lebih banyak opsi yang mendukung klien sinkron dan asinkron. Dibandingkan dengan klien HTTP lainnya, AWS CRT menawarkan:

- Waktu startup SDK lebih cepat
- Jejak memori yang lebih kecil
- Mengurangi waktu latensi
- Manajemen kesehatan koneksi
- Penyeimbangan beban DNS

Untuk informasi tentang mengonfigurasi *`AwsCrtHttpClient`* dan *`AwsCrtAsyncHttpClient`* kelas, lihat Mengkonfigurasi klien HTTP AWS berbasis CRT di Panduan Pengembang AWS SDK for Java 2.x .

Klien HTTP AWS berbasis CRT bukan default karena itu akan merusak kompatibilitas mundur untuk aplikasi yang ada. Namun, untuk DynamoDB kami menyarankan Anda menggunakan klien HTTP berbasis CRT untuk penggunaan AWS sinkronisasi dan asinkron.

Untuk pengenalan klien HTTP AWS berbasis CRT, lihat Mengumumkan ketersediaan AWS CRT HTTP Client di Blog Alat Pengembang AWS SDK for Java 2.x.AWS

Mengkonfigurasi klien HTTP

Saat mengonfigurasi klien, Anda dapat memberikan berbagai opsi konfigurasi, termasuk:

- Menyetel batas waktu untuk berbagai aspek panggilan API.
- Mengaktifkan TCP Keep-Alive.
- Mengontrol kebijakan coba lagi saat mengalami kesalahan.
- Menentukan atribut eksekusi yang dapat dimodifikasi oleh instance pencegat Eksekusi. Pencegat eksekusi dapat menulis kode yang mencegat eksekusi permintaan dan tanggapan API Anda. Ini memungkinkan Anda untuk melakukan tugas-tugas seperti menerbitkan metrik dan memodifikasi permintaan dalam penerbangan.
- Menambahkan atau memanipulasi header HTTP.
- Mengaktifkan pelacakan metrik kinerja sisi klien. Menggunakan fitur ini membantu Anda mengumpulkan metrik tentang klien layanan di aplikasi Anda dan menganalisis output di Amazon CloudWatch.
- Menentukan layanan pelaksana alternatif yang akan digunakan untuk menjadwalkan tugas, seperti percobaan ulang async dan tugas batas waktu.

Anda mengontrol konfigurasi dengan menyediakan `ClientOverrideConfiguration`objek ke `Builder` kelas klien layanan. Anda akan melihat ini di beberapa contoh kode di bagian berikut.

`ClientOverrideConfiguration`Ini menyediakan pilihan konfigurasi standar. Klien HTTP pluggable yang berbeda memiliki kemungkinan konfigurasi khusus implementasi juga.

Topik di bagian ini

- Konfigurasi waktu habis
- RetryMode
- DefaultsMode
- Konfigurasi Keep-Alive

Konfigurasi waktu habis

Anda dapat menyesuaikan konfigurasi klien untuk mengontrol berbagai batas waktu yang terkait dengan panggilan layanan. DynamoDB memberikan latensi yang lebih rendah dibandingkan dengan yang lain. Layanan AWS Oleh karena itu, Anda mungkin ingin menyesuaikan properti ini untuk menurunkan nilai batas waktu sehingga Anda dapat gagal dengan cepat jika ada masalah jaringan.

Anda dapat menyesuaikan perilaku terkait latensi menggunakan `ClientOverrideConfiguration` pada klien DynamoDB atau dengan mengubah opsi konfigurasi terperinci pada implementasi klien HTTP yang mendasarinya.

Anda dapat mengonfigurasi properti berdampak berikut menggunakan: `ClientOverrideConfiguration`

- `apiCallAttemptTimeout`— Jumlah waktu untuk menunggu satu upaya untuk menyelesaikan permintaan HTTP sebelum menyerah dan waktu habis.
- `apiCallTimeout`— Jumlah waktu yang dimiliki klien untuk sepenuhnya menjalankan panggilan API. Ini termasuk eksekusi penangan permintaan yang terdiri dari semua permintaan HTTP, termasuk percobaan ulang.

AWS SDK for Java 2.x Ini memberikan nilai default untuk beberapa opsi batas waktu, seperti batas waktu koneksi dan batas waktu soket. SDK tidak memberikan nilai default untuk batas waktu panggilan API atau batas waktu percobaan panggilan API individual. Jika batas waktu ini tidak disetel dalam`ClientOverrideConfiguration`, maka SDK menggunakan nilai batas waktu soket sebagai gantinya untuk batas waktu panggilan API secara keseluruhan. Batas waktu soket memiliki nilai default 30 detik.

RetryMode

Konfigurasi lain yang terkait dengan konfigurasi batas waktu yang harus Anda pertimbangkan adalah objek `RetryMode` konfigurasi. Objek konfigurasi ini berisi kumpulan perilaku coba lagi.

SDK for Java 2.x mendukung mode coba lagi berikut:

- `legacy`— Mode coba lagi default jika Anda tidak mengubahnya secara eksplisit. Mode coba lagi ini khusus untuk Java SDK. Ini ditandai dengan hingga tiga percobaan ulang, atau lebih untuk layanan seperti DynamoDB, yang memiliki hingga delapan percobaan ulang.
- `standard`— Dinamakan “standar” karena lebih konsisten dengan yang lain AWS SDKs. Mode ini menunggu jumlah waktu acak mulai dari 0ms hingga 1.000 ms untuk percobaan ulang pertama.

Jika percobaan lagi diperlukan, maka mode ini mengambil waktu acak lain dari 0ms menjadi 1.000 ms dan mengalikannya dengan dua. Jika percobaan ulang tambahan diperlukan, maka ia melakukan pengambilan acak yang sama dikalikan empat, dan seterusnya. Setiap penantian dibatasi pada 20 detik. Mode ini melakukan percobaan ulang pada kondisi kegagalan yang lebih terdeteksi daripada `legacy` mode. Untuk DynamoDB, ia melakukan hingga tiga upaya maksimal total kecuali Anda mengganti dengan. numRetries

- `adaptive`— Dibangun pada `standard` mode dan secara dinamis membatasi tingkat AWS permintaan untuk memaksimalkan tingkat keberhasilan. Ini dapat terjadi dengan mengorbankan latensi permintaan. Kami tidak merekomendasikan mode coba lagi adaptif ketika latensi yang dapat diprediksi penting.

Anda dapat menemukan definisi yang diperluas dari mode coba lagi ini dalam topik perilaku Coba lagi di Panduan Referensi Alat AWS SDKs dan Alat.

Coba lagi kebijakan

Semua `RetryMode` konfigurasi memiliki `RetryPolicy`, yang dibangun berdasarkan satu atau lebih `RetryCondition`konfigurasi. `TokenBucketRetryCondition`Ini sangat penting untuk perilaku coba lagi implementasi klien DynamoDB SDK. Kondisi ini membatasi jumlah percobaan ulang yang dibuat SDK menggunakan algoritma token bucket. Bergantung pada mode coba lagi yang dipilih, pengecualian pembatasan mungkin atau mungkin tidak mengurangi token dari. `TokenBucket`

Ketika klien menemukan kesalahan yang dapat dicoba ulang, seperti pengecualian pembatasan atau kesalahan server sementara, SDK secara otomatis mencoba ulang permintaan tersebut. Anda dapat mengontrol berapa kali dan seberapa cepat percobaan ulang ini terjadi.

Saat mengkonfigurasi klien, Anda dapat memberikan `RetryPolicy` yang mendukung parameter berikut:

- `numRetries`— Jumlah maksimum percobaan ulang yang harus diterapkan sebelum permintaan dianggap gagal. Nilai defaultnya adalah 8 terlepas dari mode coba lagi yang Anda gunakan.

  > **Warning**
  >
  > Pastikan Anda mengubah nilai default ini setelah mempertimbangkan.

- `backoffStrategy`— `BackoffStrategy`Untuk diterapkan pada percobaan ulang, dengan `FullJitterBackoffStrategy`menjadi strategi default. Strategi ini melakukan penundaan eksponensial antara percobaan ulang tambahan berdasarkan jumlah saat ini atau percobaan

ulang, penundaan dasar, dan waktu backoff maksimum. Kemudian menambahkan jitter untuk memberikan sedikit keacakan. Penundaan dasar yang digunakan dalam penundaan eksponensial adalah 25 ms terlepas dari mode coba lagi.

- `retryCondition`— `RetryCondition`Menentukan apakah akan mencoba kembali permintaan sama sekali. Secara default, ia mencoba ulang satu set kode status HTTP tertentu dan pengecualian yang diyakini dapat dicoba ulang. Untuk sebagian besar situasi, konfigurasi default harus cukup.

Kode berikut menyediakan kebijakan coba lagi alternatif. Ini menentukan total lima percobaan ulang (enam permintaan total). Percobaan ulang pertama harus dilakukan setelah penundaan sekitar 100 ms, dengan setiap percobaan lagi tambahan menggandakan waktu itu secara eksponensial, hingga penundaan maksimum satu detik.

```
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

Properti batas waktu yang `ClientOverrideConfiguration` dan `RetryMode` tidak dikelola biasanya dikonfigurasi secara implisit dengan menentukan a. `DefaultsMode`

AWS SDK for Java 2.x (versi 2.17.102 atau yang lebih baru) memperkenalkan dukungan untuk. `DefaultsMode` Fitur ini menyediakan serangkaian nilai default untuk pengaturan umum yang dapat dikonfigurasi, seperti pengaturan komunikasi HTTP, perilaku coba lagi, pengaturan titik akhir Regional layanan, dan kemungkinan konfigurasi terkait SDK apa pun. Saat Anda menggunakan fitur ini, Anda bisa mendapatkan default konfigurasi baru yang disesuaikan dengan skenario penggunaan umum.

Mode default distandarisasi di semua. AWS SDKs SDK for Java 2.x mendukung mode default berikut:

- `legacy`— Menyediakan pengaturan default yang bervariasi menurut AWS SDK dan yang ada sebelum `DefaultsMode` dibuat.
- `standard`— Menyediakan pengaturan default yang tidak dioptimalkan untuk sebagian besar skenario.
- `in-region`— Dibangun pada mode standar dan termasuk pengaturan yang disesuaikan untuk aplikasi yang memanggil Layanan AWS dari dalam yang sama. Wilayah AWS
- `cross-region`— Dibangun pada mode standar dan termasuk pengaturan dengan batas waktu tinggi untuk aplikasi yang memanggil Layanan AWS di Wilayah yang berbeda.
- `mobile`— Dibangun pada mode standar dan mencakup pengaturan dengan batas waktu tinggi yang disesuaikan untuk aplikasi seluler dengan latensi lebih tinggi.
- `auto`— Dibangun pada mode standar dan termasuk fitur eksperimental. SDK mencoba menemukan lingkungan runtime untuk menentukan pengaturan yang sesuai secara otomatis. Deteksi otomatis berbasis heuristik dan tidak memberikan akurasi 100%. Jika lingkungan runtime tidak dapat ditentukan, maka mode standar digunakan. Deteksi otomatis mungkin menanyakan metadata Instance dan data pengguna, yang mungkin memperkenalkan latensi. Jika latensi startup sangat penting untuk aplikasi Anda, sebaiknya pilih yang eksplisit`DefaultsMode`.

Anda dapat mengonfigurasi mode default dengan cara berikut:

- Langsung pada klien, melalui`AwsClientBuilder.Builder#defaultsMode(DefaultsMode)`.
- Pada profil konfigurasi, melalui properti file `defaults_mode` profil.
- Secara global, melalui properti `aws.defaultsMode` sistem.
- Secara global, melalui variabel `AWS_DEFAULTS_MODE` lingkungan.

> **Note**
>
> Untuk mode apa pun selain`legacy`, nilai default yang dijual dapat berubah seiring berkembangnya praktik terbaik. Oleh karena itu, jika Anda menggunakan mode selain`legacy`, maka kami mendorong Anda untuk melakukan pengujian saat memutakhirkan SDK.

Konfigurasi Smart default di AWS SDKs and Tools Reference Guide menyediakan daftar properti konfigurasi dan nilai defaultnya dalam mode default yang berbeda.

Anda memilih nilai mode default berdasarkan karakteristik aplikasi Anda dan interaksi dengan Layanan AWS aplikasi.

Nilai-nilai ini dikonfigurasi dengan berbagai pilihan Layanan AWS dalam pikiran. Untuk penerapan DynamoDB tipikal di mana tabel dan aplikasi DynamoDB Anda digunakan dalam satu Wilayah, mode default `in-region` paling relevan di antara mode default. `standard`

Example Konfigurasi klien DynamoDB SDK disetel untuk panggilan latensi rendah

Contoh berikut menyesuaikan batas waktu ke nilai yang lebih rendah untuk panggilan DynamoDB latensi rendah yang diharapkan.

```
DynamoDbAsyncClient asyncClient = DynamoDbAsyncClient.builder()
    .defaultsMode(DefaultsMode.IN_REGION)
    .httpClientBuilder(AwsCrtAsyncHttpClient.builder())
    .overrideConfiguration(ClientOverrideConfiguration.builder()
        .apiCallTimeout(Duration.ofSeconds(3))
        .apiCallAttemptTimeout(Duration.ofMillis(500))
        .build())
    .build();
```

Implementasi klien HTTP individual dapat memberi Anda kontrol yang lebih terperinci atas batas waktu dan perilaku penggunaan koneksi. Misalnya, untuk klien AWS berbasis CRT, Anda dapat mengaktifkan`ConnectionHealthConfiguration`, yang memungkinkan klien untuk secara aktif memantau kesehatan koneksi yang digunakan. Untuk informasi selengkapnya, lihat Konfigurasi lanjutan klien HTTP AWS berbasis CRT di Panduan AWS SDK for Java 2.x Pengembang.

Konfigurasi Keep-Alive

Mengaktifkan keep-alive dapat mengurangi latensi dengan menggunakan kembali koneksi. Ada dua jenis keep-alive: HTTP Keep-Alive dan TCP Keep-Alive.

- HTTP Keep-Alive mencoba mempertahankan koneksi HTTPS antara klien dan server sehingga permintaan selanjutnya dapat menggunakan kembali koneksi tersebut. Ini melewatkan otentikasi HTTPS kelas berat pada permintaan selanjutnya. HTTP Keep-Alive diaktifkan secara default pada semua klien.
- TCP Keep-Alive meminta agar sistem operasi yang mendasarinya mengirimkan paket kecil melalui koneksi soket untuk memberikan jaminan ekstra bahwa soket tetap hidup dan segera mendeteksi tetesan apa pun. Ini memastikan bahwa permintaan selanjutnya tidak akan menghabiskan waktu mencoba menggunakan soket yang terjatuh. Secara default, TCP Keep-Alive dinonaktifkan

pada semua klien. Contoh kode berikut menunjukkan cara mengaktifkannya pada setiap klien HTTP. Ketika diaktifkan untuk semua klien HTTP berbasis non-CRT, mekanisme Keep-Alive yang sebenarnya bergantung pada sistem operasi. Oleh karena itu, Anda harus mengonfigurasi nilai TCP Keep-Alive tambahan, seperti batas waktu dan jumlah paket, melalui sistem operasi. Anda dapat melakukan ini menggunakan `sysctl` di Linux atau macOS, atau menggunakan nilai registri di Windows.

Example untuk mengaktifkan TCP Keep-Alive pada klien HTTP berbasis Apache

```
DynamoDbClient client = DynamoDbClient.builder()
    .httpClientBuilder(ApacheHttpClient.builder().tcpKeepAlive(true))
    .build();
```

`URLConnection`berbasis HTTP klien

Setiap klien sinkron yang menggunakan klien HTTP `URLConnection` berbasis `HttpURLConnection`tidak memiliki mekanisme untuk mengaktifkan keep-alive.

Example untuk mengaktifkan TCP Keep-Alive pada klien HTTP berbasis Netty

```
DynamoDbAsyncClient client = DynamoDbAsyncClient.builder()
    .httpClientBuilder(NettyNioAsyncHttpClient.builder().tcpKeepAlive(true))
    .build();
```

Example untuk mengaktifkan TCP Keep-Alive pada klien HTTP berbasis CRT AWS

Dengan klien HTTP AWS berbasis CRT, Anda dapat mengaktifkan TCP keep-alive dan mengontrol durasinya.

```
DynamoDbClient client = DynamoDbClient.builder()
    .httpClientBuilder(AwsCrtHttpClient.builder()
    .tcpKeepAliveConfiguration(TcpKeepAliveConfiguration.builder()
        .keepAliveInterval(Duration.ofSeconds(50))
        .keepAliveTimeout(Duration.ofSeconds(5))
        .build()))
    .build();
```

Saat menggunakan klien DynamoDB asinkron, Anda dapat mengaktifkan TCP Keep-Alive seperti yang ditunjukkan pada kode berikut.

```
DynamoDbAsyncClient client = DynamoDbAsyncClient.builder()
    .httpClientBuilder(AwsCrtAsyncHttpClient.builder()
    .tcpKeepAliveConfiguration(TcpKeepAliveConfiguration.builder()
        .keepAliveInterval(Duration.ofSeconds(50))
        .keepAliveTimeout(Duration.ofSeconds(5))
        .build()))
    .build();
```

Penanganan kesalahan

Ketika datang ke penanganan pengecualian, AWS SDK for Java 2.x menggunakan pengecualian runtime (tidak dicentang).

Pengecualian dasar, yang mencakup semua pengecualian SDK, adalah `SdkServiceException`, yang meluas dari Java tidak dicentang. `RuntimeException` Jika Anda menangkap ini, Anda akan menangkap semua pengecualian yang dilemparkan SDK.

`SdkServiceException`memiliki subclass yang disebut `AwsServiceException`. Subkelas ini menunjukkan masalah apa pun dalam komunikasi dengan. Layanan AWS Ini memiliki subclass yang disebut `DynamoDbException`, yang menunjukkan masalah dalam komunikasi dengan DynamoDB. Jika Anda menangkap ini, Anda akan menangkap semua pengecualian yang terkait dengan DynamoDB, tetapi tidak ada pengecualian SDK lainnya.

Ada jenis pengecualian yang lebih spesifik di bawah`DynamoDbException`. Beberapa jenis pengecualian ini berlaku untuk operasi bidang kontrol seperti. `TableAlreadyExistsException` Lainnya berlaku untuk operasi data-plane. Berikut ini adalah contoh pengecualian bidang data umum:

- `ConditionalCheckFailedException`— Anda menentukan kondisi dalam permintaan yang dievaluasi menjadi false. Misalnya, Anda mungkin telah mencoba melakukan pembaruan bersyarat pada suatu item, tetapi nilai atribut sebenarnya tidak cocok dengan nilai yang diharapkan dalam kondisi yang dihadapi. Permintaan yang gagal dengan cara ini tidak dicoba lagi.

Situasi lain tidak memiliki pengecualian khusus yang ditentukan. Misalnya, ketika permintaan Anda dibatasi, spesifik `ProvisionedThroughputExceededException` mungkin dilemparkan, sementara dalam kasus lain permintaan yang lebih umum `DynamoDbException` dilemparkan. Dalam kedua kasus tersebut, Anda dapat menentukan apakah pelambatan menyebabkan pengecualian dengan memeriksa apakah `isThrottlingException()` pengembalian. `true`

Tergantung pada kebutuhan aplikasi Anda, Anda dapat menangkap semua `AwsServiceException` atau `DynamoDbException` instance. Namun, Anda sering membutuhkan perilaku yang berbeda dalam situasi yang berbeda. Logika untuk menangani kegagalan pemeriksaan kondisi berbeda dari itu untuk menangani pelambatan. Tentukan jalur luar biasa mana yang ingin Anda tangani dan pastikan untuk menguji jalur alternatif. Ini membantu Anda memastikan bahwa Anda dapat menangani semua skenario yang relevan.

Untuk daftar kesalahan umum yang mungkin Anda temui, lihatPenanganan kesalahan dengan DynamoDB. Lihat juga Kesalahan Umum di Referensi Amazon DynamoDB API. Referensi API juga menyediakan kesalahan yang tepat untuk setiap operasi API, seperti untuk `Query`operasi. Untuk informasi tentang menangani pengecualian, lihat Penanganan pengecualian untuk Panduan AWS SDK for Java 2.x Pengembang. AWS SDK for Java 2.x

AWS ID permintaan

Setiap permintaan menyertakan ID permintaan, yang dapat berguna untuk ditarik jika Anda bekerja dengan AWS Dukungan untuk mendiagnosis masalah. Setiap pengecualian yang berasal dari `SdkServiceException` memiliki `requestId()`metode yang tersedia untuk mengambil ID permintaan.

Pencatatan log

Menggunakan logging asalkan SDK menyediakan dapat berguna baik untuk menangkap pesan penting dari pustaka klien dan untuk tujuan debugging yang lebih mendalam. Logger bersifat hierarkis dan SDK digunakan `software.amazon.awssdk` sebagai root logger-nya. Anda dapat mengkonfigurasi level dengan salah satu dari`TRACE`,`DEBUG`,`INFO`,`WARN`,`ERROR`,`ALL`, atau`OFF`. Level yang dikonfigurasi berlaku untuk logger itu dan turun ke hierarki logger.

Untuk logging nya, AWS SDK for Java 2.x menggunakan Simple Logging Façade untuk Java (SLF4J). Ini bertindak sebagai lapisan abstraksi di sekitar logger lain, dan Anda dapat menggunakannya untuk mencolokkan logger yang Anda inginkan. Untuk petunjuk tentang memasukkan logger, lihat manual pengguna SLF4 J.

Setiap logger memiliki perilaku tertentu. Secara default, logger Log4j 2.x membuat`ConsoleAppender`, yang menambahkan peristiwa log ke `System.out` dan default ke tingkat log. `ERROR`

SimpleLogger Logger yang disertakan dalam output SLF4 J secara default ke `System.err` dan default ke tingkat log. `INFO`

Sebaiknya Anda mengatur level WARN untuk setiap penerapan produksi `software.amazon.awssdk` untuk menangkap pesan penting apa pun dari pustaka klien SDK sambil membatasi kuantitas keluaran.

Jika SLF4 J tidak dapat menemukan logger yang didukung di jalur kelas (tidak ada pengikatan SLF4 J), maka defaultnya adalah implementasi tanpa operasi. Implementasi ini menghasilkan logging pesan untuk `System.err` menjelaskan bahwa SLF4 J tidak dapat menemukan implementasi logger di classpath. Untuk mencegah situasi ini, Anda harus menambahkan implementasi logger. Untuk melakukan ini, Anda dapat menambahkan ketergantungan di Apache Maven Anda `pom.xml` pada artefak, seperti atau. `org.slf4j.slf4j-simple` `org.apache.logging.log4j.log4j-slf4j2-imp`

Untuk informasi tentang cara mengonfigurasi logging di SDK, termasuk menambahkan dependensi logging ke konfigurasi aplikasi Anda, lihat Logging dengan SDK for Java 2.x di Panduan Pengembang.AWS SDK untuk Java

Konfigurasi berikut dalam `Log4j2.xml` file menunjukkan cara menyesuaikan perilaku logging jika Anda menggunakan logger Apache Log4j 2. Konfigurasi ini menetapkan level root logger keWARN. Semua logger dalam hierarki mewarisi level log ini, termasuk logger. `software.amazon.awssdk`

Secara default, output masuk ke`System.out`. Dalam contoh berikut, kita masih mengganti appender Log4j keluaran default untuk menerapkan log4j yang disesuaikan. `PatternLayout`

Contoh file **`Log4j2.xml`** konfigurasi

Konfigurasi berikut mencatat pesan ke konsol di WARN tingkat ERROR dan untuk semua hierarki logger.

```
<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 minta pencatatan ID

Ketika terjadi kesalahan, Anda dapat menemukan permintaan IDs dalam pengecualian. Namun, jika Anda menginginkan permintaan IDs untuk permintaan yang tidak menghasilkan pengecualian, Anda dapat menggunakan logging.

Permintaan keluaran `software.amazon.awssdk.request` logger IDs di tingkat tersebut. `DEBUG` Contoh berikut memperluas sebelumnya configuration example untuk menjaga level root logger pada`ERROR`, level at`WARN`, dan level at. `software.amazon.awssdk` `software.amazon.awssdk.request` `DEBUG` Menyetel level ini membantu menangkap permintaan IDs dan detail terkait permintaan lainnya, seperti titik akhir dan kode status.

```
<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>
```

Berikut adalah contoh output log:

```
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
```

Paginasi

Beberapa permintaan, seperti `Query`dan `Scan`, membatasi ukuran data yang dikembalikan pada satu permintaan dan mengharuskan Anda membuat permintaan berulang untuk menarik halaman berikutnya.

Anda dapat mengontrol jumlah maksimum item untuk dibaca untuk setiap halaman dengan `Limit` parameter. Misalnya, Anda dapat menggunakan `Limit` parameter untuk mengambil hanya 10 item terakhir. Batas ini menentukan berapa banyak item untuk membaca dari tabel sebelum penyaringan diterapkan. Jika Anda menginginkan tepat 10 item setelah pemfilteran, tidak ada cara untuk menentukannya. Anda hanya dapat mengontrol jumlah yang telah difilter sebelumnya dan memeriksa sisi klien saat Anda benar-benar mengambil 10 item. Terlepas dari batasnya, respons selalu memiliki ukuran maksimum 1 MB.

A `LastEvaluatedKey` mungkin disertakan dalam respons API. Ini menunjukkan bahwa respons berakhir karena mencapai batas hitungan atau batas ukuran. Kunci ini adalah kunci terakhir yang dievaluasi untuk respons itu. Dengan berinteraksi langsung dengan API, Anda dapat mengambil ini `LastEvaluatedKey` dan meneruskannya ke panggilan tindak lanjut `ExclusiveStartKey` untuk membaca potongan berikutnya dari titik awal itu. Jika no `LastEvaluatedKey` dikembalikan, itu berarti tidak ada lagi item yang cocok dengan panggilan `Scan` API `Query` atau.

Contoh berikut menggunakan antarmuka tingkat rendah untuk membatasi item hingga 100 berdasarkan `keyConditionExpression` parameter.

```
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());
}
```

Ini AWS SDK for Java 2.x dapat menyederhanakan interaksi ini dengan DynamoDB dengan menyediakan metode pagination otomatis yang membuat beberapa panggilan layanan untuk secara otomatis mendapatkan halaman hasil berikutnya untuk Anda. Ini menyederhanakan kode Anda, tetapi menghilangkan beberapa kontrol penggunaan sumber daya yang akan Anda simpan dengan membaca halaman secara manual.

Dengan menggunakan `Iterable` metode yang tersedia di klien DynamoDB, `QueryPaginator`seperti `ScanPaginator`dan, SDK menangani pagination. Jenis pengembalian metode ini adalah iterable khusus yang dapat Anda gunakan untuk mengulangi semua halaman. SDK secara internal menangani panggilan layanan untuk Anda. Menggunakan Java Stream API, Anda dapat menangani hasil `QueryPaginator` seperti yang ditunjukkan pada contoh berikut.

```
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();
```

Anotasi kelas data

Java SDK menyediakan beberapa anotasi yang dapat Anda masukkan pada atribut kelas data Anda. Anotasi ini memengaruhi cara SDK berinteraksi dengan atribut. Dengan menambahkan anotasi, Anda dapat memiliki atribut yang berperilaku sebagai penghitung atom implisit, mempertahankan nilai stempel waktu yang dibuat secara otomatis, atau melacak nomor versi item. Untuk informasi selengkapnya, lihat Anotasi kelas data.

Penanganan kesalahan dengan DynamoDB

Bagian ini menjelaskan kesalahan runtime dan cara menanganinya. Kode dan pesan kesalahan yang khusus untuk Amazon DynamoDB juga dijelaskan dalam bagian ini. Untuk daftar kesalahan umum yang berlaku untuk semua AWS layanan, lihat Manajemen Akses

Topik

- Komponen kesalahan
- Kesalahan transaksional
- Pesan dan kode kesalahan
- Penanganan kesalahan dalam aplikasi Anda
- Percobaan ulang kesalahan dan penundaan eksponensial
- Operasi batch dan penanganan kesalahan

Komponen kesalahan

Ketika program Anda mengirimkan permintaan, DynamoDB mencoba untuk memprosesnya. Jika permintaan berhasil, DynamoDB akan menghasilkan kode status sukses HTTP (`200 OK`), bersama dengan hasil dari operasi yang diminta.

Jika permintaan tidak berhasil, DynamoDB menampilkan kesalahan. Setiap kesalahan memiliki tiga komponen:

- Kode status HTTP (seperti `400`).
- Nama pengecualian (seperti `ResourceNotFoundException`).
- Pesan kesalahan (seperti `Requested resource not found: Table: tablename not found`).

AWS SDKs Jaga penyebaran kesalahan ke aplikasi Anda sehingga Anda dapat mengambil tindakan yang tepat. Misalnya, dalam program Java, Anda bisa menulis logika `try-catch` untuk menangani `ResourceNotFoundException`.

Jika Anda tidak menggunakan AWS SDK, Anda perlu mengurai konten respons tingkat rendah dari DynamoDB. Berikut ini adalah contoh respons tersebut.

```
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"}
```

Kesalahan transaksional

Untuk informasi tentang kesalahan transaksional, lihat Penanganan konflik transaksi di DynamoDB

Pesan dan kode kesalahan

Berikut ini adalah daftar pengecualian yang dikembalikan oleh DynamoDB, dikelompokkan berdasarkan kode status HTTP. Jika Boleh diulang? adalah Ya, Anda dapat mengirimkan permintaan yang sama lagi. Jika Boleh diulang? adalah Tidak, Anda harus memperbaiki masalah di sisi klien sebelum mengirimkan permintaan baru.

Kode status HTTP 400

Kode status `400` HTTP menunjukkan adanya masalah dengan permintaan Anda, seperti kegagalan autentikasi, parameter yang diperlukan tidak ada, atau melebihi throughput yang disediakan tabel. Anda harus memperbaiki masalah di aplikasi Anda sebelum mengirimkan permintaan lagi.

AccessDeniedException

Pesan: Akses ditolak.

Klien salah menandatangani permintaan. Jika Anda menggunakan AWS SDK, permintaan ditandatangani untuk Anda secara otomatis; jika tidak, buka Proses penandatanganan Tanda Tangan versi 4 di Referensi Umum AWS.

Boleh diulang? Tidak

ConditionalCheckFailedException

Pesan: Permintaan bersyarat gagal.

Anda menentukan syarat yang dievaluasi ke false. Misalnya, Anda mungkin telah mencoba melakukan pembaruan bersyarat pada suatu item, tetapi nilai atribut sebenarnya tidak cocok dengan nilai yang diharapkan dalam kondisi yang dihadapi.

Boleh diulang? Tidak

IncompleteSignatureException

Pesan: Tanda tangan permintaan tidak sesuai dengan standar AWS .

Tanda tangan permintaan tidak menyertakan semua komponen yang diperlukan. Jika Anda menggunakan AWS SDK, permintaan ditandatangani untuk Anda secara otomatis; jika tidak, buka proses penandatanganan Versi Tanda Tangan 4 di Referensi Umum AWS.

Ingin mengulang? Tidak

ItemCollectionSizeLimitExceededException

Pesan: Ukuran koleksi terlampaui.

Untuk tabel dengan indeks sekunder lokal, sekelompok item dengan nilai kunci partisi yang sama telah melampaui batas ukuran maksimum 10 GB. Untuk informasi selengkapnya tentang kumpulan item, lihat Kumpulan item dalam Indeks Sekunder Lokal.

Boleh diulang? Ya

LimitExceededException

Pesan: Terlalu banyak operasi untuk pelanggan tertentu.

Terlalu banyak operasi bidang kontrol secara bersamaan. Jumlah kumulatif tabel dan indeks dalam `CREATING`, `DELETING`, atau keadaan `UPDATING` tidak boleh melebihi 500.

Boleh diulang? Ya

MissingAuthenticationTokenException

Pesan: Permintaan harus berisi ID Kunci Akses AWS yang valid (terdaftar).

Permintaan tidak menyertakan header otorisasi yang diperlukan, atau formatnya salah. Lihat API tingkat rendah DynamoDB.

Boleh diulang? Tidak

ProvisionedThroughputExceededException

Pesan: Anda melampaui throughput maksimum yang diizinkan untuk sebuah tabel atau untuk satu atau beberapa indeks sekunder global. Untuk melihat metrik kinerja untuk throughput yang disediakan vs. throughput yang dikonsumsi, buka konsol Amazon. CloudWatch

Contoh: Tingkat permintaan Anda terlalu tinggi. AWS SDKs Untuk DynamoDB secara otomatis mencoba kembali permintaan yang menerima pengecualian ini. Permintaan Anda akhirnya berhasil, kecuali antrean percobaan ulang Anda terlalu besar untuk diselesaikan. Kurangi frekuensi permintaan menggunakan Percobaan ulang kesalahan dan penundaan eksponensial.

Boleh diulang? Ya

RequestLimitExceeded

Pesan: Throughput melebihi batas throughput saat ini untuk akun Anda. Untuk meminta peningkatan batas, hubungi AWS Support di https://aws.amazon.com/support.

Contoh: Tingkat permintaan sesuai permintaan melebihi throughput akun yang diizinkan dan tabel tidak dapat diskalakan lebih lanjut.

Boleh diulang? Ya

ResourceInUseException

Pesan: Sumber daya yang Anda coba ubah sedang digunakan.

Contoh: Anda mencoba untuk membuat ulang tabel yang sudah ada, atau menghapus tabel yang saat ini dalam status `CREATING`.

Boleh diulang? Tidak

ResourceNotFoundException

Pesan: Sumber daya yang diminta tidak ditemukan.

Contoh: Tabel yang diminta tidak ada, atau terlalu dini dalam status `CREATING`.

Boleh diulang? Tidak

ThrottlingException

Pesan: Tingkat permintaan melebihi throughput yang diizinkan.

Pengecualian ini dikembalikan sebagai AmazonServiceException respons dengan kode status THROTTLING_EXCEPTION. Pengecualian ini mungkin dikembalikan jika Anda melakukan operasi API bidang kontrol terlalu cepat.

Untuk tabel yang menggunakan mode sesuai permintaan, pengecualian ini mungkin dikembalikan untuk setiap operasi API bidang data jika tingkat permintaan Anda terlalu tinggi. Untuk mempelajari lebih lanjut tentang penskalaan sesuai permintaan, lihat. Throughput awal dan properti penskalaan

Boleh diulang? Ya

UnrecognizedClientException

Pesan: ID Kunci Akses atau token keamanan tidak valid.

Tanda tangan permintaan tidak benar. Penyebab yang paling mungkin adalah ID kunci AWS akses atau kunci rahasia yang tidak valid.

Boleh diulang? Ya

ValidationException

Pesan: Bervariasi, tergantung pada kesalahan tertentu yang dialami

Kesalahan ini dapat terjadi karena beberapa alasan, seperti parameter yang diperlukan tidak ada, nilai di luar rentang, atau jenis data tidak cocok. kesalahan berisi detail tentang bagian tertentu dari permintaan yang menyebabkan kesalahan.

Boleh diulang? Tidak

Kode status HTTP 5xx

Kode status 5xx HTTP menunjukkan masalah yang harus diselesaikan oleh AWS. Kemungkinan ini adalah kesalahan sementara, sehingga Anda dapat mencoba lagi permintaan Anda hingga berhasil. Jika tidak, masuk ke Service Health Dashboard AWS untuk melihat apakah ada masalah operasional dengan layanan ini.

Untuk informasi selengkapnya, lihat Bagaimana cara mengatasi kesalahan HTTP 5xx di Amazon DynamoDB?

InternalServerError (HTTP 500)

DynamoDB tidak dapat memproses permintaan Anda.

Boleh diulang? Ya

> **Note**
>
> Anda mungkin mengalami kesalahan server internal saat menggunakan item. Ini diduga terjadi selama masa aktif tabel. Setiap permintaan yang gagal dapat segera dicoba kembali.
> Ketika Anda menerima kode status 500 pada operasi tulis, operasi tersebut mungkin berhasil atau gagal. Jika operasi tulis adalah permintaan `TransactWriteItem`, maka boleh saja mencoba ulang operasi tersebut. Jika operasi tulis adalah permintaan tulis item tunggal seperti `PutItem`, `UpdateItem`, atau `DeleteItem`, maka aplikasi Anda akan membaca status item sebelum mencoba kembali operasi tersebut, dan/atau menggunakan Contoh CLI ekspresi kondisi DynamoDB untuk memastikan item tetap dalam status yang benar setelah mencoba ulang, terlepas dari apakah operasi sebelumnya berhasil atau gagal. Jika idempotensi adalah persyaratan untuk operasi tulis, gunakan `TransactWriteItem`, yang mendukung permintaan idempoten dengan menentukan `ClientRequestToken` secara otomatis untuk membedakan beberapa upaya untuk melakukan tindakan yang sama.

ServiceUnavailable (HTTP 503)

DynamoDB saat ini tidak tersedia. (Ini harus merupakan status sementara.)

Boleh diulang? Ya

Penanganan kesalahan dalam aplikasi Anda

Agar aplikasi Anda berjalan lancar, Anda perlu menambahkan logika untuk menangkap dan merespons kesalahan. Pendekatan umum termasuk menggunakan blok `try-catch` atau pernyataan `if-then`.

AWS SDKs Melakukan percobaan ulang dan pengecekan kesalahan mereka sendiri. Jika Anda menemukan kesalahan saat menggunakan salah satu AWS SDKs, kode kesalahan dan deskripsi dapat membantu Anda memecahkan masalah itu.

Anda juga akan melihat `Request ID` dalam respons. Ini `Request ID` dapat membantu jika Anda perlu bekerja dengan AWS Support untuk mendiagnosis suatu masalah.

Percobaan ulang kesalahan dan penundaan eksponensial

Banyak komponen di jaringan, seperti server DNS, sakelar, penyeimbang beban, dan lainnya, dapat menghasilkan kesalahan di mana pun selama permintaan tertentu. Teknik biasa untuk menangani respons kesalahan ini dalam lingkungan jaringan adalah dengan menerapkan percobaan ulang dalam aplikasi klien. Teknik ini meningkatkan keandalan aplikasi.

Setiap AWS SDK mengimplementasikan logika coba ulang secara otomatis. Anda dapat mengubah parameter percobaan ulang sesuai kebutuhan Anda. Misalnya, pertimbangkan aplikasi Java yang memerlukan strategi gagal cepat (fail fast), dengan tidak memperbolehkan percobaan ulang jika terjadi kesalahan. Dengan AWS SDK untuk Java, Anda dapat menggunakan `ClientConfiguration` kelas dan memberikan `maxErrorRetry` nilai `0` untuk mematikan percobaan ulang. Untuk informasi selengkapnya, lihat dokumentasi AWS SDK untuk bahasa pemrograman Anda.

Jika Anda tidak menggunakan AWS SDK, Anda harus mencoba kembali permintaan asli yang menerima kesalahan server (5xx). Namun, kesalahan klien (4xx, selain `ThrottlingException` atau `ProvisionedThroughputExceededException`) menunjukkan bahwa Anda perlu merevisi permintaan itu sendiri untuk memperbaiki masalah sebelum melakukan percobaan ulang.

Selain percobaan ulang sederhana, setiap AWS SDK mengimplementasikan algoritma backoff eksponensial untuk kontrol aliran yang lebih baik. Konsep di balik penundaan eksponensial adalah menggunakan waktu tunggu yang lebih lama di antara percobaan ulang untuk respon kesalahan berturut-turut. Sebagai contoh, hinga 50 milidetik sebelum percobaan ulang pertama, hingga 100 milidetik sebelum percobaan ulang kedua, hingga 200 milidetik sebelum percobaan ulang ketiga, dan seterusnya. Namun, setelah satu menit, jika permintaan tidak berhasil, masalahnya mungkin pada ukuran permintaan yang melebihi throughput yang disediakan, dan bukan pada tingkat permintaan. Tetapkan jumlah maksimum percobaan ulang untuk berhenti sekitar satu menit. Jika permintaan tidak berhasil, selidiki opsi throughput yang disediakan.

> **Note**
>
> AWS SDKs Mengimplementasikan logika coba ulang otomatis dan backoff eksponensial.

Kebanyakan algoritma penundaan eksponensial menggunakan jitter (penundaan acak) untuk mencegah tabrakan berturut-turut. Karena Anda tidak berusaha menghindari tabrakan seperti itu dalam kasus ini, Anda tidak perlu menggunakan nomor acak ini. Namun, jika Anda menggunakan klien secara bersamaan, jitter dapat membantu permintaan Anda berhasil lebih cepat. Untuk informasi selengkapnya, lihat postingan blog tentang Penundaan eksponensial dan jitter.

Operasi batch dan penanganan kesalahan

API tingkat rendah DynamoDB mendukung operasi batch untuk baca dan tulis. `BatchGetItem` membaca item dari satu atau beberapa tabel, dan `BatchWriteItem` memasukkan atau menghapus item dalam satu atau beberapa tabel. Operasi batch ini diimplementasikan sebagai pembungkus di sekitar operasi DynamoDB non-batch lainnya. Dengan kata lain, `BatchGetItem` menginvokasi `GetItem` sekali untuk setiap item dalam batch. Demikian pula, `BatchWriteItem` menginvokasi `DeleteItem` atau `PutItem`, jika sesuai, untuk setiap item dalam batch.

Operasi batch dapat menoleransi kegagalan permintaan individual dalam batch. Misalnya, pertimbangkan `BatchGetItem` untuk membaca lima item. Meskipun beberapa permintaan `GetItem` yang mendasarinya gagal, ini tidak menyebabkan seluruh operasi `BatchGetItem` gagal. Namun, jika kelima operasi baca gagal, maka seluruh `BatchGetItem` akan gagal pula.

Operasi batch mengembalikan informasi tentang masing-masing permintaan yang gagal sehingga Anda dapat mendiagnosis masalah dan mencoba kembali operasi tersebut. Untuk `BatchGetItem`, tabel dan kunci primer yang dimaksud dikembalikan dalam nilai `UnprocessedKeys` respons. Untuk `BatchWriteItem`, informasi serupa dikembalikan dalam `UnprocessedItems`.

Kemungkinan besar penyebab gagalnya baca atau tulis adalah throttling. Untuk `BatchGetItem`, satu atau beberapa tabel dalam permintaan batch tidak memiliki kapasitas baca yang memadai untuk mendukung operasi. Untuk `BatchWriteItem`, satu atau beberapa tabel tidak memiliki kapasitas tulis yang memadai.

Jika DynamoDB mengembalikan item yang belum diproses, Anda harus mencoba ulang operasi batch pada item tersebut. Namun, kami sangat merekomendasikan agar Anda menggunakan algoritma penundaan eksponensial. Jika Anda segera mencoba ulang operasi batch, permintaan baca atau tulis yang mendasarinya masih bisa gagal karena throttling pada masing-masing tabel. Jika

Anda menunda operasi batch menggunakan penundaan eksponensial, masing-masing permintaan dalam batch kemungkinan besar akan berhasil.

Menggunakan DynamoDB dengan SDK AWS

AWS kit pengembangan perangkat lunak (SDKs) tersedia untuk banyak bahasa pemrograman populer. Setiap SDK menyediakan API, contoh kode, dan dokumentasi yang memudahkan developer untuk membangun aplikasi dalam bahasa pilihan mereka.

Dokumentasi SDK	Contoh kode
AWS SDK untuk C++	AWS SDK untuk C++ contoh kode
AWS CLI	AWS CLI contoh kode
AWS SDK untuk Go	AWS SDK untuk Go contoh kode
AWS SDK untuk Java	AWS SDK untuk Java contoh kode
AWS SDK untuk JavaScript	AWS SDK untuk JavaScript contoh kode
AWS SDK untuk Kotlin	AWS SDK untuk Kotlin contoh kode
AWS SDK untuk .NET	AWS SDK untuk .NET contoh kode
AWS SDK untuk PHP	AWS SDK untuk PHP contoh kode
Alat AWS untuk PowerShell	Alat untuk contoh PowerShell kode
AWS SDK untuk Python (Boto3)	AWS SDK untuk Python (Boto3) contoh kode
AWS SDK untuk Ruby	AWS SDK untuk Ruby contoh kode
AWS SDK for Rust	AWS SDK for Rust contoh kode
AWS SDK untuk SAP ABAP	AWS SDK untuk SAP ABAP contoh kode
AWS SDK for Swift	AWS SDK for Swift contoh kode

Untuk contoh khusus DynamoDB, lihat Contoh kode untuk DynamoDB menggunakan AWS SDKs.

Ketersediaan contoh

Tidak dapat menemukan apa yang Anda butuhkan? Minta contoh kode menggunakan tautan Berikan umpan balik di bagian bawah halaman ini.

Bekerja dengan tabel, item, kueri, pindaian, dan indeks

Bagian ini menyediakan detail tentang bekerja dengan tabel, item, kueri, dan banyak lagi di Amazon DynamoDB.

Topik

- Bekerja dengan tabel dan data di DynamoDB
- Tabel global - Replikasi multi-Wilayah untuk DynamoDB
- Bekerja dengan item dan atribut di DynamoDB
- Meningkatkan akses data dengan indeks sekunder di DynamoDB
- Mengelola alur kerja kompleks dengan DynamoDB Transactions
- Tangkapan data perubahan dengan Amazon DynamoDB

Bekerja dengan tabel dan data di DynamoDB

Bagian ini menjelaskan cara menggunakan AWS Command Line Interface (AWS CLI) dan AWS SDKs untuk membuat, memperbarui, dan menghapus tabel di Amazon DynamoDB.

> **Note**
>
> Anda juga dapat melakukan tugas-tugas yang sama ini menggunakan AWS Management Console. Untuk informasi selengkapnya, lihat Menggunakan konsol.

Bagian ini juga menyediakan informasi selengkapnya tentang kapasitas throughput menggunakan penskalaan otomatis DynamoDB atau mengatur throughput yang disediakan secara manual.

Topik

- Operasi dasar pada tabel DynamoDB
- Pertimbangan saat memilih kelas tabel di DynamoDB
- Menambahkan tag dan label ke sumber daya di DynamoDB

Operasi dasar pada tabel DynamoDB

Mirip dengan sistem basis data lainnya, Amazon DynamoDB menyimpan data dalam tabel. Anda dapat mengelola tabel Anda menggunakan beberapa operasi dasar.

Topik

- Membuat tabel
- Menjelaskan tabel
- Memperbarui tabel
- Menghapus tabel
- Menggunakan perlindungan penghapusan
- Mencantumkan nama tabel
- Menggambarkan kuota throughput yang disediakan

Membuat tabel

Gunakan operasi `CreateTable` untuk membuat tabel di Amazon DynamoDB. Untuk membuat tabel, Anda harus memberikan informasi berikut:

- Nama tabel. Nama harus sesuai dengan aturan penamaan DynamoDB, dan harus unik untuk akun saat ini dan Wilayah. AWS Misalnya, Anda dapat membuat tabel `People` di AS Timur (Virginia Utara) dan tabel `People` lainnya di Eropa (Irlandia). Namun, kedua tabel ini akan benar-benar berbeda satu sama lain. Untuk informasi selengkapnya, lihat Jenis data dan aturan penamaan yang didukung di Amazon DynamoDB.
- Kunci primer. Kunci primer dapat terdiri dari satu atribut (kunci partisi) atau dua atribut (kunci partisi dan kunci urutan). Anda perlu memberikan nama atribut, jenis daya, dan peran dari masing-masing atribut: `HASH` (untuk kunci partisi) dan `RANGE` (untuk kunci urutan). Untuk informasi selengkapnya, lihat Kunci primer.
- Pengaturan throughput (untuk tabel yang disediakan). Jika menggunakan mode yang disediakan, Anda harus menentukan pengaturan throughput tulis dan baca awal untuk tabel. Anda dapat mengubah pengaturan ini di waktu lain, atau mengaktifkan penskalaan otomatis DynamoDB untuk mengelola pengaturan untuk Anda. Untuk informasi selengkapnya, lihat Mode kapasitas yang disediakan DynamoDB dan Mengelola kapasitas throughput secara otomatis dengan penskalaan otomatis DynamoDB.

Contoh 1: Buat tabel sesuai permintaan

Untuk membuat tabel `Music` yang sama menggunakan mode sesuai permintaan.

```
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
```

Operasi `CreateTable` mengembalikan metadata untuk tabel, seperti yang ditunjukkan berikut ini.

```
{
    "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**
>
> Ketika memanggil `DescribeTable` pada tabel sesuai permintaan, unit kapasitas baca dan unit kapasitas tulis ditetapkan ke 0.

Contoh 2: Buat tabel yang disediakan

AWS CLI Contoh berikut menunjukkan cara membuat tabel (`Music`). Kunci primer terdiri dari `Artist` (kunci partisi) dan `SongTitle` (kunci urutan), masing-masing memiliki jenis daya `String`. Throughput maksimum untuk tabel ini adalah 10 unit kapasitas baca dan 5 unit kapasitas tulis.

```
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
```

Operasi `CreateTable` mengembalikan metadata untuk tabel, seperti yang ditunjukkan berikut ini.

```
{
    "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
    }
}
```

Elemen `TableStatus` menunjukkan status tabel saat ini (`CREATING`). Mungkin perlu beberapa saat untuk membuat tabel, bergantung pada nilai yang Anda tentukan untuk `ReadCapacityUnits` dan `WriteCapacityUnits`. Nilai yang lebih besar untuk ini memerlukan DynamoDB mengalokasikan lebih banyak sumber daya untuk tabel.

Contoh 3: Membuat tabel menggunakan kelas tabel akses standar-jarang DynamoDB

Untuk membuat tabel `Music` yang sama menggunakan kelas tabel akses standar-jarang DynamoDB.

```
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
```

Operasi `CreateTable` mengembalikan metadata untuk tabel, seperti yang ditunjukkan berikut ini.

```
{
    "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
    }
}
```

Menjelaskan tabel

Untuk melihat detail tentang tabel, gunakan operasi `DescribeTable`. Anda harus memberikan nama tabel. Output dari `DescribeTable` adalah dalam format yang sama seperti dari `CreateTable`. Ini mencakup stempel waktu pembuatan tabel, skema kunci, pengaturan throughput yang disediakan, perkiraan ukuran, dan indeks sekunder apa pun yang ada.

> **Important**
>
> Ketika menjalankan `DescribeTable` pada tabel sesuai permintaan, unit kapasitas baca dan unit kapasitas tulis ditetapkan ke 0.

Example

```
aws dynamodb describe-table --table-name Music
```

Tabel siap digunakan ketika `TableStatus` berubah dari `CREATING` menjadi `ACTIVE`.

> **Note**
>
> Jika Anda mengeluarkan permintaan `DescribeTable` segera setelah permintaan `CreateTable`, DynamoDB mungkin mengembalikan kesalahan (`ResourceNotFoundException`). Hal ini karena `DescribeTable` menggunakan kueri yang akhirnya konsisten, dan metadata untuk tabel Anda mungkin tidak tersedia pada saat itu. Tunggu selama beberapa detik, kemudian coba permintaan `DescribeTable` kembali. Untuk tujuan penagihan, biaya penyimpanan DynamoDB mencakup overhead per item sebesar 100 byte. (Untuk informasi selengkapnya, buka Harga DynamoDB.) Tambahan

100 byte per item ini tidak digunakan dalam perhitungan unit kapasitas atau oleh operasi `DescribeTable`.

Memperbarui tabel

Operasi `UpdateTable` memungkinkan Anda melakukan salah satu hal berikut:

- Memodifikasi pengaturan throughput tabel yang disediakan (untuk tabel mode yang disediakan).
- Mengubah mode kapasitas baca/tulis tabel.
- Memanipulasi indeks sekunder global pada tabel (lihat Menggunakan Indeks Sekunder Global di DynamoDB).
- Mengaktifkan atau menonaktifkan DynamoDB Streams pada tabel (lihat Tangkapan data perubahan DynamoDB Streams).

Example

AWS CLI Contoh berikut menunjukkan cara memodifikasi pengaturan throughput disediakan tabel.

```
aws dynamodb update-table --table-name Music \
    --provisioned-throughput ReadCapacityUnits=20,WriteCapacityUnits=10
```

> **Note**
>
> Ketika Anda mengeluarkan permintaan `UpdateTable`, status tabel berubah dari `AVAILABLE` menjadi `UPDATING`. Tabel tetap sepenuhnya tersedia untuk digunakan saat `UPDATING`. Setelah proses ini selesai, status tabel berubah dari `UPDATING` menjadi `AVAILABLE`.

Example

AWS CLI Contoh berikut menunjukkan cara memodifikasi mode kapasitas baca/tulis tabel ke mode on-demand.

```
aws dynamodb update-table --table-name Music \
    --billing-mode PAY_PER_REQUEST
```

Menghapus tabel

Anda dapat menghapus tabel yang tidak digunakan dengan operasi `DeleteTable`. Menghapus tabel adalah operasi yang tidak dapat dipulihkan. Untuk menghapus tabel menggunakan AWS Management Console, lihatthe section called “Langkah 6: (Opsional) membersihkan”.

Example

AWS CLI Contoh berikut menunjukkan cara menghapus tabel.

```
aws dynamodb delete-table --table-name Music
```

Ketika Anda mengeluarkan permintaan `DeleteTable`, status tabel berubah dari `ACTIVE` menjadi `DELETING`. Mungkin perlu beberapa saat untuk menghapus tabel, bergantung pada sumber daya yang digunakan (seperti data yang disimpan dalam tabel, dan setiap aliran atau indeks pada tabel).

Di akhir operasi `DeleteTable`, tabel tidak akan ada lagi di DynamoDB.

Menggunakan perlindungan penghapusan

Anda dapat melindungi tabel dari penghapusan yang tidak disengaja dengan properti perlindungan penghapusan. Mengaktifkan properti ini untuk tabel membantu memastikan bahwa tabel tersebut tidak terhapus secara tidak sengaja selama operasi manajemen tabel normal administrator. Hal ini juga membantu mencegah gangguan terhadap operasi bisnis normal.

Pemilik tabel atau administrator yang berwenang mengontrol properti perlindungan penghapusan untuk setiap tabel. Properti perlindungan penghapusan untuk setiap tabel dinonaktifkan secara default. Ini termasuk replika global, dan tabel yang dipulihkan dari cadangan. Ketika perlindungan penghapusan dinonaktifkan untuk sebuah tabel, tabel tersebut dapat dihapus oleh pengguna mana pun yang diberi wewenang oleh kebijakan Manajemen Identitas dan Akses (IAM). Ketika perlindungan penghapusan diaktifkan untuk sebuah tabel, tabel tersebut tidak dapat dihapus oleh siapa pun.

Untuk mengubah pengaturan ini, buka Pengaturan tambahan tabel, arahkan ke panel Perlindungan Penghapusan dan pilih Aktifkan perlindungan penghapusan.

Properti proteksi penghapusan didukung oleh konsol DynamoDB, API, CLI/SDK dan. AWS CloudFormation`CreateTable` API mendukung properti proteksi penghapusan pada waktu pembuatan tabel, dan `UpdateTable` API mendukung perubahan properti proteksi penghapusan untuk tabel yang ada.

> **Note**
>
> - Jika AWS akun dihapus, semua data akun tersebut termasuk tabel masih dihapus dalam waktu 90 hari.
> - Jika DynamoDB kehilangan akses ke kunci yang dikelola pelanggan yang digunakan untuk mengenkripsi tabel, DynamoDB akan tetap mengarsipkan tabel tersebut. Pengarsipan melibatkan membuat cadangan tabel dan menghapus aslinya.

Mencantumkan nama tabel

`ListTables`Operasi mengembalikan nama-nama tabel DynamoDB untuk akun AWS saat ini dan Wilayah.

Example

AWS CLI Contoh berikut menunjukkan bagaimana untuk daftar nama tabel DynamoDB.

```
aws dynamodb list-tables
```

Menggambarkan kuota throughput yang disediakan

`DescribeLimits`Operasi mengembalikan kuota kapasitas baca dan tulis saat ini untuk AWS akun saat ini dan Wilayah.

Example

AWS CLI Contoh berikut menunjukkan bagaimana mendeskripsikan kuota throughput yang disediakan saat ini.

```
aws dynamodb describe-limits
```

Output menunjukkan kuota atas unit kapasitas baca dan tulis untuk AWS akun saat ini dan Wilayah.

Untuk informasi selengkapnya tentang kuota ini, dan cara meminta peningkatan kuota, lihat Kuota default throughput.

Pertimbangan saat memilih kelas tabel di DynamoDB

DynamoDB menawarkan dua kelas tabel yang dirancang untuk membantu Anda mengoptimalkan biaya. Kelas tabel Standar DynamoDB adalah defaultnya, dan direkomendasikan untuk sebagian besar beban kerja. Kelas tabel DynamoDB Standard-Infrequent Access (DynamoDB Standard-IA) dioptimalkan untuk tabel yang biayanya didominasi oleh penyimpanan. Misalnya, tabel yang menyimpan data yang jarang diakses, seperti log aplikasi, postingan media sosial lama, riwayat pesanan e-niaga, dan pencapaian game sebelumnya, merupakan kandidat yang baik untuk kelas tabel Standard-IA.

Setiap tabel DynamoDB dikaitkan dengan kelas tabel. Semua indeks sekunder yang terkait dengan tabel menggunakan kelas tabel yang sama. Anda dapat mengatur kelas tabel Anda saat membuat tabel Anda (DynamoDB Standard secara default) dan memperbarui kelas tabel tabel yang ada menggunakan AWS Management Console, AWS CLI, atau SDK. AWS DynamoDB juga mendukung pengelolaan kelas tabel Anda AWS CloudFormation menggunakan tabel satu wilayah (tabel yang bukan tabel global). Setiap kelas tabel menawarkan harga yang berbeda untuk penyimpanan data serta permintaan baca dan tulis. Saat memilih kelas tabel untuk tabel Anda, ingatlah hal berikut:

- Kelas tabel DynamoDB Standard menawarkan biaya throughput yang lebih rendah daripada DynamoDB Standard-IA dan merupakan opsi yang paling hemat biaya untuk tabel di mana throughput adalah biaya dominan.
- Kelas tabel DynamoDB Standard-IA menawarkan biaya penyimpanan yang lebih rendah dibandingkan DynamoDB Standard, dan merupakan pilihan paling hemat biaya untuk tabel di mana penyimpanan merupakan biaya dominan. Ketika penyimpanan melebihi 50% biaya throughput (baca dan tulis) tabel yang menggunakan kelas tabel Standar DynamoDB, kelas tabel IA-Standar DynamoDB dapat membantu Anda mengurangi total biaya tabel.
- Tabel DynamoDB Standard-IA menawarkan performa, ketahanan, dan ketersediaan yang sama dengan tabel DynamoDB Standard.
- Beralih antara kelas tabel DynamoDB Standard dan DynamoDB Standard-IA tidak memerlukan perubahan kode aplikasi Anda. Anda menggunakan APIs DynamoDB dan titik akhir layanan yang sama terlepas dari kelas tabel yang digunakan tabel Anda.
- Tabel DynamoDB Standard-IA kompatibel dengan semua fitur DynamoDB yang ada seperti auto scaling, on-demand mode, (TTL), on-demand backup, recovery (PITR) time-to-live, dan indeks sekunder global. point-in-time

Kelas tabel yang paling hemat biaya untuk tabel Anda bergantung pada penyimpanan yang diharapkan dan pola penggunaan throughput tabel Anda. Anda dapat melihat penyimpanan historis dan biaya throughput dan penggunaan tabel Anda dengan Laporan AWS Biaya dan Penggunaan dan AWS Cost Explorer. Gunakan data historis ini untuk menentukan kelas tabel yang paling hemat biaya untuk tabel Anda. Untuk mempelajari lebih lanjut tentang menggunakan Laporan AWS Biaya dan Penggunaan serta AWS Cost Explorer, lihat Dokumentasi AWS Billing and Cost Management. Lihat Harga Amazon DynamoDB untuk detail tentang harga kelas tabel.

> **Note**
>
> Pembaruan kelas tabel adalah proses latar belakang. Anda masih dapat mengakses tabel Anda secara normal selama pembaruan kelas tabel. Waktu untuk memperbarui kelas tabel Anda bergantung pada lalu lintas tabel, ukuran penyimpanan, dan variabel terkait lainnya. Tidak lebih dari dua pembaruan kelas tabel di tabel Anda diperbolehkan dalam periode akhir 30 hari.

Menambahkan tag dan label ke sumber daya di DynamoDB

Anda dapat memberi label pada sumber daya Amazon DynamoDB menggunakan tag. Tanda memungkinkan Anda mengategorikan sumber daya dengan berbagai cara, misalnya berdasarkan tujuan, pemilik, lingkungan, atau kriteria lainnya. Tanda membantu Anda melakukan hal berikut:

- Identifikasi sumber daya dengan cepat berdasarkan tanda yang Anda tetapkan padanya.
- Lihat AWS tagihan yang dipecah berdasarkan tag.

> **Note**
>
> Setiap indeks sekunder lokal (LSI) dan indeks sekunder global (GSI) yang terkait dengan tabel yang diberi tanda akan diberi label dengan tanda yang sama secara otomatis. Saat ini, penggunaan DynamoDB Streams tidak dapat diberi tag.

Penandaan didukung oleh AWS layanan seperti Amazon EC2, Amazon S3, DynamoDB, dan banyak lagi. Penandaan yang efisien dapat memberikan wawasan biaya dengan memungkinkan Anda membuat laporan di seluruh layanan yang membawa tanda tertentu.

Untuk memulai penandaan, lakukan hal berikut:

1. Pahami Pembatasan penandaan di DynamoDB.
2. Buat tanda menggunakan Penandaan pada sumber daya di DynamoDB.
3. Gunakan Menggunakan tag DynamoDB untuk membuat laporan alokasi biaya untuk melacak AWS biaya Anda per tag aktif.

Terakhir, merupakan praktik yang baik untuk mengikuti strategi penandaan yang optimal. Untuk informasi selengkapnya, lihat Strategi penandaan AWS.

Pembatasan penandaan di DynamoDB

Setiap tanda terdiri dari kunci dan nilai, yang keduanya Anda tentukan. Pembatasan berikut berlaku:

- Setiap tabel DynamoDB hanya dapat memiliki satu tanda dengan kunci yang sama. Jika Anda mencoba menambahkan tanda yang ada (kunci yang sama), nilai tanda yang ada akan diperbarui ke nilai baru.
- Kunci dan nilai tanda peka huruf besar-kecil.
- Panjang kunci maksimum adalah 128 karakter Unicode.
- Panjang nilai maksimum adalah 256 karakter Unicode.
- Karakter yang diperbolehkan adalah huruf, spasi kosong, dan angka, ditambah karakter khusus berikut: `+ - = . _ : /`
- Jumlah maksimum tanda per sumber daya adalah 50.
- Ukuran maksimum yang didukung untuk semua tag dalam tabel adalah 10 KB.
- AWS-Assigned nama tag dan nilai secara otomatis diberi `aws:` awalan, yang tidak dapat Anda tetapkan. AWS-nama tag yang ditetapkan tidak dihitung terhadap batas tag 50 atau batas ukuran maksimum 10 KB. Nama tanda yang ditetapkan pengguna memiliki prefiks `user:` dalam laporan alokasi biaya.
- Anda tidak dapat mengubah penerapan tanda ke versi sebelumnya.

Penandaan pada sumber daya di DynamoDB

Anda dapat menggunakan konsol Amazon DynamoDB atau AWS Command Line Interface AWS CLI() untuk menambahkan, membuat daftar, mengedit, atau menghapus tag. Anda kemudian dapat mengaktifkan tanda yang ditetapkan pengguna ini sehingga mereka muncul di konsol AWS Manajemen Penagihan dan Biaya untuk pelacakan alokasi biaya. Untuk informasi selengkapnya, lihat Menggunakan tag DynamoDB untuk membuat laporan alokasi biaya.

Untuk pengeditan massal, Anda juga dapat menggunakan Editor Tanda di AWS Management Console. Untuk informasi selengkapnya, silakan lihat Bekerja dengan Editor Tag.

Untuk menggunakan DynamoDB API, lihat operasi berikut di Referensi API Amazon DynamoDB:

- TagResource
- UntagResource
- ListTagsOfResource

Topik

- Menetapkan izin untuk memfilter menurut tag
- Menambahkan tanda ke tabel baru atau yang sudah ada (AWS Management Console)
- Menambahkan tanda ke tabel baru atau yang sudah ada (AWS CLI)

Menetapkan izin untuk memfilter menurut tag

Untuk menggunakan tanda guna memfilter daftar tabel Anda di konsol DynamoDB, pastikan kebijakan pengguna Anda mencakup akses ke operasi berikut:

- `tag:GetTagKeys`
- `tag:GetTagValues`

Anda dapat mengakses operasi ini dengan melampirkan kebijakan IAM baru untuk pengguna Anda dengan mengikuti langkah-langkah di bawah ini.

1. Buka konsol IAM dengan pengguna Admin.
2. Pilih "Kebijakan" di menu navigasi kiri.
3. Pilih "Buat kebijakan".
4. Tempelkan kebijakan berikut ke editor JSON.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
```

```
                "tag:GetTagKeys",
                "tag:GetTagValues"
            ],
            "Resource": "*"
        }
    ]
}
```

5. Selesaikan panduan dan tetapkan nama untuk kebijakan (misalnya, `TagKeysAndValuesReadAccess`).
6. Pilih "Pengguna" di menu navigasi kiri.
7. Dari daftar, pilih pengguna yang biasanya Anda gunakan untuk mengakses konsol DynamoDB.
8. Pilih "Tambahkan izin".
9. Pilih "Lampirkan kebijakan yang ada secara langsung".
10. Dari daftar, pilih kebijakan yang Anda buat sebelumnya.
11. Selesaikan panduan.

Menambahkan tanda ke tabel baru atau yang sudah ada (AWS Management Console)

Anda dapat menggunakan konsol DynamoDB untuk menambahkan tanda ke tabel baru saat Anda membuatnya, atau untuk menambahkan, mengedit, atau menghapus tanda untuk tabel yang ada.

Untuk memberi tanda pada sumber daya saat pembuatan (konsol)

1. Masuk ke AWS Management Console dan buka konsol DynamoDB di. https://console.aws.amazon.com/dynamodb/
2. Di panel navigasi, pilih Tabel, lalu pilih Buat tabel.
3. Pada halaman Buat tabel DynamoDB, masukkan nama dan kunci primer. Di bagian Tanda, pilih Tambahkan tanda baru dan masukkan tanda yang ingin Anda gunakan.

   Untuk informasi selengkapnya tentang struktur tag, lihat Pembatasan penandaan di DynamoDB.

   Untuk informasi selengkapnya tentang membuat tabel, lihat Operasi dasar pada tabel DynamoDB.

Untuk memberi tanda pada sumber daya yang ada (konsol)

Buka konsol DynamoDB di. https://console.aws.amazon.com/dynamodb/

1. Di panel navigasi, pilih Tabel.
2. Pilih tabel dalam daftar, lalu pilih tab Pengaturan tambahan. Anda dapat menambahkan, mengedit, atau menghapus tanda di bagian bawah halaman.

Menambahkan tanda ke tabel baru atau yang sudah ada (AWS CLI)

Contoh berikut menunjukkan cara menggunakan tag AWS CLI untuk menentukan saat Anda membuat tabel dan indeks, dan untuk menandai sumber daya yang ada.

Untuk memberi tanda pada sumber daya saat pembuatan (AWS CLI)

- Contoh berikut membuat tabel `Movies` baru dan menambahkan tanda `Owner` dengan nilai `blueTeam`:

```
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
```

Untuk memberi tanda pada sumber daya yang ada (AWS CLI)

- Contoh berikut menambahkan tanda `Owner` dengan nilai `blueTeam` untuk tabel `Movies`:

```
aws dynamodb tag-resource \
    --resource-arn arn:aws:dynamodb:us-east-1:123456789012:table/Movies \
    --tags Key=Owner,Value=blueTeam
```

Untuk mencantumkan semua tanda untuk tabel (AWS CLI)

- Contoh berikut mencantumkan semua tanda yang terkait dengan tabel `Movies`:

```
aws dynamodb list-tags-of-resource \
    --resource-arn arn:aws:dynamodb:us-east-1:123456789012:table/Movies
```

Menggunakan tag DynamoDB untuk membuat laporan alokasi biaya

AWS menggunakan tag untuk mengatur biaya sumber daya pada laporan alokasi biaya Anda. AWS menyediakan dua jenis tag alokasi biaya:

- Tag AWS yang dihasilkan. AWS mendefinisikan, membuat, dan menerapkan tag ini untuk Anda.
- Tanda yang ditentukan pengguna. Anda menentukan, membuat, dan menerapkan tanda ini.

Anda harus mengaktifkan kedua jenis tanda secara terpisah sebelum tanda tersebut muncul di Cost Explorer atau laporan alokasi biaya.

Untuk mengaktifkan tag AWS yang dihasilkan:

1. Masuk ke AWS Management Console dan buka konsol Billing and Cost Management https://console.aws.amazon.com/billing/di rumah#/.
2. Di panel navigasi, pilih Tanda Alokasi Biaya.
3. Di bagian Tanda Alokasi Biaya yang Dihasilkan AWS, pilih Aktifkan.

Untuk mengaktifkan tanda yang ditentukan pengguna:

1. Masuk ke AWS Management Console dan buka konsol Billing and Cost Management https://console.aws.amazon.com/billing/di rumah#/.
2. Di panel navigasi, pilih Tanda Alokasi Biaya.
3. Di bagian Tanda Alokasi Biaya yang Ditentukan Pengguna, pilih Aktifkan.

Setelah Anda membuat dan mengaktifkan tag, AWS buat laporan alokasi biaya dengan penggunaan dan biaya yang dikelompokkan berdasarkan tag aktif Anda. Laporan alokasi biaya mencakup semua AWS biaya Anda untuk setiap periode penagihan. Laporan ini mencakup sumber daya yang diberi tanda dan tidak diberi tanda, sehingga Anda dapat mengatur biaya untuk sumber daya dengan jelas.

> **Note**
>
> Saat ini, data apa pun yang ditransfer dari DynamoDB tidak akan diperinci menurut tanda pada laporan alokasi biaya.

Untuk informasi selengkapnya, lihat Menggunakan tanda alokasi biaya.

Tabel global - Replikasi multi-Wilayah untuk DynamoDB

Tabel global Amazon DynamoDB adalah opsi basis data terkelola penuh, multi-wilayah, dan multi-aktif yang memberikan performa baca dan penulisan performa cepat dan lokal untuk aplikasi global berskala besar.

Tabel global menyediakan solusi terkelola penuh untuk men-deploy basis data multi-Wilayah dan multi-aktif, tanpa harus membangun dan memelihara solusi replikasi Anda sendiri. Anda dapat menentukan AWS Wilayah tempat Anda ingin tabel tersedia dan DynamoDB akan menyebarkan perubahan data yang sedang berlangsung ke semuanya. Tabel global tersedia di semua Wilayah.

Manfaat khusus penggunaan tabel global meliputi:

- Mereplikasi tabel DynamoDB Anda secara otomatis di AWS Wilayah pilihan Anda
- Menghilangkan kesulitan dalam mereplikasi data antarWilayah dan menyelesaikan konflik pembaruan, sehingga Anda dapat fokus pada logika bisnis aplikasi Anda.
- Membantu aplikasi Anda tetap tersedia bahkan ketika terjadi isolasi atau degradasi di seluruh Wilayah.

Tabel global DynamoDB ideal untuk aplikasi berskala besar dengan pengguna yang tersebar secara global. Di lingkungan seperti itu, pengguna mengharapkan performa aplikasi yang sangat cepat. Tabel global menyediakan replikasi multi-aktif otomatis ke AWS Wilayah di seluruh dunia. Tabel global memungkinkan Anda memberikan akses data latensi rendah kepada pengguna Anda di mana pun mereka berada.

Video berikut akan memberi Anda gambaran pengantar tentang tabel global.

Anda dapat mengatur tabel global di AWS Management Console atau AWS CLI. Tabel global menggunakan DynamoDB APIs yang ada, jadi tidak ada perubahan aplikasi yang diperlukan. Anda hanya membayar untuk sumber daya yang disediakan tanpa biaya atau komitmen di muka.

Tabel global untuk replikasi antar-Wilayah

Topik

- Mereplikasi data secara mulus di Wilayah dengan tabel global
- Memberikan keamanan dan akses untuk tabel global Anda dengan AWS KMS
- Tabel global DynamoDB: Cara kerjanya
- Praktik dan persyaratan terbaik untuk mengelola tabel global DynamoDB

- Tutorial: Membuat tabel global
- Memantau tabel global DynamoDB
- Menggunakan IAM dengan tabel global DynamoDB
- Menentukan versi tabel global DynamoDB yang Anda gunakan
- Memutakhirkan tabel global DynamoDB Anda dari versi 2017.11.29 (Legacy) ke versi 2019.11.21 (Saat ini)
- Memahami penagihan Amazon DynamoDB untuk tabel global

Mereplikasi data secara mulus di Wilayah dengan tabel global

Misalkan Anda memiliki basis pelanggan besar yang tersebar di tiga wilayah geografis—Pantai Timur AS, Pantai Barat AS, dan Eropa Barat. Pelanggan tersebut dapat memperbarui informasi profil mereka menggunakan aplikasi Anda. Untuk memenuhi kasus penggunaan ini, Anda perlu membuat tiga tabel DynamoDB identik bernama `CustomerProfiles`, dalam tiga Wilayah AWS berbeda tempat pelanggan berada. Ketiga tabel ini akan sepenuhnya terpisah satu sama lain—perubahan pada data di satu tabel tidak akan terlihat di tabel lainnya. Tanpa solusi replikasi terkelola, Anda harus menulis kode untuk mereplikasi perubahan data. Namun, melakukan hal ini akan memakan waktu dan tenaga.

Daripada menulis kode Anda sendiri, Anda bisa membuat tabel global yang terdiri dari tiga tabel `CustomerProfiles` khusus Wilayah. DynamoDB kemudian akan otomatis mereplikasi perubahan data di antara tabel tersebut sehingga perubahan pada data `CustomerProfiles` di satu Wilayah akan menyebar ke Wilayah lain dengan lancar. Selain itu, jika salah satu AWS Wilayah menjadi tidak tersedia sementara, pelanggan Anda masih dapat mengakses `CustomerProfiles` data yang sama di Wilayah lain.

> **Note**
>
> - Dukungan Wilayah untuk tabel global Versi tabel global 2017.11.29 (Legacy) terbatas untuk AS Timur (Virginia Utara), AS Timur (Ohio), AS Barat (California Utara), AS Barat (Oregon), Eropa (Irlandia), Eropa (London), Eropa (Frankfurt), Asia Pasifik (Singapura), Asia Pasifik (Sydney), Asia Pasifik (Tokyo), dan Asia Pasifik (Seoul).
> - Operasi transaksional memberikan jaminan atomisitas, konsistensi, isolasi, dan daya tahan (ACID) hanya di wilayah tempat penulisan dilakukan pada awalnya. Transaksi tidak didukung di seluruh wilayah dalam tabel global. Misalnya, jika Anda memiliki tabel global dengan replika di wilayah AS Timur (Ohio) dan AS Barat (Oregon) dan melakukan

TransactWriteItems operasi di Wilayah AS Timur (Virginia N.), Anda dapat mengamati transaksi yang diselesaikan sebagian di Wilayah AS Barat (Oregon) saat perubahan direplikasi. Perubahan hanya akan direplikasi ke wilayah lain setelah perubahan telah dilakukan di wilayah sumber.

- Jika Anda menonaktifkan AWS Region, DynamoDB akan menghapus replika ini dari grup replikasi, 20 jam setelah mendeteksi Wilayah sebagai tidak dapat diakses. AWS Replika tidak akan dihapus dan replikasi dari dan ke wilayah ini akan dihentikan. Tabel ini akan dikonversi ke tabel regional.
- Anda harus menunggu 24 jam setelah Anda menambahkan replika baca agar berhasil menghapus tabel sumber. Jika mencoba menghapus tabel dalam 24 jam pertama setelah menambahkan replika baca, Anda akan menerima pesan kesalahan yang menyatakan: "Replika tidak dapat dihapus karena telah bertindak sebagai wilayah sumber untuk replika baru yang ditambahkan dalam tabel dalam 24 jam terakhir".
- Tidak ada dampak performa pada wilayah sumber saat menambahkan replika baru.
- Jika Anda mengubah kapasitas baca dan tulis replika, kapasitas tulis baru akan tercermin ke replika lain yang disinkronkan tetapi kapasitas baca baru tidak.

Untuk informasi tentang ketersediaan dan harga AWS Wilayah, lihat Harga Amazon DynamoDB.

Memberikan keamanan dan akses untuk tabel global Anda dengan AWS KMS

- Anda dapat melakukan AWS KMS operasi pada tabel global Anda dengan menggunakan peran `AWSServiceRoleForDynamoDBReplication` terkait layanan terhadap kunci yang dikelola pelanggan atau yang Kunci yang dikelola AWSdigunakan untuk mengenkripsi replika.
- Jika kunci yang dikelola pelanggan yang digunakan untuk mengenkripsi replika tidak dapat diakses, DynamoDB akan menghapus replika ini dari grup replikasi. Replika tidak akan dihapus dan replikasi dari dan ke wilayah ini akan dihentikan, 20 jam setelah mendeteksi bahwa kunci KMS tidak dapat diakses.
- Jika ingin menonaktifkan kunci yang dikelola pelanggan yang digunakan untuk mengenkripsi tabel replika, Anda harus melakukannya hanya jika kunci tidak lagi digunakan untuk mengenkripsi tabel replika. Setelah mengeluarkan perintah untuk menghapus tabel replika, Anda harus menunggu operasi penghapusan selesai dan tabel global menjadi `Active` sebelum menonaktifkan kunci. Tidak melakukannya dapat mengakibatkan replikasi data parsial dari dan ke tabel replika.

- Jika ingin mengubah atau menghapus kebijakan peran IAM untuk tabel replika, Anda harus melakukannya ketika tabel replika berada dalam status `Active`. Jika tidak, pembuatan, pembaruan, atau penghapusan tabel replika bisa gagal.
- Tabel global dibuat dengan perlindungan penghapusan dinonaktifkan secara default. Meskipun perlindungan penghapusan diaktifkan untuk tabel global, setiap replika tabel tersebut akan dimulai dengan perlindungan penghapusan dinonaktifkan secara default.
- Saat perlindungan penghapusan dinonaktifkan untuk suatu tabel, tabel tersebut dapat terhapus secara tidak sengaja. Saat perlindungan penghapusan diaktifkan untuk suatu tabel, tidak ada yang bisa menghapusnya.
- Mengubah pengaturan perlindungan penghapusan untuk satu tabel replika tidak akan memperbarui replika lain dalam grup.

> **Note**
>
> Kunci yang dikelola pelanggan tidak didukung di Versi tabel global 2017.11.29 (Legacy). Jika Anda ingin menggunakan kunci yang dikelola pelanggan di DynamoDB Global Table, Anda perlu memutakhirkan tabel ke Tabel Global versi 2019.11.21 (Saat Ini) dan kemudian mengaktifkannya.

Tabel global DynamoDB: Cara kerjanya

Bagian berikut menjelaskan konsep dan perilaku tabel global di Amazon DynamoDB.

Konsep tabel global

Tabel global adalah kumpulan dari satu atau lebih tabel replika, semuanya dimiliki oleh satu AWS akun.

Tabel replika (atau replika, untuk singkatnya) adalah tabel DynamoDB tunggal yang berfungsi sebagai bagian dari tabel global. Setiap replika menyimpan set item data yang sama. Setiap tabel global tertentu hanya dapat memiliki satu tabel replika per Wilayah AWS . Untuk informasi selengkapnya tentang cara mulai menggunakan tabel global, lihat Tutorial: Membuat tabel global.

Saat Anda membuat tabel global DynamoDB, tabel tersebut terdiri dari beberapa tabel replika (satu per Wilayah) yang diperlakukan DynamoDB sebagai satu unit. Setiap replika memiliki nama tabel

yang sama dan skema kunci primer yang sama. Saat aplikasi menulis data ke tabel replika di satu Wilayah, DynamoDB menyebarkan penulisan ke tabel replika lainnya di Wilayah lain secara otomatis. AWS

Anda dapat menambahkan tabel replika ke tabel global sehingga dapat tersedia di Wilayah tambahan.

Dengan Versi 2019.11.21 (Terbaru), saat Anda membuat Indeks Sekunder Global di satu Wilayah, indeks tersebut otomatis direplikasi ke Wilayah lain serta di-backfill secara otomatis.

Tugas umum

Tugas umum untuk tabel global berfungsi sebagai berikut.

Anda dapat menghapus tabel replika tabel global sama seperti tabel biasa. Ini akan menghentikan replikasi ke Wilayah itu dan menghapus salinan tabel yang disimpan di Wilayah itu. Anda tidak dapat memutuskan replikasi dan memiliki salinan tabel sebagai entitas independen. Anda dapat menyalin tabel global ke tabel lokal di Wilayah tersebut, lalu menghapus replika global untuk Wilayah tersebut.

> **Note**
>
> Anda tidak akan dapat menghapus tabel sumber hingga setidaknya 24 jam setelah tabel tersebut digunakan untuk memulai Wilayah baru. Jika mencoba menghapusnya terlalu cepat, Anda akan menerima pesan kesalahan.

Konflik dapat terjadi jika aplikasi memperbarui item yang sama di Wilayah yang berbeda pada waktu yang sama. Untuk membantu memastikan konsistensi akhirnya, tabel global DynamoDB menggunakan metode “penulis terakhir menang” untuk merekonsiliasi antara pembaruan yang dilakukan secara bersamaan. Semua replika akan menyetujui pembaruan terkini dan berkumpul menuju status ketika semua replika memiliki data yang identik.

> **Note**
>
> Ada beberapa cara untuk menghindari konflik, antara lain:
>
> - Hanya mengizinkan penulisan ke tabel dalam satu Wilayah.
> - Merutekan lalu lintas pengguna ke Wilayah yang berbeda sesuai dengan kebijakan tulis Anda, untuk memastikan tidak ada konflik.

- Menghindari penggunaan pembaruan non-idempoten seperti Bookmark = Bookmark + 1, dan mendukung pembaruan statis seperti Bookmark=25.
- Perlu diingat bahwa saat Anda merutekan penulisan atau pembacaan ke satu Wilayah, terserah aplikasi Anda untuk memastikan bahwa aliran ditegakkan.

Memantau tabel global

Anda dapat menggunakan CloudWatch untuk mengamati metrik`ReplicationLatency`. CloudWatch melacak waktu yang berlalu antara ketika item ditulis ke tabel replika, dan ketika item tersebut muncul di replika lain di tabel global. Waktu tersebut dinyatakan dalam milidetik dan dipancarkan untuk setiap pasangan Wilayah-sumber dan Wilayah-tujuan. Metrik ini disimpan di Wilayah sumber. Ini adalah satu-satunya CloudWatch metrik yang disediakan oleh Global Tables v2.

Latensi replikasi yang akan Anda alami tergantung pada jarak antara yang Anda pilih Wilayah AWS, serta variabel lainnya. Jika tabel asli Anda berada di Wilayah AS Barat (California Utara) (us-west-1), replika di Wilayah yang lebih dekat, seperti Wilayah AS Barat (Oregon) (us-west-2), akan memiliki latensi replikasi yang lebih rendah dibandingkan dengan replika di Wilayah yang jauh lebih jauh, seperti Afrika (Cape Town) (af-south-1 1) Wilayah.

> **Note**
>
> Latensi replikasi tidak memengaruhi latensi API. Jika Anda memiliki klien dan tabel di Wilayah A dan Anda menambahkan replika tabel global di Wilayah B, klien dan tabel di Wilayah A akan memiliki latensi yang sama seperti sebelum menambahkan Wilayah B. Jika Anda memanggil operasi PutItemAPI di Wilayah B, akhirnya akan tersedia untuk dibaca di Wilayah A setelah penundaan kira-kira `ReplicationLatency` statistik yang tersedia di Amazon. CloudWatch Sebelum direplikasi, Anda akan menerima respons kosong dan setelah direplikasi, Anda akan menerima item; kedua panggilan akan memiliki latensi API yang kira-kira sama.

Waktu Untuk Tayang (TTL)

Anda dapat menggunakan Waktu Untuk Tayang (TTL) untuk menentukan nama atribut yang nilainya menunjukkan waktu kedaluwarsa item tersebut. Nilai ini ditentukan sebagai angka dalam hitungan detik sejak awal jangka waktu Unix. Setelah itu, DynamoDB dapat menghapus item tanpa menimbulkan biaya tulis.

Dengan tabel global, Anda mengonfigurasi TTL di satu Wilayah, dan pengaturan tersebut otomatis direplikasi ke Wilayah lainnya. Ketika sebuah item dihapus melalui aturan TTL, pekerjaan tersebut dilakukan tanpa menggunakan Unit Tulis pada tabel sumber - tetapi tabel target akan dikenakan biaya Unit Tulis yang Direplikasi.

Ketahuilah bahwa jika tabel sumber dan target memiliki kapasitas penulisan Provisioned yang sangat rendah, ini dapat menyebabkan pelambatan, karena penghapusan TTL memerlukan kapasitas tulis.

Aliran dan transaksi dengan tabel global

Setiap tabel global menghasilkan aliran independen berdasarkan semua tulisannya, terlepas dari titik asal untuk penulisan tersebut. Anda dapat memilih untuk menggunakan aliran DynamoDB ini di satu Wilayah atau di semua Wilayah secara independen.

Jika Anda ingin memproses penulisan lokal tetapi tidak direplikasi tulis, Anda dapat menambahkan atribut Region Anda sendiri ke setiap item. Kemudian, Anda dapat menggunakan filter peristiwa Lambda untuk hanya menginvokasi Lambda untuk penulisan di Wilayah lokal.

Operasi transaksional memberikan jaminan ACID (Atomicity, Consistency, Isolation, and Durability) hanya di Wilayah tempat penulisan awalnya dibuat. Transaksi tidak didukung di seluruh Wilayah dalam tabel global.

Misalnya, jika Anda memiliki tabel global dengan replika di Wilayah AS Timur (Ohio) dan AS Barat (Oregon) dan melakukan `TransactWriteItems` operasi di Wilayah AS Timur (Ohio), Anda dapat mengamati transaksi yang diselesaikan sebagian di Wilayah AS Barat (Oregon) saat perubahan direplikasi. Perubahan hanya akan direplikasi ke Wilayah lain setelah dilakukan di Wilayah sumber.

> **Note**
>
> - Tabel global “write-around” DynamoDB Accelerator dengan memperbarui DynamoDB secara langsung. Akibatnya DAX tidak akan menyadari bahwa mereka menyimpan data basi. Cache DAX hanya akan disegarkan ketika TTL cache kedaluwarsa.
> - Tag pada tabel global tidak secara otomatis menyebar.

Throughput baca dan tulis

Tabel global mengelola throughput baca dan tulis dengan cara berikut.

- Kapasitas tulis harus sama di semua instans tabel di seluruh Wilayah.

- Dengan Versi 2019.11.21 (Saat Ini), jika tabel diatur untuk mendukung penskalaan otomatis atau dalam mode sesuai permintaan maka kapasitas tulis secara otomatis tetap sinkron. Artinya, perubahan kapasitas tulis pada satu tabel akan mereplikasi tabel lainnya.
- Kapasitas baca dapat berbeda antarWilayah karena baca mungkin tidak sama. Saat menambahkan replika global ke tabel, kapasitas Wilayah sumber disebarkan. Setelah pembuatan, Anda dapat menyesuaikan kapasitas baca untuk satu replika, dan pengaturan baru ini tidak ditransfer ke sisi lain.

Konsistensi dan resolusi konflik

Setiap perubahan yang dibuat pada item apa pun di tabel replika mana pun akan direplikasi ke semua replika lain dalam tabel global yang sama. Dalam tabel global, item yang baru ditulis biasanya disebarkan ke semua tabel replika dalam hitungan detik.

Dengan tabel global, setiap tabel replika menyimpan set item data yang sama. DynamoDB tidak mendukung replikasi parsial hanya beberapa item.

Aplikasi dapat membaca dan menulis data ke tabel replika mana pun. Jika aplikasi Anda hanya menggunakan pembacaan yang konsisten pada akhirnya dan hanya masalah yang dibaca terhadap satu AWS Wilayah, itu akan berfungsi tanpa modifikasi apa pun. Namun, jika aplikasi Anda membutuhkan bacaan sangat konsisten, aplikasi harus melakukan semua penulisan dan bacaan sangat konsisten di Wilayah yang sama. DynamoDB tidak mendukung pembacaan yang sangat konsisten di seluruh Wilayah. Oleh karena itu, jika Anda menulis ke satu Wilayah dan membaca dari Wilayah lain, respons baca mungkin menyertakan data usang yang tidak mencerminkan hasil penulisan yang baru saja diselesaikan di Wilayah lain.

Jika aplikasi memperbarui item yang sama di Wilayah yang berbeda pada waktu yang sama, konflik dapat terjadi. Untuk membantu memastikan konsistensi akhir, tabel global DynamoDB menggunakan rekonsiliasi penulis terakhir menang antara pembaruan serentak, dengan DynamoDB melakukan upaya terbaik untuk menentukan penulis terakhir. Ini dilakukan di tingkat item. Dengan mekanisme resolusi konflik ini, semua replika akan menyetujui pembaruan terkini dan berkumpul menuju status ketika semua replika memiliki data yang identik.

Ketersediaan dan daya tahan

Jika satu AWS Wilayah menjadi terisolasi atau terdegradasi, aplikasi Anda dapat mengarahkan ulang ke Wilayah lain dan melakukan pembacaan dan penulisan terhadap tabel replika yang berbeda. Anda

dapat menerapkan logika bisnis khusus untuk menentukan kapan harus mengalihkan permintaan ke Wilayah lain.

Jika Region menjadi terisolasi atau terdegradasi, DynamoDB melacak penulisan apa pun yang telah dilakukan tetapi belum disebarkan ke semua tabel replika. Setelah Wilayah kembali online, DynamoDB melanjutkan penyebaran penulisan yang tertunda dari Wilayah tersebut ke tabel replika di Wilayah lain. Ini juga melanjutkan menyebarkan tulisan dari tabel replika lain ke Wilayah yang sekarang kembali online.

Praktik dan persyaratan terbaik untuk mengelola tabel global DynamoDB

Menggunakan tabel global Amazon DynamoDB, Anda dapat mereplikasi data tabel Anda di seluruh Wilayah. AWS Tabel replika dan indeks sekunder di tabel global Anda harus memiliki pengaturan kapasitas tulis yang identik untuk memastikan replikasi data yang tepat.

Untuk kejelasan di masa mendatang, sebaiknya jangan mencantumkan Wilayah dalam nama tabel mana pun yang mungkin suatu saat akan diubah menjadi tabel global.

> **Warning**
>
> Nama tabel untuk setiap tabel global harus unik dalam AWS akun Anda.

Versi tabel global

Untuk menentukan versi tabel global yang Anda gunakan, lihat[Menentukan versi tabel global DynamoDB yang Anda gunakan](#).

Persyaratan untuk Mengelola Kapasitas

Tabel global harus memiliki kapasitas throughput yang dikonfigurasi dengan salah satu dari dua cara berikut:

1. Mode kapasitas sesuai permintaan, diukur dalam unit permintaan tulis yang direplikasi (r) WRUs
2. Mode kapasitas yang disediakan dengan penskalaan otomatis, diukur dalam unit kapasitas tulis yang direplikasi (r) WCUs

Penggunaan mode kapasitas yang disediakan dengan penskalaan otomatis atau mode kapasitas sesuai permintaan membantu memastikan tabel global memiliki kapasitas yang memadai untuk melakukan replikasi tulis ke semua wilayah tabel global.

> **Note**
>
> Beralih dari satu mode kapasitas tabel ke mode kapasitas lainnya di Wilayah mana pun akan mengalihkan mode untuk semua replika.

Men-deploy tabel global

Dalam AWS CloudFormation, setiap tabel global dikendalikan oleh satu tumpukan dalam satu Wilayah. Hal ini terlepas dari jumlah replika. Ketika Anda menerapkan template Anda, CloudFormation akan membuat/memperbarui semua replika sebagai bagian dari operasi tumpukan tunggal. Oleh karena itu, Anda tidak boleh men-deploy sumber daya `AWS::DynamoDB::GlobalTable` yang sama di beberapa Wilayah. Tindakan tersebut tidak didukung dan akan mengakibatkan kesalahan.

Jika men-deploy templat aplikasi di beberapa Wilayah, Anda dapat menggunakan syarat untuk membuat sumber daya hanya di satu Wilayah. Alternatifnya, Anda dapat memilih untuk menentukan sumber daya `AWS::DynamoDB::GlobalTable` Anda dalam tumpukan yang terpisah dari tumpukan aplikasi Anda dan memastikan sumber daya tersebut hanya disebarkan ke satu Wilayah. Untuk informasi selengkapnya lihat tabel Global di CloudFormation

Tabel DynamoDB disebut sebagai `AWS::DynamoDB::Table`, dan tabel global adalah `AWS::DynamoDB::GlobalTable`. CloudFormation Sejauh menyangkut, ini pada dasarnya menjadikan mereka dua sumber daya yang berbeda. Dengan demikian, salah satu pendekatannya adalah membuat semua tabel yang mungkin bersifat global menggunakan konstruksi `GlobalTable`. Anda kemudian dapat menyimpannya sebagai tabel mandiri untuk memulai, dan menambahkannya nanti ke Wilayah jika diperlukan.

Jika Anda memiliki tabel biasa dan Anda ingin mengonversinya saat menggunakan CloudFormation, metode yang disarankan adalah:

1. Menetapkan kebijakan penghapusan untuk dipertahankan.
2. Menghapus tabel dari tumpukan.
3. Mengonversi tabel menjadi Tabel Global di konsol.

4. Mengimpor tabel global sebagai sumber daya baru ke tumpukan.

> **Note**
>
> Replikasi lintas akun tidak didukung saat ini.

Menggunakan tabel global untuk membantu menangani potensi gangguan Wilayah

Memiliki atau dapat dengan cepat membuat salinan independen dari tumpukan eksekusi Anda di Wilayah alternatif, masing-masing mengakses titik akhir DynamoDB lokalnya.

Gunakan Route53 atau rute AWS Global Accelerator ke Wilayah sehat terdekat. Sebagai alternatif, buat klien mengetahui beberapa titik akhir yang mungkin digunakannya.

Gunakan pemeriksaan kondisi di setiap Wilayah yang akan dapat menentukan secara andal apakah ada masalah dengan tumpukan, termasuk apakah DynamoDB mengalami degradasi. Misalnya, jangan hanya melakukan ping bahwa titik akhir DynamoDB sudah aktif. Lakukan panggilan untuk memastikan aliran basis data berhasil sepenuhnya.

Jika pemeriksaan kondisi gagal, lalu lintas dapat merutekan ke Wilayah lain (dengan memperbarui entri DNS dengan Route53, dengan merutekan Akselerator Global secara berbeda, atau dengan meminta klien memilih titik akhir yang berbeda). Tabel global memiliki RPO (sasaran titik pemulihan) yang baik karena data terus disinkronkan dan RTO yang baik (sasaran waktu pemulihan) karena kedua Wilayah selalu menyiapkan tabel untuk lalu lintas baca dan tulis.

Untuk informasi selengkapnya tentang pemeriksaan kondisi, lihat Jenis pemeriksaan kondisi.

> **Note**
>
> DynamoDB adalah layanan inti tempat layanan lain sering membangun operasi bidang kontrolnya, sehingga kecil kemungkinannya Anda akan menghadapi skenario ketika DynamoDB mengalami penurunan layanan di suatu Wilayah sementara layanan lain tidak terkena dampak.

Mencadangkan tabel global

Saat mencadangkan tabel global, pencadangan tabel di satu Wilayah sudah cukup dan mencadangkan semua tabel di semua Wilayah seharusnya tidak diperlukan. Jika tujuannya adalah untuk dapat memulihkan data yang terhapus atau diubah secara keliru, maka PITR di satu Wilayah sudah cukup. Demikian pula, ketika menyimpan snapshot untuk tujuan historis seperti persyaratan peraturan, maka pencadangan di satu Wilayah sudah cukup. Data yang dicadangkan dapat direplikasi ke beberapa Wilayah melalui AWS Backup.

Replika dan menghitung unit tulis

Untuk perencanaan, Anda harus mengambil jumlah penulisan yang akan dilakukan oleh satu Wilayah dan menambahkannya ke jumlah penulisan yang terjadi di setiap Wilayah lainnya. Ini sangat penting karena setiap penulisan yang dilakukan di satu Wilayah juga harus dilakukan di setiap Wilayah replika. Jika Anda tidak memiliki kapasitas yang mamadai untuk menangani semua penulisan, pengecualian kapasitas akan terjadi. Selain itu, waktu tunggu replikasi antarwilayah akan meningkat.

Misalnya, anggaplah Anda mengharapkan 5 aktivitas tulis per detik untuk tabel replika Anda di Ohio, 10 aktivitas tulis per detik untuk tabel replika Anda di Virginia Utara, dan 5 aktivitas tulis per detik untuk tabel replika Anda di Irlandia. Dalam hal ini, Anda harus mengharapkan untuk mengkonsumsi 20 r WCUs atau r WRUs di setiap Wilayah: Ohio, Virginia N., dan Irlandia. Dengan kata lain, Anda harus mengharapkan untuk mengkonsumsi WCUs total 60 r di ketiga Wilayah.

Untuk detail tentang kapasitas yang disediakan dengan penskalaan otomatis dan DynamoDB, lihat Mengelola kapasitas throughput secara otomatis dengan penskalaan otomatis DynamoDB.

> **Note**
>
> Jika tabel berjalan dalam mode kapasitas yang disediakan dengan penskalaan otomatis, kapasitas tulis yang disediakan diperbolehkan untuk mengambang dalam pengaturan penskalaan otomatis tersebut untuk setiap Wilayah.

Tutorial: Membuat tabel global

Bagian ini menjelaskan cara membuat tabel menggunakan konsol Amazon DynamoDB atau AWS Command Line Interface (AWS CLI).

Membuat tabel global (Konsol)

Ikuti langkah-langkah ini untuk membuat tabel global menggunakan AWS Management Console. Contoh berikut membuat tabel global dengan tabel replika di Amerika Serikat dan Eropa.

1. Buka konsol DynamoDB di rumah. https://console.aws.amazon.com/dynamodb/ Untuk contoh ini, pilih Wilayah Timur AS (Ohio).
2. Di panel navigasi di sisi kiri konsol, pilih Tabel.
3. Pilih Buat Tabel.
4. Pada halaman Buat tabel, lakukan hal berikut:

   a. Untuk Nama tabel, masukkan **`Music`**.

   b. Untuk kunci Partisi, masukkan**`Artist`**.

   c. Untuk tombol Sortir, masukkan**`SongTitle`**.

   d. Pertahankan pilihan default String untuk kedua kunci Partition dan Sort key.

   e. Simpan pilihan default lainnya di halaman, lalu pilih Buat tabel.

      Tabel baru ini berfungsi sebagai tabel replika pertama dalam tabel global baru. Ini adalah prototipe untuk tabel replika lain yang Anda tambahkan nanti.
5. Pada halaman Tabel, pilih tabel Musik yang baru dibuat, lalu lakukan hal berikut:

   a. Pilih tab tabel Global, lalu pilih Buat replika.

   b. Dalam daftar tarik-turun Wilayah Replikasi yang Tersedia, pilih US West (Oregon) us-west-2.

      Konsol memeriksa untuk memastikan bahwa tabel dengan nama yang sama tidak ada di Wilayah yang dipilih. Jika ada tabel dengan nama yang sama, Anda harus menghapus tabel yang ada sebelum dapat membuat tabel replika baru di Wilayah tersebut.

   c. Pilih Buat replika. Ini memulai proses pembuatan tabel di Wilayah AS Barat (Oregon) us-west-2.

      Tab tabel Global untuk tabel Musik (dan untuk tabel replika lainnya) menunjukkan bahwa tabel telah direplikasi di beberapa Wilayah.

   d. Tambahkan Wilayah lain sehingga tabel global Anda direplikasi dan disinkronkan di seluruh Amerika Serikat dan Eropa. Untuk melakukan ini, ulangi langkah 5.b, tetapi kali ini, tentukan Eropa (Frankfurt) eu-central-1 alih-alih AS Barat (Oregon) us-west-2.
6. Pastikan Anda masih menggunakan Wilayah AWS Management Console Timur AS (Ohio). Kemudian, lakukan hal berikut:

a. Pilih Jelajahi item tabel.

b. Pilih Buat item.

c. Untuk Artist, masukkan **`item_1`**.

d. Untuk SongTitle, masukkan **`Song Value 1`**.

e. Untuk menyimpan item, pilih Buat item.

7. Setelah beberapa saat, item direplikasi di ketiga Wilayah tabel global Anda. Untuk memverifikasi ini, di pemilih Wilayah di sudut kanan atas di konsol, pilih Eropa (Frankfurt). Tabel Musik di Eropa (Frankfurt) harus berisi item baru.
8. Ulangi langkah 7 dan pilih US West (Oregon) untuk memverifikasi replikasi di Wilayah itu.

Membuat tabel global (AWS CLI)

Ikuti langkah-langkah ini untuk membuat tabel global `Music` menggunakan AWS CLI. Contoh berikut membuat tabel global, dengan tabel replika di Amerika Serikat dan Eropa.

1. Buat tabel baru (`Music`) di AS Timur (Ohio), dengan DynamoDB Streams diaktifkan (`NEW_AND_OLD_IMAGES`).

```
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 \
    --stream-specification StreamEnabled=true,StreamViewType=NEW_AND_OLD_IMAGES \
    --region us-east-2
```

2. Buat tabel `Music` identik di AS Timur (Virginia Utara).

```
aws dynamodb update-table --table-name Music --cli-input-json  \
'{
  "ReplicaUpdates":
  [
    {
      "Create": {
```

```
        "RegionName": "us-east-1"
      }
    }
  ]
}' \
--region=us-east-2
```

3. Ulangi langkah 2 untuk membuat tabel di Eropa (Irlandia) (eu-west-1).
4. Anda dapat melihat daftar replika yang dibuat menggunakan `describe-table`.

```
aws dynamodb describe-table --table-name Music --region us-east-2
```

5. Untuk memverifikasi bahwa replikasi berfungsi, tambahkan item baru ke tabel `Music` di AS Timur (Ohio).

```
aws dynamodb put-item \
    --table-name Music \
    --item '{"Artist": {"S":"item_1"},"SongTitle": {"S":"Song Value 1"}}' \
    --region us-east-2
```

6. Tunggu beberapa detik, lalu periksa untuk melihat apakah item tersebut telah berhasil direplikasi ke AS Timur (Virginia Utara) dan Eropa (Irlandia).

```
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
```

7. Hapus tabel replika di Wilayah Eropa (Irlandia).

```
aws dynamodb update-table --table-name Music --cli-input-json \
'{
  "ReplicaUpdates":
  [
    {
      "Delete": {
```

```
        "RegionName": "eu-west-1"
      }
    }
  ]
}'
```

Membuat tabel global (Java)

Berikut sampel kode java, buat tabel `Music` di wilayah Eropa (Irlandia), lalu buat replika di wilayah Asia Pasifik (Seoul).

```
package com.amazonaws.codesamples.gtv2
import java.util.logging.Logger;
import com.amazonaws.auth.profile.ProfileCredentialsProvider;
import com.amazonaws.regions.Regions;
import com.amazonaws.services.dynamodbv2.AmazonDynamoDB;
import com.amazonaws.services.dynamodbv2.AmazonDynamoDBClientBuilder;
import com.amazonaws.services.dynamodbv2.model.AmazonDynamoDBException;
import com.amazonaws.services.dynamodbv2.model.AttributeDefinition;
import com.amazonaws.services.dynamodbv2.model.BillingMode;
import com.amazonaws.services.dynamodbv2.model.CreateReplicationGroupMemberAction;
import com.amazonaws.services.dynamodbv2.model.CreateTableRequest;
import com.amazonaws.services.dynamodbv2.model.DescribeTableRequest;
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.ProjectionType;
import com.amazonaws.services.dynamodbv2.model.ProvisionedThroughput;
import com.amazonaws.services.dynamodbv2.model.ProvisionedThroughputOverride;
import com.amazonaws.services.dynamodbv2.model.ReplicaGlobalSecondaryIndex;
import com.amazonaws.services.dynamodbv2.model.ReplicationGroupUpdate;
import com.amazonaws.services.dynamodbv2.model.ScalarAttributeType;
import com.amazonaws.services.dynamodbv2.model.StreamSpecification;
import com.amazonaws.services.dynamodbv2.model.StreamViewType;
import com.amazonaws.services.dynamodbv2.model.UpdateTableRequest;
import com.amazonaws.waiters.WaiterParameters;


public class App
{
    private final static Logger LOGGER = Logger.getLogger(Logger.GLOBAL_LOGGER_NAME);
```

```
   public static void main( String[] args )
   {

       String tableName = "Music";
       String indexName = "index1";

       Regions calledRegion = Regions.EU_WEST_1;
       Regions destRegion = Regions.AP_NORTHEAST_2;

       AmazonDynamoDB ddbClient = AmazonDynamoDBClientBuilder.standard()
               .withCredentials(new ProfileCredentialsProvider("default"))
               .withRegion(calledRegion)
               .build();


       LOGGER.info("Creating a regional table - TableName: " + tableName +",
IndexName: " + indexName + " .....");
       ddbClient.createTable(new CreateTableRequest()
               .withTableName(tableName)
               .withAttributeDefinitions(
                       new AttributeDefinition()

.withAttributeName("Artist").withAttributeType(ScalarAttributeType.S),
                       new AttributeDefinition()

.withAttributeName("SongTitle").withAttributeType(ScalarAttributeType.S))
               .withKeySchema(
                       new
KeySchemaElement().withAttributeName("Artist").withKeyType(KeyType.HASH),
                       new
KeySchemaElement().withAttributeName("SongTitle").withKeyType(KeyType.RANGE))
               .withBillingMode(BillingMode.PAY_PER_REQUEST)
               .withGlobalSecondaryIndexes(new GlobalSecondaryIndex()
                       .withIndexName(indexName)
                       .withKeySchema(new KeySchemaElement()
                               .withAttributeName("SongTitle")
                               .withKeyType(KeyType.HASH))
                       .withProjection(new
Projection().withProjectionType(ProjectionType.ALL)))
               .withStreamSpecification(new StreamSpecification()
                       .withStreamEnabled(true)
                       .withStreamViewType(StreamViewType.NEW_AND_OLD_IMAGES)));
```

```
        LOGGER.info("Waiting for ACTIVE table status .....");
        ddbClient.waiters().tableExists().run(new WaiterParameters<>(new
 DescribeTableRequest(tableName)));


        LOGGER.info("Testing parameters for adding a new Replica in " + destRegion +
 " .....");

        CreateReplicationGroupMemberAction createReplicaAction = new
 CreateReplicationGroupMemberAction()
                .withRegionName(destRegion.getName())
                .withGlobalSecondaryIndexes(new ReplicaGlobalSecondaryIndex()
                        .withIndexName(indexName)
                        .withProvisionedThroughputOverride(new
 ProvisionedThroughputOverride()
                                .withReadCapacityUnits(15L)));


        ddbClient.updateTable(new UpdateTableRequest()
                .withTableName(tableName)
                .withReplicaUpdates(new ReplicationGroupUpdate()
                        .withCreate(createReplicaAction.withKMSMasterKeyId(null))));




    }
}
```

Memantau tabel global DynamoDB

Anda dapat menggunakan Amazon CloudWatch untuk memantau perilaku dan kinerja tabel global. Amazon DynamoDB menerbitkan metrik `ReplicationLatency` untuk setiap replika dalam tabel global.

- **`ReplicationLatency`**—Waktu yang berlalu antara saat item ditulis ke tabel replika, dan saat item tersebut muncul di replika lain dalam tabel global. `ReplicationLatency` dinyatakan dalam milidetik dan dipancarkan untuk setiap pasangan Wilayah sumber dan tujuan.

  Selama operasi normal, `ReplicationLatency` akan cukup konstan. Nilai `ReplicationLatency` yang tinggi dapat menunjukkan bahwa pembaruan dari satu replika tidak disebarkan ke tabel replika lain pada waktu yang tepat. Seiring waktu, hal ini dapat mengakibatkan

tabel replika lainnya tertinggal karena tidak lagi menerima pembaruan secara konsisten. Dalam hal ini, Anda harus memverifikasi bahwa unit kapasitas baca (RCUs) dan unit kapasitas tulis (WCUs) identik untuk masing-masing tabel replika. Selain itu, saat memilih pengaturan WCU, ikuti rekomendasi di Versi tabel global.

`ReplicationLatency`dapat meningkat jika suatu AWS Wilayah menjadi terdegradasi dan Anda memiliki tabel replika di Wilayah itu. Dalam hal ini, Anda dapat mengalihkan sementara aktivitas baca dan tulis aplikasi Anda ke AWS Wilayah yang berbeda.

Untuk informasi selengkapnya, lihat Dimensi dan Metrik DynamoDB.

Menggunakan IAM dengan tabel global DynamoDB

Ketika Anda membuat tabel global untuk pertama kalinya, Amazon DynamoDB otomatis membuat peran tertaut layanan AWS Identity and Access Management (IAM) untuk Anda. Peran ini bernama `AWSServiceRoleForDynamoDBReplication`, dan memungkinkan DynamoDB untuk mengelola replikasi lintas-Wilayah untuk tabel global atas nama Anda. Jangan hapus peran tertaut layanan ini. Jika Anda melakukannya, semua tabel global Anda tidak akan lagi berfungsi.

Untuk informasi selengkapnya tentang peran tertaut layanan, lihat Menggunakan peran tertaut layanan di Panduan Pengguna IAM.

Untuk membuat tabel replika di DynamoDB, Anda harus memiliki izin berikut di wilayah sumber.

- `dynamodb:UpdateTable`

Untuk membuat tabel replika di DynamoDB, Anda harus memiliki izin berikut di wilayah tujuan.

- `dynamodb:CreateTable`
- `dynamodb:CreateTableReplica`
- `dynamodb:Scan`
- `dynamodb:Query`
- `dynamodb:UpdateItem`
- `dynamodb:PutItem`
- `dynamodb:GetItem`
- `dynamodb:DeleteItem`
- `dynamodb:BatchWriteItem`

Untuk menghapus tabel replika di DynamoDB, Anda harus memiliki izin berikut di wilayah tujuan.

- `dynamodb:DeleteTable`
- `dynamodb:DeleteTableReplica`

Untuk memperbarui kebijakan penskalaan otomatis replika`UpdateTableReplicaAutoScaling`, Anda harus memiliki izin berikut di semua Wilayah tempat replika tabel ada

- `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`

Untuk menggunakan `UpdateTimeToLive` Anda harus memiliki izin untuk `dynamodb:UpdateTimeToLive` di semua Wilayah tempat replika tabel berada.

> **Important**
>
> the section called “Ketentuan penggunaan”tidak didukung untuk membatasi replikasi tabel global. Menerapkan kondisi kebijakan untuk kontrol akses berbutir halus ke prinsipal layanan DynamoDB atau peran terkait layanan yang digunakan untuk replikasi tabel global dapat mengganggu replikasi dalam tabel global.

Contoh: Menambahkan replika

Kebijakan IAM berikut memberikan izin untuk memungkinkan Anda menambahkan replika ke tabel global.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "VisualEditor0",
            "Effect": "Allow",
            "Action": [
                "dynamodb:CreateTable",
                "dynamodb:DescribeTable",
                "dynamodb:UpdateTable",
                "dynamodb:CreateTableReplica",
                "iam:CreateServiceLinkedRole"
            ],
            "Resource": "*"
        }
    ]
}
```

Contoh: AutoScaling Kebijakan pembaruan

Kebijakan IAM berikut memberikan izin untuk memungkinkan Anda memperbarui kebijakan penskalaan otomatis replika.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "VisualEditor0",
            "Effect": "Allow",
            "Action": [
                "application-autoscaling:RegisterScalableTarget",
                "application-autoscaling:DeleteScheduledAction",
                "application-autoscaling:DescribeScalableTargets",
                "application-autoscaling:DescribeScalingActivities",
                "application-autoscaling:DescribeScalingPolicies",
                "application-autoscaling:PutScalingPolicy",
                "application-autoscaling:DescribeScheduledActions",
                "application-autoscaling:DeleteScalingPolicy",
                "application-autoscaling:PutScheduledAction",
                "application-autoscaling:DeregisterScalableTarget"
            ],
            "Resource": "*"
        }
```

```
    ]
}
```

Contoh: Memungkinkan pembuatan replika untuk wilayah dan nama tabel tertentu

Kebijakan IAM berikut memberikan izin yang memungkinkan pembuatan tabel dan replika untuk tabel `Customers` dengan replika di tiga Wilayah.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "VisualEditor0",
            "Effect": "Allow",
            "Action": [
                "dynamodb:CreateTable",
                "dynamodb:DescribeTable",
                "dynamodb:UpdateTable"
            ],

            "Resource": [
                "arn:aws:dynamodb:us-east-1:123456789012:table/Customers",
                "arn:aws:dynamodb:us-west-1:123456789012:table/Customers",
                "arn:aws:dynamodb:eu-east-2:123456789012:table/Customers"
            ]
        }
    ]
}
```

Menentukan versi tabel global DynamoDB yang Anda gunakan

Ada dua versi tabel global DynamoDB yang tersedia: Tabel Global versi 2019.11.21 (Saat Ini) dan. Versi tabel global 2017.11.29 (Legacy) Kami merekomendasikan menggunakan Tabel Global versi 2019.11.21 (Saat Ini). Ini lebih efisien dan mengonsumsi lebih sedikit kapasitas tulis daripada Versi tabel global 2017.11.29 (Legacy). Keuntungan dari versi terbaru meliputi:

- Tabel sumber dan target dipertahankan bersama-sama dan tetap selaras secara otomatis untuk throughput, pengaturan TTL, pengaturan penskalaan otomatis, dan atribut berguna lainnya.
- Indeks sekunder global juga tetap selaras.
- Anda dapat menambahkan tabel replika baru secara dinamis dari tabel yang diisi dengan data

- Atribut metadata yang diperlukan untuk mengontrol replikasi disembunyikan sehingga membantu mencegah penulisan atribut tersebut yang dapat menyebabkan masalah pada replikasi.
- Versi terbaru mendukung lebih banyak Wilayah daripada versi lama, dan memungkinkan Anda menambahkan atau menghapus Wilayah pada tabel yang ada, sedangkan versi lama tidak.
- Versi Tabel Global 2019.11.21 (Saat Ini) lebih efisien dan mengkonsumsi kapasitas tulis lebih sedikit daripadaVersi tabel global 2017.11.29 (Legacy), dan karenanya lebih hemat biaya. Secara spesifik:
  - Memasukkan item baru di satu Wilayah dan kemudian mereplikasi ke Wilayah lain memerlukan 2 r WCUs per wilayah untuk Versi 2017.11.29 (Warisan), tetapi hanya 1 untuk Versi 2019.11.21 (Saat Ini).
  - Memperbarui item membutuhkan 2 r WCUs di Wilayah sumber dan 1 kemudian RWCu per Wilayah tujuan di Versi 2017.11.29 (Warisan), tetapi hanya 1 RWCu per sumber atau tujuan di Versi 2019.11.21 (Saat Ini).
  - Menghapus item memerlukan 1 rWCu di Wilayah sumber dan kemudian 2 r WCUs per wilayah tujuan di Versi 2017.11.29 (Warisan), tetapi hanya 1 rWCu per sumber atau tujuan di Versi 2019.11.21 (Saat Ini).

  Untuk informasi selengkapnya, lihat Harga Amazon DynamoDB.

Menentukan versi melalui CLI

Untuk mengetahui versi tabel global mana yang Anda gunakanmelalui AWS CLI, periksa `DescribeTable` dan. `DescribeGlobalTable` `DescribeTable`akan menampilkan versi tabel jika itu adalah Versi 2019.11.21 (Saat ini), dan `DescribeGlobalTable` properti akan menampilkan versi tabel jika itu adalah Versi 2017.11.29 (Legacy).

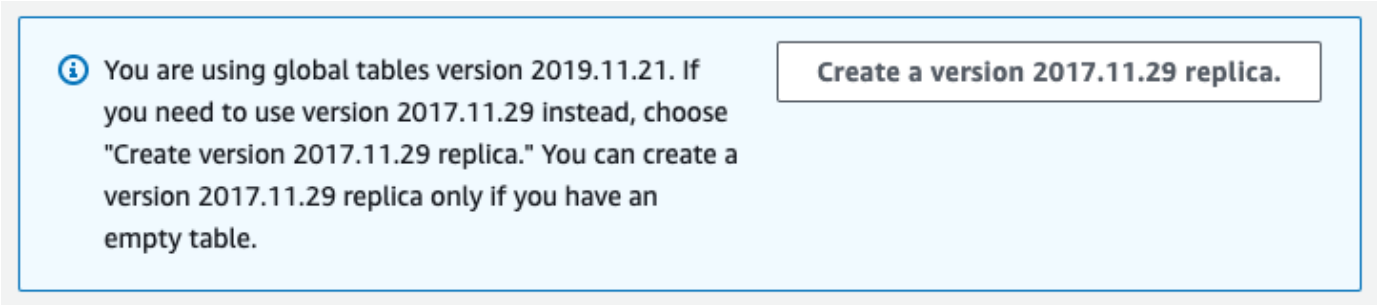
Menentukan versi melalui konsol

Menemukan versi melalui konsol

Untuk mengetahui versi tabel global yang Anda gunakan melalui konsol, lakukan langkah berikut:

1. Buka konsol DynamoDB di rumah. https://console.aws.amazon.com/dynamodb/
2. Di panel navigasi di sisi kiri konsol, pilih Tabel.
3. Pilih tabel yang ingin Anda gunakan.
4. Pilih tab Tabel Global.

Versi tabel global menampilkan versi tabel global yang sedang digunakan:



Untuk meningkatkan dari tabel global Versi 2017.11.29 (Lama) ke Versi 2019.11.21 (Terbaru), ikuti langkah-langkahnya di sini. Proses peningkatan secara keseluruhan akan berjalan tanpa mengganggu tabel langsung, dan akan selesai dalam waktu kurang dari satu jam. Untuk informasi selengkapnya, lihat Memperbarui ke Versi 2019.11.21 (Terbarui)

Note

- Jika pesan Versi tabel global tidak muncul di konsol, artinya ada tabel lain dengan nama yang sama di Wilayah lain. Dalam hal ini, tabel saat ini tidak dapat dibuat menjadi tabel global. Tabel saat ini harus disalin ke tabel baru dengan nama unik, atau semua tabel lain dengan nama yang sama harus dihapus.
- Jika Anda menggunakan Global Tables versi 2019.11.21 (Saat ini) dari tabel global dan Anda juga menggunakan fitur Time to Live, DynamoDB mereplikasi penghapusan TTL ke semua tabel replika. Penghapusan TTL awal tidak mengonsumsi kapasitas tulis di wilayah tempat TTL kedaluwarsa. Namun, penghapusan TTL yang direplikasi untuk tabel replika mengonsumsi unit kapasitas tulis yang direplikasi ketika menggunakan kapasitas yang disediakan, atau kapasitas tulis yang direplikasi ketika menggunakan mode kapasitas sesuai permintaan, di masing-masing wilayah replika dan biaya akan berlaku.
- Dalam Tabel Global versi 2019.11.21 (Saat ini), ketika penghapusan TTL terjadi, itu direplikasi ke semua wilayah replika. Tulisan yang direplikasi ini tidak mengandung properti `type` atau `principalID`. Hal ini dapat menyulitkan untuk membedakan penghapusan TTL dari penghapusan pengguna dalam tabel yang direplikasi.

Memutakhirkan tabel global DynamoDB Anda dari versi 2017.11.29 (Legacy) ke versi 2019.11.21 (Saat ini)

> **Note**
>
> Ada dua versi tabel global DynamoDB yang tersedia: Tabel Global versi 2019.11.21 (Saat Ini) dan. Versi tabel global 2017.11.29 (Legacy) Pelanggan harus menggunakan versi 2019.11.21 (Saat Ini) jika memungkinkan, karena memberikan fleksibilitas yang lebih besar, efisiensi yang lebih tinggi, dan mengkonsumsi kapasitas tulis yang lebih sedikit daripada 2017.11.29 (Legacy). Untuk menentukan versi mana yang Anda gunakan, lihatMenentukan versi tabel global DynamoDB yang Anda gunakan.

Bagian ini menjelaskan cara memutakhirkan tabel global Anda ke versi 2019.11.21 (Saat ini) menggunakan konsol DynamoDB. Memutakhirkan dari versi 2017.11.29 (Legacy) ke versi 2019.11.21 (Saat ini) adalah tindakan satu kali dan Anda tidak dapat membalikkannya. Saat ini, Anda dapat memutakhirkan tabel global hanya menggunakan konsol.

Topik

- Perbedaan perilaku antara versi Legacy dan Current
- Prasyarat peningkatan
- Izin yang diperlukan untuk peningkatan tabel global
- Apa yang diharapkan selama upgrade
- Perilaku DynamoDB Streams sebelum, selama, dan setelah peningkatan
- Memutakhirkan ke versi 2019.11.21 (Saat ini)

Perbedaan perilaku antara versi Legacy dan Current

Daftar berikut menjelaskan perbedaan perilaku antara versi Legacy dan Current dari tabel global.

- versi 2019.11.21 (Saat ini) mengkonsumsi lebih sedikit kapasitas tulis untuk beberapa operasi DynamoDB dibandingkan dengan versi 2017.11.29 (Legacy), dan oleh karena itu, lebih hemat biaya bagi sebagian besar pelanggan. Perbedaan untuk operasi DynamoDB ini adalah sebagai berikut:

- PutItemMemanggil item 1KB di Wilayah dan mereplikasi ke Wilayah lain membutuhkan 2 r WRUs per wilayah untuk 2017.11.29 (Warisan), tetapi hanya 1 rWRU untuk 2019.11.21 (Saat Ini).
- Memanggil UpdateItemitem 1KB membutuhkan 2 r WRUs di Wilayah sumber dan 1 RwRU per Wilayah tujuan untuk 2017.11.29 (Warisan), tetapi hanya 1 rWRU untuk Wilayah sumber dan tujuan untuk 2019.11.21 (Saat Ini).
- Memanggil DeleteItemitem 1KB membutuhkan 1 rWRU di Wilayah sumber dan 2 r WRUs per Wilayah tujuan untuk 2017.11.29 (Warisan), tetapi hanya 1 rWRU untuk Wilayah sumber atau tujuan untuk 2019.11.21 (Saat Ini).

Tabel berikut menunjukkan konsumsi RwRU tabel 2017.11.29 (Legacy) dan 2019.11.21 (Saat Ini) untuk item 1KB di dua Wilayah.

Operasi	2017.11.29 (Warisan)	2019.11.21 (Saat ini)	Penghematan
PutItem	4 r WRUs	2 r WRUs	50%
UpdateItem	3 r WRUs	2 r WRUs	33%
DeleteItem	3 r WRUs	2 r WRUs	33%

- versi 2017.11.29 (Legacy) hanya tersedia di 11. Wilayah AWS Namun, versi 2019.11.21 (Saat ini) tersedia di semua. Wilayah AWS
- Anda membuat tabel global versi 2017.11.29 (Legacy) dengan terlebih dahulu membuat satu set tabel Regional kosong, lalu memanggil CreateGlobalTableAPI untuk membentuk tabel global. Anda membuat tabel global versi 2019.11.21 (Saat ini) dengan menjalankan UpdateTableAPI untuk menambahkan replika ke tabel Regional yang ada.
- versi 2017.11.29 (Legacy) mengharuskan Anda untuk mengosongkan semua replika dalam tabel sebelum menambahkan replika di Wilayah baru (termasuk selama pembuatan). versi 2019.11.21 (Saat ini) mendukung Anda untuk menambah dan menghapus replika ke Wilayah pada tabel yang sudah berisi data.
- versi 2017.11.29 (Legacy) menggunakan kumpulan bidang APIs kontrol khusus berikut untuk mengelola replika:
  - CreateGlobalTable
  - DescribeGlobalTable
  - DescribeGlobalTableSettings

- ListGlobalTables
- UpdateGlobalTable
- UpdateGlobalTableSettings

versi 2019.11.21 (Saat ini) menggunakan DescribeTabledan UpdateTable APIs untuk mengelola replika.

- versi 2017.11.29 (Legacy) menerbitkan dua catatan DynamoDB Streams untuk setiap penulisan. versi 2019.11.21 (Saat ini) hanya menerbitkan satu catatan DynamoDB Streams untuk setiap penulisan.
- versi 2017.11.29 (Legacy) mengisi dan memperbaruiaws:rep:deleting,aws:rep:updateregion, dan aws:rep:updatetime atribut. versi 2019.11.21 (Saat ini) tidak mengisi atau memperbarui atribut ini.
- versi 2017.11.29 (Legacy) tidak menyinkronkan pengaturan di seluruh replika. versi 2019.11.21 (Saat ini) menyinkronkan Menggunakan time to live (TTL) di DynamoDB pengaturan TTL di seluruh replika.
- versi 2017.11.29 (Legacy) tidak mereplikasi penghapusan TTL ke replika lain. versi 2019.11.21 (Saat ini) mereplikasi penghapusan TTL ke semua replika.
- versi 2017.11.29 (Legacy) tidak menyinkronkan pengaturan penskalaan otomatis di seluruh replika. versi 2019.11.21 (Saat ini) menyinkronkan pengaturan penskalaan otomatis di seluruh replika.
- versi 2017.11.29 (Legacy) tidak menyinkronkan pengaturan indeks sekunder global (GSI) di seluruh replika. versi 2019.11.21 (Saat ini) menyinkronkan pengaturan GSI di seluruh replika.
- versi 2017.11.29 (Legacy) tidak menyinkronkan enkripsi pada pengaturan istirahat di seluruh replika. versi 2019.11.21 (Saat ini) menyinkronkan enkripsi pada pengaturan istirahat di seluruh replika.
- versi 2017.11.29 (Legacy) menerbitkan PendingReplicationCount metrik. versi 2019.11.21 (Saat ini) tidak mempublikasikan metrik ini.

Prasyarat peningkatan

Sebelum Anda mulai meningkatkan ke versi 2019.11.21 (Saat ini) tabel global, Anda harus memenuhi prasyarat berikut:

- Menggunakan time to live (TTL) di DynamoDBpengaturan pada replika konsisten di seluruh Wilayah.
- Definisi indeks sekunder global (GSI) pada replika konsisten di seluruh Wilayah.

- Enkripsi pada pengaturan istirahat pada replika konsisten di seluruh Wilayah.
- DynamoDB auto scaling WCUs diaktifkan untuk semua replika, atau mode kapasitas sesuai permintaan diaktifkan untuk semua replika.
- Aplikasi tidak memerlukan kehadiran dari`aws:rep:deleting`,`aws:rep:updateregion`, dan `aws:rep:updatetime` atribut dalam item tabel.

Izin yang diperlukan untuk peningkatan tabel global

Untuk meningkatkan ke versi 2019.11.21 (Saat ini), Anda harus memiliki `dynamodb:UpdateGlobalTableversion` izin di semua Wilayah dengan replika. Izin ini diperlukan selain izin yang diperlukan untuk mengakses konsol DynamoDB dan tabel tampilan.

Kebijakan IAM berikut memberikan izin untuk memutakhirkan tabel global apa pun ke versi 2019.11.21 (Saat Ini).

```
{
    "version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "dynamodb:UpdateGlobalTableversion",
            "Resource": "*"
        }
    ]
}
```

Kebijakan IAM berikut memberikan izin untuk memutakhirkan hanya tabel `Music` global dengan replika di dua Wilayah ke versi 2019.11.21 (Saat Ini).

```
{
    "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"
```

```
            ]
        }
    ]
}
```

Apa yang diharapkan selama upgrade

- Semua replika tabel global akan terus memproses lalu lintas baca dan tulis saat meningkatkan.
- Proses upgrade membutuhkan antara beberapa menit hingga beberapa jam tergantung pada ukuran tabel dan jumlah replika.
- Selama proses upgrade, nilai TableStatusakan berubah dari `ACTIVE` ke`UPDATING`. Anda dapat melihat status tabel dengan menjalankan DescribeTableAPI, atau dengan tampilan Tabel di konsol DynamoDB.
- Penskalaan otomatis tidak akan menyesuaikan pengaturan kapasitas yang disediakan untuk tabel global saat tabel sedang ditingkatkan. Kami sangat menyarankan Anda mengatur tabel ke mode kapasitas sesuai permintaan selama peningkatan.
- Jika Anda memilih untuk menggunakan mode kapasitas yang disediakan dengan penskalaan otomatis selama peningkatan, Anda harus meningkatkan throughput baca dan tulis minimum pada kebijakan Anda untuk mengakomodasi peningkatan lalu lintas yang diharapkan untuk menghindari pembatasan selama peningkatan.
- `ReplicationLatency`Metrik dapat melaporkan lonjakan latensi sementara atau menghentikan pelaporan data metrik selama proses peningkatan. Lihat,the section called “ReplicationLatency”, untuk informasi lebih lanjut.
- Ketika proses upgrade selesai, status tabel Anda akan berubah menjadi`ACTIVE`.

Perilaku DynamoDB Streams sebelum, selama, dan setelah peningkatan

Operasi	Wilayah Replika	Perilaku sebelum upgrade	Perilaku selama peningkatan	Perilaku setelah peningkatan
Masukkan atau Perbarui	Sumber	Populasi stempel waktu terjadi menggunakan. UpdateItem	Populasi stempel waktu terjadi menggunakan. PutItem	Tidak ada stempel waktu yang terlihat pelanggan yang dihasilkan.

Operasi	Wilayah Replika	Perilaku sebelum upgrade	Perilaku selama peningkatan	Perilaku setelah peningkatan
		Dua catatan Streams dihasilkan. Catatan pertama berisi atribut tertulis pelanggan. Catatan kedua berisi `aws:rep:*` atribut.	Dua catatan Streams dihasilkan. Catatan pertama berisi atribut tertulis pelanggan. Catatan kedua berisi `aws:rep:*` atribut.	Catatan Streams tunggal dihasilkan yang berisi atribut yang ditulis pelanggan.
		Dua r WCUs dikonsumsi untuk setiap pelanggan menulis.	Dua r WCUs dikonsumsi untuk setiap pelanggan menulis.	Satu RWCu dikonsumsi untuk setiap pelanggan menulis.
		`ReplicationLatency` dan `PendingReplicationCount` metrik dipublikasikan di CloudWatch.	`ReplicationLatency` dan `PendingReplicationCount` metrik dipublikasikan di CloudWatch.	`ReplicationLatency` metrik diterbitkan di CloudWatch.
	Destinasi	Replikasi terjadi dengan menggunakan PutItem.	Replikasi terjadi dengan menggunakan PutItem.	Replikasi terjadi dengan menggunakan PutItem.

Operasi	Wilayah Replika	Perilaku sebelum upgrade	Perilaku selama peningkatan	Perilaku setelah peningkatan
		Catatan Streams tunggal dihasilkan, yang berisi atribut yang ditulis pelanggan dan atribut. `aws:rep:*`	Catatan Streams tunggal dihasilkan, yang berisi atribut yang ditulis pelanggan dan atribut. `aws:rep:*`	Rekaman Streams tunggal dihasilkan, yang berisi atribut yang ditulis pelanggan saja dan tidak ada atribut replikasi.
		Satu RWCu dikonsumsi jika item tersebut ada di Wilayah tujuan. Dua r WCUs dikonsumsi jika item tidak ada di Wilayah tujuan.	Satu RWCu dikonsumsi jika item tersebut ada di Wilayah tujuan. Dua r WCUs dikonsumsi jika item tidak ada di Wilayah tujuan.	Satu RWCu dikonsumsi untuk setiap pelanggan menulis.
		`ReplicationLatency` dan `PendingReplicationCount` metrik dipublikasikan di CloudWatch.	`ReplicationLatency` dan `PendingReplicationCount` metrik dipublikasikan di CloudWatch.	`ReplicationLatency` metrik diterbitkan di CloudWatch.
Hapus	Sumber	Hapus item apa pun dengan stempel waktu yang lebih kecil menggunakan. DeleteItem	Hapus item apa pun dengan stempel waktu yang lebih kecil menggunakan. DeleteItem	Hapus item apa pun dengan stempel waktu yang lebih kecil menggunakan. DeleteItem

Operasi	Wilayah Replika	Perilaku sebelum upgrade	Perilaku selama peningkatan	Perilaku setelah peningkatan
		Catatan Streams tunggal dihasilkan, yang berisi atribut yang ditulis pelanggan dan atribut. `aws:rep:*`	Catatan Streams tunggal dihasilkan, yang berisi atribut yang ditulis pelanggan dan atribut. `aws:rep:*`	Rekaman Streams tunggal dihasilkan, yang berisi atribut yang ditulis pelanggan.
		Satu RWCu dikonsumsi untuk setiap penghapusan pelanggan.	Satu RWCu dikonsumsi untuk setiap penghapusan pelanggan.	Satu RWCu dikonsumsi untuk setiap penghapusan pelanggan.
		`ReplicationLatency` dan `PendingReplicationCount` metrik dipublikasikan di CloudWatch.	`ReplicationLatency` dan `PendingReplicationCount` metrik dipublikasikan di CloudWatch.	`ReplicationLatency` metrik diterbitkan di CloudWatch.
	Destinasi	Penghapusan dua fase terjadi: • Di Fase 1, UpdateItem atur bendera penghapusan. • Di Fase 2, DeleteItem menghapus item.	Menghapus item menggunakan DeleteItem.	Menghapus item menggunakan DeleteItem.

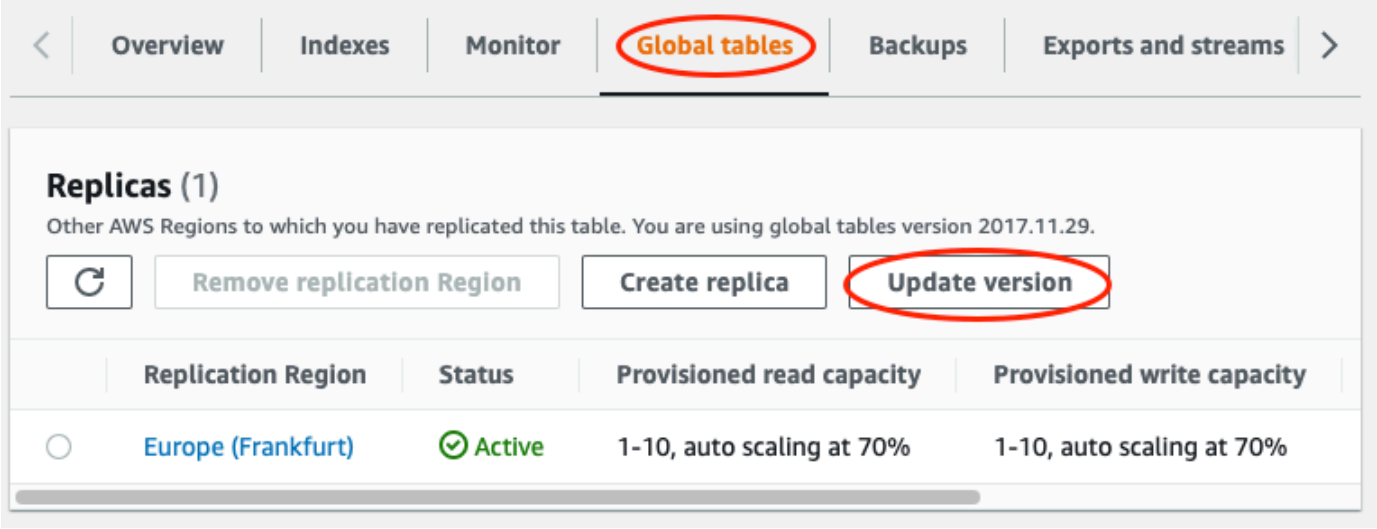
Operasi	Wilayah Replika	Perilaku sebelum upgrade	Perilaku selama peningkatan	Perilaku setelah peningkatan
		Dua catatan Streams dihasilkan. Catatan pertama berisi perubahan ke `aws:rep:deleting` bidang. Catatan kedua berisi atribut yang ditulis pelanggan dan atribut. `aws:rep:*`	Rekaman Stream tunggal dihasilkan, yang berisi atribut yang ditulis pelanggan.	Rekaman Stream tunggal dihasilkan, yang berisi atribut yang ditulis pelanggan.
		Dua r WCUs dikonsumsi untuk setiap penghapusan pelanggan.	Satu RWCu dikonsumsi untuk setiap penghapusan pelanggan.	Satu RWCu dikonsumsi untuk setiap penghapusan pelanggan.
		`ReplicationLatency` dan `PendingReplicationCount` metrik dipublikasikan di CloudWatch.	`ReplicationLatency` metrik diterbitkan di CloudWatch.	`ReplicationLatency` metrik diterbitkan di CloudWatch.

Memutakhirkan ke versi 2019.11.21 (Saat ini)

Lakukan langkah-langkah berikut untuk memutakhirkan versi tabel global DynamoDB Anda menggunakan. AWS Management Console

Untuk meningkatkan tabel global ke versi 2019.11.21 (Saat ini)

1. Buka konsol DynamoDB di rumah. https://console.aws.amazon.com/dynamodb/
2. Di panel navigasi di sisi kiri konsol, pilih Tabel, lalu pilih tabel global yang ingin Anda tingkatkan ke versi 2019.11.21 (Saat Ini).
3. Pilih tab Tabel Global.
4. Pilih Perbarui versi.



5. Baca dan setujui persyaratan baru, lalu pilih Perbarui versi.
6. Setelah proses pemutakhiran selesai, versi tabel global yang muncul di konsol berubah menjadi 2019.11.21.

Memahami penagihan Amazon DynamoDB untuk tabel global

Panduan ini menjelaskan cara kerja penagihan DynamoDB untuk tabel global, mengidentifikasi komponen yang berkontribusi pada biaya tabel global, termasuk contoh praktis.

Tabel global Amazon DynamoDB adalah database yang dikelola sepenuhnya, tanpa server, Multi-wilayah, dan multi-aktif. Tabel global dirancang untuk ketersediaan 99,999%, memberikan peningkatan ketahanan aplikasi, dan kelangsungan bisnis yang lebih baik. Tabel global mereplikasi tabel DynamoDB Anda secara otomatis di seluruh wilayah pilihan AWS Anda sehingga Anda dapat mencapai kinerja baca dan tulis lokal yang cepat.

Cara kerjanya

Model penagihan untuk tabel global berbeda dari tabel DynamoDB wilayah tunggal. Operasi tulis untuk tabel DynamoDB wilayah tunggal ditagih menggunakan unit berikut:

- Tulis Unit Permintaan (WRUs) untuk mode kapasitas sesuai permintaan, di mana satu WRU dikenakan biaya untuk setiap penulisan hingga 1KB
- Tulis Unit Kapasitas (WCUs) untuk mode kapasitas yang disediakan, di mana satu WCU menyediakan satu penulisan per detik hingga 1 KB

Saat Anda membuat tabel global dengan menambahkan tabel replika ke tabel Single-region yang ada, tabel Single-region itu menjadi tabel replika, yang berarti unit yang digunakan untuk menagih penulisan ke tabel juga berubah. Operasi tulis ke tabel replika ditagih menggunakan unit berikut:

- Unit Permintaan Tulis Replikasi (rWRUs) untuk mode kapasitas sesuai permintaan, di mana satu RWRU per tabel replika dikenakan biaya untuk setiap penulisan hingga 1KB
- Unit Kapasitas Tulis Replikasi (rWCUs) untuk mode kapasitas yang disediakan, di mana satu WCU per tabel replika menyediakan satu penulisan per detik hingga 1 KB

Pembaruan untuk Global Secondary Indexes (GSIs) ditagih menggunakan unit yang sama dengan tabel DynamoDB wilayah Tunggal, bahkan jika tabel dasar untuk GSI adalah tabel replika. Operasi pembaruan untuk GSIs ditagih menggunakan unit berikut:

- Tulis Unit Permintaan (WRUs) untuk mode kapasitas sesuai permintaan, di mana satu WRU dikenakan biaya untuk setiap penulisan hingga 1KB
- Tulis Unit Kapasitas (WCUs) untuk mode kapasitas yang disediakan, di mana satu WCU menyediakan satu penulisan per detik hingga 1 KB

Unit tulis yang direplikasi (r WCUs dan rWRUs) dihargai sama dengan unit tulis wilayah tunggal (WCUs dan). WRUs Biaya transfer data Lintas Wilayah berlaku untuk tabel global karena data direplikasi di seluruh Wilayah. Biaya tulis yang direplikasi (RWCu atau RwRU) terjadi di setiap Wilayah yang berisi tabel replika untuk tabel global.

Membaca operasi dari tabel Single-region dan dari tabel replika menggunakan unit berikut:

- Baca Unit Permintaan (RRUs) untuk mode kapasitas sesuai permintaan, di mana satu RRU dibebankan untuk setiap pembacaan yang sangat konsisten hingga 4KB

- Baca Unit Kapasitas (RCUs) untuk tabel yang disediakan, di mana satu RCU menyediakan satu pembacaan per detik yang sangat konsisten hingga 4KB

Contoh penagihan tabel global DynamoDB

Mari kita telusuri skenario contoh multi-hari untuk melihat cara kerja penagihan permintaan tulis tabel global dalam praktiknya (perhatikan bahwa contoh ini hanya mempertimbangkan permintaan tulis, dan tidak termasuk pengembalian tabel dan biaya transfer data lintas wilayah yang akan dikeluarkan dalam contoh):

Hari 1 - Tabel Wilayah Tunggal: Anda memiliki tabel DynamoDB sesuai permintaan wilayah tunggal bernama Table_A di Wilayah us-west-2. Anda menulis 100 item 1KB ke Table_A. Untuk operasi penulisan satu wilayah ini, Anda dikenakan biaya 1 unit permintaan tulis (WRU) per 1KB tertulis. Biaya hari 1 Anda adalah:

- 100 WRUs di Wilayah us-west-2 untuk wilayah Tunggal menulis

Total unit permintaan dibebankan pada hari 1: 100 WRUs.

Hari 2 - Membuat tabel global: Anda membuat tabel global dengan menambahkan replika ke Table_A di Wilayah us-east-2. Table_A sekarang menjadi tabel global dengan dua tabel replika; satu di Wilayah us-west-2, dan satu di Wilayah us-east-2. Anda menulis 150 item 1KB ke tabel replika di Wilayah us-west-2. Biaya hari 2 Anda adalah:

- 150 r WRUs di Wilayah us-west-2 untuk penulisan yang direplikasi
- 150 r WRUs di Wilayah us-east-2 untuk penulisan yang direplikasi

Total unit permintaan dibebankan pada hari ke 2: 300 r WRUs.

Hari 3 - Menambahkan Indeks Sekunder Global: Anda menambahkan indeks sekunder global (GSI) ke tabel replika di Wilayah us-east-2 yang memproyeksikan semua atribut dari tabel dasar (replika). Tabel global secara otomatis membuat GSI pada tabel replika di Wilayah us-west-2 untuk Anda. Anda menulis 200 catatan 1KB baru ke tabel replika di Wilayah us-west-2. Biaya hari 3 Anda adalah:

- • 200 r WRUs di Wilayah us-west-2 untuk penulisan yang direplikasi
- • 200 WRUs di Wilayah us-west-2 untuk pembaruan GSI
- • 200 r WRUs di Wilayah us-east-2 untuk penulisan yang direplikasi

- • 200 WRUs di Wilayah us-east-2 untuk pembaruan GSI

Total unit permintaan tulis dibebankan pada hari ke 3: 400 WRUs dan 400 r WRUs.

Total biaya unit tulis untuk ketiga hari adalah 500 WRUs (100 WRU pada hari 1 + 400 WRUs pada hari ke 3) dan 700 r WRUs (300 r WRUs pada Hari2 + 400 r WRUs pada Hari 3).

Singkatnya, operasi penulisan tabel replika ditagih dalam unit tulis yang direplikasi di semua Wilayah yang berisi tabel replika. Jika Anda memiliki indeks sekunder global, Anda dikenakan biaya unit tulis untuk pembaruan GSIs di semua wilayah yang berisi GSI (yang dalam tabel global adalah semua Wilayah yang berisi tabel replika).

Bekerja dengan item dan atribut di DynamoDB

Di Amazon DynamoDB, item adalah koleksi atribut. Setiap atribut mempunyai nama dan nilai. Nilai atribut dapat berupa skalar, himpunan, atau jenis dokumen. Untuk informasi selengkapnya, lihat Amazon DynamoDB: Cara kerjanya.

DynamoDB menyediakan empat operasi untuk fungsionalitas dasar buat, baca, perbarui, dan hapus (CRUD). Semua operasi ini bersifat atom.

- `PutItem` — Membuat item.
- `GetItem` — Membaca item.
- `UpdateItem` — Memperbarui item.
- `DeleteItem` — Menghapus item.

Setiap operasi ini mengharuskan Anda menentukan kunci primer item yang ingin Anda gunakan. Misalnya, untuk membaca item menggunakan `GetItem`, Anda harus menentukan kunci partisi dan kunci urutan (jika ada) untuk item tersebut.

Selain empat operasi CRUD dasar, DynamoDB juga menyediakan hal berikut ini:

- `BatchGetItem` — Membaca hingga 100 item dari satu tabel atau lebih.
- `BatchWriteItem` — Membuat atau menghapus hingga 25 item dalam satu atau beberapa tabel.

Operasi batch ini menggabungkan beberapa operasi CRUD ke dalam satu permintaan. Selain itu, operasi batch membaca dan menulis item secara paralel untuk meminimalkan latensi respons.

Bagian ini menjelaskan cara menggunakan operasi ini dan mencakup topik terkait, seperti pembaruan bersyarat dan penghitung atom. Bagian ini juga mencakup contoh kode yang menggunakan file AWS SDKs.

Topik

- Ukuran dan format item DynamoDB
- Membaca item
- Menulis item
- Nilai yang dikembalikan
- Operasi batch
- Penghitung atom
- Penulisan bersyarat
- Menggunakan ekspresi di DynamoDB
- Menggunakan time to live (TTL) di DynamoDB
- Menanyakan tabel di DynamoDB
- Memindai tabel di DynamoDB
- PartiQL - Bahasa kueri yang kompatibel dengan SQL untuk Amazon DynamoDB
- Bekerja dengan item: Java
- Bekerja dengan item: .NET

Ukuran dan format item DynamoDB

Tabel DynamoDB tidak memiliki skema, kecuali kunci primer, sehingga semua item dalam tabel dapat memiliki atribut, ukuran, dan jenis data yang berbeda.

Ukuran total suatu item adalah jumlah panjang nama dan nilai atributnya, ditambah overhead yang berlaku seperti dijelaskan di bawah. Anda dapat menggunakan panduan berikut untuk memperkirakan ukuran atribut:

- String adalah Unicode dengan pengodean biner UTF-8. Ukuran string adalah (jumlah byte UTF-8 yang dikodekan dari nama atribut) + (jumlah byte yang dikodekan UTF-8).
- Angka memiliki panjang yang bervariasi, hingga 38 angka penting. Angka nol di depan dan di belakang dipangkas. Ukuran angka kira-kira (jumlah byte yang dikodekan UTF-8 dari nama atribut) + (1 byte per dua digit signifikan) + (1 byte).

- Nilai biner harus dikodekan dalam format base64 sebelum dapat dikirim ke DynamoDB, namun panjang byte mentah nilai tersebut digunakan untuk menghitung ukuran. Ukuran atribut biner adalah (jumlah byte UTF-8 yang dikodekan dari nama atribut) + (jumlah byte mentah).
- Ukuran atribut null atau atribut Boolean adalah (jumlah byte UTF-8 yang dikodekan dari nama atribut) + (1 byte).
- Atribut jenis `List` atau `Map` memerlukan overhead 3 byte, apa pun kontennya. Ukuran a `List` or `Map` is (jumlah byte UTF-8 yang dikodekan dari nama atribut) + jumlah (ukuran elemen bersarang) + (3 byte). Ukuran kosong `List` atau `Map` is (jumlah byte UTF-8 yang dikodekan dari nama atribut) + (3 byte).
- Masing-masing elemen `List` atau `Map` juga membutuhkan 1 byte overhead.

> **Note**
>
> Sebaiknya Anda memilih nama atribut yang lebih pendek. Ini membantu Anda mengurangi jumlah penyimpanan yang diperlukan, tetapi juga dapat menurunkan jumlah RCU/yang WCUs Anda gunakan.

Untuk tujuan penagihan penyimpanan, setiap item mencakup overhead penyimpanan per item yang bergantung pada fitur yang telah Anda aktifkan.

- Semua item di DynamoDB memerlukan overhead penyimpanan sebesar 100 byte untuk pengindeksan.
- Beberapa fitur DynamoDB (tabel global, transaksi, perubahan pengambilan data untuk Kinesis Data Streams dengan DynamoDB) memerlukan overhead penyimpanan tambahan untuk memperhitungkan atribut yang dibuat sistem akibat pengaktifan fitur tersebut. Misalnya, tabel global memerlukan tambahan overhead penyimpanan sebesar 48 byte.

Membaca item

Untuk membaca item dari tabel DynamoDB, gunakan operasi `GetItem`. Anda harus memberikan nama tabel, beserta kunci primer item yang Anda inginkan.

Example

AWS CLI Contoh berikut menunjukkan cara membaca item dari `ProductCatalog` tabel.

```
aws dynamodb get-item \
    --table-name ProductCatalog \
    --key '{"Id":{"N":"1"}}'
```

> **Note**
>
> Dengan `GetItem`, Anda harus menentukan seluruh kunci primer, bukan hanya sebagian darinya. Misalnya, jika tabel memiliki kunci primer gabungan (kunci partisi dan kunci urutan), Anda harus menyediakan nilai untuk kunci partisi dan nilai untuk kunci urutan.

Pada akhirnya, permintaan `GetItem` melakukan pembacaan konsisten secara default. Sebagai gantinya, Anda dapat menggunakan parameter `ConsistentRead` untuk meminta bacaan sangat konsisten. (Ini mengkonsumsi unit kapasitas baca tambahan, tetapi mengembalikan sebagian besar up-to-date versi item.)

`GetItem` mengembalikan semua atribut item. Anda dapat menggunakan ekspresi proyeksi untuk mengembalikan sebagian atribut saja. Untuk informasi selengkapnya, lihat Menggunakan ekspresi proyeksi di DynamoDB.

Untuk mengembalikan jumlah unit kapasitas baca yang dikonsumsi oleh `GetItem`, tetapkan parameter `ReturnConsumedCapacity` menjadi `TOTAL`.

Example

Berikut AWS Command Line Interface (AWS CLI) contoh menunjukkan beberapa `GetItem` parameter opsional.

```
aws dynamodb get-item \
    --table-name ProductCatalog \
    --key '{"Id":{"N":"1"}}' \
    --consistent-read \
    --projection-expression "Description, Price, RelatedItems" \
    --return-consumed-capacity TOTAL
```

Menulis item

Untuk membuat, memperbarui, atau menghapus item dalam tabel DynamoDB, gunakan salah satu operasi berikut:

- `PutItem`
- `UpdateItem`
- `DeleteItem`

Untuk setiap operasi ini, Anda harus menentukan seluruh kunci primer, bukan hanya sebagian saja. Misalnya, jika tabel memiliki kunci primer gabungan (kunci partisi dan kunci urutan), Anda harus memberikan nilai untuk kunci partisi dan nilai untuk kunci urutan.

Untuk mengembalikan jumlah unit kapasitas tulis yang digunakan oleh salah satu operasi ini, atur parameter `ReturnConsumedCapacity` ke salah satu dari berikut ini:

- `TOTAL` — Mengembalikan jumlah total unit kapasitas tulis yang dikonsumsi.
- `INDEXES` — Mengembalikan jumlah total unit kapasitas tulis yang digunakan, dengan subtotal untuk tabel dan indeks sekunder apa pun yang terpengaruh oleh operasi.
- `NONE` — Tidak ada detail kapasitas tulis dikembalikan. (Ini adalah pengaturan default.)

PutItem

`PutItem` membuat item baru. Jika item dengan kunci yang sama sudah ada di tabel, item tersebut akan diganti dengan item baru.

Example

Tulis item baru ke tabel `Thread`. Kunci primer untuk `Thread` terdiri dari `ForumName` (kunci partisi) dan `Subject` (kunci urutan).

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

Argumen untuk `--item` disimpan dalam file `item.json`.

```
{
    "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

Jika item dengan kunci tertentu tidak ada, `UpdateItem` akan membuat item baru. Jika tidak, ini akan mengubah atribut item yang sudah ada.

Anda menggunakan ekspresi pembaruan untuk menentukan atribut yang ingin Anda ubah dan nilai barunya. Untuk informasi selengkapnya, lihat Menggunakan ekspresi pembaruan di DynamoDB.

Dalam ekspresi pembaruan, Anda menggunakan nilai atribut ekspresi sebagai pengganti nilai aktual. Untuk informasi selengkapnya, lihat Menggunakan nilai atribut ekspresi di DynamoDB.

Example

Modifikasi berbagai atribut dalam item `Thread`. Parameter `ReturnValues` opsional menunjukkan item seperti yang muncul setelah pembaruan. Untuk informasi selengkapnya, lihat Nilai yang dikembalikan.

```
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
```

Argumen untuk `--key` disimpan dalam file `key.json`.

```
{
    "ForumName": {"S": "Amazon DynamoDB"},
    "Subject": {"S": "New discussion thread"}
}
```

Argumen untuk `--expression-attribute-values` disimpan dalam file `expression-attribute-values.json`.

```
{
    ":zero": {"N":"0"},
    ":lastpostedby": {"S":"barney@example.com"}
}
```

DeleteItem

`DeleteItem` menghapus item dengan kunci tertentu.

Example

AWS CLI Contoh berikut menunjukkan cara menghapus `Thread` item.

```
aws dynamodb delete-item \
    --table-name Thread \
    --key file://key.json
```

Nilai yang dikembalikan

Dalam beberapa kasus, Anda mungkin ingin DynamoDB mengembalikan nilai-nilai atribut tertentu seperti nilai yang muncul sebelum atau setelah Anda memodifikasi mereka. Operasi `PutItem`, `UpdateItem`, dan `DeleteItem` memiliki parameter `ReturnValues` yang dapat Anda gunakan untuk mengembalikan nilai atribut sebelum atau setelah dimodifikasi.

Nilai default untuk `ReturnValues` adalah `NONE`, yang berarti DynamoDB tidak mengembalikan informasi apa pun tentang atribut yang dimodifikasi.

Berikut ini adalah pengaturan valid lainnya untuk `ReturnValues`, yang diatur oleh operasi DynamoDB API.

PutItem

- `ReturnValues`: `ALL_OLD`
  - Jika Anda menimpa item yang ada, `ALL_OLD` akan mengembalikan seluruh item seperti item yang muncul sebelum penimpaan.
  - Jika Anda menulis item yang tidak ada, `ALL_OLD` tidak akan berpengaruh.

UpdateItem

Penggunaan yang paling umum untuk `UpdateItem` adalah untuk memperbarui item yang ada. Namun, `UpdateItem` benar-benar melakukan upsert, yang berarti bahwa ini secara otomatis membuat item jika sudah tidak ada.

- `ReturnValues`: `ALL_OLD`

- Jika Anda memperbarui item yang ada, `ALL_OLD` akan mengembalikan seluruh item seperti item yang muncul sebelum pembaruan.
- Jika Anda memperbarui item yang tidak ada (upsert), `ALL_OLD` tidak akan berpengaruh.

- `ReturnValues`: `ALL_NEW`
  - Jika Anda memperbarui item yang ada, `ALL_NEW` menampilkan seluruh item seperti item yang muncul setelah pembaruan.
  - Jika Anda memperbarui item tidak ada (upsert), `ALL_NEW` akan mengembalikan seluruh item.
- `ReturnValues`: `UPDATED_OLD`
  - Jika Anda memperbarui item yang ada, `UPDATED_OLD` hanya mengembalikan atribut yang diperbarui seperti atribut yang muncul sebelum pembaruan.
  - Jika Anda memperbarui item yang tidak ada (upsert), `UPDATED_OLD` tidak akan berpengaruh.
- `ReturnValues`: `UPDATED_NEW`
  - Jika Anda memperbarui item yang ada, `UPDATED_NEW` hanya menampilkan atribut yang terpengaruh, seperti yang muncul setelah pembaruan.
  - Jika Anda memperbarui item tidak ada (upsert), `UPDATED_NEW` hanya mengembalikan atribut diperbarui, seperti yang muncul setelah pembaruan.

DeleteItem

- `ReturnValues`: `ALL_OLD`
  - Jika Anda menghapus item yang ada, `ALL_OLD` akan mengembalikan seluruh item seperti yang muncul sebelum Anda menghapusnya.
  - Jika Anda menghapus item yang tidak ada, `ALL_OLD` tidak akan menampilkan data apa pun.

Operasi batch

Untuk aplikasi yang memerlukan membaca atau menulis beberapa item, DynamoDB menyediakan operasi `BatchGetItem` dan `BatchWriteItem`. Dengan menggunakan operasi ini, Anda dapat mengurangi jumlah perjalanan bolak-balik jaringan dari aplikasi Anda ke DynamoDB. Selain itu, DynamoDB melakukan operasi baca atau tulis individu secara paralel. Aplikasi Anda mendapatkan keuntungan dari paralelisme ini tanpa perlu mengelola konkurensi atau threading.

Operasi batch pada dasarnya merupakan pembungkus di sekitar beberapa permintaan baca atau tulis. Misalnya, jika permintaan `BatchGetItem` berisi lima item, DynamoDB melakukan lima operasi

`GetItem` atas nama Anda. Demikian pula, jika permintaan `BatchWriteItem` berisi dua permintaan put dan empat permintaan penghapusan, DynamoDB akan melakukan dua permintaan `PutItem` dan empat `DeleteItem`.

Secara umum, operasi batch tidak gagal kecuali semua permintaan dalam batch mengalami kegagalan. Misalnya, bayangkan Anda melakukan operasi `BatchGetItem`, tetapi salah satu permintaan `GetItem` individu dalam batch mengalami kegagalan. Dalam kasus ini, `BatchGetItem` mengembalikan kunci dan data dari permintaan `GetItem` yang gagal. Permintaan `GetItem` lainnya dalam batch tidak akan terpengaruh.

BatchGetItem

Satu operasi `BatchGetItem` dapat berisi hingga 100 permintaan `GetItem` individu dapat mengambil data hingga 16 MB. Selain itu, operasi `BatchGetItem` dapat mengambil item dari beberapa tabel.

Example

Ambil dua item dari tabel `Thread`, menggunakan ekspresi proyeksi untuk mengembalikan beberapa atribut saja.

```
aws dynamodb batch-get-item \
    --request-items file://request-items.json
```

Argumen untuk `--request-items` disimpan dalam file `request-items.json`.

```
{
    "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

Operasi `BatchWriteItem` dapat berisi hingga 25 permintaan `PutItem` dan `DeleteItem` individu, serta dapat menulis data hingga 16 MB. (Ukuran maksimum item individu adalah sebesar 400 KB.) Selain itu, operasi `BatchWriteItem` dapat menempatkan atau menghapus item dalam beberapa tabel.

> **Note**
>
> `BatchWriteItem` tidak mendukung permintaan `UpdateItem`.

Example

Tulis dua item ke tabel `ProductCatalog`.

```
aws dynamodb batch-write-item \
    --request-items file://request-items.json
```

Argumen untuk `--request-items` disimpan dalam file `request-items.json`.

```
{
    "ProductCatalog": [
        {
            "PutRequest": {
                "Item": {
                    "Id": { "N": "601" },
                    "Description": { "S": "Snowboard" },
                    "QuantityOnHand": { "N": "5" },
                    "Price": { "N": "100" }
                }
            }
        },
        {
            "PutRequest": {
                "Item": {
                    "Id": { "N": "602" },
                    "Description": { "S": "Snow shovel" }
                }
```

```
            }
        }
    ]
}
```

Penghitung atom

Anda dapat menggunakan operasi `UpdateItem` untuk mengimplementasikan penghitung atom—atribut numerik yang bertambah, tanpa syarat, tanpa mengganggu permintaan tulis lainnya. (Semua permintaan tulis diterapkan sesuai urutan penerimaannya.) Dengan penghitung atom, pembaruan tidak idempotent. Dengan kata lain, nilai numerik bertambah atau berkurang setiap kali Anda menelepon. `UpdateItem` Jika nilai kenaikan yang digunakan untuk memperbarui penghitung atom positif, maka itu dapat menyebabkan penghitungan berlebihan. Jika nilai kenaikan negatif, maka dapat menyebabkan kekurangan perhitungan.

Anda dapat menggunakan penghitung atom untuk melacak jumlah pengunjung ke situs web. Dalam hal ini, aplikasi Anda akan menaikkan nilai numerik, terlepas dari nilai saat ini. Jika operasi `UpdateItem` gagal, aplikasi hanya dapat mencoba ulang operasi. Tindakan ini akan berisiko memperbarui penghitung dua kali, tetapi Anda mungkin bisa menoleransi sedikit kelebihan penghitungan atau kekurangan penghitungan dari pengunjung situs web.

Penghitung atom tidak akan cocok ketika kelebihan penghitungan atau kekurangan penghitungan tidak dapat ditoleransi (misalnya, dalam aplikasi perbankan). Dalam kasus ini, lebih aman menggunakan pembaruan bersyarat, bukannya penghitung atom.

Untuk informasi selengkapnya, lihat Menambahkan dan mengurangi atribut numerik.

Example

AWS CLI Contoh berikut menambah produk `Price` sebesar 5. Untuk contoh ini, item diketahui ada sebelum penghitung diperbarui. Karena `UpdateItem` tidak idempotent, `Price` meningkat setiap kali Anda menjalankan kode ini.

```
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
```

Penulisan bersyarat

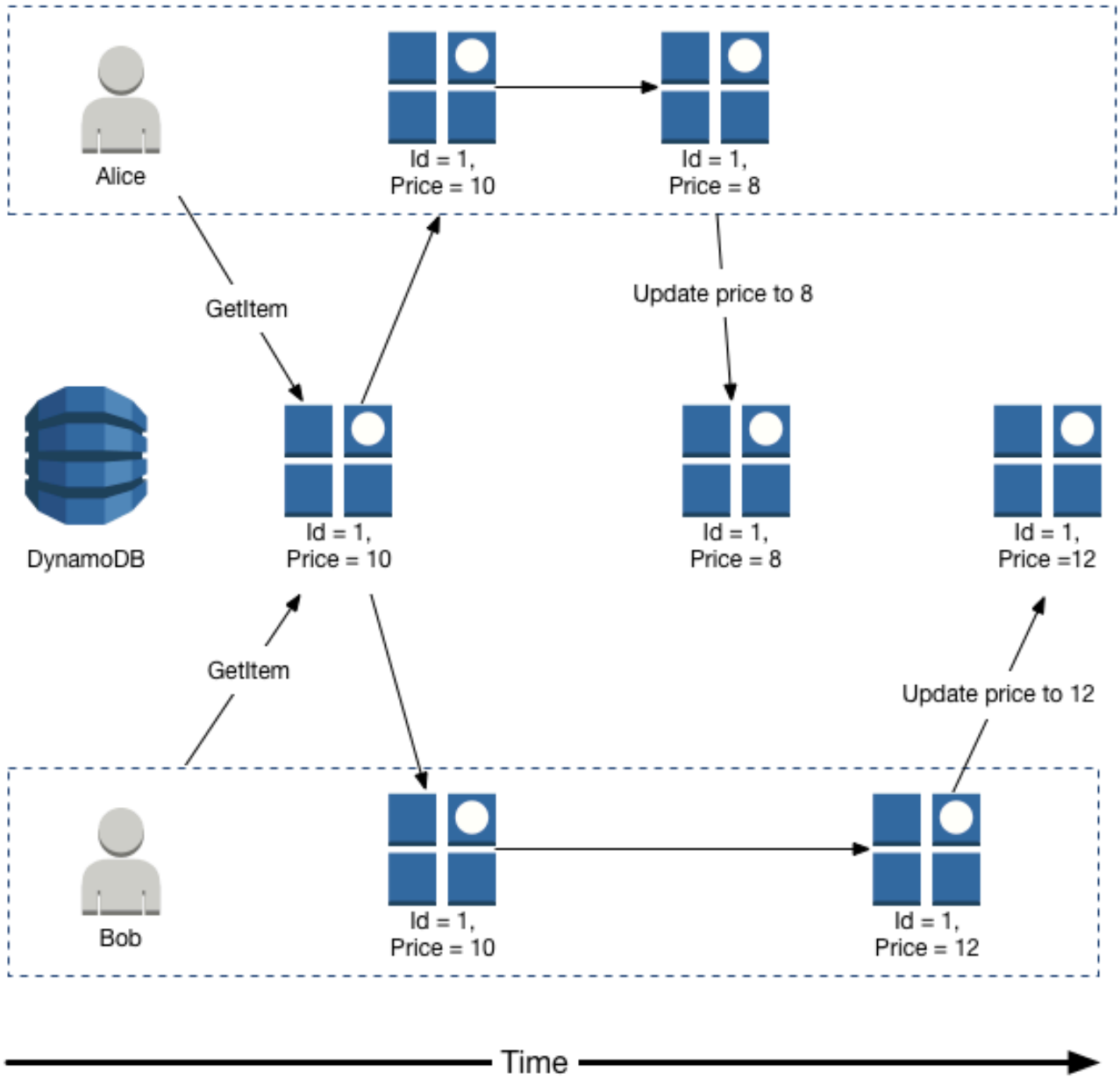
Secara default, operasi tulis DynamoDB (`PutItem`, `UpdateItem`, `DeleteItem`) bersifat tanpa syarat: Setiap operasi menimpa item yang ada yang memiliki kunci primer tertentu.

DynamoDB secara opsional mendukung penulisan bersyarat untuk operasi ini. Penulisan bersyarat akan berhasil hanya jika atribut item memenuhi satu atau beberapa kondisi yang diharapkan. Jika tidak, operasi ini akan mengembalikan kesalahan.

Penulisan bersyarat memeriksa kondisinya terhadap versi item yang terbaru diperbarui. Perhatikan bahwa jika item sebelumnya tidak ada atau jika operasi sukses terbaru terhadap item tersebut adalah penghapusan, maka penulisan bersyarat tidak akan menemukan item sebelumnya.

Penulisan bersyarat sangat membantu dalam banyak situasi. Misalnya, Anda mungkin ingin operasi `PutItem` bisa berhasil hanya jika sudah tidak ada item dengan kunci primer yang sama. Atau Anda dapat mencegah operasi `UpdateItem` memodifikasi item jika salah satu atributnya memiliki nilai tertentu.

Penulisan bersyarat berguna jika beberapa pengguna mencoba mengubah item yang sama. Perhatikan diagram berikut, di mana dua pengguna (Alice dan Bob) bekerja dengan item yang sama dari tabel DynamoDB.



Misalkan Alice menggunakan AWS CLI untuk memperbarui `Price` atribut ke 8.

```
aws dynamodb update-item \
    --table-name ProductCatalog \
    --key '{"Id":{"N":"1"}}' \
    --update-expression "SET Price = :newval" \
```

```
    --expression-attribute-values file://expression-attribute-values.json
```

Argumen untuk `--expression-attribute-values` disimpan dalam file `expression-attribute-values.json`:

```
{
    ":newval":{"N":"8"}
}
```

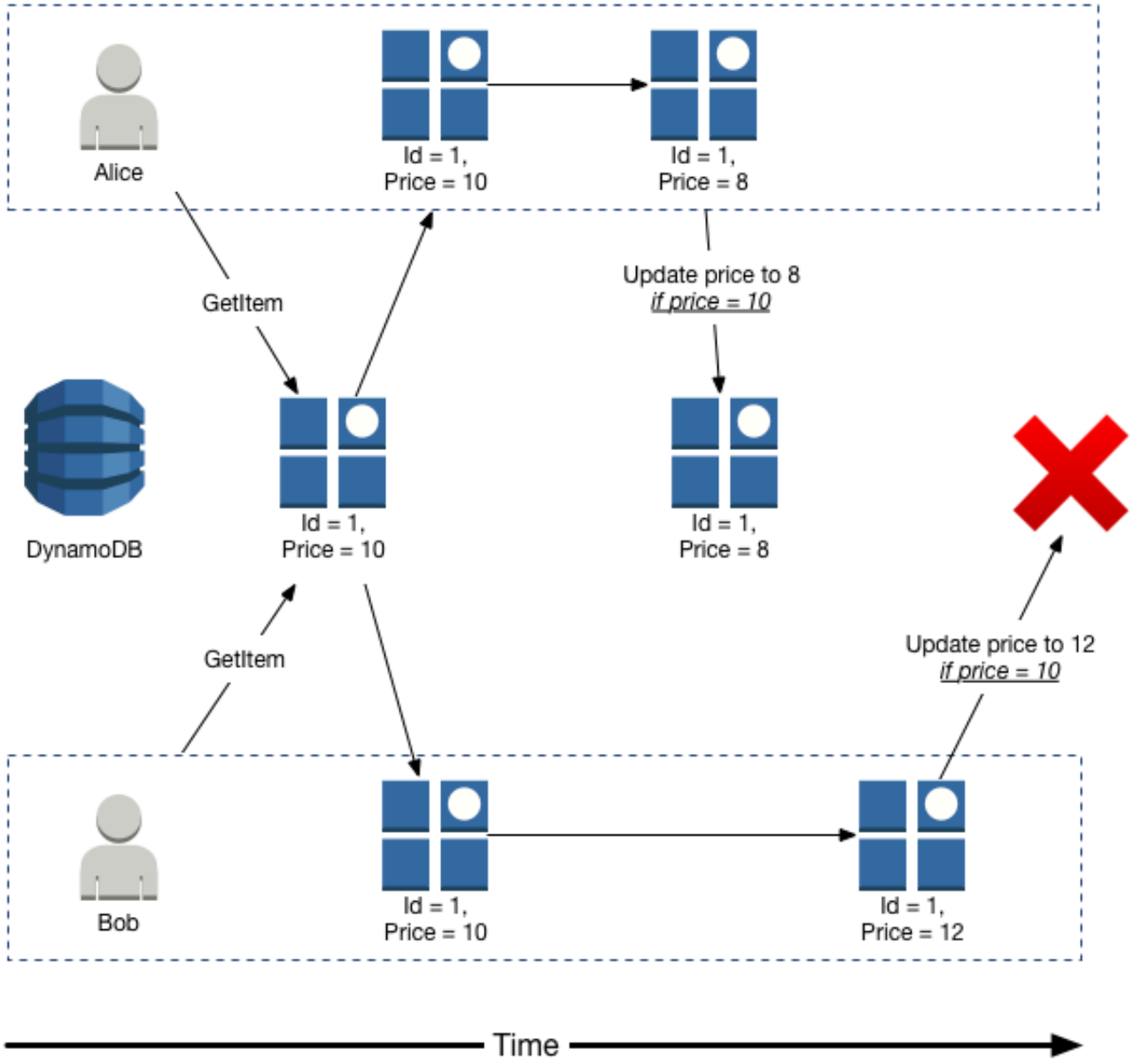
Sekarang anggaplah Bob mengeluarkan permintaan `UpdateItem` yang serupa kemudian, tetapi mengubah `Price` menjadi 12. Untuk Bob, parameter `--expression-attribute-values` akan terlihat seperti berikut ini.

```
{
    ":newval":{"N":"12"}
}
```

Permintaan Bob berhasil, tetapi pembaruan Alice sebelumnya hilang.

Untuk meminta `PutItem`, `DeleteItem`, atau `UpdateItem` bersyarat, Andalah yang menentukan ekspresi kondisi. Ekspresi kondisi adalah string yang berisi nama atribut, operator bersyarat, dan fungsi bawaan. Seluruh ekspresi harus bernilai benar. Jika tidak, operasi gagal.

Sekarang pertimbangkan diagram berikut yang menunjukkan bagaimana penulisan bersyarat akan mencegah pembaruan Alice ditimpa.



Alice mencoba pertama kali untuk memperbarui `Price` menjadi 8, tetapi hanya jika `Price` saat ini adalah 10.

```
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
```

Argumen untuk `--expression-attribute-values` disimpan dalam file `expression-attribute-values.json`.

```
{
    ":newval":{"N":"8"},
    ":currval":{"N":"10"}
}
```

Pembaruan Alice berhasil karena kondisi bernilai benar.

Berikutnya, Bob mencoba memperbarui `Price` menjadi 12, tetapi hanya jika `Price` saat ini adalah 10. Untuk Bob, parameter `--expression-attribute-values` akan terlihat seperti berikut ini.

```
{
    ":newval":{"N":"12"},
    ":currval":{"N":"10"}
}
```

Karena Alice sebelumnya telah mengubah `Price` ke 8, ekspresi kondisi bernilai SALAH, dan pembaruan Bob mengalami kegagalan.

Untuk informasi selengkapnya, lihat Contoh CLI ekspresi kondisi DynamoDB.

Idempotensi penulisan bersyarat

Penulisan bersyarat dapat menjadi idempotent jika pemeriksaan bersyarat berada pada atribut yang sama dan sedang diperbarui. Ini berarti bahwa DynamoDB melakukan permintaan tulis yang diberikan hanya jika nilai atribut tertentu dalam item cocok dengan tempat mereka berada pada saat permintaan sesuai harapan Anda.

Misalnya, misalnya Anda mengeluarkan permintaan `UpdateItem` untuk meningkatkan `Price` dari item sebesar 3, tetapi hanya jika `Price` saat ini adalah 20. Setelah Anda mengirimkan permintaan, tetapi sebelum Anda mendapatkan hasil kembali, terjadi kesalahan jaringan, dan Anda tidak tahu apakah permintaan tersebut berhasil. Karena penulisan bersyarat ini idempotent, Anda dapat mencoba kembali permintaan `UpdateItem` yang sama, dan DynamoDB memperbarui item hanya jika `Price` saat ini adalah 20.

Unit kapasitas yang digunakan oleh penulisan bersyarat

Jika `ConditionExpression` bernilai false selama penulisan bersyarat, DynamoDB masih menggunakan kapasitas tulis dari tabel. Jumlah yang dikonsumsi tergantung dari besar kecilnya barang yang ada (atau minimal 1). Misalnya, jika item yang ada berukuran 300kb dan item baru yang Anda coba buat atau perbarui berukuran 310kb, unit kapasitas tulis yang digunakan akan menjadi 300 jika kondisi gagal, dan 310 jika kondisi berhasil. Jika ini adalah item baru (belum ada item yang ada), maka unit kapasitas tulis yang digunakan adalah 1 jika kondisi gagal dan 310 jika kondisi berhasil.

> **Note**
>
> Operasi tulis hanya mengonsumsi unit kapasitas tulis. Operasi tersebut tidak pernah mengonsumsi unit kapasitas baca.

Penulisan bersyarat yang gagal akan menampilkan `ConditionalCheckFailedException`. Ketika ini terjadi, Anda tidak menerima informasi apa pun dalam tanggapan tentang kapasitas tulis yang dikonsumsi.

Untuk mengembalikan jumlah unit kapasitas tulis yang dikonsumsi selama penulisan bersyarat, Anda menggunakan parameter `ReturnConsumedCapacity`:

- `TOTAL` — Mengembalikan jumlah total unit kapasitas tulis yang dikonsumsi.
- `INDEXES` — Mengembalikan jumlah total unit kapasitas tulis yang digunakan, dengan subtotal untuk tabel dan indeks sekunder apa pun yang terpengaruh oleh operasi.
- `NONE` — Tidak ada detail kapasitas tulis dikembalikan. (Ini menjadi opsi default.)

> **Note**
>
> Tidak seperti indeks sekunder global, indeks sekunder lokal berbagi kapasitas throughput yang disediakan dengan tabelnya. Aktivitas baca dan tulis pada indeks sekunder lokal mengonsumsi kapasitas throughput yang disediakan dari tabel.

Menggunakan ekspresi di DynamoDB

Di Amazon DynamoDB, Anda dapat menggunakan ekspresi untuk menentukan atribut mana yang akan dibaca dari item, menulis data saat kondisi terpenuhi, menentukan cara memperbarui item, menentukan kueri, dan memfilter hasil kueri.

Tabel ini menjelaskan tata bahasa ekspresi dasar dan jenis ekspresi yang tersedia.

Jenis ekspresi	Deskripsi
Ekspresi proyeksi	Ekspresi proyeksi mengidentifikasi atribut yang ingin Anda ambil dari item saat Anda menggunakan operasi seperti GetItem, Kueri, atau Pindai.
Ekspresi kondisi	Ekspresi kondisi menentukan item mana yang harus dimodifikasi saat Anda menggunakan PutItem, UpdateItem, dan DeleteItem operasi.
Ekspresi perbarui	Ekspresi pembaruan menentukan bagaimana UpdateItem akan memodifikasi atribut item — misalnya, menetapkan nilai skalar atau menghapus elemen dari daftar atau peta.
Ekspresi kondisi kunci	Ekspresi kondisi kunci menentukan item mana yang akan dibaca kueri dari tabel atau indeks.
Ekspresi filter	Ekspresi filter menentukan item mana di antara hasil Query yang harus dikembalikan kepada Anda. Semua hasil lainnya dibuang.

Untuk informasi tentang sintaks ekspresi dan informasi lebih rinci tentang setiap jenis ekspresi, lihat bagian berikut.

Topik

- Mengacu pada atribut item saat menggunakan ekspresi di DynamoDB
- Nama atribut ekspresi (alias) di DynamoDB
- Menggunakan nilai atribut ekspresi di DynamoDB

- Menggunakan ekspresi proyeksi di DynamoDB
- Menggunakan ekspresi pembaruan di DynamoDB
- Kondisi dan filter ekspresi, operator, dan fungsi di DynamoDB
- Contoh CLI ekspresi kondisi DynamoDB

> **Note**
>
> Untuk kompatibilitas ke belakang, DynamoDB juga mendukung parameter bersyarat yang tidak menggunakan ekspresi. Untuk informasi selengkapnya, lihat Parameter bersyarat DynamoDB lama.
> Aplikasi baru harus menggunakan ekspresi, bukan parameter warisan.

Mengacu pada atribut item saat menggunakan ekspresi di DynamoDB

Bagian ini menjelaskan cara merujuk ke atribut item dalam suatu ekspresi di Amazon DynamoDB. Anda dapat bekerja dengan atribut apa pun, meskipun itu sangat bersarang dalam beberapa daftar dan peta.

Topik

- Atribut tingkat atas
- Atribut bersarang
- Jalur dokumen

Item Sampel: ProductCatalog

Contoh di halaman ini menggunakan item contoh berikut dalam `ProductCatalog` tabel. (Tabel ini dijelaskan dalam Contoh tabel dan data untuk digunakan di DynamoDB.)

```
{
    "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"
}
```

Perhatikan hal-hal berikut:

- Nilai kunci partisi (`Id`) adalah `123`. Tidak ada kunci urutan.
- Sebagian besar atribut memiliki jenis daya skalar, seperti `String`, `Number`, `Boolean`, dan `Null`.
- Satu atribut (`Color`) adalah `String Set`.
- Atribut berikut adalah jenis daya dokumen:
  - Daftar dari `RelatedItems`. Setiap elemen adalah `Id` untuk produk terkait.
  - Peta dari `Pictures`. Setiap elemen adalah deskripsi singkat dari gambar, bersama dengan URL untuk file gambar yang sesuai.
  - Peta dari `ProductReviews`. Setiap elemen mewakili penilaian dan daftar ulasan yang sesuai dengan penilaian tersebut. Awalnya, peta ini diisi dengan ulasan bintang lima dan bintang satu.

Atribut tingkat atas

Suatu atribut dikatakan tingkat atas jika tidak melekat pada atribut lain. Untuk item `ProductCatalog`, atribut tingkat atas adalah sebagai berikut:

- `Id`
- `Title`
- `Description`
- `BicycleType`
- `Brand`
- `Price`
- `Color`
- `ProductCategory`
- `InStock`
- `QuantityOnHand`
- `RelatedItems`
- `Pictures`
- `ProductReviews`
- `Comment`
- `Safety.Warning`

Semua atribut tingkat atas ini bersifat skalar, kecuali untuk `Color` (daftar), `RelatedItems` (daftar), `Pictures` (peta), dan `ProductReviews` (peta).

Atribut bersarang

Suatu atribut dikatakan bersarang jika melekat pada atribut lain. Untuk mengakses atribut bersarang, Anda menggunakan operator dereferensi:

- `[n]` — untuk elemen daftar
- `.` (titik) — untuk elemen peta

Mengakses elemen daftar

Operator dereferensi untuk elemen daftar adalah [N], dengan n adalah nomor elemen. Elemen daftar berbasis nol, sehingga [0] mewakili elemen pertama dalam daftar, [1] mewakili elemen kedua, dan seterusnya. Berikut ini adalah beberapa contohnya:

- `MyList[0]`
- `AnotherList[12]`
- `ThisList[5][11]`

Elemen `ThisList[5]` itu sendiri adalah daftar bersarang. Karena itu, `ThisList[5][11]` merujuk pada elemen ke-12 dalam daftar itu.

Angka di dalam tanda kurung siku harus berupa angka bulat bukan negatif. Oleh karena itu, ekspresi berikut ini tidak valid:

- `MyList[-1]`
- `MyList[0.4]`

Mengakses elemen peta

Operator dereferensi untuk elemen peta adalah . (titik). Gunakan satu titik sebagai pemisah antara unsur-unsur dalam peta:

- `MyMap.nestedField`
- `MyMap.nestedField.deeplyNestedField`

Jalur dokumen

Dalam sebuah ekspresi, Anda menggunakan jalur dokumen untuk memberi tahu DynamoDB lokasi penemuan atribut. Untuk atribut tingkat atas, jalur dokumen hanyalah nama atribut. Untuk atribut bersarang, Anda mengonstruksi jalur dokumen menggunakan operator dereferensi.

Berikut adalah beberapa contoh dari jalur dokumen. (Lihat item yang ditampilkan dalam Mengacu pada atribut item saat menggunakan ekspresi di DynamoDB.)

- Atribut skalar tingkat atas.

`Description`

- Atribut daftar tingkat atas. (Ini mengembalikan seluruh daftar, bukan hanya beberapa elemen.)

`RelatedItems`

- Elemen ketiga dari daftar `RelatedItems`. (Ingat bahwa elemen daftar berbasis nol.)

`RelatedItems[2]`

- Gambar tampilan depan produk.

`Pictures.FrontView`

- Semua ulasan bintang lima.

`ProductReviews.FiveStar`

- Yang pertama dari ulasan bintang lima.

`ProductReviews.FiveStar[0]`

> **Note**
>
> Kedalaman maksimum untuk jalur dokumen adalah 32. Oleh karena itu, jumlah operator dereferensi dalam suatu jalur tidak dapat melebihi batas ini.

Anda dapat menggunakan nama atribut apa pun di jalur dokumen asalkan memenuhi persyaratan berikut:

- Karakter pertama adalah `a-z` atau `A-Z` dan atau `0-9`
- Karakter kedua (jika ada) adalaha`-z`, `A-Z`

> **Note**
>
> Jika nama atribut tidak memenuhi persyaratan ini, Anda harus menentukan nama atribut ekspresi sebagai placeholder.

Untuk informasi selengkapnya, lihat Nama atribut ekspresi (alias) di DynamoDB.

Nama atribut ekspresi (alias) di DynamoDB

Nama atribut ekspresi adalah alias (atau placeholder) yang Anda gunakan dalam ekspresi Amazon DynamoDB sebagai alternatif dari nama atribut yang sebenarnya. Nama atribut ekspresi harus dimulai dengan tanda pound (#) dan diikuti oleh satu atau lebih karakter alfanumerik. Karakter underscore (_) juga diperbolehkan.

Bagian ini menjelaskan beberapa situasi ketika Anda harus menggunakan nama atribut ekspresi.

> **Note**
>
> Contoh di bagian ini menggunakan AWS Command Line Interface (AWS CLI).

Topik

- Kata yang dicadangkan
- Nama atribut yang berisi karakter khusus
- Atribut bersarang
- Berulang kali mereferensikan nama atribut

Kata yang dicadangkan

Terkadang Anda mungkin perlu menulis ekspresi yang berisi nama atribut yang bertentangan dengan kata terpesan DynamoDB. (Untuk daftar lengkap kata-kata yang dicadangkan, lihat Kata-kata penggunaan khusus di DynamoDB.)

Misalnya, AWS CLI contoh berikut akan gagal karena `COMMENT` merupakan kata yang dicadangkan.

```
aws dynamodb get-item \
    --table-name ProductCatalog \
    --key '{"Id":{"N":"123"}}' \
    --projection-expression "Comment"
```

Untuk mengatasi hal ini, Anda dapat mengganti `Comment` dengan nama atribut ekspresi seperti `#c`. # (tanda pagar) diperlukan dan menunjukkan bahwa ini adalah placeholder untuk nama atribut. AWS CLI Contohnya sekarang akan terlihat seperti berikut ini.

```
aws dynamodb get-item \
```

```
    --table-name ProductCatalog \
    --key '{"Id":{"N":"123"}}' \
    --projection-expression "#c" \
    --expression-attribute-names '{"#c":"Comment"}'
```

> **Note**
>
> Jika nama atribut dimulai dengan nomor, berisi spasi, atau berisi kata terpesan, Anda harus menggunakan nama atribut ekspresi untuk menggantikan nama atribut dalam ekspresi tersebut.

Nama atribut yang berisi karakter khusus

Dalam suatu ekspresi, titik (".") ditafsirkan sebagai karakter pemisah pada jalur dokumen. Namun, DynamoDB juga memungkinkan Anda menggunakan karakter titik dan karakter khusus lainnya, seperti tanda hubung (“-”) sebagai bagian dari nama atribut. Dalam beberapa kasus, ini bisa menjadi ambigu. Untuk menggambarkan, misalkan bahwa Anda ingin mengambil atribut `Safety.Warning` dari item `ProductCatalog` (lihat Mengacu pada atribut item saat menggunakan ekspresi di DynamoDB).

Misalkan Anda ingin mengakses `Safety.Warning` menggunakan ekspresi proyeksi.

```
aws dynamodb get-item \
    --table-name ProductCatalog \
    --key '{"Id":{"N":"123"}}' \
    --projection-expression "Safety.Warning"
```

DynamoDB akan mengembalikan hasil kosong, bukan string yang diharapkan ("`Always wear a helmet`"). Hal ini karena DynamoDB menafsirkan titik dalam ekspresi sebagai pemisah jalur dokumen. Dalam hal ini, Anda harus menentukan nama atribut ekspresi (seperti `#sw`) sebagai pengganti untuk `Safety.Warning`. Anda kemudian dapat menggunakan ekspresi proyeksi berikut.

```
aws dynamodb get-item \
    --table-name ProductCatalog \
    --key '{"Id":{"N":"123"}}' \
    --projection-expression "#sw" \
    --expression-attribute-names '{"#sw":"Safety.Warning"}'
```

DynamoDB kemudian akan mengembalikan hasil yang benar.

> **Note**
>
> Jika nama atribut berisi titik (".") atau tanda hubung ("-"), Anda harus menggunakan nama atribut ekspresi untuk menggantikan nama atribut dalam ekspresi tersebut.

Atribut bersarang

Misalkan Anda ingin mengakses atribut `ProductReviews.OneStar` bersarang. Dalam nama atribut ekspresi, DynamoDB memperlakukan titik (".") sebagai karakter dalam nama atribut. Untuk mereferensikan atribut bersarang, tentukan nama atribut ekspresi untuk setiap elemen di jalur dokumen:

- `#pr – ProductReviews`
- `#1star – OneStar`

Anda kemudian dapat menggunakan `#pr.#1star` untuk ekspresi proyeksi.

```
aws dynamodb get-item \
    --table-name ProductCatalog \
    --key '{"Id":{"N":"123"}}' \
    --projection-expression "#pr.#1star"  \
    --expression-attribute-names '{"#pr":"ProductReviews", "#1star":"OneStar"}'
```

DynamoDB kemudian akan mengembalikan hasil yang benar.

Berulang kali mereferensikan nama atribut

Nama atribut ekspresi sangat membantu ketika Anda perlu merujuk ke nama atribut yang sama berulang-ulang. Misalnya, pertimbangkan ekspresi berikut untuk mengambil beberapa ulasan dari item `ProductCatalog`.

```
aws dynamodb get-item \
    --table-name ProductCatalog \
    --key '{"Id":{"N":"123"}}' \
    --projection-expression "ProductReviews.FiveStar, ProductReviews.ThreeStar,
 ProductReviews.OneStar"
```

Untuk membuat hal ini lebih jelas, Anda dapat mengganti `ProductReviews` dengan nama atribut ekspresi seperti `#pr`. Ekspresi yang direvisi kini akan terlihat seperti berikut ini.

- `#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"}'
```

Jika Anda menentukan nama atribut ekspresi, Anda harus menggunakannya secara konsisten di seluruh ekspresi. Selain itu, Anda tidak dapat menghilangkan simbol `#`.

Menggunakan nilai atribut ekspresi di DynamoDB

Nilai atribut ekspresi di Amazon DynamoDB bertindak sebagai variabel. Mereka adalah pengganti nilai aktual yang ingin Anda bandingkan — nilai yang mungkin tidak Anda ketahui sampai runtime. Nilai atribut ekspresi harus dimulai dengan titik dua (`:`) dan diikuti oleh satu atau beberapa karakter alfanumerik.

Misalnya, anggaplah Anda ingin menampilkan semua item `ProductCatalog` yang tersedia dalam `Black` dan berbiaya `500` atau lebih sedikit. Anda dapat menggunakan operasi `Scan` dengan ekspresi filter, seperti dalam contoh AWS Command Line Interface (AWS CLI).

```
aws dynamodb scan \
    --table-name ProductCatalog \
    --filter-expression "contains(Color, :c) and Price <= :p" \
    --expression-attribute-values file://values.json
```

Argumen untuk `--expression-attribute-values` disimpan dalam file `values.json`.

```
{
    ":c": { "S": "Black" },
    ":p": { "N": "500" }
}
```

Jika Anda menentukan nilai atribut ekspresi, Anda harus menggunakannya secara konsisten di seluruh ekspresi. Selain itu, Anda tidak dapat menghilangkan simbol `:`.

Nilai atribut ekspresi digunakan dengan ekspresi kondisi kunci, ekspresi kondisi, ekspresi pembaruan, dan ekspresi filter.

Menggunakan ekspresi proyeksi di DynamoDB

Untuk membaca data dari tabel, Anda menggunakan operasi seperti `GetItem`, `Query`, atau `Scan`. Amazon DynamoDB mengembalikan semua atribut item secara default. Untuk mendapatkan beberapa saja, bukan semua atribut, gunakan ekspresi proyeksi.

Ekspresi proyeksi adalah string yang mengidentifikasi atribut yang Anda inginkan. Untuk mengambil atribut tunggal, tentukan namanya. Untuk beberapa atribut, nama harus dipisahkan koma.

Berikut adalah beberapa contoh ekspresi proyeksi, berdasarkan item `ProductCatalog` dari Mengacu pada atribut item saat menggunakan ekspresi di DynamoDB:

- Atribut tingkat atas tunggal.

  ```
  Title
  ```

- Tiga atribut tingkat atas. DynamoDB mengambil seluruh set `Color`.

  ```
  Title, Price, Color
  ```

- Empat atribut tingkat atas. DynamoDB mengembalikan seluruh konten `RelatedItems` dan `ProductReviews`.

  ```
  Title, Description, RelatedItems, ProductReviews
  ```

> **Note**
>
> Ekspresi proyeksi tidak berpengaruh pada konsumsi throughput yang disediakan. DynamoDB menentukan unit kapasitas yang dikonsumsi berdasarkan ukuran item, bukan jumlah data yang dikembalikan ke aplikasi.

Kata-kata yang dicadangkan dan karakter khusus

DynamoDB telah memesan kata-kata dan karakter khusus. DynamoDB memungkinkan Anda untuk menggunakan kata-kata yang dicadangkan ini dan karakter khusus untuk nama, tetapi kami menyarankan Anda menghindari melakukannya karena Anda harus menggunakan alias untuk mereka setiap kali Anda menggunakan nama-nama ini dalam sebuah ekspresi. Untuk daftar lengkap, lihat Kata-kata penggunaan khusus di DynamoDB.

Anda harus menggunakan nama atribut ekspresi sebagai pengganti nama sebenarnya jika:

- Nama atribut ada di daftar kata-kata yang dicadangkan di DynamoDB.
- Nama atribut tidak memenuhi persyaratan bahwa karakter pertama adalah `a-z` atau `A-Z` dan bahwa karakter kedua (jika ada) adalah`a-Z`,`A-Z`, atau`0-9`.
- Nama atribut berisi # (hash) atau: (titik dua).

AWS CLI Contoh berikut menunjukkan bagaimana menggunakan ekspresi proyeksi dengan `GetItem` operasi. Ekspresi proyeksi ini mengambil atribut skalar tingkat atas (`Description`), elemen pertama dalam daftar (`RelatedItems[0]`), dan daftar bersarang dalam peta (`ProductReviews.FiveStar`).

```
aws dynamodb get-item \
    --table-name ProductCatalog \
    --key '"Id": { "N": "123" } \
    --projection-expression "Description, RelatedItems[0], ProductReviews.FiveStar"
```

JSON berikut akan dikembalikan untuk contoh ini.

```
{
    "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"
                }
```

```
            ]
        }
    }
}
```

Menggunakan ekspresi pembaruan di DynamoDB

Operasi `UpdateItem` memperbarui item yang sudah ada, atau menambahkan item baru ke tabel jika belum ada. Anda harus memberikan kunci item yang ingin Anda perbarui. Anda juga harus menyediakan ekspresi pembaruan, yang menunjukkan atribut yang ingin Anda ubah dan nilai yang ingin Anda tetapkan padanya.

Ekspresi pembaruan menentukan cara `UpdateItem` mengubah atribut item—misalnya, menetapkan nilai skalar atau menghapus elemen dari daftar atau peta.

Berikut ini adalah ringkasan sintaksis untuk ekspresi pembaruan.

```
update-expression ::=
    [ SET action [, action] ... ]
    [ REMOVE action [, action] ...]
    [ ADD action [, action] ... ]
    [ DELETE action [, action] ...]
```

Ekspresi pembaruan terdiri dari satu atau lebih klausa. Setiap klausa dimulai dengan kata kunci `SET`, `REMOVE`, `ADD`, atau `DELETE`. Anda dapat memasukkan salah satu klausa ini dalam ekspresi pembaruan, dalam urutan apa pun. Namun, setiap kata kunci tindakan hanya dapat muncul satu kali.

Dalam setiap klausa, ada satu atau lebih tindakan yang dipisahkan dengan koma. Setiap tindakan mewakili modifikasi data.

Contoh dalam bagian ini didasarkan pada item `ProductCatalog` yang ditampilkan dalam Menggunakan ekspresi proyeksi di DynamoDB.

Topik di bawah ini mencakup beberapa kasus penggunaan yang berbeda untuk `SET` tindakan tersebut.

Topik

- SET — mengubah atau menambahkan atribut item
- REMOVE—menghapus atribut dari item
- ADD — memperbarui nomor dan set

- DELETE — menghapus elemen dari set
- Menggunakan beberapa ekspresi pembaruan

SET — mengubah atau menambahkan atribut item

Gunakan tindakan SET dalam ekspresi pembaruan untuk menambahkan satu atau beberapa atribut ke item. Jika salah satu atribut ini sudah ada, atribut tersebut akan ditimpa dengan nilai baru. Jika Anda ingin menghindari penimpaan atribut yang ada, Anda dapat menggunakan SET dengan fungsi `if_not_exists`. Fungsi `if_not_exists` khusus untuk tindakan SET dan hanya dapat digunakan dalam ekspresi pembaruan.

Saat Anda menggunakan SET untuk memperbarui elemen daftar, konten elemen akan diganti dengan data baru yang Anda tentukan. Jika elemen tersebut belum ada, SET menambahkan elemen baru pada akhir daftar.

Jika Anda menambahkan beberapa elemen dalam satu operasi SET, elemen diurutkan berdasarkan nomor elemen.

Anda juga dapat menggunakan SET untuk menambahkan atau mengurangi dari atribut yang berjenis `Number`. Untuk melakukan beberapa tindakan SET, pisahkan dengan koma.

Dalam ringkasan sintaks berikut:

- *`path`*Elemen adalah jalur dokumen ke item.
- *`operand`*Elemen dapat berupa jalur dokumen ke item atau fungsi.

```
set-action ::=
    path = value

value ::=
    operand
    | operand '+' operand
    | operand '-' operand

operand ::=
    path | function

function ::=
    if_not_exists (path, value)
```

Jika item tidak berisi atribut di jalur yang ditentukan, `if_not_exists` evaluasi kevalue. Kalau tidak, itu mengevaluasi. `path`

Operasi `PutItem` berikut membuat item sampel yang dirujuk oleh contoh.

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

Argumen untuk `--item` disimpan dalam file `item.json`. (Untuk kemudahan, hanya beberapa atribut item yang digunakan.)

```
{
    "Id": {"N": "789"},
    "ProductCategory": {"S": "Home Improvement"},
    "Price": {"N": "52"},
    "InStock": {"BOOL": true},
    "Brand": {"S": "Acme"}
}
```

Topik

- Memodifikasi atribut
- Menambahkan daftar dan peta
- Menambahkan elemen ke daftar
- Menambahkan atribut peta bersarang
- Menambahkan dan mengurangi atribut numerik
- Melekatkan elemen ke daftar
- Mencegah penimpaan atribut yang ada

Memodifikasi atribut

Example

Perbarui atribut `ProductCategory` dan `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
```

Argumen untuk `--expression-attribute-values` disimpan dalam file `values.json`.

```
{
    ":c": { "S": "Hardware" },
    ":p": { "N": "60" }
}
```

> **Note**
>
> Dalam operasi `UpdateItem`, `--return-values ALL_NEW` menyebabkan DynamoDB untuk mengembalikan item seperti yang muncul setelah pembaruan.

Menambahkan daftar dan peta

Example

Tambahkan daftar baru dan peta baru.

```
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
```

Argumen untuk `--expression-attribute-values` disimpan dalam file `values.json`.

```
{
    ":ri": {
        "L": [
            { "S": "Hammer" }
        ]
    },
    ":pr": {
        "M": {
            "FiveStar": {
                "L": [
```

```
                    { "S": "Best product ever!" }
                ]
            }
        }
    }
}
```

Menambahkan elemen ke daftar

Example

Tambahkan atribut baru ke daftar `RelatedItems`. (Ingatlah bahwa elemen daftar berbasis nol, sehingga [0] mewakili elemen pertama dalam daftar, [1] mewakili elemen kedua, dan seterusnya.)

```
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
```

Argumen untuk `--expression-attribute-values` disimpan dalam file `values.json`.

```
{
    ":ri": { "S": "Nails" }
}
```

> **Note**
>
> Saat Anda menggunakan `SET` untuk memperbarui elemen daftar, konten elemen akan diganti dengan data baru yang Anda tentukan. Jika elemen tersebut belum ada, `SET` menambahkan elemen baru pada akhir daftar.
> Jika Anda menambahkan beberapa elemen dalam satu operasi `SET`, elemen diurutkan berdasarkan nomor elemen.

Menambahkan atribut peta bersarang

Example

Tambahkan beberapa atribut peta bersarang.

```
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
```

Argumen untuk `--expression-attribute-names` disimpan dalam file `names.json`.

```
{
    "#pr": "ProductReviews",
    "#5star": "FiveStar",
    "#3star": "ThreeStar"
}
```

Argumen untuk `--expression-attribute-values` disimpan dalam file `values.json`.

```
{
    ":r5": { "S": "Very happy with my purchase" },
    ":r3": {
        "L": [
            { "S": "Just OK - not that great" }
        ]
    }
}
```

Menambahkan dan mengurangi atribut numerik

Anda dapat menambahkan atau mengurangi dari atribut numerik yang sudah ada. Untuk melakukannya, gunakan operator + (plus) dan - (minus).

Example

Kurangi `Price` dari item.

```
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
```

Untuk meningkatkan `Price`, Anda akan menggunakan operator + dalam ekspresi pembaruan.

Melekatkan elemen ke daftar

Anda dapat menambahkan elemen ke akhir daftar. Untuk melakukannya, gunakan `SET` dengan fungsi `list_append`. (Nama fungsi ini peka huruf besar/kecil.) Fungsi `list_append` khusus untuk tindakan `SET` dan hanya dapat digunakan dalam ekspresi pembaruan. Sintaksnya adalah sebagai berikut.

- `list_append (`*`list1`*`, `*`list2`*`)`

Fungsi ini mengambil dua daftar sebagai input dan menambahkan semua elemen dari *`list2`* ke *`list1`*.

Example

Dalam Menambahkan elemen ke daftar, Anda mencantumkan `RelatedItems` dan mengisinya dengan dua elemen: `Hammer` dan `Nails`. Sekarang Anda menambahkan dua elemen lagi ke akhir `RelatedItems`.

```
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
```

Argumen untuk `--expression-attribute-values` disimpan dalam file `values.json`.

```
{
    ":vals": {
        "L": [
            { "S": "Screwdriver" },
            {"S": "Hacksaw" }
        ]
    }
```

```
}
```

Akhirnya, Anda menambahkan satu elemen lagi ke awal dari `RelatedItems`. Untuk melakukan hal ini, tukar urutan elemen `list_append`. (Ingatlah bahwa `list_append` mengambil dua daftar sebagai input dan menambahkan daftar kedua ke yang pertama.)

```
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
```

Atribut `RelatedItems` yang dihasilkan sekarang berisi lima elemen, dalam urutan berikut: `Chisel`, `Hammer`, `Nails`, `Screwdriver`, `Hacksaw`.

Mencegah penimpaan atribut yang ada

Example

Tetapkan `Price` dari suatu item, tetapi hanya jika item tersebut belum memiliki atribut `Price`. (Jika `Price` sudah ada, tidak ada yang terjadi.)

```
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—menghapus atribut dari item

Gunakan tindakan `REMOVE` dalam ekspresi pembaruan untuk menghapus satu atau beberapa atribut dari item di Amazon DynamoDB. Untuk melakukan beberapa tindakan `REMOVE`, pisahkan dengan koma.

Berikut ini adalah ringkasan sintaks untuk `REMOVE` dalam ekspresi pembaruan. Satu-satunya operand adalah jalur dokumen untuk atribut yang ingin Anda hapus.

```
remove-action ::=
```

```
    path
```

Example

Hapus beberapa atribut dari item. (Jika atribut tidak ada, tidak ada yang terjadi.)

```
aws dynamodb update-item \
    --table-name ProductCatalog \
    --key '{"Id":{"N":"789"}}' \
    --update-expression "REMOVE Brand, InStock, QuantityOnHand" \
    --return-values ALL_NEW
```

Menghapus elemen dari daftar

Anda dapat menggunakan `REMOVE` untuk menghapus elemen individu dari daftar.

Example

Dalam Melekatkan elemen ke daftar, Anda mengubah atribut daftar (`RelatedItems`) sehingga berisi lima elemen:

- `[0]—Chisel`
- `[1]—Hammer`
- `[2]—Nails`
- `[3]—Screwdriver`
- `[4]—Hacksaw`

Berikut AWS Command Line Interface (AWS CLI) contoh menghapus `Hammer` dan `Nails` dari daftar.

```
aws dynamodb update-item \
    --table-name ProductCatalog \
    --key '{"Id":{"N":"789"}}' \
    --update-expression "REMOVE RelatedItems[1], RelatedItems[2]" \
    --return-values ALL_NEW
```

Setelah `Hammer` dan `Nails` dihapus, elemen yang tersisa digeser. Daftarnya sekarang berisi hal-hal berikut:

- `[0]—Chisel`

- `[1]—Screwdriver`
- `[2]—Hacksaw`

ADD — memperbarui nomor dan set

> **Note**
>
> Secara umum, sebaiknya gunakan `SET` daripada `ADD`.

Gunakan tindakan `ADD` dalam ekspresi pembaruan untuk menambahkan atribut baru dan nilainya untuk item.

Jika atribut sudah ada, perilaku `ADD` bergantung pada jenis daya atribut ini:

- Jika atribut adalah angka, dan nilai yang Anda tambahkan juga angka, nilai matematis ditambahkan ke atribut yang ada. (Jika nilai adalah angka negatif, nilai tersebut dikurangi dari atribut yang ada.)
- Jika atribut adalah suatu set, dan nilai yang Anda tambahkan juga suatu set, nilai ditambahkan ke set yang ada.

> **Note**
>
> Tindakan `ADD` hanya mendukung jenis daya angka dan set.

Untuk melakukan beberapa tindakan `ADD`, pisahkan dengan koma.

Dalam ringkasan sintaks berikut:

- *`path`*Elemen adalah jalur dokumen ke atribut. Atribut harus berupa `Number` atau jenis daya set.
- *`value`*Elemen adalah angka yang ingin Anda tambahkan ke atribut (untuk tipe `Number` data), atau set untuk ditambahkan ke atribut (untuk jenis set).

```
add-action ::=
    path value
```

Topik di bawah ini mencakup beberapa kasus penggunaan yang berbeda untuk `ADD` tindakan tersebut.

Topik

- Menambahkan nomor
- Menambahkan elemen ke set

Menambahkan nomor

Anggaplah atribut `QuantityOnHand` tidak ada. AWS CLI Contoh berikut ditetapkan `QuantityOnHand` ke 5.

```
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
```

Kini setelah `QuantityOnHand` ada, Anda dapat menjalankan kembali contoh untuk menaikkan `QuantityOnHand` sebesar 5 setiap kali melakukannya.

Menambahkan elemen ke set

Anggaplah atribut `Color` tidak ada. Contoh AWS CLI berikut menetapkan `Color` untuk satu set string dengan dua elemen.

```
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
```

Kini setelah `Color` ada, Anda dapat menambahkan lebih banyak elemen untuknya.

```
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 — menghapus elemen dari set

> **Important**
>
> Tindakan `DELETE` hanya mendukung jenis daya `Set`.

Gunakan tindakan `DELETE` dalam ekspresi pembaruan untuk menghapus satu atau beberapa elemen dari satu set. Untuk melakukan beberapa tindakan `DELETE`, pisahkan dengan koma.

Dalam ringkasan sintaks berikut:

- *`path`*Elemen adalah jalur dokumen ke atribut. Atribut harus berupa jenis daya set.
- *`subset`*Ini adalah satu atau lebih elemen yang ingin Anda hapus*`path`*. Anda harus menentukan *`subset`* sebagai tipe set.

```
delete-action ::=
    path subset
```

Example

Dalam Menambahkan elemen ke set, Anda membuat set string `Color`. Contoh ini akan menghapus beberapa elemen dari set tersebut.

```
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
```

Menggunakan beberapa ekspresi pembaruan

Anda dapat menggunakan beberapa ekspresi pembaruan dalam pernyataan tunggal.

Example

Jika Anda ingin mengubah nilai atribut dan menghapus atribut lain sepenuhnya, Anda dapat menggunakan tindakan SET dan REMOVE dalam satu pernyataan. Operasi ini akan mengurangi `Price` nilai menjadi 15 sekaligus menghapus `InStock` atribut dari item.

```
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

Jika Anda ingin menambahkan ke daftar sambil juga mengubah nilai atribut lain, Anda bisa menggunakan dua tindakan SET dalam satu pernyataan. Operasi ini akan menambahkan “Nails” ke atribut daftar `RelatedItems` dan juga menetapkan nilai `Price` ke 21.

```
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
```

Kondisi dan filter ekspresi, operator, dan fungsi di DynamoDB

Untuk memanipulasi data dalam tabel DynamoDB, Anda menggunakan,, dan operasi`PutItem`. `UpdateItem` `DeleteItem` Untuk operasi manipulasi data ini, Anda dapat menentukan ekspresi kondisi untuk menentukan item mana yang harus dimodifikasi. Jika ekspresi kondisi dievaluasi ke true, operasi berhasil. Jika tidak, operasi gagal.

Bagian ini mencakup fungsi dan kata kunci bawaan untuk menulis ekspresi filter dan ekspresi kondisi di Amazon DynamoDB. Untuk informasi lebih rinci mengenai fungsi dan pemrograman dengan DynamoDB, lihat Pemrograman dengan DynamoDB dan AWS SDKs dan Referensi API DynamoDB.

Topik

- Sintaks untuk ekspresi filter dan kondisi

- Membuat perbandingan
- Fungsi
- Evaluasi logika
- Tanda kurung
- Prioritas dalam kondisi

Sintaks untuk ekspresi filter dan kondisi

Dalam ringkasan sintaks berikut, *operand* dapat berupa sebagai berikut:

- Nama atribut tingkat atas, seperti `Id`, `Title`, `Description`, atau `ProductCategory`
- Jalur dokumen yang mereferensikan atribut bersarang

```
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)
```

Membuat perbandingan

Gunakan komparator ini untuk membandingkan operan dengan nilai tunggal:

- *a* = *b*- Benar *a* jika sama dengan*b*.
- *a* <> *b*Benar jika *a* tidak sama dengan*b*.
- *a* < *b*- Benar *a* jika kurang dari*b*.
- *a* <= *b*Benar jika *a* kurang dari atau sama dengan*b*.
- *a* > *b*Benar jika *a* lebih besar dari*b*.
- *a* >= *b*Benar jika *a* lebih besar dari atau sama dengan*b*.

Gunakan kata kunci `BETWEEN` dan `IN` untuk membandingkan operan dengan rentang nilai atau daftar nilai yang disebutkan:

- *a* `BETWEEN` *b* `AND` *c*Benar jika *a* lebih besar dari atau sama dengan*b*, dan kurang dari atau sama dengan*c*.
- *a* `IN (`*b*`,` *c*`,` *d*`)` — Benar jika *a* sama dengan nilai apa pun dalam daftar—misalnya, salah satu dari, *bc*, atau. *d* Daftar ini dapat berisi hingga 100 nilai, dipisahkan dengan koma.

Fungsi

Gunakan fungsi berikut untuk menentukan apakah atribut berada dalam item, atau untuk mengevaluasi nilai atribut. Nama fungsi ini peka huruf besar/kecil. Untuk atribut bersarang, Anda harus menyediakan jalur dokumen yang lengkap.

Fungsi	Deskripsi
`attribute_exists (`*path*`)`	True jika item berisi atribut yang ditentukan oleh `path`. Contoh: Periksa apakah item dalam tabel `Product` memiliki tampilan tampak samping. • `attribute_exists (#Pictures.#SideView)`

Fungsi	Deskripsi
`attribute_not_exists (path)`	True jika atribut yang ditentukan oleh `path` tidak ada dalam item. Contoh: Periksa apakah item memiliki atribut `Manufacturer` . • `attribute_not_exists (Manufacturer)`

Fungsi	Deskripsi
`attribute_type ( path, type)`	True jika atribut di jalur yang ditentukan merupakan jenis daya tertentu. Parameter `type` harus salah satu dari hal berikut: • `S` – String • `SS` – Set string • `N` – Nomor • `NS` – Set nomor • `B` – Biner • `BS` – Set biner • `BOOL` – Boolean • `NULL` – Null • `L` – Daftar • `M` – Peta Anda harus menggunakan nilai atribut ekspresi untuk parameter `type`. Contoh: Periksa apakah atribut `Quantity0nHand` berjenis Daftar. Dalam contoh ini, `:v_sub` adalah placeholder untuk string `L`. • `attribute_type (ProductReviews.FiveStar, :v_sub)`

Fungsi	Deskripsi
	Anda harus menggunakan nilai atribut ekspresi untuk parameter `type`.
`begins_with ( path, substr)`	True jika atribut yang ditentukan oleh `path` dimulai dengan substring tertentu. Contoh: Periksa apakah beberapa karakter pertama dari URL gambar tampilan depan adalah `http://`. • `begins_with (Pictures.FrontView, :v_sub)` Nilai atribut ekspresi `:v_sub` adalah placeholder untuk `http://`.

Fungsi	Deskripsi
`contains (path, operand)`	True jika atribut yang ditentukan oleh `path` adalah salah satu hal berikut: • `String` yang berisi substring tertentu. • `Set` yang berisi elemen tertentu dalam set. • `List` yang berisi elemen tertentu dalam daftar. Jika atribut yang ditentukan oleh `path` adalah `String`, `operand` harus `String`. Jika atribut yang ditentukan oleh `path` adalah `Set`, `operand` harus merupakan jenis elemen set. Jalan dan operan harus berbeda. Artinya, `contains (a, a)` mengembalikan kesalahan. Contoh: Periksa apakah atribut `Brand` berisi substring `Company`. • `contains (Brand, :v_sub)` Nilai atribut ekspresi `:v_sub` adalah placehold er untuk `Company`. Contoh: Periksa apakah produk tersedia dalam warna merah. • `contains (Color, :v_sub)`

Fungsi	Deskripsi
	Nilai atribut ekspresi `:v_sub` adalah placeholder untuk `Red`.

Fungsi	Deskripsi
`size (path)`	Mengembalikan angka yang mewakili ukuran atribut ini. Berikut ini adalah jenis daya yang valid untuk digunakan dengan `size`. Jika atribut adalah jenis `String`, `size` menampilkan panjang string. Contoh: Periksa apakah string `Brand` kurang dari atau sama dengan 20 karakter. Nilai atribut ekspresi `:v_sub` adalah placeholder untuk `20`. • `size (Brand) <= :v_sub` Jika atribut adalah jenis `Binary`, `size` menghasilkan jumlah byte dalam nilai atribut. Contoh: Misalkan item `ProductCatalog` memiliki atribut binari yang disebut `VideoClip` yang berisi video singkat dari produk yang digunakan. Ekspresi berikut memeriksa apakah `VideoClip` melebihi 64.000 byte. Nilai atribut ekspresi `:v_sub` adalah placeholder untuk `64000`. • `size(VideoClip) > :v_sub` Jika atribut adalah jenis daya `Set`, `size` menghasilkan jumlah elemen dalam set. Contoh: Periksa apakah produk tersedia dalam lebih dari satu warna. Nilai atribut ekspresi `:v_sub` adalah placeholder untuk `1`.

Fungsi	Deskripsi
	• `size (Color) < :v_sub` Jika atribut adalah jenis `List` atau `Map`, `size` menghasilkan jumlah elemen turunan. Contoh: Periksa apakah jumlah ulasan `OneStar` telah melampaui ambang batas tertentu. Nilai atribut ekspresi `:v_sub` adalah placeholder untuk `3`. • `size(ProductReviews.OneStar) > :v_sub`

Evaluasi logika

Gunakan kata kunci `AND`, `OR`, dan `NOT` untuk melakukan evaluasi logika. Dalam daftar berikut, *a* dan *b* mewakili kondisi yang akan dievaluasi.

- *a* `AND` *b*Benar jika *a* dan *b* keduanya benar.
- *a* `OR` *b*Benar jika salah satu *a* atau *b* (atau keduanya) benar.
- `NOT` *a*Benar jika *a* salah. Salah jika *a* benar.

Berikut ini adalah contoh kode AND dalam suatu operasi.

```
dynamodb-local (*)> select * from exprtest where a > 3 and a < 5;
```

Tanda kurung

Gunakan tanda kurung untuk mengubah prioritas evaluasi logis. Misalnya, anggaplah bahwa kondisi *a* dan *b* benar, dan kondisi *c* itu salah. Ekspresi berikut bernilai True:

- *a* `OR` *b* `AND` *c*

Namun, jika Anda mengapit kondisi dalam tanda kurung, kondisi tersebut akan dievaluasi terlebih dahulu. Misalnya, nilai berikut bernilai false:

- `(a OR b) AND c`

> **Note**
>
> Anda dapat menyarangkan tanda kurung dalam sebuah ekspresi. Bagian paling dalam akan dievaluasi terlebih dahulu.

Berikut ini adalah contoh kode dengan tanda kurung dalam evaluasi logis.

```
dynamodb-local (*)> select * from exprtest where attribute_type(b, string)
or ( a = 5 and c = “coffee”);
```

Prioritas dalam kondisi

DynamoDB mengevaluasi kondisi dari kiri ke kanan menggunakan aturan prioritas berikut:

- `= <> < <= > >=`
- `IN`
- `BETWEEN`
- `attribute_exists attribute_not_exists begins_with contains`
- Tanda kurung
- `NOT`
- `AND`
- `OR`

Contoh CLI ekspresi kondisi DynamoDB

Berikut ini adalah beberapa AWS Command Line Interface (AWS CLI) contoh penggunaan ekspresi kondisi. Contoh-contoh ini berdasarkan tabel `ProductCatalog`, yang diperkenalkan dalam Mengacu pada atribut item saat menggunakan ekspresi di DynamoDB. Kunci partisi untuk tabel ini adalah `Id`, dan tidak ada kunci urutan. Operasi `PutItem` berikut membuat item `ProductCatalog` sampel yang dirujuk oleh contoh.

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

Argumen untuk `--item` disimpan dalam file `item.json`. (Untuk kemudahan, hanya beberapa atribut item yang digunakan.)

```
{
    "Id": {"N": "456" },
    "ProductCategory": {"S": "Sporting Goods" },
    "Price": {"N": "650" }
}
```

Topik

- Put bersyarat
- Penghapusan bersyarat
- Pembaruan bersyarat
- Contoh ekspresi bersyarat

Put bersyarat

Operasi `PutItem` menimpa item dengan kunci primer yang sama (jika ada). Jika Anda ingin menghindari hal ini, gunakan ekspresi kondisi. Hal ini memungkinkan penulisan untuk dilanjutkan hanya jika item yang dimaksud belum memiliki kunci primer yang sama.

Contoh berikut digunakan `attribute_not_exists()` untuk memeriksa apakah kunci primer ada dalam tabel sebelum mencoba operasi tulis.

> **Note**
>
> Jika kunci utama Anda terdiri dari kedua kunci partisi (pk) dan kunci sortir (sk), parameter akan memeriksa apakah `attribute_not_exists(pk)` DAN `attribute_not_exists(sk)` mengevaluasi menjadi benar atau salah sebagai pernyataan keseluruhan sebelum mencoba operasi tulis.

```
aws dynamodb put-item \
    --table-name ProductCatalog \
```

```
    --item file://item.json \
    --condition-expression "attribute_not_exists(Id)"
```

Jika ekspresi kondisi bernilai false, DynamoDB mengembalikan pesan kesalahan berikut: Permintaan bersyarat gagal.

> **Note**
>
> Untuk informasi selengkapnya tentang fungsi `attribute_not_exists` dan lainnya, lihat Kondisi dan filter ekspresi, operator, dan fungsi di DynamoDB.

Penghapusan bersyarat

Untuk melakukan penghapusan bersyarat, Anda menggunakan operasi `DeleteItem` dengan ekspresi kondisi. Ekspresi kondisi harus bernilai true agar operasi berhasil; jika tidak, operasi akan gagal.

Pertimbangkan item yang didefinisikan di atas.

Misalkan Anda ingin menghapus item, tetapi hanya dalam kondisi berikut:

- `ProductCategory` adalah "Alat Olahraga" atau "Perlengkapan Berkebun."
- `Price` adalah antara 500 dan 600.

Contoh berikut mencoba menghapus versi item.

```
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
```

Argumen untuk `--expression-attribute-values` disimpan dalam file `values.json`.

```
{
    ":cat1": {"S": "Sporting Goods"},
    ":cat2": {"S": "Gardening Supplies"},
    ":lo": {"N": "500"},
    ":hi": {"N": "600"}
```

```
}
```

> **Note**
>
> Dalam ekspresi kondisi, `:` (karakter titik dua) menunjukkan nilai atribut ekspresi—placeholder untuk nilai aktual. Untuk informasi selengkapnya, lihat Menggunakan nilai atribut ekspresi di DynamoDB.
> Untuk informasi selengkapnya tentang `IN`, `AND`, dan kata kunci lainnya, lihat Kondisi dan filter ekspresi, operator, dan fungsi di DynamoDB.

Dalam contoh ini, perbandingan `ProductCategory` bernilai true, tetapi perbandingan `Price` bernilai false. Hal ini menyebabkan ekspresi kondisi untuk bernilai false dan operasi `DeleteItem` gagal.

Pembaruan bersyarat

Untuk melakukan pembaruan bersyarat, Anda menggunakan operasi `UpdateItem` dengan ekspresi kondisi. Ekspresi kondisi harus bernilai true agar operasi berhasil; jika tidak, operasi akan gagal.

> **Note**
>
> `UpdateItem` juga mendukung ekspresi pembaruan, di mana Anda menentukan modifikasi yang ingin Anda buat ke item. Untuk informasi selengkapnya, lihat Menggunakan ekspresi pembaruan di DynamoDB.

Misalkan Anda mulai dengan item yang didefinisikan di atas.

Contoh berikut melakukan operasi `UpdateItem`. Operasi ini mencoba untuk mengurangi `Price` dari produk sebesar 75—tetapi ekspresi kondisi mencegah pembaruan jika `Price` saat ini kurang dari atau sama dengan 500.

```
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
```

Argumen untuk `--expression-attribute-values` disimpan dalam file `values.json`.

```
{
    ":discount": { "N": "75"},
    ":limit": {"N": "500"}
}
```

Jika `Price` awal adalah 650, operasi `UpdateItem` mengurangi `Price` menjadi 575. Jika Anda menjalankan operasi `UpdateItem` lagi, `Price` berkurang menjadi 500. Jika Anda menjalankannya untuk kali ketiga, ekspresi kondisi bernilai false, dan pembaruan gagal.

> **Note**
>
> Dalam ekspresi kondisi, `:` (karakter titik dua) menunjukkan nilai atribut ekspresi—placeholder untuk nilai aktual. Untuk informasi selengkapnya, lihat Menggunakan nilai atribut ekspresi di DynamoDB.
> Untuk informasi selengkapnya tentang ">" dan operator lainnya, lihat Kondisi dan filter ekspresi, operator, dan fungsi di DynamoDB.

Contoh ekspresi bersyarat

Untuk informasi selengkapnya tentang fungsi yang digunakan dalam contoh berikut, lihat Kondisi dan filter ekspresi, operator, dan fungsi di DynamoDB. Jika Anda ingin tahu lebih banyak tentang cara menentukan jenis atribut yang berbeda dalam suatu ekspresi, lihat Mengacu pada atribut item saat menggunakan ekspresi di DynamoDB.

Memeriksa atribut dalam item

Anda dapat memeriksa keberadaan (atau ketiadaan) atribut apa pun. Jika ekspresi kondisi bernilai true, operasi berhasil; jika tidak, operasi akan gagal.

Contoh berikut menggunakan `attribute_not_exists` untuk menghapus produk hanya jika operasi tidak memiliki atribut `Price`.

```
aws dynamodb delete-item \
    --table-name ProductCatalog \
    --key '{"Id": {"N": "456"}}' \
    --condition-expression "attribute_not_exists(Price)"
```

DynamoDB juga menyediakan fungsi `attribute_exists`. Contoh berikut menghapus produk hanya jika operasi telah menerima ulasan buruk.

```
aws dynamodb delete-item \
    --table-name ProductCatalog \
    --key '{"Id": {"N": "456"}}' \
    --condition-expression "attribute_exists(ProductReviews.OneStar)"
```

Memeriksa jenis atribut

Anda dapat memeriksa jenis daya nilai atribut dengan menggunakan fungsi `attribute_type`. Jika ekspresi kondisi bernilai true, operasi berhasil; jika tidak, operasi akan gagal.

Contoh berikut menggunakan `attribute_type` untuk menghapus produk hanya jika memiliki atribut `Color` jenis Set String.

```
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
```

Argumen untuk `--expression-attribute-values` disimpan dalam expression-attribute-values file.json.

```
{
    ":v_sub":{"S":"SS"}
}
```

Memeriksa nilai awal string

Anda dapat memeriksa apakah nilai atribut String dimulai dengan substring tertentu dengan menggunakan fungsi `begins_with`. Jika ekspresi kondisi bernilai true, operasi berhasil; jika tidak, operasi akan gagal.

Contoh berikut menggunakan `begins_with` untuk menghapus produk hanya jika elemen `FrontView` dari peta `Pictures` dimulai dengan nilai tertentu.

```
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
```

Argumen untuk `--expression-attribute-values` disimpan dalam expression-attribute-values file.json.

```
{
    ":v_sub":{"S":"http://"}
}
```

Memeriksa elemen dalam set

Anda dapat memeriksa elemen dalam set atau mencari substring dalam string dengan menggunakan fungsi `contains`. Jika ekspresi kondisi bernilai true, operasi berhasil; jika tidak, operasi akan gagal.

Contoh berikut menggunakan `contains` untuk menghapus produk hanya jika elemen Set String `Color` memiliki elemen dengan nilai tertentu.

```
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
```

Argumen untuk `--expression-attribute-values` disimpan dalam expression-attribute-values file.json.

```
{
    ":v_sub":{"S":"Red"}
}
```

Memeriksa ukuran nilai atribut

Anda dapat memeriksa ukuran nilai atribut dengan menggunakan fungsi `size`. Jika ekspresi kondisi bernilai true, operasi berhasil; jika tidak, operasi akan gagal.

Contoh berikut menggunakan `size` untuk menghapus produk hanya jika ukuran atribut Binari `VideoClip` lebih besar dari `64000` byte.

```
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
```

Argumen untuk `--expression-attribute-values` disimpan dalam expression-attribute-values file.json.

```
{
    ":v_sub":{"N":"64000"}
}
```

Menggunakan time to live (TTL) di DynamoDB

Time To Live (TTL) untuk DynamoDB adalah metode hemat biaya untuk menghapus item yang tidak lagi relevan. TTL memungkinkan Anda untuk menentukan stempel waktu kedaluwarsa per item yang menunjukkan kapan item tidak lagi diperlukan. DynamoDB secara otomatis menghapus item yang kedaluwarsa dalam beberapa hari dari waktu kedaluwarsa, tanpa menghabiskan throughput penulisan.

Untuk menggunakan TTL, pertama-tama aktifkan pada tabel dan kemudian tentukan atribut tertentu untuk menyimpan stempel waktu kedaluwarsa TTL. Stempel waktu harus disimpan dalam format waktu epoch Unix pada perincian detik. Setiap kali item dibuat atau diperbarui, Anda dapat menghitung waktu kedaluwarsa dan menyimpannya di atribut TTL.

Item dengan atribut TTL yang valid dan kedaluwarsa dapat dihapus oleh sistem kapan saja, biasanya dalam beberapa hari setelah kedaluwarsa. Anda masih dapat memperbarui item kedaluwarsa yang menunggu penghapusan, termasuk mengubah atau menghapus atribut TTL mereka. Saat memperbarui item yang kedaluwarsa, kami menyarankan Anda menggunakan ekspresi kondisi untuk memastikan item tersebut belum dihapus selanjutnya. Gunakan ekspresi filter untuk menghapus item kedaluwarsa dari hasil Pindai dan Kueri.

Item yang dihapus bekerja sama dengan yang dihapus melalui operasi penghapusan tipikal. Setelah dihapus, item masuk ke DynamoDB Streams sebagai penghapusan layanan, bukan penghapusan pengguna, dan dihapus dari indeks sekunder lokal dan indeks sekunder global seperti operasi penghapusan lainnya.

Jika Anda menggunakan Tabel Global versi 2019.11.21 (Saat ini) dari tabel global dan Anda juga menggunakan fitur TTL, DynamoDB mereplikasi penghapusan TTL ke semua tabel replika.

Penghapusan TTL awal tidak menggunakan Write Capacity Units (WCU) di wilayah di mana TTL kedaluwarsa terjadi. Namun, penghapusan TTL yang direplikasi ke tabel replika menggunakan Unit Kapasitas Tulis yang direplikasi saat menggunakan kapasitas yang disediakan, atau Unit Tulis Replikasi saat menggunakan mode kapasitas sesuai permintaan, di setiap wilayah replika dan biaya yang berlaku akan berlaku.

Untuk informasi selengkapnya tentang TTL, lihat topik berikut:

Topik

- Aktifkan waktu untuk hidup (TTL) di DynamoDB
- Menghitung waktu untuk hidup (TTL) di DynamoDB
- Bekerja dengan item kedaluwarsa dan waktu untuk hidup (TTL)

Aktifkan waktu untuk hidup (TTL) di DynamoDB

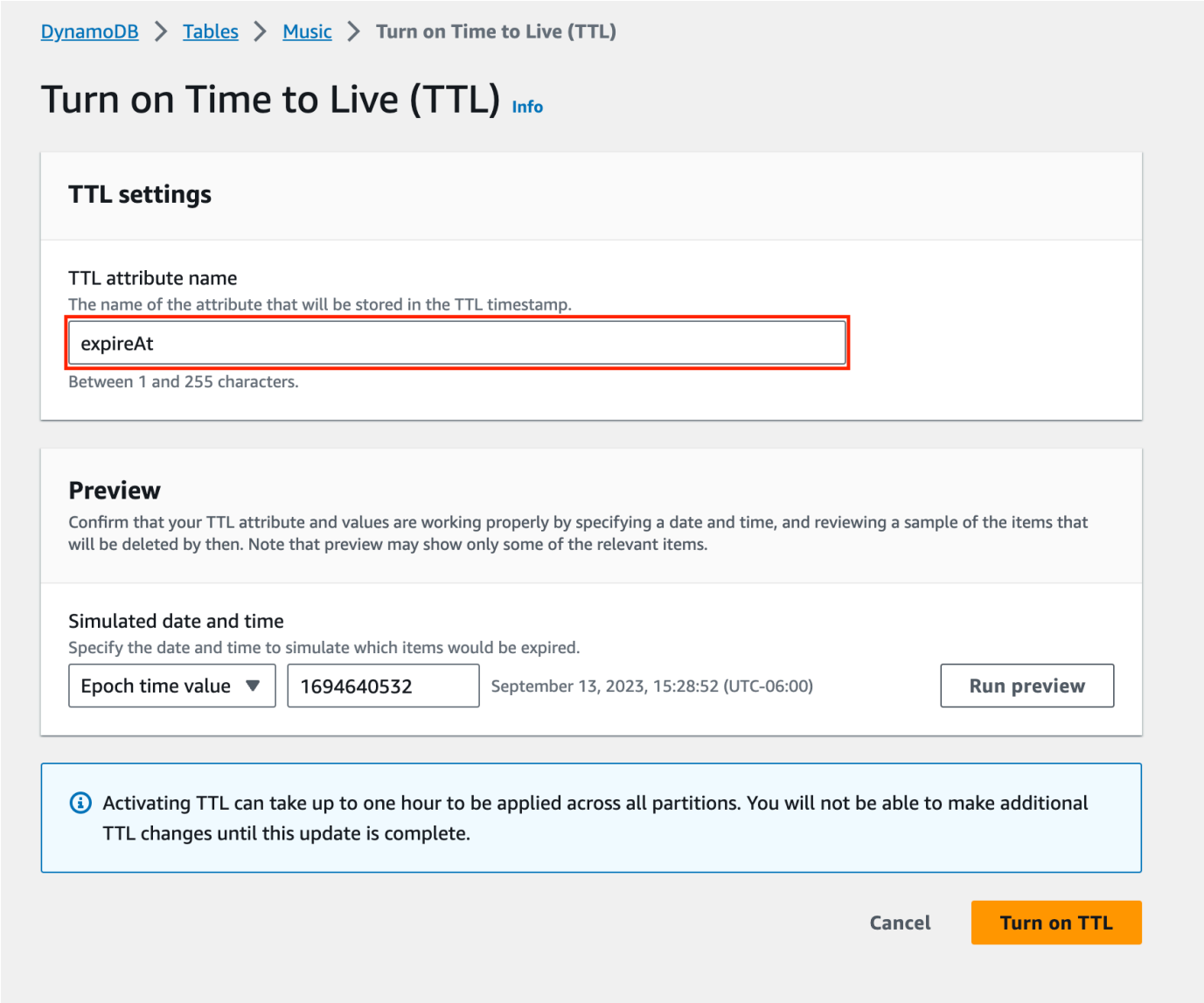
> **Note**
>
> Untuk membantu debugging dan verifikasi pengoperasian fitur TTL yang benar, nilai yang disediakan untuk item TTL dicatat dalam plaintext di log diagnostik DynamoDB.

Anda dapat mengaktifkan TTL di Amazon DynamoDB Console AWS Command Line Interface , (), atau menggunakan AWS CLI Referensi Amazon DynamoDB API dengan salah satu yang seharusnya. AWS SDKs Dibutuhkan sekitar satu jam untuk mengaktifkan TTL di semua partisi.

Aktifkan DynamoDB TTL menggunakan konsol AWS

1. Masuk ke AWS Management Console dan buka konsol DynamoDB di. https://console.aws.amazon.com/dynamodb/
2. Pilih Tabel, lalu pilih tabel yang ingin Anda ubah.
3. Di tab Pengaturan tambahan, di bagian Time to Live (TTL), pilih Aktifkan untuk mengaktifkan TTL.
4. Saat mengaktifkan TTL pada tabel, DynamoDB mengharuskan Anda mengidentifikasi nama atribut tertentu yang akan dicari layanan saat menentukan apakah suatu item memenuhi syarat untuk kedaluwarsa. Nama atribut TTL, yang ditunjukkan di bawah ini, peka huruf besar/kecil dan harus cocok dengan atribut yang ditentukan dalam operasi baca dan tulis Anda. Ketidakcocokan akan menyebabkan item yang kedaluwarsa tidak dihapus. Mengganti nama

atribut TTL mengharuskan Anda untuk menonaktifkan TTL dan kemudian mengaktifkannya kembali dengan atribut baru di masa mendatang. TTL akan terus memproses penghapusan selama kurang lebih 30 menit setelah dinonaktifkan. TTL harus dikonfigurasi ulang pada tabel yang dipulihkan.



5. (Opsional) Anda dapat melakukan tes dengan mensimulasikan tanggal dan waktu kedaluwarsa dan mencocokkan beberapa item. Ini memberi Anda daftar sampel item dan mengonfirmasi bahwa ada item yang berisi nama atribut TTL yang disediakan bersama dengan waktu kedaluwarsa.

Setelah TTL diaktifkan, atribut TTL ditandai TTL saat Anda melihat item di konsol DynamoDB. Anda dapat melihat tanggal dan waktu kedaluwarsa suatu item dengan mengarahkan kursor ke atribut tersebut.

Aktifkan DynamoDB TTL menggunakan API

Python

Anda dapat mengaktifkan TTL dengan kode, menggunakan UpdateTimeToLiveoperasi.

```
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')
```

Anda dapat mengonfirmasi TTL diaktifkan dengan menggunakan DescribeTimeToLiveoperasi, yang menggambarkan status TTL di atas meja. `TimeToLive`Statusnya adalah salah satu `ENABLED` atau`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

Anda dapat mengaktifkan TTL dengan kode, menggunakan UpdateTimeToLiveCommandoperasi.

```
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');
```

Aktifkan Waktu untuk Hidup menggunakan AWS CLI

1. Aktifkan TTL pada tabel `TTLExample`.

```
aws dynamodb update-time-to-live --table-name TTLExample --time-to-live-
specification "Enabled=true, AttributeName=ttl"
```

2. Jelaskan TTL pada tabel `TTLExample`.

```
aws dynamodb describe-time-to-live --table-name TTLExample
{
    "TimeToLiveDescription": {
        "AttributeName": "ttl",
        "TimeToLiveStatus": "ENABLED"
    }
}
```

3. Tambahkan item ke tabel `TTLExample` dengan atribut Waktu untuk Tayang yang diatur menggunakan shell BASH dan AWS CLI.

```
EXP=`date -d '+5 days' +%s`
aws dynamodb put-item --table-name "TTLExample" --item '{"id": {"N": "1"}, "ttl":
 {"N": "'$EXP'"}}'
```

Contoh ini dimulai dengan tanggal saat ini dan menambahkan 5 hari untuk membuat waktu kedaluwarsa. Kemudian, contoh ini mengubah waktu kedaluwarsa menjadi format jangka waktu yang pada akhirnya menambahkan item ke "`TTLExample`".

> **Note**
>
> Salah satu cara untuk menetapkan nilai kedaluwarsa Waktu untuk Tayang adalah dengan menghitung jumlah detik yang ditambahkan ke waktu kedaluwarsa. Misalnya, 5 hari adalah 432.000 detik. Namun, seringkali lebih baik memulai dengan suatu tanggal dan bekerja dari sana.

Cukup mudah untuk mendapatkan waktu saat ini dalam format jangka waktu, seperti pada contoh berikut.

- Terminal Linux: `date +%s`
- Python: `import time; int(time.time())`
- Java: `System.currentTimeMillis() / 1000L`
- JavaScript: `Math.floor(Date.now() / 1000)`

Aktifkan DynamoDB TTL menggunakan AWS 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
```

Detail tambahan tentang penggunaan TTL dalam AWS CloudFormation template Anda dapat ditemukan di sini.

Menghitung waktu untuk hidup (TTL) di DynamoDB

Cara umum untuk menerapkan TTL adalah dengan menetapkan waktu kedaluwarsa untuk item berdasarkan kapan mereka dibuat atau terakhir diperbarui. Ini dapat dilakukan dengan menambahkan waktu ke `createdAt` dan `updatedAt` stempel waktu. Misalnya, TTL untuk item yang baru dibuat dapat diatur `createdAt` ke+90 hari. Saat item diperbarui, TTL dapat dihitung ulang menjadi +90 hari. `updatedAt`

Waktu kedaluwarsa yang dihitung harus dalam format epoch, dalam hitungan detik. Untuk dipertimbangkan untuk kedaluwarsa dan penghapusan, TTL tidak boleh lebih dari lima tahun yang lalu. Jika Anda menggunakan format lain, proses TTL akan mengabaikan item. Jika Anda menetapkan tanggal kedaluwarsa ke sometime in the future ketika Anda ingin item kedaluwarsa, item akan kedaluwarsa setelah waktu itu. Misalnya, Anda menetapkan tanggal kedaluwarsa ke 1724241326 (yaitu Senin, 21 Agustus 2024 11:55:26 (GMT)). Item akan kedaluwarsa setelah waktu yang ditentukan.

Topik

- Buat item dan atur Waktu ke Hidup
- Perbarui item dan segarkan Waktu untuk Hidup

Buat item dan atur Waktu ke Hidup

Contoh berikut menunjukkan bagaimana menghitung waktu kedaluwarsa saat membuat item baru, menggunakan `expireAt` sebagai nama atribut TTL. Pernyataan penugasan memperoleh waktu saat ini sebagai variabel. Dalam contoh, waktu kedaluwarsa dihitung sebagai 90 hari dari waktu saat ini. Waktu kemudian dikonversi ke format epoch dan disimpan sebagai tipe data integer dalam atribut TTL.

Contoh kode berikut menunjukkan cara membuat item dengan TTL.

Java

SDK untuk 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;
        }
    }
}
```

- Untuk detail API, lihat PutItemdi Referensi AWS SDK for Java 2.x API.

JavaScript

SDK untuk 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');
```

- Untuk detail API, lihat PutItemdi Referensi AWS SDK untuk JavaScript API.

Python

SDK untuk 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"
)
```

- Untuk detail API, lihat PutItemdi AWS SDK for Python (Boto3) Referensi API.

Perbarui item dan segarkan Waktu untuk Hidup

Contoh ini merupakan kelanjutan dari yang dari bagian sebelumnya. Waktu kedaluwarsa dapat dihitung ulang jika item diperbarui. Contoh berikut menghitung ulang `expireAt` stempel waktu menjadi 90 hari dari waktu saat ini.

Contoh kode berikut menunjukkan cara memperbarui TTL item.

Java

SDK untuk Java 2.x

Perbarui TTL pada item DynamoDB yang ada dalam tabel.

```
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;
        }
```

```
    }
```

- Untuk detail API, lihat UpdateItemdi Referensi AWS SDK for Java 2.x API.

JavaScript

SDK untuk 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');
```

- Untuk detail API, lihat UpdateItemdi Referensi AWS SDK untuk JavaScript API.

Python

SDK untuk 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"
)
```

- Untuk detail API, lihat UpdateItemdi AWS SDK for Python (Boto3) Referensi API.

Contoh TTL yang dibahas dalam pendahuluan ini menunjukkan metode untuk memastikan hanya item yang baru diperbarui disimpan dalam tabel. Item yang diperbarui memiliki masa pakainya diperpanjang, sedangkan item yang tidak diperbarui pasca-pembuatan kedaluwarsa dan dihapus tanpa biaya, mengurangi penyimpanan dan mempertahankan tabel bersih.

Bekerja dengan item kedaluwarsa dan waktu untuk hidup (TTL)

Item kedaluwarsa yang tertunda penghapusan dapat difilter dari operasi baca dan tulis. Ini berguna dalam skenario ketika data kedaluwarsa tidak lagi valid dan tidak boleh digunakan. Jika tidak difilter, mereka akan terus ditampilkan dalam operasi baca dan tulis sampai dihapus oleh proses latar belakang.

> **Note**
>
> Barang-barang ini masih diperhitungkan untuk penyimpanan dan biaya baca sampai dihapus.

Penghapusan TTL dapat diidentifikasi di DynamoDB Streams, tetapi hanya di Wilayah tempat penghapusan terjadi. Penghapusan TTL yang direplikasi ke wilayah tabel global tidak dapat diidentifikasi dalam aliran DynamoDB di wilayah tempat penghapusan direplikasi.

Filter item kedaluwarsa dari operasi baca

Untuk operasi baca seperti Pindai dan Kueri, ekspresi filter dapat memfilter item kedaluwarsa yang menunggu penghapusan. Seperti yang ditunjukkan dalam cuplikan kode berikut, ekspresi filter dapat menyaring item di mana waktu TTL sama dengan atau kurang dari waktu saat ini. Misalnya, kode SDK Python menyertakan pernyataan penugasan yang memperoleh waktu saat ini sebagai variabel (now), dan mengubahnya menjadi format waktu epoch. `int`

Contoh kode berikut menunjukkan cara query untuk item TTL.

Java

SDK untuk Java 2.x

Query Filtered Expression untuk mengumpulkan item TTL dalam tabel DynamoDB menggunakan. 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;
        }
```

- Untuk detail API, lihat Kueri di Referensi API AWS SDK for Java 2.x .

JavaScript

SDK untuk JavaScript (v3)

Query Filtered Expression untuk mengumpulkan item TTL dalam tabel DynamoDB menggunakan. AWS SDK untuk 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');
```

- Untuk detail API, lihat Kueri di Referensi API AWS SDK untuk JavaScript .

Python

SDK untuk Python (Boto3)

Query Filtered Expression untuk mengumpulkan item TTL dalam tabel DynamoDB menggunakan. AWS SDK untuk 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")
```

- Untuk detail API, lihat Kueri di Referensi API AWS SDK untuk Python (Boto3).

Menulis secara kondisional ke item yang kedaluwarsa

Ekspresi kondisi dapat digunakan untuk menghindari penulisan terhadap item yang kedaluwarsa. Cuplikan kode di bawah ini adalah pembaruan bersyarat yang memeriksa apakah waktu kedaluwarsa lebih besar dari waktu saat ini. Jika benar, operasi tulis akan berlanjut.

Contoh kode berikut menunjukkan cara memperbarui TTL item secara kondisional.

Java

SDK untuk Java 2.x

Perbarui TTL pada Item DynamoDB yang ada dalam tabel, dengan kondisi.

```
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;
        }
    }
}
```

- Untuk detail API, lihat UpdateItemdi Referensi AWS SDK for Java 2.x API.

JavaScript

SDK untuk JavaScript (v3)

Perbarui TTL pada Item DynamoDB yang ada dalam tabel, dengan kondisi.

```
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');
```

- Untuk detail API, lihat UpdateItemdi Referensi AWS SDK untuk JavaScript API.

Python

SDK untuk Python (Boto3)

Perbarui TTL pada Item DynamoDB yang ada dalam tabel, dengan kondisi.

```
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",
)
```

- Untuk detail API, lihat UpdateItemdi AWS SDK for Python (Boto3) Referensi API.

Mengidentifikasi item yang dihapus di DynamoDB Streams

Catatan stream berisi bidang identitas pengguna `Records[<index>].userIdentity`. Item yang dihapus oleh proses TTL memiliki bidang berikut:

```
Records[<index>].userIdentity.type
"Service"

Records[<index>].userIdentity.principalId
"dynamodb.amazonaws.com"
```

JSON berikut menunjukkan bagian yang relevan dari catatan aliran tunggal:

```
"Records": [
  {
 ...
  "userIdentity": {
  "type": "Service",
       "principalId": "dynamodb.amazonaws.com"
    }
   ...
 }
]
```

Menanyakan tabel di DynamoDB

Anda dapat menggunakan operasi API `Query` di Amazon DynamoDB untuk menemukan item berdasarkan nilai kunci primer.

Anda harus memberikan nama atribut kunci partisi dan satu nilai untuk atribut tersebut. `Query` mengembalikan semua item dengan nilai kunci partisi tersebut. Secara opsional, Anda dapat memberikan atribut kunci urutan dan menggunakan operator perbandingan untuk menyaring hasil pencarian.

Untuk informasi selengkapnya tentang cara menggunakan `Query`, seperti sintaks permintaan, parameter respons, dan contoh tambahan, lihat Kueri di Referensi API Amazon DynamoDB.

Topik

- Ekspresi kondisi kunci untuk operasi Query di DynamoDB
- Ekspresi filter untuk operasi Query di DynamoDB

- Hasil query tabel paginasi di DynamoDB
- Aspek lain dari bekerja dengan operasi Query di DynamoDB

Ekspresi kondisi kunci untuk operasi Query di DynamoDB

Anda dapat menggunakan nama atribut apa pun dalam ekspresi kondisi kunci, asalkan karakter pertama adalah `a-z` atau `A-Z` dan karakter lainnya (mulai dari karakter kedua, jika ada) adalah `a-z`, `A-Z`, or `0-9`. Selain itu, nama atribut tidak boleh berupa kata khusus DynamoDB. (Untuk daftar lengkapnya, lihat Kata-kata penggunaan khusus di DynamoDB.) Jika nama atribut tidak memenuhi persyaratan ini, Anda harus menentukan nama atribut ekspresi sebagai placeholder. Untuk informasi selengkapnya, lihat Nama atribut ekspresi (alias) di DynamoDB.

Untuk item dengan nilai kunci partisi tertentu, DynamoDB menyimpan item ini berdekatan, dalam urutan berdasarkan nilai kunci urutan. Dalam operasi `Query`, DynamoDB mengambil item dalam urutan yang diurutkan, lalu memproses item menggunakan `KeyConditionExpression` dan `FilterExpression` apa pun yang mungkin ada. Baru setelah itu hasil `Query` dikirim kembali ke klien.

Operasi `Query` selalu mengembalikan kumpulan hasil. Jika tidak ditemukan item yang cocok, set hasil kosong.

Hasil `Query` selalu diurutkan berdasarkan nilai kunci urutan. Jika jenis daya kunci urutan adalah `Number`, hasilnya dikembalikan dalam urutan numerik. Jika tidak, hasilnya akan dikembalikan dalam urutan UTF-8 byte. Secara default, urutannya menaik. Untuk membalikkan urutan, atur parameter `ScanIndexForward` ke `false`.

Operasi `Query` tunggal dapat mengambil data maksimum 1 MB. Batas ini berlaku sebelum `FilterExpression` atau `ProjectionExpression` diterapkan pada hasil. Jika `LastEvaluatedKey` ada dalam respons dan bukan null, Anda harus memberi nomor halaman pada kumpulan hasil (lihat Hasil query tabel paginasi di DynamoDB).

Contoh ekspresi kondisi kunci

Untuk menentukan kriteria pencarian, Anda menggunakan ekspresi kondisi kunci—string yang menentukan item yang akan dibaca dari tabel atau indeks.

Anda harus menentukan nama dan nilai kunci partisi sebagai syarat kesetaraan. Anda tidak dapat menggunakan atribut bukan kunci dalam ekspresi kondisi kunci.

Anda juga dapat memberikan kondisi kedua untuk kunci urutan (jika ada). Kondisi kunci urutan harus menggunakan salah satu operator perbandingan berikut:

- *a* = *b*— true jika atribut *a* sama dengan nilai *b*
- *a* < *b*— benar *a* jika kurang dari *b*
- *a* <= *b*— benar jika *a* kurang dari atau sama dengan *b*
- *a* > *b*Benar jika *a* lebih besar dari *b*
- *a* >= *b*— benar jika *a* lebih besar dari atau sama dengan *b*
- *a* BETWEEN *b* AND *c*— benar jika *a* lebih besar dari atau sama dengan*b*, dan kurang dari atau sama dengan*c*.

Fungsi berikut ini juga didukung:

- begins_with (*a*, *substr*)— true jika nilai atribut *a* dimulai dengan substring tertentu.

Contoh berikut AWS Command Line Interface (AWS CLI) menunjukkan penggunaan ekspresi kondisi kunci. Ekspresi ini menggunakan placeholder (seperti :name dan :sub) dan bukan nilai aktual. Untuk informasi selengkapnya, lihat Nama atribut ekspresi (alias) di DynamoDB dan Menggunakan nilai atribut ekspresi di DynamoDB.

Example

Ajukan kueri tabel Thread untuk ForumName (kunci partisi) tertentu. Semua item dengan nilai ForumName tersebut dibaca oleh kueri karena kunci urutan (Subject) tidak termasuk dalamKeyConditionExpression.

```
aws dynamodb query \
    --table-name Thread \
    --key-condition-expression "ForumName = :name" \
    --expression-attribute-values  '{":name":{"S":"Amazon DynamoDB"}}'
```

Example

Ajukan kueri tabel Thread untuk ForumName kunci partisi), tetapi kali ini hanya mengembalikan item dengan Subject(kunci urutan) tertentu.

```
aws dynamodb query \
```

```
    --table-name Thread \
    --key-condition-expression "ForumName = :name and Subject = :sub" \
    --expression-attribute-values  file://values.json
```

Argumen untuk `--expression-attribute-values` disimpan dalam file `values.json`.

```
{
    ":name":{"S":"Amazon DynamoDB"},
    ":sub":{"S":"DynamoDB Thread 1"}
}
```

Example

Kueri tabel `Reply` untuk `Id` tertentu (kunci partisi), tetapi hanya kembalikan item yang `ReplyDateTime` (kunci urutan) dimulai dengan karakter tertentu.

```
aws dynamodb query \
    --table-name Reply \
    --key-condition-expression "Id = :id and begins_with(ReplyDateTime, :dt)" \
    --expression-attribute-values  file://values.json
```

Argumen untuk `--expression-attribute-values` disimpan dalam file `values.json`.

```
{
    ":id":{"S":"Amazon DynamoDB#DynamoDB Thread 1"},
    ":dt":{"S":"2015-09"}
}
```

Ekspresi filter untuk operasi Query di DynamoDB

Jika Anda perlu menyempurnakan hasil `Query` lebih lanjut, Anda dapat memberikan ekspresi filter secara opsional. Ekspresi filter menentukan item mana dalam hasil `Query` yang harus dikembalikan kepada Anda. Semua hasil lainnya dibuang.

Filter ekspresi diterapkan setelah `Query` selesai, namun sebelum hasilnya dikembalikan. Oleh karena itu, `Query` menggunakan jumlah kapasitas baca yang sama, terlepas dari apakah ada ekspresi filter.

Operasi `Query` dapat mengambil data maksimal 1 MB. Batasan ini berlaku sebelum ekspresi filter dievaluasi.

Filter ekspresi tidak boleh berisi atribut kunci partisi atau kunci urutan. Anda perlu menentukan atribut tersebut dalam ekspresi kondisi kunci, bukan ekspresi filter.

Sintaks untuk ekspresi filter mirip dengan ekspresi kondisi kunci. Ekspresi filter dapat menggunakan pembanding, fungsi, dan operator logika yang sama sebagai ekspresi kondisi utama. Selain itu, ekspresi filter dapat menggunakan operator tak sama dengan <>(), operator `OR`, operator `CONTAINS`, operator `IN`, operator `BEGINS_WITH`, operator `BETWEEN`, operator `EXISTS`, dan operator `SIZE`. Untuk informasi selengkapnya, lihat Ekspresi kondisi kunci untuk operasi Query di DynamoDB dan Sintaks untuk ekspresi filter dan kondisi.

Example

AWS CLI Contoh berikut query `Thread` tabel untuk tertentu `ForumName` (kunci partisi) dan `Subject` (sort key). Dari item yang ditemukan, hanya rangkaian diskusi terpopuler yang dikembalikan—dengan kata lain, hanya rangkaian diskusi yang memiliki jumlah `Views` lebih dari tertentu.

```
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
```

Argumen untuk `--expression-attribute-values` disimpan dalam file `values.json`.

```
{
    ":fn":{"S":"Amazon DynamoDB"},
    ":sub":{"S":"DynamoDB Thread 1"},
    ":num":{"N":"3"}
}
```

Perhatikan bahwa `Views` adalah kata yang dicadangkan di DynamoDB (lihat Kata-kata penggunaan khusus di DynamoDB), sehingga contoh ini menggunakan #v sebagai placeholder. Untuk informasi selengkapnya, lihat Nama atribut ekspresi (alias) di DynamoDB.

> **Note**
>
> Ekspresi filter menghapus item dari set hasil `Query`. Jika memungkinkan, hindari penggunaan `Query` saat Anda ingin mengambil item dalam jumlah besar namun juga harus membuang sebagian besar item tersebut.

Hasil query tabel paginasi di DynamoDB

DynamoDB memberi nomor halaman hasil dari operasi `Query`. Dengan pemberian nomor halaman, hasil `Query` dibagi menjadi "halaman" data berukuran 1 MB (atau kurang). Aplikasi bisa memproses halaman pertama hasil, lalu halaman kedua, dan seterusnya.

`Query` tunggal hanya mengembalikan set hasil yang sesuai dalam batas ukuran 1 MB. Untuk menentukan apakah terdapat lebih banyak hasil, dan untuk mengambilkan satu halaman sekaligus, aplikasi harus melakukan hal berikut:

1. Periksa hasil `Query` tingkat rendah:
   - Jika hasilnya berisi elemen `LastEvaluatedKey` dan elemen tersebut non-null, lanjutkan ke langkah 2.
   - Jika tidak ada `LastEvaluatedKey` pada hasilnya, tidak ada lagi item yang dapat diambil.
2. Membangun `Query` dengan yang sama`KeyConditionExpression`. Namun, kali ini, ambil nilai `LastEvaluatedKey` dari langkah 1 dan gunakan sebagai parameter `ExclusiveStartKey` dalam permintaan `Query` baru.
3. Jalankan permintaan `Query` baru.
4. Lanjutkan ke langkah 1.

Dengan kata lain, `LastEvaluatedKey` dari respons `Query` harus digunakan sebagai `ExclusiveStartKey` untuk permintaan `Query` berikutnya. Jika tidak ada elemen `LastEvaluatedKey` dalam respons `Query`, maka Anda telah mengambil halaman hasil akhir. Jika `LastEvaluatedKey` tidak kosong, bukan berarti ada lebih banyak data di set hasil. Satu-satunya cara untuk mengetahui kapan Anda telah mencapai akhir kumpulan hasil adalah ketika `LastEvaluatedKey` kosong.

Anda dapat menggunakan AWS CLI untuk melihat perilaku ini. AWS CLI Mengirimkan `Query` permintaan tingkat rendah ke DynamoDB berulang kali, `LastEvaluatedKey` hingga tidak ada lagi dalam hasil. Perhatikan AWS CLI contoh berikut yang mengambil judul film dari tahun tertentu.

```
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
```

Biasanya, AWS CLI menangani pagination secara otomatis. Namun, dalam contoh ini, AWS CLI `--page-size` parameter membatasi jumlah item per halaman. Parameter `--debug` mencetak informasi tingkat rendah tentang permintaan dan respons.

Jika Anda menjalankan contoh ini, respons pertama dari DynamoDB terlihat serupa dengan yang berikut ini.

```
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}'
```

`LastEvaluatedKey` dalam responsnya menunjukkan bahwa tidak semua item telah diambil. AWS CLI Kemudian mengeluarkan `Query` permintaan lain ke DynamoDB. Pola permintaan dan respons ini berlanjut hingga respons akhir.

```
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}'
```

Tidak adanya `LastEvaluatedKey` menunjukkan bahwa tidak ada lagi item yang dapat diambil.

> **Note**
>
> AWS SDKs Menangani respons DynamoDB tingkat rendah (termasuk ada atau tidak adanya) dan memberikan berbagai `LastEvaluatedKey` abstraksi untuk hasil paginasi. `Query` Misalnya, antarmuka dokumen SDK untuk Java menyediakan dukungan `java.util.Iterator` sehingga Anda dapat mempelajari hasilnya satu per satu. Untuk contoh kode dalam berbagai bahasa pemrograman, lihat Panduan Memulai Amazon DynamoDB dan dokumentasi AWS SDK untuk bahasa Anda.

Anda juga dapat mengurangi ukuran halaman dengan membatasi jumlah item dalam set hasil, dengan parameter `Limit` operasi `Query`.

Untuk informasi selengkapnya tentang pembuatan kueri dengan DynamoDB, lihat Menanyakan tabel di DynamoDB.

Aspek lain dari bekerja dengan operasi Query di DynamoDB

Bagian ini mencakup aspek tambahan dari operasi Query DynamoDB, termasuk membatasi ukuran hasil, menghitung item yang dipindai vs. dikembalikan, memantau konsumsi kapasitas baca, dan mengontrol konsistensi baca.

Membatasi jumlah item dalam set hasil

Dengan operasi `Query`, Anda dapat membatasi jumlah item yang dibaca. Untuk melakukannya, tetapkan parameter `Limit` ke jumlah maksimum item yang Anda inginkan.

Misalnya, anggaplah bahwa Anda `Query` tabel, dengan nilai `Limit` dari 6, dan tanpa ekspresi filter. Hasil `Query` berisi enam item pertama dari tabel yang cocok dengan ekspresi kondisi kunci dari permintaan.

Sekarang anggap bahwa Anda menambahkan ekspresi filter ke `Query`. Dalam kasus ini, DynamoDB membaca hingga enam item, lalu hanya mengembalikan item yang cocok dengan ekspresi filter. Hasil `Query` akhir berisi enam item atau lebih sedikit, meskipun lebih banyak item akan cocok dengan ekspresi filter jika DynamoDB terus membaca lebih banyak item.

Menghitung item dalam hasil

Selain item yang sesuai dengan kriteria Anda, respons `Query` berisi elemen berikut:

- `ScannedCount` — Jumlah item yang cocok dengan ekspresi kondisi kunci sebelum ekspresi filter (jika ada) diterapkan.
- `Count` — Jumlah item yang tersisa setelah ekspresi filter (jika ada) diterapkan.

> **Note**
>
> Jika Anda tidak menggunakan ekspresi filter, `ScannedCount` dan `Count` memiliki nilai yang sama.

Jika ukuran set hasil `Query` lebih besar dari 1 MB, `ScannedCount` dan `Count` mewakili hanya jumlah parsial dari total item. Anda harus melakukan beberapa operasi `Query` untuk mengambil semua hasil (lihat Hasil query tabel paginasi di DynamoDB).

Setiap respons `Query` berisi `ScannedCount` dan `Count` untuk item yang diproses oleh permintaan `Query` khusus. Untuk mendapatkan total keseluruhan untuk semua permintaan `Query`, Anda dapat menyimpan penghitungan `ScannedCount` dan `Count`.

Unit kapasitas yang digunakan oleh kueri

Anda dapat melakukan `Query` tabel atau indeks sekunder apa pun, selama Anda memberikan nama atribut kunci partisi dan satu nilai untuk atribut tersebut. `Query` mengembalikan semua item dengan nilai kunci partisi tersebut. Secara opsional, Anda dapat memberikan atribut kunci urutan dan menggunakan operator perbandingan untuk menyaring hasil pencarian. `Query` Operasi API menggunakan unit kapasitas baca, sebagai berikut.

Jika Anda **`Query`**...	DynamoDB menggunakan unit kapasitas baca dari...
Tabel	Kapasitas baca yang ditetapkan tabel.
Indeks sekunder global	Kapasitas baca yang ditetapkan indeks.
Indeks sekunder lokal	Kapasitas baca yang ditetapkan tabel dasar.

Secara default, operasi `Query` tidak menampilkan data tentang berapa banyak kapasitas baca yang digunakan. Namun, Anda dapat menentukan parameter `ReturnConsumedCapacity` dalam sebuah permintaan `Query` untuk mendapatkan informasi ini. Berikut ini adalah pengaturan yang valid untuk `ReturnConsumedCapacity`:

- `NONE` — Tidak ada penggunaan data kapasitas yang ditampilkan. (Ini menjadi opsi default.)
- `TOTAL` — Respons mencakup jumlah agregat unit kapasitas baca yang digunakan.
- `INDEXES` — Respons menunjukkan jumlah agregat unit kapasitas baca yang digunakan, beserta dengan kapasitas yang digunakan untuk setiap tabel dan indeks yang diakses.

DynamoDB menghitung jumlah unit kapasitas baca yang dikonsumsi berdasarkan jumlah item dan ukuran item tersebut, bukan pada jumlah data yang dikembalikan ke aplikasi. Dengan alasan ini, jumlah unit kapasitas yang digunakan adalah sama, entah Anda meminta semua atribut (perilaku default) atau hanya beberapa dari atribut tersebut (menggunakan ekspresi proyeksi). Jumlahnya juga sama apakah Anda menggunakan ekspresi filter atau tidak. `Query`mengkonsumsi unit kapasitas baca minimum untuk melakukan satu pembacaan yang sangat konsisten per detik, atau dua

pembacaan yang konsisten per detik untuk item hingga 4 KB. Jika Anda perlu membaca item yang lebih besar dari 4 KB, DynamoDB memerlukan unit permintaan baca tambahan. Tabel kosong dan tabel yang sangat besar yang memiliki jumlah kunci partisi yang jarang mungkin melihat beberapa tambahan RCUs dibebankan di luar jumlah data yang ditanyakan. Ini mencakup biaya melayani `Query` permintaan, bahkan jika tidak ada data.

Konsistensi baca untuk kueri

Operasi `Query` melakukan bacaan akhir konsisten, secara default. Ini berarti bahwa hasil `Query` mungkin tidak mencerminkan perubahan karena operasi `PutItem` atau `UpdateItem` baru saja selesai. Untuk informasi selengkapnya, lihat DynamoDB membaca konsistensi.

Jika Anda memerlukan bacaan sangat konsisten, atur parameter `ConsistentRead` ke `true` dalam permintaan `Query`.

Memindai tabel di DynamoDB

Operasi `Scan` di Amazon DynamoDB membaca setiap item dalam tabel atau indeks sekunder. Secara default, operasi `Scan` mengembalikan semua atribut data untuk setiap item dalam tabel atau indeks. Anda dapat menggunakan parameter `ProjectionExpression` sehingga `Scan` hanya mengembalikan beberapa atribut, bukan semuanya.

`Scan` selalu mengembalikan set hasil. Jika tidak ditemukan item yang cocok, set hasil kosong.

Permintaan `Scan` tunggal dapat mengambil data maksimum 1 MB. Atau, DynamoDB dapat menerapkan ekspresi filter pada data ini, mengerucutkan hasil sebelum dikembalikan ke pengguna.

Topik

- Ekspresi filter untuk pemindaian
- Membatasi jumlah item dalam set hasil
- Memberi nomor halaman hasil
- Menghitung item dalam hasil
- Unit kapasitas dikonsumsi oleh pemindaian
- Konsistensi baca untuk kueri
- Pemindaian paralel

Ekspresi filter untuk pemindaian

Jika Anda perlu menyempurnakan hasil `Scan` lebih lanjut, Anda dapat memberikan ekspresi filter secara opsional. Ekspresi filter menentukan item mana dalam hasil `Scan` yang harus dikembalikan kepada Anda. Semua hasil lainnya dibuang.

Filter ekspresi diterapkan setelah `Scan` selesai, tetapi sebelum hasilnya dikembalikan. Oleh karena itu, `Scan` menggunakan jumlah kapasitas baca yang sama, terlepas dari apakah ada ekspresi filter.

Operasi `Scan` dapat mengambil data maksimal 1 MB. Batasan ini berlaku sebelum ekspresi filter dievaluasi.

Dengan `Scan`, Anda dapat menentukan atribut apa pun dalam ekspresi filter, termasuk atribut kunci partisi dan kunci urutan.

Sintaks untuk ekspresi filter mirip dengan ekspresi kondisi kunci. Ekspresi filter dapat menggunakan pembanding, fungsi, dan operator logika yang sama sebagai ekspresi kondisi utama. Lihat Kondisi dan filter ekspresi, operator, dan fungsi di DynamoDB untuk informasi selengkapnya tentang operator logika.

Example

Contoh berikut AWS Command Line Interface (AWS CLI) memindai `Thread` tabel dan hanya mengembalikan item yang terakhir diposting oleh pengguna tertentu.

```
aws dynamodb scan \
     --table-name Thread \
     --filter-expression "LastPostedBy = :name" \
     --expression-attribute-values '{":name":{"S":"User A"}}'
```

Membatasi jumlah item dalam set hasil

Operasi `Scan` memungkinkan Anda membatasi jumlah item yang dikembalikan dalam hasil. Untuk melakukannya, tetapkan parameter `Limit` pada jumlah maksimum item yang Anda inginkan untuk dikembalikan operasi `Scan`, sebelum evaluasi ekspresi filter.

Misalnya, anggaplah bahwa Anda melakukan `Scan` pada tabel, dengan nilai `Limit` adalah 6, dan tanpa ekspresi filter. Hasil `Scan` berisi enam item pertama dari tabel.

Sekarang anggap bahwa Anda menambahkan ekspresi filter ke `Scan`. Dalam kasus ini, DynamoDB menerapkan ekspresi filter untuk enam item yang dikembalikan, membuang item yang tidak cocok. Hasil `Scan` akhir berisi enam item atau lebih sedikit, tergantung jumlah item yang disaring.

Memberi nomor halaman hasil

DynamoDB memberi nomor halaman hasil dari operasi `Scan`. Dengan pemberian nomor halaman, hasil `Scan` dibagi menjadi "halaman" data berukuran 1 MB (atau kurang). Aplikasi bisa memproses halaman pertama hasil, lalu halaman kedua, dan seterusnya.

`Scan` tunggal hanya mengembalikan set hasil yang sesuai dalam batas ukuran 1 MB.

Untuk menentukan apakah terdapat lebih banyak hasil, dan untuk mengambilkan satu halaman sekaligus, aplikasi harus melakukan hal berikut:

1. Periksa hasil `Scan` tingkat rendah:
   - Jika hasilnya berisi elemen `LastEvaluatedKey`, lanjutkan ke langkah 2.
   - Jika tidak ada `LastEvaluatedKey` pada hasilnya, tidak ada lagi item yang dapat diambil.
2. Buat permintaan `Scan` baru, dengan parameter yang sama seperti yang sebelumnya. Namun, kali ini, ambil nilai `LastEvaluatedKey` dari langkah 1 dan gunakan sebagai parameter `ExclusiveStartKey` dalam permintaan `Scan` baru.
3. Jalankan permintaan `Scan` baru.
4. Lanjutkan ke langkah 1.

Dengan kata lain, `LastEvaluatedKey` dari respons `Scan` harus digunakan sebagai `ExclusiveStartKey` untuk permintaan `Scan` berikutnya. Jika tidak ada elemen `LastEvaluatedKey` dalam respons `Scan`, Anda telah mengambil halaman hasil akhir. (Tidak adanya `LastEvaluatedKey` adalah satu-satunya cara untuk mengetahui bahwa Anda telah mencapai akhir set hasil.)

Anda dapat menggunakan AWS CLI untuk melihat perilaku ini. AWS CLI Mengirimkan `Scan` permintaan tingkat rendah ke DynamoDB, berulang kali, `LastEvaluatedKey` hingga tidak lagi hadir dalam hasil. Perhatikan AWS CLI contoh berikut yang memindai seluruh `Movies` tabel tetapi hanya mengembalikan film dari genre tertentu.

```
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
```

Biasanya, AWS CLI menangani pagination secara otomatis. Namun, dalam contoh ini, AWS CLI `--page-size` parameter membatasi jumlah item per halaman. Parameter `--debug` mencetak informasi tingkat rendah tentang permintaan dan respons.

> **Note**
>
> Hasil pemberian nomor halaman Anda juga akan berbeda berdasarkan parameter input yang Anda lewati.
>
> - Menggunakan `aws dynamodb scan --table-name Prices --max-items 1` mengembalikan `NextToken`
> - Menggunakan `aws dynamodb scan --table-name Prices --limit 1` mengembalikan `LastEvaluatedKey`.
>
> Ketahuilah juga bahwa menggunakan `--starting-token` khususnya membutuhkan nilai `NextToken`.

Jika Anda menjalankan contoh ini, respons pertama dari DynamoDB terlihat serupa dengan yang berikut ini.

```
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}'
```

`LastEvaluatedKey` dalam responsnya menunjukkan bahwa tidak semua item telah diambil. AWS CLI Kemudian mengeluarkan `Scan` permintaan lain ke DynamoDB. Pola permintaan dan respons ini berlanjut hingga respons akhir.

```
2017-07-07 12:19:17,830 - MainThread - botocore.parsers - DEBUG - Response body:
b'{"Count":1,"Items":[{"title":{"S":"WarGames"}}],"ScannedCount":6}'
```

Tidak adanya `LastEvaluatedKey` menunjukkan bahwa tidak ada lagi item yang dapat diambil.

> **Note**
>
> AWS SDKs Menangani respons DynamoDB tingkat rendah (termasuk ada atau tidak adanya) dan memberikan berbagai `LastEvaluatedKey` abstraksi untuk hasil paginasi. `Scan` Misalnya, antarmuka dokumen SDK untuk Java menyediakan dukungan `java.util.Iterator` sehingga Anda dapat mempelajari hasilnya satu per satu. Untuk contoh kode dalam berbagai bahasa pemrograman, lihat Panduan Memulai Amazon DynamoDB dan dokumentasi AWS SDK untuk bahasa Anda.

Menghitung item dalam hasil

Selain item yang sesuai dengan kriteria Anda, respons `Scan` berisi elemen berikut:

- `ScannedCount` — Jumlah item yang dievaluasi, sebelum `ScanFilter` diterapkan. Nilai `ScannedCount` yang tinggi dengan sedikit, atau tanpa hasil, `Count` hasil menunjukkan operasi `Scan` yang tidak efisien. Jika Anda tidak menggunakan filter dalam permintaan, `ScannedCount` sama dengan `Count`.
- `Count` — Jumlah item yang tersisa setelah ekspresi filter (jika ada) diterapkan.

> **Note**
>
> Jika Anda tidak menggunakan ekspresi filter, `ScannedCount` dan `Count` memiliki nilai yang sama.

Jika ukuran set hasil `Scan` lebih besar dari 1 MB, `ScannedCount` dan `Count` mewakili hanya jumlah parsial dari total item. Anda harus melakukan beberapa operasi `Scan` untuk mengambil semua hasil (lihat Memberi nomor halaman hasil).

Setiap respons `Scan` berisi `ScannedCount` dan `Count` untuk item yang diproses oleh permintaan `Scan` khusus. Untuk mendapatkan total keseluruhan untuk semua permintaan `Scan`, Anda dapat menyimpan penghitungan `ScannedCount` dan `Count`.

Unit kapasitas dikonsumsi oleh pemindaian

Anda dapat melakukan `Scan` pada setiap tabel atau indeks sekunder. Operasi `Scan` mengonsumsi unit kapasitas baca, sebagai berikut.

Jika Anda **`Scan`**...	DynamoDB menggunakan unit kapasitas baca dari...
Tabel	Kapasitas baca yang ditetapkan tabel.
Indeks sekunder global	Kapasitas baca yang ditetapkan indeks.
Indeks sekunder lokal	Kapasitas baca yang ditetapkan tabel dasar.

> **Note**
>
> Akses lintas akun untuk operasi pemindaian indeks sekunder saat ini tidak didukung dengan kebijakan berbasis sumber daya.

Secara default, operasi `Scan` tidak menampilkan data tentang berapa banyak kapasitas baca yang digunakan. Namun, Anda dapat menentukan parameter `ReturnConsumedCapacity` dalam sebuah permintaan `Scan` untuk mendapatkan informasi ini. Berikut ini adalah pengaturan yang valid untuk `ReturnConsumedCapacity`:

- `NONE` — Tidak ada penggunaan data kapasitas yang ditampilkan. (Ini menjadi opsi default.)
- `TOTAL` — Respons mencakup jumlah agregat unit kapasitas baca yang digunakan.
- `INDEXES` — Respons menunjukkan jumlah agregat unit kapasitas baca yang digunakan, beserta dengan kapasitas yang digunakan untuk setiap tabel dan indeks yang diakses.

DynamoDB menghitung jumlah unit kapasitas baca yang dikonsumsi berdasarkan jumlah item dan ukuran item tersebut, bukan pada jumlah data yang dikembalikan ke aplikasi. Dengan alasan ini, jumlah unit kapasitas yang digunakan adalah sama, entah Anda meminta semua atribut (perilaku default) atau hanya beberapa dari atribut tersebut (menggunakan ekspresi proyeksi). Jumlahnya juga sama apakah Anda menggunakan ekspresi filter atau tidak. `Scan`mengkonsumsi unit kapasitas baca minimum untuk melakukan satu pembacaan yang sangat konsisten per detik, atau dua pembacaan

yang konsisten per detik untuk item hingga 4 KB. Jika Anda perlu membaca item yang lebih besar dari 4 KB, DynamoDB memerlukan unit permintaan baca tambahan. Tabel kosong dan tabel yang sangat besar yang memiliki jumlah kunci partisi yang jarang mungkin melihat beberapa tambahan yang RCUs dibebankan melebihi jumlah data yang dipindai. Ini mencakup biaya melayani `Scan` permintaan, bahkan jika tidak ada data.

Konsistensi baca untuk kueri

Operasi `Scan` melakukan bacaan akhir konsisten, secara default. Ini berarti bahwa hasil `Scan` mungkin tidak mencerminkan perubahan karena operasi `PutItem` atau `UpdateItem` baru saja selesai. Untuk informasi selengkapnya, lihat DynamoDB membaca konsistensi.

Jika Anda memerlukan bacaan sangat konsisten, pada saat `Scan` dimulai, atur parameter `ConsistentRead` menjadi `true` dalam permintaan `Scan`. Hal ini memastikan bahwa semua operasi tulis yang diselesaikan sebelum `Scan` dimulai termasuk dalam respons `Scan`.

Mengatur `ConsistentRead` ke `true` dapat berguna dalam skenario pencadangan atau replikasi tabel, dalam hubungannya dengan DynamoDB Streams. Anda pertama kali menggunakan `Scan` dengan `ConsistentRead` yang diatur ke true untuk mendapatkan salinan data yang konsisten dalam tabel. Selama `Scan`, DynamoDB Streams mencatat aktivitas tulis tambahan yang terjadi pada tabel. Setelah `Scan` selesai, Anda dapat menerapkan aktivitas tulis dari stream ke table.

> **Note**
>
> Operasi `Scan` dengan `ConsistentRead` yang diatue ke `true` mengonsumsi dua kali lebih banyak unit kapasitas baca dibandingkan dengan meninggalkan `ConsistentRead` pada nilai defaultnya (`false`).

Pemindaian paralel

Secara default, operasi `Scan` memproses data secara berurutan. Amazon DynamoDB mengembalikan data ke aplikasi dalam kenaikan 1 MB, dan aplikasi melakukan operasi `Scan` tambahan untuk mengambil 1 MB data berikutnya.

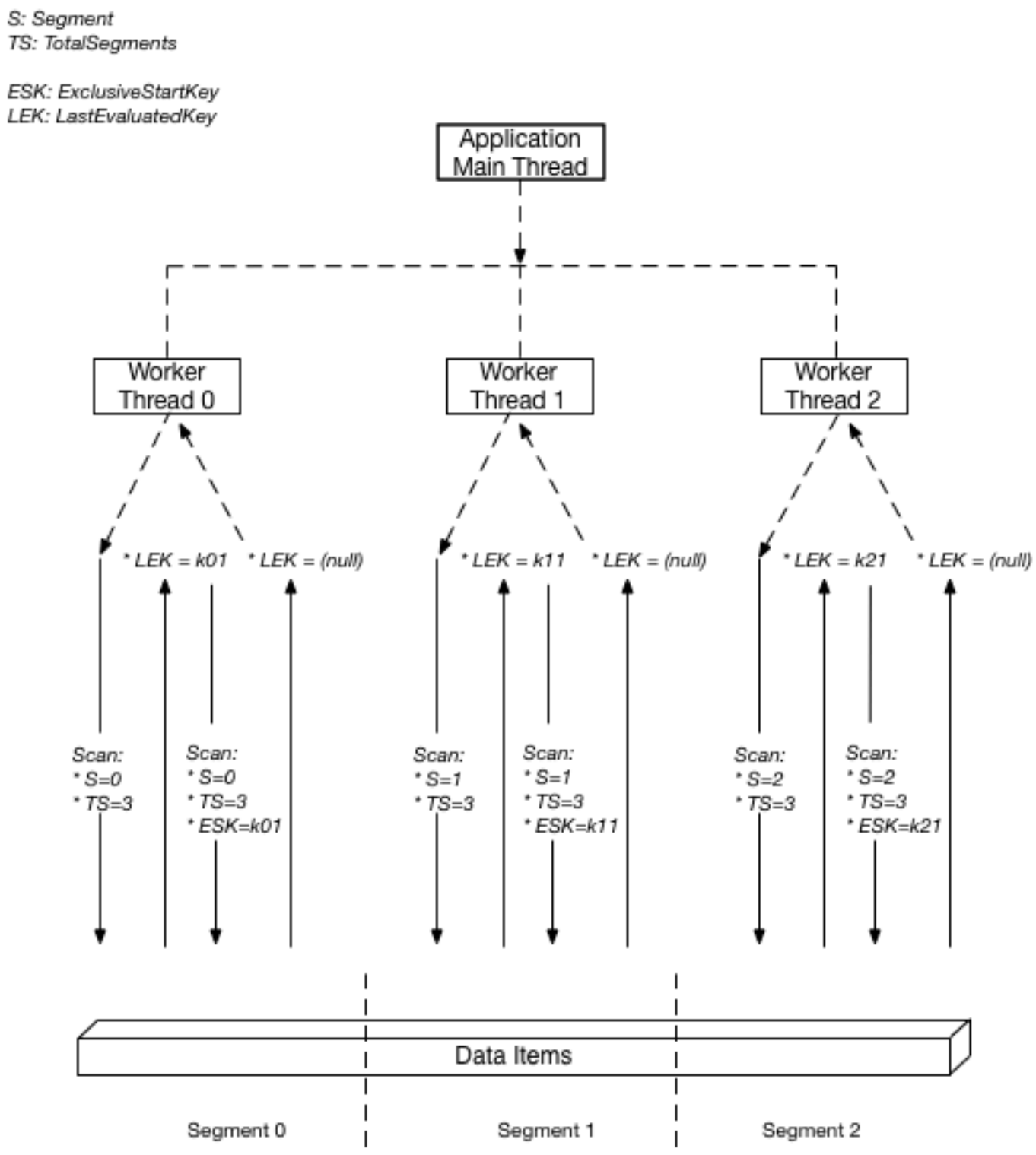
Makin besar tabel atau indeks yang dipindai, makin banyak waktu yang diperlukan untuk menyelesaikan `Scan`. Selain itu, `Scan` berurutan mungkin tidak selalu dapat sepenuhnya menggunakan kapasitas throughput baca yang disediakan: Meskipun DynamoDB mendistribusikan

data tabel besar ke beberapa partisi fisik, operasi `Scan` hanya dapat membaca satu partisi dalam satu waktu. Karena alasan ini, throughput `Scan` dibatasi oleh throughput maksimum dari satu partisi.

Untuk mengatasi masalah ini, operasi `Scan` dapat secara logis membagi tabel atau indeks sekunder menjadi beberapa segmen, dengan beberapa pekerja aplikasi memindai segmen tersebut secara paralel. Setiap pekerja dapat berupa thread (dalam bahasa pemrograman yang mendukung multithreading) atau proses sistem operasi. Untuk melakukan pemindaian paralel, setiap pekerja mengeluarkan permintaan `Scan`-nya sendiri dengan parameter berikut:

- `Segment` — Segmen yang akan dipindai oleh pekerja tertentu. Setiap pekerja harus menggunakan nilai yang berbeda untuk `Segment`.
- `TotalSegments` — Jumlah total segmen untuk pemindaian paralel. Nilai ini harus sama dengan jumlah pekerja yang akan digunakan aplikasi Anda.

Diagram berikut menunjukkan bagaimana aplikasi multithreaded melakukan `Scan` paralel dengan tiga derajat paralelisme.



Dalam diagram ini, aplikasi memunculkan tiga thread dan memberikan angka pada setiap thread. (Segmen berbasis nol, sehingga angka pertama selalu 0.) Setiap thread menerbitkan permintaan `Scan`, mengatur `Segment` ke angka yang ditunjuk dan mengatur `TotalSegments` ke 3. Setiap

thread memindai segmen yang ditunjuk, mengambil data 1 MB pada satu waktu, dan mengembalikan data ke thread utama aplikasi.

Nilai untuk `Segment` dan `TotalSegments` diterapkan untuk permintaan `Scan` individual, dan Anda dapat menggunakan nilai yang berbeda setiap saat. Anda mungkin perlu bereksperimen dengan nilai-nilai ini, dan jumlah pekerja yang Anda gunakan, sampai aplikasi Anda mencapai performa terbaiknya.

> **Note**
>
> Pemindaian paralel dengan sejumlah besar pekerja dapat dengan mudah menggunakan semua throughput yang disediakan untuk tabel atau indeks yang sedang dipindai. Sebaiknya hindari pemindaian seperti itu jika tabel atau indeks juga mengalami aktivitas baca atau tulis yang berat dari aplikasi lain.
> Untuk mengontrol jumlah data yang dikembalikan per permintaan, gunakan parameter `Limit`. Hal ini dapat membantu mencegah situasi di mana satu pekerja mengkonsumsi semua throughput yang ditetapkan, dengan mengorbankan semua pekerja lainnya.

PartiQL - Bahasa kueri yang kompatibel dengan SQL untuk Amazon DynamoDB

Amazon DynamoDB mendukung PartiQL, bahasa kueri yang kompatibel dengan SQL untuk memilih, menyisipkan, memperbarui, dan menghapus data di Amazon DynamoDB. Menggunakan PartiQL, Anda dapat dengan mudah berinteraksi dengan tabel DynamoDB dan menjalankan kueri ad hoc menggunakan, AWS Management Console NoSQL Workbench,, dan DynamoDB untuk PartiQL. AWS Command Line Interface APIs

Operasi PartiQL memberikan ketersediaan, latensi, dan performa yang sama seperti operasi bidang data DynamoDB lainnya.

Bagian berikut menjelaskan implementasi DynamoDB PartiQL.

Topik

- Apa yang dimaksud dengan PartiQL?
- PartiQL di Amazon DynamoDB
- Mulai menggunakan PartiQL untuk DynamoDB

- Jenis data PartiQL untuk DynamoDB
- Pernyataan PartiQL untuk DynamoDB
- Gunakan fungsi PartiQL dengan DynamoDB
- Aritmatika PartiQL, perbandingan, dan operator logika untuk DynamoDB
- Melakukan transaksi dengan PartiQL untuk DynamoDB
- Menjalankan operasi batch dengan PartiQL untuk DynamoDB
- Kebijakan keamanan IAM dengan PartiQL untuk DynamoDB

Apa yang dimaksud dengan PartiQL?

PartiQL menyediakan akses kueri yang kompatibel dengan SQL di beberapa penyimpanan data yang berisi data terstruktur, data semi terstruktur, dan data bersarang. Ini banyak digunakan di Amazon dan sekarang tersedia sebagai bagian dari banyak AWS layanan, termasuk DynamoDB.

Untuk spesifikasi PartiQL dan tutorial tentang bahasa kueri inti, lihat Dokumentasi PartiQL.

> **Note**
>
> - Amazon DynamoDB mendukung subset dari bahasa kueri PartiQL.
> - Amazon DynamoDB tidak mendukung format data Amazon ion atau literal Amazon Ion.

PartiQL di Amazon DynamoDB

Untuk menjalankan kueri PartiQL di DynamoDB, Anda dapat menggunakan:

- Konsol DynamoDB
- NoSQL Workbench
- AWS Command Line Interface (AWS CLI)
- DynamoDB APIs

Untuk informasi tentang cara menggunakan metode ini guna mengakses DynamoDB, lihat Mengakses DynamoDB.

Mulai menggunakan PartiQL untuk DynamoDB

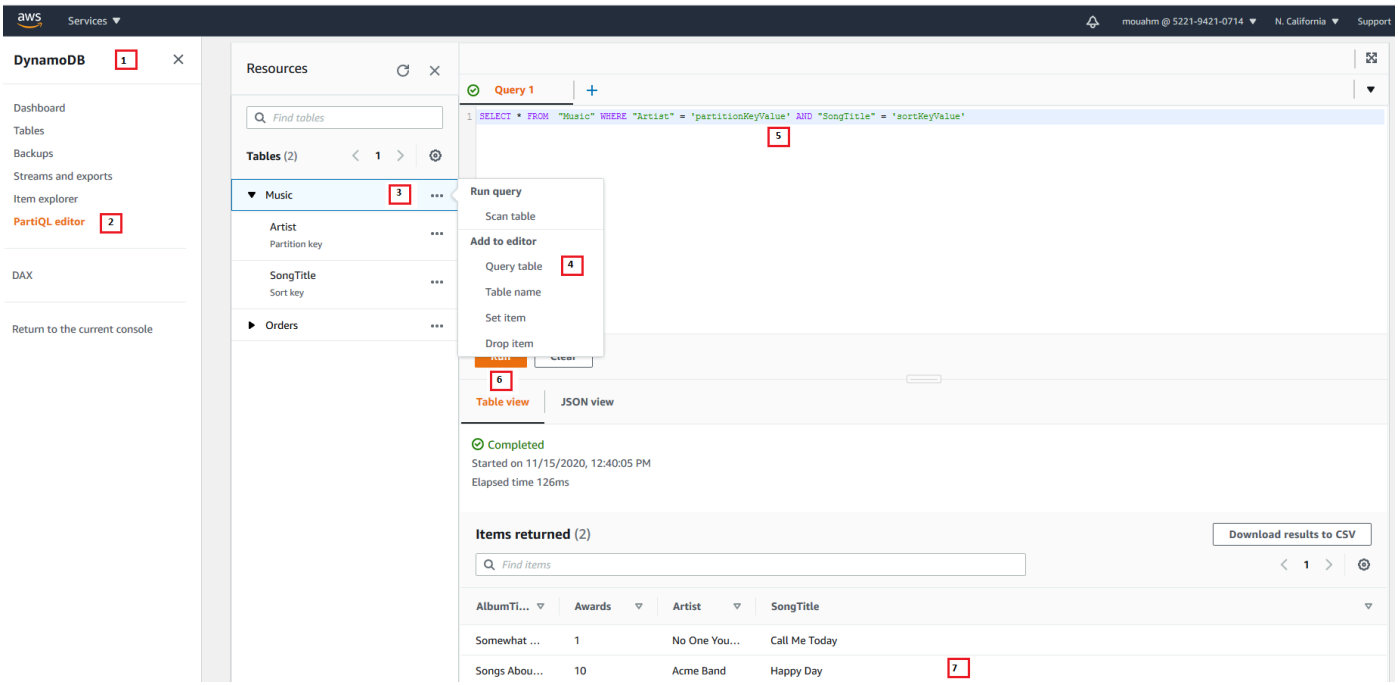
Bagian ini menjelaskan cara menggunakan PartiQL untuk DynamoDB dari konsol Amazon DynamoDB, (), dan DynamoDB. AWS Command Line Interface AWS CLI APIs

Dalam contoh berikut, tabel DynamoDB yang ditentukan dalam tutorial Mulai menggunakan DynamoDB adalah prasyarat.

Untuk informasi tentang menggunakan konsol DynamoDB, AWS Command Line Interface, atau DynamoDB untuk mengakses DynamoDB, lihat Mengakses APIs DynamoDB.

Untuk mengunduh dan menggunakan NoSQL workbench untuk membuat pernyataan PartiQL untuk DynamoDB, pilih operasi PartiQL di sudut kanan atas pembangun Operasi NoSQL Workbench untuk DynamoDB.

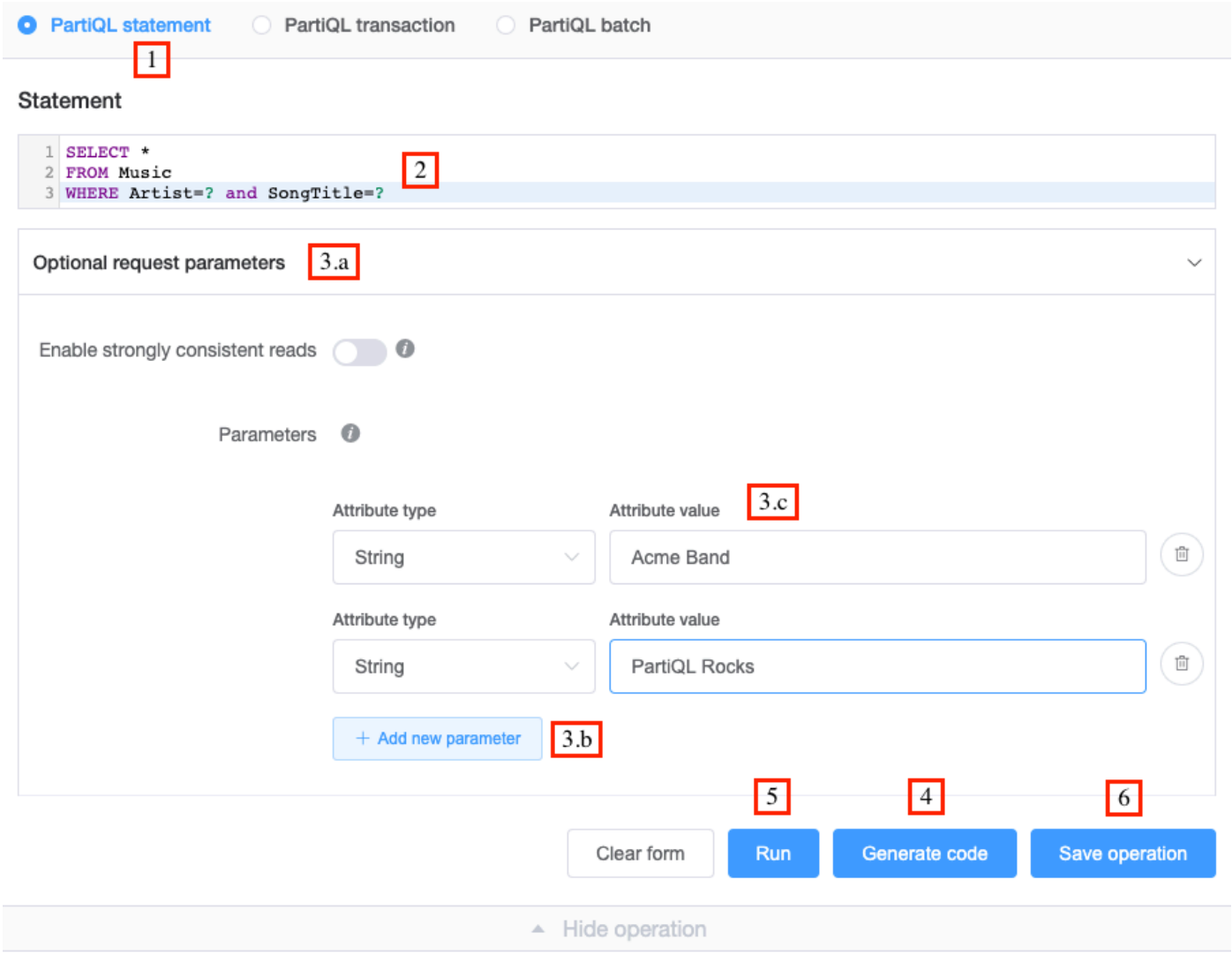
Console



1. Masuk ke AWS Management Console dan buka konsol DynamoDB di. https://console.aws.amazon.com/dynamodb/
2. Di panel navigasi di sisi kiri konsol, pilih editor PartiQL.
3. Pilih tabel Musik.
4. Pilih tabel Kueri. Tindakan ini menghasilkan kueri yang tidak akan menghasilkan pemindaian tabel penuh.

5. Ganti `partitionKeyValue` dengan nilai string `Acme Band`. Ganti `sortKeyValue` dengan nilai string `Happy Day`.
6. Pilih tombol Jalankan.
7. Anda dapat melihat hasil kueri dengan memilih tombol Tampilan tabel atau Tampilan JSON.

NoSQL workbench



1. Pilih pernyataan PartiQL.
2. Masukkan pernyataan SELECT PartiQL berikut

```
SELECT *
FROM Music
WHERE Artist=? and SongTitle=?
```

3. Untuk menentukan nilai parameter `Artist` dan `SongTitle`:

   a. Pilih Parameter permintaan opsional.

   b. Pilih Tambahkan parameter baru.

   c. Pilih jenis atribut string dan nilai `Acme Band`.

   d. Ulangi langkah b dan c, lalu pilih jenis string dan nilai `PartiQL Rocks`.

4. Jika Anda ingin membuat kode, pilih Buat kode.

   Pilih bahasa yang Anda inginkan dari tab yang ditampilkan. Sekarang Anda dapat menyalin kode ini dan menggunakannya dalam aplikasi Anda.

5. Jika Anda ingin operasi segera dijalankan, pilih Jalankan.

AWS CLI

1. Buat item dalam tabel `Music` menggunakan pernyataan PartiQL INSERT.

```
aws dynamodb execute-statement --statement "INSERT INTO Music  \
         VALUE  \
         {'Artist':'Acme Band','SongTitle':'PartiQL Rocks'}"
```

2. Ambil item dari tabel Musik menggunakan pernyataan PartiQL SELECT.

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

3. Perbarui item dalam tabel `Music` menggunakan pernyataan PartiQL UPDATE.

```
aws dynamodb execute-statement --statement "UPDATE Music  \
                                            SET AwardsWon=1  \
                                            SET AwardDetail={'Grammys':[2020,
 2018]}  \
                                            WHERE Artist='Acme Band' AND
 SongTitle='PartiQL Rocks'"
```

   Tambahkan nilai daftar untuk item dalam tabel `Music`.

```
aws dynamodb execute-statement --statement "UPDATE Music  \
```

```
                                            SET AwardDetail.Grammys
 =list_append(AwardDetail.Grammys,[2016])  \
                                            WHERE Artist='Acme Band' AND
 SongTitle='PartiQL Rocks'"
```

Hapus nilai daftar untuk item dalam tabel `Music`.

```
aws dynamodb execute-statement --statement "UPDATE Music  \
                                            REMOVE AwardDetail.Grammys[2]  \
                                            WHERE Artist='Acme Band' AND
 SongTitle='PartiQL Rocks'"
```

Tambahkan anggota peta baru untuk item dalam tabel `Music`.

```
aws dynamodb execute-statement --statement "UPDATE Music  \
                                            SET AwardDetail.BillBoard=[2020]  \
                                            WHERE Artist='Acme Band' AND
 SongTitle='PartiQL Rocks'"
```

Tambahkan atribut set string baru untuk item dalam tabel `Music`.

```
aws dynamodb execute-statement --statement "UPDATE Music  \
                                            SET BandMembers =<<'member1',
 'member2'>>  \
                                            WHERE Artist='Acme Band' AND
 SongTitle='PartiQL Rocks'"
```

Perbarui atribut set string untuk item dalam tabel `Music`.

```
aws dynamodb execute-statement --statement "UPDATE Music  \
                                            SET BandMembers
 =set_add(BandMembers, <<'newmember'>>)  \
                                            WHERE Artist='Acme Band' AND
 SongTitle='PartiQL Rocks'"
```

4. Hapus item dari tabel `Music` menggunakan pernyataan PartiQL DELETE.

```
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 com.amazonaws.services.dynamodbv2.AmazonDynamoDB;
import com.amazonaws.services.dynamodbv2.AmazonDynamoDBClientBuilder;
import com.amazonaws.services.dynamodbv2.model.AttributeValue;
import com.amazonaws.services.dynamodbv2.model.ConditionalCheckFailedException;
import com.amazonaws.services.dynamodbv2.model.ExecuteStatementRequest;
import com.amazonaws.services.dynamodbv2.model.ExecuteStatementResult;
import com.amazonaws.services.dynamodbv2.model.InternalServerErrorException;
import
 com.amazonaws.services.dynamodbv2.model.ItemCollectionSizeLimitExceededException;
import
 com.amazonaws.services.dynamodbv2.model.ProvisionedThroughputExceededException;
import com.amazonaws.services.dynamodbv2.model.RequestLimitExceededException;
import com.amazonaws.services.dynamodbv2.model.ResourceNotFoundException;
import com.amazonaws.services.dynamodbv2.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());
        }
```

```
    }

}
```

Jenis data PartiQL untuk DynamoDB

Tabel berikut mencantumkan jenis data yang dapat Anda gunakan dengan PartiQL untuk DynamoDB.

Jenis data DynamoDB	Representasi PartiQL	Catatan
`Boolean`	TRUE | FALSE	Tidak peka terhadap huruf besar-kecil.
`Binary`	N/A	Hanya didukung melalui kode.
`List`	[value1, value2,...]	Tidak ada batasan pada jenis daya yang dapat disimpan dalam elemen daftar, dan elemen dalam elemen daftar tidak harus berjenis sama.
`Map`	{ 'name' : value }	Tidak ada batasan pada jenis data yang dapat disimpan dalam elemen peta, dan elemen dalam peta tidak harus berjenis sama.
`Null`	NULL	Tidak peka terhadap huruf besar-kecil.
`Number`	1, 1.0, 1e0	Angka bisa positif, negatif, atau nol. Angka dapat memiliki hingga 38 digit presisi.
`Number Set`	<<number1, number2>>	Elemen dalam sejumlah set harus dari berjenis Angka.

Jenis data DynamoDB	Representasi PartiQL	Catatan
`String Set`	<<'string1', 'string2'>>	Elemen dalam suatu set string harus berjenis String.
`String`	‘nilai string’	Tanda kutip tunggal harus digunakan untuk menentukan nilai String.

Contoh

Pernyataan berikut menunjukkan cara memasukkan jenis data berikut: `String`, `Number`, `Map`, `List`, `Number Set`, dan `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'>>
}
```

Pernyataan berikut menunjukkan cara menyisipkan elemen baru ke dalam jenis `Map`, `List`, `Number Set`, and `String Set` serta mengubah nilai jenis `Number`.

```
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'
```

Pernyataan berikut menunjukkan cara menghapus elemen dari jenis `Map`, `List`, `Number Set`, and `String Set` serta mengubah nilai jenis `Number`.

```
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'
```

Untuk informasi selengkapnya, lihat jenis data DynamoDB.

Pernyataan PartiQL untuk DynamoDB

Amazon DynamoDB mendukung pernyataan PartiQL berikut.

> **Note**
>
> DynamoDB tidak mendukung semua pernyataan PartiQL.
> Referensi ini memberikan sintaks dasar dan contoh penggunaan pernyataan PartiQL yang Anda jalankan secara manual menggunakan or. AWS CLI APIs

Bahasa manipulasi data (DML) adalah kumpulan pernyataan PartiQL yang Anda gunakan untuk mengelola data dalam tabel DynamoDB. Anda menggunakan pernyataan DML untuk menambah, mengubah, atau menghapus data dalam sebuah tabel.

Berikut DML dan kueri bahasa pernyataan yang didukung:

- Pernyataan pemilihan PartiQL untuk DynamoDB
- Pernyataan pembaruan PartiQL untuk DynamoDB
- Pernyataan sisipkan PartiQL untuk DynamoDB
- Pernyataan hapus PartiQL untuk DynamoDB

Melakukan transaksi dengan PartiQL untuk DynamoDB and Menjalankan operasi batch dengan PartiQL untuk DynamoDB juga didukung oleh PartiQL untuk DynamoDB.

Pernyataan pemilihan PartiQL untuk DynamoDB

Gunakan pernyataan `SELECT` untuk mengambil data dari tabel di Amazon DynamoDB.

Menggunakan `SELECT` pernyataan dapat menghasilkan pemindaian tabel lengkap jika kondisi kesetaraan atau IN dengan kunci partisi tidak disediakan dalam klausa WHERE. Operasi pemindaian memeriksa setiap item untuk nilai yang diminta dan dapat menggunakan throughput yang disediakan untuk indeks atau tabel besar dalam satu operasi.

Jika ingin menghindari pemindaian tabel secara lengkap di PartiQL, Anda dapat:

- Menulis pernyataan SELECT Anda agar tidak memindai tabel secara lengkap dengan memastikan ketentuan klausul WHERE Anda dikonfigurasi dengan semestinya.
- Menonaktifkan pemindaian tabel lengkap menggunakan kebijakan IAM yang ditentukan di Contoh: Mengizinkan pernyataan pilihan dan menolak pernyataan pemindaian tabel lengkap di PartiQL untuk DynamoDB, dalam panduan developer DynamoDB.

Untuk informasi selengkapnya, lihat Praktik terbaik untuk Mengkueri dan memindai data, dalam panduan developer DynamoDB.

Topik

- Sintaks
- Parameter
- Contoh

Sintaks

```
SELECT expression  [, ...]
FROM table[.index]
[ WHERE condition ] [ [ORDER BY key [DESC|ASC] , ...]
```

Parameter

expression

(Diperlukan) Proyeksi yang terbentuk dari wildcard * atau daftar proyeksi dari satu nama atribut atau lebih atau jalur dokumen dari set hasil. Ekspresi dapat terdiri dari panggilan ke Gunakan fungsi PartiQL dengan DynamoDB atau bidang yang diubah oleh Aritmatika PartiQL, perbandingan, dan operator logika untuk DynamoDB .

table

(Diperlukan) Nama tabel untuk kueri.

index

(Opsional) Nama indeks untuk kueri.

> **Note**
>
> Anda harus menambahkan tanda kutip ganda ke nama tabel dan nama indeks saat mengkueri indeks.
>
> ```
> SELECT *
> FROM "TableName"."IndexName"
> ```

condition

(Opsional) Kriteria seleksi untuk kueri.

> **Important**
>
> Untuk memastikan bahwa pernyataan `SELECT` tidak menghasilkan pemindaian tabel secara lengkap, kondisi klausul `WHERE` harus menentukan kunci partisi. Gunakan kesetaraan atau operator IN.
> Misalnya, jika Anda memiliki tabel `Orders` dengan kunci partisi `OrderID` dan atribut non-kunci lainnya, termasuk `Address`, pernyataan berikut tidak akan menghasilkan pemindaian tabel secara lengkap:
>
> ```
> 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]
> ```
>
> Namun, pernyataan `SELECT` berikut akan menghasilkan pemindaian tabel secara lengkap:

```
SELECT *
FROM "Orders"
WHERE OrderID > 1

SELECT *
FROM "Orders"
WHERE Address='some address'

SELECT *
FROM "Orders"
WHERE OrderID = 100 OR Address='some address'
```

key

(Opsional) Kunci hash atau kunci urutan untuk digunakan dalam mengurutkan hasil yang dikembalikan. Urutan defaultnya adalah naik (ASC) tentukan DESC jika Anda ingin hasil dikembalikan dalam urutan turun.

> **Note**
>
> Jika Anda menghapus klausul WHERE, semua item dalam tabel akan diambil.

Contoh

Kueri berikut mengembalikan satu item, jika ada, dari tabel `Orders` dengan menentukan kunci partisi, `OrderID`, dan menggunakan operator kesetaraan.

```
SELECT OrderID, Total
FROM "Orders"
WHERE OrderID = 1
```

Kueri berikut mengembalikan semua item dalam tabel `Orders` yang memiliki kunci partisi spesifik, `OrderID`, nilai menggunakan operator OR.

```
SELECT OrderID, Total
FROM "Orders"
```

```
WHERE OrderID = 1 OR OrderID = 2
```

Kueri berikut mengembalikan semua item dalam tabel `Orders` yang memiliki kunci partisi spesifik, `OrderID`, nilai menggunakan operator IN. Hasil yang dikembalikan dalam urutan turun, berdasarkan nilai atribut kunci `OrderID`.

```
SELECT OrderID, Total
FROM "Orders"
WHERE OrderID IN [1, 2, 3] ORDER BY OrderID DESC
```

Kueri berikut menunjukkan pemindaian tabel secara lengkap yang mengembalikan semua item dari tabel `Orders` yang memiliki `Total` lebih dari 500, dengan `Total` adalah atribut non-kunci.

```
SELECT OrderID, Total
FROM "Orders"
WHERE Total > 500
```

Kueri berikut menunjukkan pemindaian tabel secara lengkap yang mengembalikan semua item dari tabel `Orders` di dalam rentang urutan `Total` tertentu, menggunakan operator IN dan atribut non-kunci `Total`.

```
SELECT OrderID, Total
FROM "Orders"
WHERE Total IN [500, 600]
```

Kueri berikut menunjukkan pemindaian tabel secara lengkap yang mengembalikan semua item dari tabel `Orders` di dalam rentang urutan `Total` tertentu, menggunakan operator BETWEEN dan atribut non-kunci `Total`.

```
SELECT OrderID, Total
FROM "Orders"
WHERE Total BETWEEN 500 AND 600
```

Kueri berikut mengembalikan tanggal pertama perangkat firestick digunakan untuk menonton dengan menentukan kunci partisi `CustomerID` dan kunci urutan `MovieID` dalam kondisi klausul WHERE dan menggunakan jalur dokumen dalam klausul SELECT.

```
SELECT Devices.FireStick.DateWatched[0]
```

```
FROM WatchList
WHERE CustomerID= 'C1' AND MovieID= 'M1'
```

Kueri berikut menunjukkan pemindaian tabel secara penuh yang mengembalikan daftar item tempat perangkat firestick pertama kali digunakan setelah 24/12/19 menggunakan jalur dokumen dalam kondisi klausul WHERE.

```
SELECT Devices
FROM WatchList
WHERE Devices.FireStick.DateWatched[0] >= '12/24/19'
```

Pernyataan pembaruan PartiQL untuk DynamoDB

Gunakan pernyataan UPDATE untuk mengubah nilai dari satu atribut atau lebih dalam item dalam tabel Amazon DynamoDB.

> **Note**
>
> Anda hanya dapat memperbarui satu item pada satu waktu; Anda tidak dapat mengeluarkan pernyataan DynamoDB PartiQL tunggal yang memperbarui beberapa item. Untuk informasi tentang memperbarui beberapa item, lihat Melakukan transaksi dengan PartiQL untuk DynamoDB atau Menjalankan operasi batch dengan PartiQL untuk DynamoDB.

Topik

- Sintaks
- Parameter
- Nilai yang dikembalikan
- Contoh

Sintaks

```
UPDATE  table
[SET | REMOVE]  path  [=  data] […]
WHERE condition [RETURNING returnvalues]
<returnvalues>  ::= [ALL OLD | MODIFIED OLD | ALL NEW | MODIFIED NEW] *
```

Parameter

table

(Diperlukan) Tabel yang berisi data yang akan diubah.

path

(Diperlukan) Nama atribut atau jalur dokumen yang akan dibuat atau diubah.

data

(Diperlukan) Nilai atribut atau hasil operasi.

Operasi yang didukung untuk digunakan bersama SET:

- LIST_APPEND: menambahkan nilai ke jenis daftar.
- SET_ADD: menambahkan nilai ke set angka atau string.
- SET_DELETE: menghapus nilai dari set angka atau string.

condition

(Diperlukan) Kriteria pemilihan untuk item yang akan diubah. Syarat ini harus diselesaikan untuk satu nilai kunci primer.

returnvalues

(Opsional) Gunakan `returnvalues` jika Anda ingin mendapatkan atribut item seperti yang muncul sebelum atau setelah diperbarui. Nilai yang valid adalah:

- `ALL OLD *` - Mengembalikan semua atribut item, saat atribut item tersebut muncul sebelum operasi pembaruan.
- `MODIFIED OLD *` - Mengembalikan hanya atribut yang diperbarui, saat atribut item tersebut muncul sebelum operasi pembaruan.
- `ALL NEW *` - Mengembalikan semua atribut item, seperti yang muncul setelah operasi pembaruan.
- `MODIFIED NEW *` - Hanya mengembalikan atribut yang diperbarui, seperti yang muncul setelah operasi `UpdateItem`.

Nilai yang dikembalikan

Pernyataan ini tidak mengembalikan nilai kecuali parameter `returnvalues` ditentukan.

> **Note**
>
> Jika klausul WHERE pernyataan UPDATE tidak dinilai true untuk item apa pun dalam tabel DynamoDB, `ConditionalCheckFailedException` akan dikembalikan.

Contoh

Memperbarui nilai atribut dalam item yang sudah ada. Jika atribut tersebut tidak ada, atribut tersebut akan dibuat.

Kueri berikut memperbarui item dalam tabel `"Music"` dengan menambahkan atribut dari nomor jenis (`AwardsWon`) dan atribut dari jenis peta (`AwardDetail`).

```
UPDATE "Music"
SET AwardsWon=1
SET AwardDetail={'Grammys':[2020, 2018]}
WHERE Artist='Acme Band' AND SongTitle='PartiQL Rocks'
```

Anda dapat menambahkan `RETURNING ALL OLD *` untuk mengembalikan atribut seperti yang muncul sebelum operasi `Update`.

```
UPDATE "Music"
SET AwardsWon=1
SET AwardDetail={'Grammys':[2020, 2018]}
WHERE Artist='Acme Band' AND SongTitle='PartiQL Rocks'
RETURNING ALL OLD *
```

Ini akan menghasilkan hal berikut:

```
{
    "Items": [
        {
            "Artist": {
                "S": "Acme Band"
            },
            "SongTitle": {
                "S": "PartiQL Rocks"
            }
        }
    ]
```

```
}
```

Anda dapat menambahkan `RETURNING ALL NEW *` untuk mengembalikan atribut seperti yang muncul setelah operasi `Update`.

```
UPDATE "Music"
SET AwardsWon=1
SET AwardDetail={'Grammys':[2020, 2018]}
WHERE Artist='Acme Band' AND SongTitle='PartiQL Rocks'
RETURNING ALL NEW *
```

Ini akan menghasilkan hal berikut:

```
{
    "Items": [
        {
            "AwardDetail": {
                "M": {
                    "Grammys": {
                        "L": [
                            {
                                "N": "2020"
                            },
                            {
                                "N": "2018"
                            }
                        ]
                    }
                }
            },
            "AwardsWon": {
                "N": "1"
            }
        }
    ]
}
```

Kueri berikut memperbarui item dalam tabel `"Music"` dengan menambahkan ke daftar `AwardDetail.Grammys`.

```
UPDATE "Music"
SET AwardDetail.Grammys =list_append(AwardDetail.Grammys,[2016])
```

```
WHERE Artist='Acme Band' AND SongTitle='PartiQL Rocks'
```

Kueri berikut memperbarui item dalam tabel `"Music"` dengan menghapus dari daftar `AwardDetail.Grammys`.

```
UPDATE "Music"
REMOVE AwardDetail.Grammys[2]
WHERE Artist='Acme Band' AND SongTitle='PartiQL Rocks'
```

Kueri berikut memperbarui item dalam tabel `"Music"` dengan menambahkan `BillBoard` ke peta `AwardDetail`.

```
UPDATE "Music"
SET AwardDetail.BillBoard=[2020]
WHERE Artist='Acme Band' AND SongTitle='PartiQL Rocks'
```

Kueri berikut memperbarui item dalam tabel `"Music"` dengan menambahkan atribut set string `BandMembers`.

```
UPDATE "Music"
SET BandMembers =<<'member1', 'member2'>>
WHERE Artist='Acme Band' AND SongTitle='PartiQL Rocks'
```

Kueri berikut memperbarui item dalam tabel `"Music"` dengan menambahkan `newbandmember` ke set string `BandMembers`.

```
UPDATE "Music"
SET BandMembers =set_add(BandMembers, <<'newbandmember'>>)
WHERE Artist='Acme Band' AND SongTitle='PartiQL Rocks'
```

Pernyataan hapus PartiQL untuk DynamoDB

Gunakan pernyataan `DELETE` untuk menghapus item yang sudah ada dari tabel Amazon DynamoDB Anda.

> **Note**
>
> Anda hanya dapat menghapus satu item dalam satu waktu. Anda tidak dapat mengeluarkan pernyataan DynamoDB PartiQL tunggal yang menghapus beberapa item. Untuk informasi

tentang menghapus beberapa item, lihat Melakukan transaksi dengan PartiQL untuk DynamoDB atau Menjalankan operasi batch dengan PartiQL untuk DynamoDB.

Topik

- Sintaks
- Parameter
- Nilai yang dikembalikan
- Contoh

Sintaks

```
DELETE FROM table
 WHERE condition [RETURNING returnvalues]
 <returnvalues>  ::= ALL OLD *
```

Parameter

table

(Diperlukan) Tabel DynamoDB yang berisi item yang akan dihapus.

condition

(Diperlukan) Kriteria seleksi untuk item yang akan dihapus; syarat ini harus diselesaikan untuk nilai kunci primer tunggal.

returnvalues

(Opsional) Gunakan `returnvalues` jika Anda ingin mendapatkan atribut item seperti yang muncul sebelum atribut item tersebut dihapus. Nilai yang valid adalah:

- `ALL OLD *` - Konten dari item lama akan dikembalikan.

Nilai yang dikembalikan

Pernyataan ini tidak mengembalikan nilai kecuali parameter `returnvalues` ditentukan.

> **Note**
>
> Jika tabel DynamoDB tidak memiliki item dengan kunci primer yang sama seperti yang item yang DELETE-nya dikeluarkan, SUCCESS dikembalikan dengan 0 item dihapus. Jika tabel memiliki item dengan kunci primer yang sama, tetapi syarat dalam klausul WHERE pernyataan dari DELETE bernilai false, `ConditionalCheckFailedException` akan dikembalikan.

Contoh

Kueri berikut akan menghapus item dalam tabel `"Music"`.

```
DELETE FROM "Music" WHERE "Artist" = 'Acme Band' AND "SongTitle" = 'PartiQL Rocks'
```

Anda dapat menambahkan parameter `RETURNING ALL OLD *` untuk mengembalikan data yang telah dihapus.

```
DELETE FROM "Music" WHERE "Artist" = 'Acme Band' AND "SongTitle" = 'PartiQL Rocks'
 RETURNING ALL OLD *
```

Pernyataan `Delete` sekarang mengembalikan yang berikut:

```
{
    "Items": [
        {
            "Artist": {
                "S": "Acme Band"
            },
            "SongTitle": {
                "S": "PartiQL Rocks"
            }
        }
    ]
}
```

Pernyataan sisipkan PartiQL untuk DynamoDB

Gunakan pernyataan `INSERT` untuk menambahkan item ke tabel di Amazon DynamoDB.

> **Note**
>
> Anda hanya dapat menyisipkan satu item pada satu waktu; Anda tidak dapat mengeluarkan pernyataan DynamoDB PartiQL tunggal yang menyisipkan beberapa item. Untuk informasi tentang cara menyisipkan beberapa item, lihat Melakukan transaksi dengan PartiQL untuk DynamoDB atau Menjalankan operasi batch dengan PartiQL untuk DynamoDB.

Topik

- Sintaks
- Parameter
- Nilai yang dikembalikan
- Contoh

Sintaks

Sisipkan satu item.

```
INSERT INTO table VALUE item
```

Parameter

table

(Diperlukan) Tabel tempat Anda ingin menyisipkan data. Tabel harus sudah ada.

item

(Diperlukan) Item DynamoDB yang valid diwakili sebagai tupel PartiQL. Anda harus menentukan hanya satu item dan setiap nama atribut dalam item adalah peka huruf besar-kecil dan dapat dilambangkan dengan tanda petik tunggal (`'...'`) di PartiQL.

Nilai string juga dilambangkan dengan tanda petik tunggal (`'...'`) di PartiQL.

Nilai yang dikembalikan

Pernyataan ini tidak mengembalikan nilai apa pun.

> **Note**
>
> Jika tabel DynamoDB sudah memiliki item dengan kunci primer yang sama sebagai kunci primer dari item yang sedang disisipkan, `DuplicateItemException` dikembalikan.

Contoh

```
INSERT INTO "Music" value {'Artist' : 'Acme Band','SongTitle' : 'PartiQL Rocks'}
```

Gunakan fungsi PartiQL dengan DynamoDB

PartiQL di Amazon DynamoDB mendukung varian bawaan fungsi standar SQL berikut.

> **Note**
>
> Fungsi SQL apa pun yang tidak termasuk dalam daftar ini saat ini tidak didukung di DynamoDB.

Fungsi agregat

- Menggunakan Fungsi SIZE dengan PartiQL untuk Amazon DynamoDB

Fungsi kondisional

- Menggunakan fungsi EXISTS dengan PartiQL untuk DynamoDB
- Menggunakan Fungsi ATTRIBUTE_TYPE dengan PartiQL untuk DynamoDB
- Menggunakan Fungsi BEGINS_WITH dengan PartiQL untuk DynamoDB
- Menggunakan fungsi COUNTAINS dengan PartiQL untuk DynamoDB
- Menggunakan fungsi MISSING dengan PartiQL untuk DynamoDB

Menggunakan fungsi EXISTS dengan PartiQL untuk DynamoDB

Anda dapat menggunakan EXISTS untuk melakukan fungsi yang sama seperti `ConditionCheck` di TransactWriteItemsAPI. Fungsi EXISTS hanya dapat digunakan dalam transaksi.

Dengan nilai, mengembalikan TRUE jika nilai adalah kumpulan yang tidak kosong. Jika tidak, mengembalikan FALSE.

> **Note**
>
> Fungsi ini hanya dapat digunakan dalam operasi transaksional.

Sintaks

```
EXISTS ( statement )
```

Pendapat

statement

(Diperlukan) Pernyataan SELECT yang dievaluasi oleh fungsi.

> **Note**
>
> Pernyataan SELECT harus menentukan kunci primer lengkap dan satu kondisi lainnya.

Jenis pengembalian

bool

Contoh

```
EXISTS(
    SELECT * FROM "Music"
    WHERE "Artist" = 'Acme Band' AND "SongTitle" = 'PartiQL Rocks')
```

Menggunakan Fungsi BEGINS_WITH dengan PartiQL untuk DynamoDB

Mengembalikan TRUE jika atribut yang ditentukan dimulai dengan substring tertentu.

Sintaks

```
begins_with(path, value )
```

Pendapat

path

(Diperlukan) Nama atribut atau jalur dokumen yang akan digunakan.

value

(Diperlukan) String untuk mencari.

Jenis pengembalian

bool

Contoh

```
SELECT * FROM "Orders" WHERE "OrderID"=1 AND begins_with("Address", '7834 24th')
```

Menggunakan fungsi MISSING dengan PartiQL untuk DynamoDB

Mengembalikan TRUE jika item tidak berisi atribut yang ditentukan. Hanya operator kesetaraan dan ketidaksetaraan yang dapat digunakan dengan fungsi ini.

Sintaks

```
 attributename IS | IS NOT  MISSING
```

Pendapat

attributename

(Diperlukan) Nama atribut yang akan dicari.

Jenis pengembalian

bool

Contoh

```
SELECT * FROM Music WHERE "Awards" is MISSING
```

Menggunakan Fungsi ATTRIBUTE_TYPE dengan PartiQL untuk DynamoDB

Mengembalikan TRUE jika atribut pada jalur yang ditentukan adalah jenis data tertentu.

Sintaks

```
attribute_type( attributename, type )
```

Pendapat

attributename

(Diperlukan) Nama atribut yang digunakan.

type

(Wajib) Jenis atribut yang akan diperiksa. Untuk daftar nilai yang valid, lihat attribute_type DynamoDB.

Jenis pengembalian

bool

Contoh

```
SELECT * FROM "Music" WHERE attribute_type("Artist", 'S')
```

Menggunakan fungsi COUNTAINS dengan PartiQL untuk DynamoDB

Mengembalikan TRUE jika atribut yang ditentukan oleh jalur tersebut adalah salah satu dari hal berikut:

- String yang berisi substring tertentu.
- Set yang berisi elemen tertentu dalam set.

Untuk informasi selengkapnya, lihat fungsi contains DynamoDB.

Sintaks

```
contains( path, substring )
```

Pendapat

path

(Diperlukan) Nama atribut atau jalur dokumen yang akan digunakan.

substring

(Diperlukan) Substring atribut atau anggota yang ditetapkan yang akan diperiksa. Untuk informasi selengkapnya, lihat fungsi contains DynamoDB.

Jenis pengembalian

`bool`

Contoh

```
SELECT * FROM "Orders" WHERE "OrderID"=1 AND contains("Address", 'Kirkland')
```

Menggunakan Fungsi SIZE dengan PartiQL untuk Amazon DynamoDB

Mengembalikan angka yang mewakili ukuran atribut dalam byte. Berikut ini adalah jenis daya yang valid untuk digunakan dengan size. Untuk informasi selengkapnya, lihat fungsi size DynamoDB.

Sintaks

```
size( path)
```

Pendapat

path

(Diperlukan) Nama atribut atau jalur dokumen.

Untuk jenis yang didukung, lihat fungsi size DynamoDB.

Jenis pengembalian

`int`

Contoh

```
SELECT * FROM "Orders" WHERE "OrderID"=1 AND size("Image") >300
```

Aritmatika PartiQL, perbandingan, dan operator logika untuk DynamoDB

PartiQL di Amazon DynamoDB mendukung Operator standar SQL berikut.

> **Note**
>
> Operator SQL apa pun yang tidak termasuk dalam daftar ini saat ini tidak didukung di DynamoDB.

Operator aritmatika

Operator	Deskripsi
+	Tambahkan
-	Kurangi

Operator perbandingan

Operator	Deskripsi
=	Sama dengan
<>	Tidak Sama dengan
!=	Tidak Sama dengan
>	Lebih besar dari
<	Kurang dari
>=	Lebih besar dari atau sama dengan
<=	Kurang dari atau sama dengan

Operator logis

Operator	Deskripsi
`AND`	`TRUE` jika semua kondisi yang dipisahkan oleh `AND` adalah `TRUE`
`BETWEEN`	`TRUE`jika operan berada dalam kisaran perbandingan. Operator ini termasuk batas bawah dan atas operan tempat Anda menerapkannya.
`IN`	`TRUE` jika operand sama dengan salah satu daftar ekspresi (maks 50 nilai atribut hash atau maksimal 100 nilai atribut non-kunci)
`IS`	`TRUE` jika operannya adalah jenis data PartiQL tertentu, termasuk `NULL` atau `MISSING`
`NOT`	Membalikkan nilai ekspresi Boolean yang diberikan
`OR`	`TRUE` jika semua kondisi yang dipisahkan oleh `OR` adalah `TRUE`

Untuk informasi selengkapnya tentang menggunakan operator logika, lihat Membuat perbandingan danEvaluasi logika.

Melakukan transaksi dengan PartiQL untuk DynamoDB

Bagian ini menjelaskan cara menggunakan transaksi dengan PartiQL untuk DynamoDB. Transaksi PartiQL dibatasi hingga 100 total pernyataan (tindakan).

Untuk informasi lebih lanjut tentang transaksi DynamoDB, lihat Mengelola alur kerja kompleks dengan transaksi DynamoDB.

> **Note**
>
> Seluruh transaksi harus terdiri dari pernyataan baca atau pernyataan tulis. Anda tidak dapat menggabungkan keduanya dalam satu transaksi. Fungsi EXISTS merupakan pengecualian. Anda dapat menggunakannya untuk memeriksa kondisi atribut tertentu dari item dengan cara yang mirip `ConditionCheck` dengan operasi TransactWriteItemsAPI.

Topik

- Sintaks
- Parameter
- Nilai yang dikembalikan
- Contoh

Sintaks

```
[
   {
      "Statement":" statement ",
      "Parameters":[
         {
            " parametertype " : " parametervalue "
         }, ...]
   } , ...
]
```

Parameter

statement

(Diperlukan) Pernyataan yang didukung PartiQL untuk DynamoDB.

> **Note**
>
> Seluruh transaksi harus terdiri dari pernyataan baca atau pernyataan tulis. Anda tidak dapat menggabungkan keduanya dalam satu transaksi.

parametertype

(Opsional) Jenis DynamoDB, jika parameter digunakan saat menentukan pernyataan PartiQL.

parametervalue

(Opsional) Nilai parameter jika parameter digunakan saat menentukan pernyataan PartiQL.

Nilai yang dikembalikan

Pernyataan ini tidak mengembalikan nilai apa pun untuk operasi Tulis (INSERT, UPDATE, atau DELETE). Namun, ia mengembalikan nilai yang berbeda untuk operasi Baca (SELECT) berdasarkan kondisi yang ditentukan dalam klausul WHERE.

> **Note**
>
> Jika salah satu operasi INSERT, UPDATE, atau DELETE singleton mengembalikan kesalahan, transaksi dibatalkan dengan pengecualian `TransactionCanceledException`, dan kode alasan pembatalan termasuk kesalahan dari operasi singleton individual.

Contoh

Contoh berikut menjalankan beberapa pernyataan sebagai sebuah transaksi.

AWS CLI

1. Simpan kode JSON berikut ke file bernama 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. Jalankan perintah berikut di prompt perintah.

```
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());
        }
    }

}
```

Contoh berikut menunjukkan nilai pengembalian yang berbeda ketika DynamoDB membaca item dengan kondisi berbeda yang ditentukan dalam klausul WHERE.

AWS CLI

1. Simpan kode JSON berikut ke file bernama 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. perintah berikut di prompt perintah.

```
aws dynamodb execute-transaction --transact-statements  file://partiql.json
```

3. Respons berikut dikembalikan:

```
{
    "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
        {}
    ]
}
```

Menjalankan operasi batch dengan PartiQL untuk DynamoDB

Bagian ini menjelaskan cara menggunakan pernyataan batch dengan PartiQL untuk DynamoDB.

> **Note**
>
> - Keseluruhan batch harus terdiri dari pernyataan baca atau pernyataan tulis; Anda tidak dapat mencampur keduanya dalam satu batch.
> - `BatchExecuteStatement` dan `BatchWriteItem` dapat melakukan tidak lebih dari 25 pernyataan per batch.

Topik

- Sintaks
- Parameter
- Contoh

Sintaks

```
[
   {
      "Statement":" statement ",
      "Parameters":[
         {
            " parametertype " : " parametervalue "
         }, ...]
   } , ...
]
```

Parameter

statement

(Diperlukan) Pernyataan yang didukung PartiQL untuk DynamoDB.

> **Note**
>
> - Keseluruhan batch harus terdiri dari pernyataan baca atau pernyataan tulis; Anda tidak dapat mencampur keduanya dalam satu batch.
> - `BatchExecuteStatement` dan `BatchWriteItem` dapat melakukan tidak lebih dari 25 pernyataan per batch.

parametertype

(Opsional) Jenis DynamoDB, jika parameter digunakan saat menentukan pernyataan PartiQL.

parametervalue

(Opsional) Nilai parameter jika parameter digunakan saat menentukan pernyataan PartiQL.

Contoh

AWS CLI

1. Simpan json berikut ke file bernama 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. Jalankan perintah berikut di prompt perintah.

```
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());
        }
    }

}
```

Kebijakan keamanan IAM dengan PartiQL untuk DynamoDB

Izin berikut diperlukan:

- Untuk membaca item menggunakan PartiQL untuk DynamoDB, Anda harus memiliki izin `dynamodb:PartiQLSelect` pada tabel atau indeks.
- Untuk menyisipkan item menggunakan PartiQL untuk DynamoDB, Anda harus memiliki izin `dynamodb:PartiQLInsert` pada tabel atau indeks.
- Untuk memperbarui item menggunakan PartiQL untuk DynamoDB, Anda harus memiliki izin `dynamodb:PartiQLUpdate` pada tabel atau indeks.
- Untuk menghapus item menggunakan PartiQL untuk DynamoDB, Anda harus memiliki izin `dynamodb:PartiQLDelete` pada tabel atau indeks.

Contoh: Izinkan semua PartiQL untuk pernyataan DynamoDB () di atas meja Select/Insert/Update/Delete

Kebijakan IAM berikut memberikan izin untuk menjalankan semua pernyataan PartiQL untuk DynamoDB pada tabel.

```
{
   "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"
         ]
      }
   ]
}
```

Contoh: Mengizinkan pernyataan pilih PartiQL untuk DynamoDB pada tabel

Kebijakan IAM berikut memberikan izin untuk menjalankan pernyataan `select` pada tabel tertentu.

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Effect":"Allow",
         "Action":[
            "dynamodb:PartiQLSelect"
         ],
         "Resource":[
            "arn:aws:dynamodb:us-west-2:123456789012:table/Music"
         ]
      }
   ]
}
```

Contoh: Mengizinkan PartiQL untuk pernyataan penyisipan DynamoDB pada indeks

Kebijakan IAM berikut memberikan izin untuk menjalankan pernyataan `insert` pada indeks tertentu.

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Effect":"Allow",
         "Action":[
            "dynamodb:PartiQLInsert"
         ],
         "Resource":[
            "arn:aws:dynamodb:us-west-2:123456789012:table/Music/index/index1"
         ]
      }
   ]
}
```

Contoh: Mengizinkan PartiQL untuk pernyataan transaksional DynamoDB hanya pada tabel

Kebijakan IAM berikut memberikan izin untuk hanya menjalankan pernyataan transaksional pada tabel tertentu.

```
{
   "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"
               ]
```

```
            }
         }
      }
   ]
}
```

Contoh: Mengizinkan PartiQL untuk pembacaan dan penulisan non-transaksional DynamoDB dan memblokir pernyataan transaksional pembacaan dan penulisan transaksional PartiQL pada tabel.

Kebijakan IAM berikut memberikan izin untuk menjalankan PartiQL untuk pembacaan dan penulisan non-transaksional DynamoDB sambil memblokir PartiQL untuk pembacaan dan penulisan transaksional DynamoDB.

```
{
   "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"
         ]
      }
   ]
}
```

Contoh: Mengizinkan pernyataan pilihan dan menolak pernyataan pemindaian tabel lengkap di PartiQL untuk DynamoDB

Kebijakan IAM berikut memberikan izin untuk menjalankan pernyataan `select` pada tabel tertentu sambil memblokir pernyataan `select` yang menghasilkan pemindaian tabel penuh.

```
{
   "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"
         ]
      }
   ]
}
```

Bekerja dengan item: Java

Anda dapat menggunakan AWS SDK untuk Java Document API untuk melakukan operasi membuat, membaca, memperbarui, dan menghapus (CRUD) pada item Amazon DynamoDB dalam tabel.

> **Note**
>
> SDK untuk Java juga menyediakan model persistensi objek, memungkinkan Anda memetakan kelas sisi klien ke tabel DynamoDB. Pendekatan ini dapat mengurangi jumlah kode yang harus Anda tulis. Untuk informasi selengkapnya, lihat Java 1.x: Dinamo DBMapper.

Bagian ini berisi contoh Java untuk melakukan beberapa tindakan item Java Document API dan beberapa contoh kerja yang lengkap.

Topik

- Menempatkan item
- Mendapatkan item
- Penulisan batch: Menempatkan dan menghapus beberapa item
- Batch get: Mendapatkan beberapa item
- Memperbarui Item
- Menghapus item
- Contoh: Operasi CRUD menggunakan AWS SDK untuk Java document API
- Contoh: Operasi batch menggunakan AWS SDK untuk Java document API
- Contoh: Menangani atribut tipe biner menggunakan API AWS SDK untuk Java dokumen

Menempatkan item

Metode `putItem` menyimpan item dalam tabel. Jika item ada, item tersebut akan menggantikan keseluruhan item. Alih-alih mengganti seluruh item, jika Anda hanya ingin memperbarui atribut tertentu, Anda dapat menggunakan metode `updateItem`. Untuk informasi selengkapnya, lihat Memperbarui Item.

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

Ikuti langkah-langkah ini:

1. Buat instans dari kelas `DynamoDB`.
2. Buat instans dari kelas `Table` untuk mewakili tabel yang ingin Anda gunakan.
3. Buat instans dari kelas `Item` untuk mewakili item baru. Anda harus menentukan kunci primer item baru dan atributnya.
4. Panggil metode `putItem` dari objek `Table` dengan menggunakan `Item` yang Anda buat pada langkah sebelumnya.

Contoh kode Java berikut menunjukkan tugas sebelumnya. Kode tersebut menulis item baru ke tabel `ProductCatalog`.

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);
```

Pada contoh sebelumnya, item memiliki atribut berupa skalar (`String`, `Number`, `Boolean`, `Null`), set (`String Set`), dan jenis dokumen (`List`, `Map`).

Menentukan parameter opsional

Selain parameter yang diperlukan, Anda juga dapat menentukan parameter opsional untuk metode `putItem`. Misalnya, contoh kode Java berikut menggunakan parameter opsional untuk menentukan kondisi untuk mengunggah item. Jika kondisi yang Anda tentukan tidak terpenuhi, AWS SDK untuk Java lemparan a`ConditionalCheckFailedException`. Contoh kode menentukan parameter opsional berikut dalam metode `putItem`:

- `ConditionExpression` yang mendefinisikan kondisi permintaan. Kode tersebut mendefinisikan kondisi bahwa item yang ada dengan kunci primer yang sama diganti hanya jika item tersebut memiliki atribut ISBN yang sama dengan nilai tertentu.
- Peta untuk `ExpressionAttributeValues` yang digunakan dalam kondisi tersebut. Dalam kasus ini, hanya ada satu substitusi yang diperlukan: placeholder `:val` dalam ekspresi kondisi diganti pada saat runtime dengan nilai ISBN aktual yang akan diperiksa.

Contoh berikut menambahkan item buku baru menggunakan parameter opsional.

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 dan dokumen JSON

Anda dapat menyimpan dokumen JSON sebagai atribut dalam tabel DynamoDB. Untuk melakukannya, gunakan metode `withJSON` `Item`. Metode ini mengurai dokumen JSON dan memetakan setiap elemen untuk jenis daya DynamoDB asli.

Misalkan Anda ingin menyimpan dokumen JSON berikut, yang berisi vendor yang dapat memenuhi pesanan untuk produk tertentu.

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"
        ]
    }
}
```

Anda dapat menggunakan metode `withJSON` untuk menyimpan ini dalam tabel `ProductCatalog`, dalam atribut `Map` yang disebut `VendorInfo`. Contoh kode Java berikut mendemonstrasikan cara melakukannya.

```
// 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);
```

Mendapatkan item

Untuk mengambil satu item, gunakan metode `getItem` dari objek `Table`. Ikuti langkah-langkah ini:

1. Buat instans dari kelas `DynamoDB`.
2. Buat instans dari kelas `Table` untuk mewakili tabel yang ingin Anda gunakan.
3. Panggil metode `getItem` dari instans `Table`. Anda harus menentukan kunci item primer yang ingin Anda ambil.

Contoh kode Java berikut mendemonstrasikan langkah sebelumnya. Kode mendapatkan item yang memiliki kunci partisi yang ditentukan.

```
AmazonDynamoDB client = AmazonDynamoDBClientBuilder.standard().build();
DynamoDB dynamoDB = new DynamoDB(client);

Table table = dynamoDB.getTable("ProductCatalog");
```

```
Item item = table.getItem("Id", 210);
```

Menentukan parameter opsional

Selain parameter yang diperlukan, Anda juga dapat menentukan parameter opsional untuk metode `getItem`. Misalnya, contoh kode Java berikut menggunakan metode opsional untuk mengambil hanya daftar atribut tertentu dan menentukan bacaan sangat konsisten. (Untuk mempelajari selengkapnya tentang konsistensi baca, lihat DynamoDB membaca konsistensi.)

Anda dapat menggunakan `ProjectionExpression` untuk mengambil hanya atribut atau elemen tertentu, bukan seluruh item. `ProjectionExpression` dapat menentukan tingkat atas atau atribut bersarang menggunakan jalur dokumen. Untuk informasi selengkapnya, lihat Menggunakan ekspresi proyeksi di DynamoDB.

Parameter dari metode `getItem` tidak membiarkan Anda menentukan konsistensi baca. Namun, Anda dapat membuat `GetItemSpec`, yang menyediakan akses penuh ke semua input ke operasi `GetItem` tingkat rendah. Contoh kode berikut membuat `GetItemSpec` dan menggunakan spesifikasi tersebut sebagai input untuk metode `getItem`.

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());
```

Untuk mencetak `Item` dalam format yang dapat dibaca manusia, gunakan metode `toJSONPretty`. Output dari contoh sebelumnya terlihat seperti berikut ini.

```
{
  "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 dan dokumen JSON

Di bagian PutItem dan dokumen JSON, Anda menyimpan dokumen JSON dalam atribut `Map` yang disebut `VendorInfo`. Anda dapat menggunakan metode `getItem` untuk mengambil seluruh dokumen dalam format JSON. Selain itu, Anda dapat menggunakan notasi jalur dokumen untuk mengambil hanya beberapa elemen dalam dokumen. Contoh kode Java berikut mendemonstrasikan teknik ini.

```
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());
```

Output dari contoh sebelumnya terlihat seperti berikut ini.

```
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**
>
> Anda dapat menggunakan metode `toJSON` untuk mengonversi item apa pun (atau atributnya) ke string berformat JSON. Kode berikut mengambil beberapa atribut tingkat atas dan bertingkat serta mencetak hasilnya sebagai JSON.
>
> ```
> GetItemSpec spec = new GetItemSpec()
>     .withPrimaryKey("Id", 210)
>     .withProjectionExpression("VendorInfo.V01, Title, Price");
>
> Item item = table.getItem(spec);
> System.out.println(item.toJSON());
> ```
>
> Output-nya akan terlihat seperti berikut.
>
> ```
> {"VendorInfo":{"V01":{"Name":"Acme Books","Offices":
> ["Seattle"]}},"Price":30,"Title":"Book 210 Title"}
> ```

Penulisan batch: Menempatkan dan menghapus beberapa item

Penulisan batch merujuk pada penempatan dan penghapusan beberapa item dalam satu batch. Metode `batchWriteItem` memungkinkan Anda untuk memasukkan dan menghapus beberapa item dari satu atau lebih tabel dalam satu panggilan. Berikut ini adalah langkah-langkah untuk menempatkan atau menghapus beberapa item menggunakan AWS SDK untuk Java Document API.

1. Buat instans dari kelas `DynamoDB`.
2. Buat instans dari kelas `TableWriteItems` yang menjelaskan semua operasi put dan hapus untuk tabel. Jika Anda ingin menulis ke beberapa tabel dalam satu operasi penulisan batch, Anda harus membuat satu instans `TableWriteItems` per tabel.
3. Jalankan metode `batchWriteItem` dengan menyediakan objek `TableWriteItems` yang Anda buat pada langkah sebelumnya.
4. Proses responsnya. Anda harus memeriksa apakah ada item permintaan yang belum diproses yang dikembalikan sebagai respons. Hal ini dapat terjadi jika Anda mencapai kuota throughput yang disediakan atau kesalahan sementara lainnya. Selain itu, DynamoDB membatasi ukuran permintaan dan jumlah operasi yang dapat Anda tentukan dalam permintaan. Jika Anda melebihi

batas ini, DynamoDB menolak permintaan tersebut. Untuk informasi selengkapnya, lihat Kuota di Amazon DynamoDB.

Contoh kode Java berikut mendemonstrasikan langkah sebelumnya. Contoh tersebut melakukan operasi `batchWriteItem` pada dua tabel: `Forum` dan `Thread`. Objek `TableWriteItems` yang sesuai menentukan tindakan berikut:

- Tempatkan satu item dalam tabel `Forum`.
- Tempatkan dan hapus satu item dalam tabel `Thread`.

Kode kemudian menjalankan `batchWriteItem` untuk melakukan operasi.

```
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
```

Untuk contoh pekerjaan, lihat Contoh: Operasi batch write menggunakan AWS SDK untuk Java document API.

Batch get: Mendapatkan beberapa item

Metode `batchGetItem` memungkinkan Anda mengambil beberapa item dari satu atau lebih tabel. Untuk mengambil satu item, Anda dapat menggunakan metode `getItem`.

Ikuti langkah-langkah ini:

1. Buat instans dari kelas `DynamoDB`.
2. Buat instans dari kelas `TableKeysAndAttributes` yang menjelaskan daftar nilai kunci primer untuk mengambil dari tabel. Jika Anda ingin membaca dari beberapa tabel dalam satu operasi penulisan batch, Anda harus membuat satu instans `TableKeysAndAttributes` per tabel.
3. Jalankan metode `batchGetItem` dengan menyediakan objek `TableKeysAndAttributes` yang Anda buat pada langkah sebelumnya.

Contoh kode Java berikut mendemonstrasikan langkah sebelumnya. Contoh tersebut mengambil dua item dari tabel `Forum` dan tiga item dari tabel `Thread`.

```
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);
    }
}
```

Menentukan parameter opsional

Bersamaan dengan parameter yang diperlukan, Anda juga dapat menentukan parameter opsional saat menggunakan `batchGetItem`. Misalnya, Anda dapat memberikan `ProjectionExpression` dengan setiap `TableKeysAndAttributes` yang Anda tentukan. Hal ini memungkinkan Anda menentukan atribut yang ingin Anda ambil dari tabel.

Contoh kode berikut mengambil dua item dari tabel `Forum`. Parameter `withProjectionExpression` menentukan bahwa hanya atribut `Threads` yang akan diambil.

Example

```
TableKeysAndAttributes forumTableKeysAndAttributes = new
 TableKeysAndAttributes("Forum")
    .withProjectionExpression("Threads");

forumTableKeysAndAttributes.addHashOnlyPrimaryKeys("Name",
    "Amazon S3",
    "Amazon DynamoDB");

BatchGetItemOutcome outcome = dynamoDB.batchGetItem(forumTableKeysAndAttributes);
```

Memperbarui Item

Metode `updateItem` dari objek `Table` dapat memperbarui nilai atribut yang ada, menambahkan atribut baru, atau menghapus atribut dari item yang ada.

Metode `updateItem` berperilaku sebagai berikut:

- Jika item tidak ada (tidak ada item dalam tabel dengan kunci primer yang ditentukan), `updateItem` menambahkan item baru ke tabel.
- Jika item ada, `updateItem` melakukan pembaruan seperti yang ditentukan oleh parameter `UpdateExpression`.

> **Note**
>
> Hal ini juga memungkinkan untuk "memperbarui" item menggunakan `putItem`. Misalnya, jika Anda memanggil `putItem` untuk menambahkan item ke tabel, tetapi sudah ada item dengan kunci primer yang ditentukan, `putItem` akan menggantikan seluruh item. Jika ada atribut

dalam item yang ada yang tidak ditentukan dalam input, `putItem` akan menghapus atribut tersebut dari item.
Secara umum, sebaiknya gunakan `updateItem` setiap kali Anda ingin memodifikasi atribut item. Metode `updateItem` hanya memodifikasi atribut item yang Anda tentukan dalam input, dan atribut lainnya dalam item tidak berubah.

Ikuti langkah-langkah ini:

1. Buat instans dari kelas `Table` untuk mewakili tabel yang ingin Anda gunakan.
2. Panggil metode `updateTable` dari instans `Table`. Anda harus menentukan kunci primer dari item yang ingin Anda ambil, bersama dengan `UpdateExpression` yang menjelaskan atribut untuk memodifikasi dan cara memodifikasinya.

Contoh kode Java berikut menunjukkan tugas sebelumnya. Kode tersebut memperbarui item buku dalam tabel `ProductCatalog`. Contoh tersebut menambahkan penulis baru ke set `Authors`, dan menghapus atribut `ISBN` yang ada. Contoh tersebut juga mengurangi harga sebanyak satu.

Peta `ExpressionAttributeValues` digunakan dalam `UpdateExpression`. Placeholder `:val1` dan `:val2` diganti pada saat runtime dengan nilai aktual untuk `Authors` dan `Price`.

```
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);
```

Menentukan parameter opsional

Bersamaan dengan parameter yang diperlukan, Anda juga dapat menentukan parameter opsional untuk metode `updateItem`, termasuk kondisi yang harus dipenuhi agar pembaruan bisa terjadi. Jika kondisi yang Anda tentukan tidak terpenuhi, AWS SDK untuk Java lemparan a`ConditionalCheckFailedException`. Misalnya, contoh kode Java berikut memperbarui harga item buku secara kondisional menjadi 25. Ini menentukan pernyataan `ConditionExpression` bahwa harga harus diperbarui hanya jika harga saat ini adalah 20.

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);
```

Penghitung atom

Anda dapat menggunakan `updateItem` untuk mengimplementasi penghitung atom, di mana Anda menambahkan atau mengurangi nilai atribut yang ada tanpa mengganggu permintaan tulis lainnya. Untuk kenaikan penghitung atom, gunakan `UpdateExpression` dengan tindakan `set` untuk menambahkan nilai numerik ke atribut jenis `Number` yang ada.

Contoh berikut menunjukkan hal ini, menambahkan satu atribut `Quantity`. Hal ini juga menunjukkan penggunaan parameter `ExpressionAttributeNames` dalam `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);
```

Menghapus item

Metode `deleteItem` menghapus item dari tabel. Anda harus menyediakan kunci item primer yang ingin Anda hapus.

Ikuti langkah-langkah ini:

1. Buat instans dari klien `DynamoDB`.
2. Panggil metode `deleteItem` dengan menyediakan kunci item yang ingin Anda hapus.

Contoh Java berikut menunjukkan tugas ini.

Example

```
AmazonDynamoDB client = AmazonDynamoDBClientBuilder.standard().build();
DynamoDB dynamoDB = new DynamoDB(client);

Table table = dynamoDB.getTable("ProductCatalog");

DeleteItemOutcome outcome = table.deleteItem("Id", 101);
```

Menentukan parameter opsional

Anda dapat menentukan parameter opsional untuk `deleteItem`. Misalnya, contoh kode Java berikut menentukan `ConditionExpression`, yang menyatakan bahwa item buku dalam `ProductCatalog` hanya dapat dihapus jika buku ini tidak lagi dalam publikasi (atribut `InPublication` adalah salah).

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);
```

Contoh: Operasi CRUD menggunakan AWS SDK untuk Java document API

Contoh kode berikut menunjukkan operasi CRUD pada item Amazon DynamoDB. Contohnya membuat item, mengambilnya, melakukan berbagai pembaruan, dan akhirnya menghapus item tersebut.

> **Note**
>
> SDK untuk Java juga menyediakan model persistensi objek, memungkinkan Anda memetakan kelas sisi klien ke tabel DynamoDB. Pendekatan ini dapat mengurangi jumlah kode yang harus Anda tulis. Untuk informasi selengkapnya, lihat Java 1.x: Dinamo DBMapper.

> **Note**
>
> Contoh kode ini mengasumsikan bahwa Anda telah memuat data ke DynamoDB untuk akun Anda dengan mengikuti petunjuk di bagian Membuat tabel dan memuat data untuk contoh kode di DynamoDB.
> Untuk step-by-step instruksi untuk menjalankan contoh berikut, lihatContoh kode Java.

```
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());
        }
    }
}
```

Contoh: Operasi batch menggunakan AWS SDK untuk Java document API

Bagian ini memberikan contoh operasi batch write dan batch get di Amazon DynamoDB menggunakan AWS SDK untuk Java Document API.

> **Note**
>
> SDK untuk Java juga menyediakan model persistensi objek, memungkinkan Anda memetakan kelas sisi klien ke tabel DynamoDB. Pendekatan ini dapat mengurangi jumlah kode yang harus Anda tulis. Untuk informasi selengkapnya, lihat Java 1.x: Dinamo DBMapper.

Topik

- Contoh: Operasi batch write menggunakan AWS SDK untuk Java document API
- Contoh: Operasi batch get menggunakan AWS SDK untuk Java document API

Contoh: Operasi batch write menggunakan AWS SDK untuk Java document API

Contoh kode Java berikut menggunakan metode `batchWriteItem` untuk melakukan operasi put dan delete berikut:

- Tempatkan satu item dalam tabel `Forum`.
- Tempatkan satu item dan hapus satu item dari tabel `Thread`.

Anda dapat menentukan sejumlah permintaan put dan delete terhadap satu atau beberapa tabel saat membuat permintaan batch write Anda. Namun, `batchWriteItem` membatasi ukuran permintaan penulisan batch dan jumlah operasi put dan delete dalam satu operasi penulisan batch. Jika permintaan Anda melebihi batas ini, permintaan Anda ditolak. Jika tabel Anda tidak memiliki cukup throughput yang disediakan untuk melayani permintaan ini, item permintaan yang belum diproses akan dikembalikan sebagai respons.

Contoh berikut memeriksa respons untuk melihat apakah ada item permintaan yang belum diproses. Jika ya, contoh tersebut akan mengulang kembali dan mengirim ulang permintaan `batchWriteItem` dengan item yang belum diproses dalam permintaan tersebut. Jika Anda mengikuti contoh dalam panduan ini, Anda seharusnya sudah membuat `Forum` dan `Thread` tabel. Anda juga dapat membuat tabel ini dan mengunggah data sampel secara terprogram. Untuk informasi selengkapnya, lihat Membuat tabel contoh dan mengunggah data menggunakan AWS SDK untuk Java.

Untuk step-by-step instruksi untuk menguji sampel berikut, lihatContoh kode Java.

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);
        }

    }

}
```

Contoh: Operasi batch get menggunakan AWS SDK untuk Java document API

Contoh kode Java berikut menggunakan metode `batchGetItem` untuk mengambil beberapa item dari tabel `Forum` dan `Thread`. `BatchGetItemRequest` menentukan nama tabel dan daftar kunci untuk setiap item yang akan didapatkan. Contoh tersebut memproses respons dengan mencetak item yang diambil.

> **Note**
>
> Contoh kode ini mengasumsikan bahwa Anda telah memuat data ke DynamoDB untuk akun Anda dengan mengikuti petunjuk di bagian Membuat tabel dan memuat data untuk contoh kode di DynamoDB.
> Untuk step-by-step instruksi untuk menjalankan contoh berikut, lihatContoh kode Java.

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());
        }

    }

}
```

Contoh: Menangani atribut tipe biner menggunakan API AWS SDK untuk Java dokumen

Contoh kode Java berikut menggambarkan penanganan atribut jenis biner. Contoh tersebut menambahkan item ke tabel `Reply`. Item termasuk atribut jenis biner (`ExtendedMessage`) yang menyimpan data terkompresi. Contoh tersebut kemudian mengambil item dan mencetak semua nilai atribut. Sebagai ilustrasi, contoh menggunakan kelas `GZIPOutputStream` untuk mengompresi aliran sampel dan menetapkannya ke atribut `ExtendedMessage`. Ketika atribut binari diambil, atribut didekompresi menggunakan kelas `GZIPInputStream`.

> **Note**
>
> SDK untuk Java juga menyediakan model persistensi objek, memungkinkan Anda memetakan kelas sisi klien ke tabel DynamoDB. Pendekatan ini dapat mengurangi jumlah kode yang harus Anda tulis. Untuk informasi selengkapnya, lihat Java 1.x: Dinamo DBMapper.

Jika Anda mengikuti bagian Membuat tabel dan memuat data untuk contoh kode di DynamoDB, semestinya Anda telah membuat tabel `Reply`. Anda juga dapat membuat tabel ini secara terprogram. Untuk informasi selengkapnya, lihat Membuat tabel contoh dan mengunggah data menggunakan AWS SDK untuk Java.

Untuk step-by-step instruksi untuk menguji sampel berikut, lihatContoh kode Java.

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;
    }
}
```

Bekerja dengan item: .NET

Anda dapat menggunakan API AWS SDK untuk .NET tingkat rendah untuk melakukan operasi membuat, membaca, memperbarui, dan menghapus (CRUD) pada item dalam tabel. Berikut ini adalah langkah-langkah umum yang Anda ikuti untuk melakukan operasi CRUD data menggunakan API tingkat rendah .NET:

1. Buat instans dari kelas `AmazonDynamoDBClient` (klien).
2. Berikan parameter khusus operasi yang diperlukan dalam objek permintaan yang sesuai.

   Misalnya, gunakan permintaan `PutItemRequest` saat mengunggah item dan gunakan permintaan `GetItemRequest` saat mengambil item yang sudah ada.

   Anda dapat menggunakan objek permintaan untuk menyediakan parameter wajib dan opsional.
3. Jalankan metode yang sesuai yang disediakan oleh klien dengan meneruskan objek permintaan yang Anda buat pada langkah sebelumnya.

   Klien `AmazonDynamoDBClient` menyediakan metode `PutItem`, `GetItem`, `UpdateItem`, dan `DeleteItem` untuk operasi CRUD.

Topik

- Menempatkan item
- Mendapatkan item
- Memperbarui Item
- Penghitung atom
- Menghapus item
- Penulisan batch: Menempatkan dan menghapus beberapa item
- Batch get: Mendapatkan beberapa item
- Contoh: Operasi CRUD menggunakan API tingkat rendah AWS SDK untuk .NET
- Contoh: Operasi batch menggunakan API tingkat rendah AWS SDK untuk .NET

- Contoh: Menangani atribut tipe biner menggunakan API AWS SDK untuk .NET tingkat rendah

Menempatkan item

Metode `PutItem` mengunggah item ke tabel. Jika item ada, item tersebut akan menggantikan keseluruhan item.

> **Note**
>
> Alih-alih mengganti seluruh item, jika Anda hanya ingin memperbarui atribut tertentu, Anda dapat menggunakan metode `UpdateItem`. Untuk informasi selengkapnya, lihat Memperbarui Item.

Berikut langkah-langkah untuk mengunggah item menggunakan .NET SDK API tingkat rendah:

1. Buat instans dari kelas `AmazonDynamoDBClient`.
2. Berikan parameter yang diperlukan dengan membuat instans kelas `PutItemRequest`.

   Untuk menaruh item, Anda harus memberikan nama tabel dan itemnya.
3. Jalankan metode `PutItem` dengan menyediakan objek `PutItemRequest` yang Anda buat pada langkah sebelumnya.

Contoh #C berikut menunjukkan langkah-langkah sebelumnya. Contoh tersebut mengunggah item ke tabel `ProductCatalog`.

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);
```

Pada contoh sebelumnya, Anda mengunggah item buku yang memiliki atribut `Id`, `Title`, `ISBN`, dan `Authors`. Perhatikan bahwa `Id` adalah atribut jenis numerik, dan semua atribut lainnya adalah jenis string. Penulis adalah set `String`.

Menentukan parameter opsional

Anda juga dapat memberikan parameter opsional menggunakan objek `PutItemRequest` seperti yang ditunjukkan dalam contoh C# berikut. Contoh tersebut menentukan parameter opsional berikut:

- `ExpressionAttributeNames`, `ExpressionAttributeValues`, dan `ConditionExpression` menentukan bahwa barang tersebut hanya dapat diganti jika barang yang ada memiliki atribut ISBN dengan nilai tertentu.
- Parameter `ReturnValues` untuk meminta item lama dalam respons.

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);
```

Untuk informasi selengkapnya, lihat PutItem.

Mendapatkan item

Metode `GetItem` mengambil item.

> **Note**
>
> Untuk mengambil beberapa item, Anda dapat menggunakan metode `BatchGetItem`. Untuk informasi selengkapnya, lihat Batch get: Mendapatkan beberapa item.

Berikut ini adalah langkah-langkah untuk mengambil item yang ada menggunakan API AWS SDK untuk .NET tingkat rendah.

1. Buat instans dari kelas `AmazonDynamoDBClient`.
2. Berikan parameter yang diperlukan dengan membuat instans kelas `GetItemRequest`.

   Untuk mendapatkan item, Anda harus memberikan nama tabel dan kunci primer item tersebut.
3. Jalankan metode `GetItem` dengan menyediakan objek `GetItemRequest` yang Anda buat pada langkah sebelumnya.

Contoh #C berikut menunjukkan langkah-langkah sebelumnya. Contoh tersebut mengambil item dari tabel `ProductCatalog`.

```
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.
```

Menentukan parameter opsional

Anda juga dapat memberikan parameter opsional menggunakan objek `GetItemRequest`, seperti yang ditunjukkan dalam contoh C# berikut. Contoh tersebut menentukan parameter opsional berikut:

- Parameter `ProjectionExpression` untuk menentukan atribut yang akan diambil.
- Parameter `ConsistentRead` untuk melakukan bacaan sangat konsisten. Untuk mempelajari selengkapnya tentang konsistensi baca, lihat DynamoDB membaca konsistensi.

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;
```

Untuk informasi selengkapnya, lihat GetItem.

Memperbarui Item

Metode `UpdateItem` memperbarui item yang ada jika ada. Anda dapat menggunakan operasi `UpdateItem` untuk memperbarui nilai atribut yang ada, menambahkan atribut baru, atau menghapus atribut dari koleksi yang ada. Jika item yang memiliki kunci primer yang ditentukan tidak ditemukan, item baru akan ditambahkan.

Operasi `UpdateItem` menggunakan panduan berikut:

- Jika item tidak ada, `UpdateItem` menambahkan item baru menggunakan kunci primer yang ditentukan dalam input.
- Jika item ada, `UpdateItem` menerapkan pembaruan sebagai berikut:
  - Menggantikan nilai atribut yang ada dengan nilai dalam pembaruan.
  - Jika atribut yang Anda berikan pada input tidak ada, atribut baru akan ditambahkan ke item tersebut.
  - Jika atribut input adalah null, atribut tersebut akan dihapus, jika ada.
  - Jika Anda menggunakan `ADD` untuk`Action`, Anda dapat menambahkan nilai ke set yang sudah ada (set string atau angka), atau menambahkan secara matematis (menggunakan angka positif) atau mengurangi (menggunakan angka negatif) dari nilai atribut numerik yang ada.

> **Note**
>
> Operasi `PutItem` juga dapat melakukan pembaruan. Untuk informasi selengkapnya, lihat Menempatkan item. Misalnya, jika Anda memanggil `PutItem` untuk mengunggah item dan kunci primer ada, operasi `PutItem` menggantikan seluruh item. Jika terdapat atribut dalam item yang ada yang tidak ditentukan dalam input, operasi `PutItem` akan menghapus atribut tersebut. Namun, `UpdateItem` hanya memperbarui atribut input yang ditentukan. Atribut lain item tersebut yang ada tidak akan berubah.

Berikut langkah-langkah untuk memperbarui item menggunakan .NET SDK API tingkat rendah:

1. Buat instans dari kelas `AmazonDynamoDBClient`.
2. Berikan parameter yang diperlukan dengan membuat instans kelas `UpdateItemRequest`.

Ini adalah objek permintaan tempat Anda menjelaskan semua pembaruan, seperti menambahkan atribut, memperbarui atribut yang ada, atau menghapus atribut. Untuk menghapus atribut yang ada, tentukan nama atribut dengan nilai null.

3. Jalankan metode `UpdateItem` dengan menyediakan objek `UpdateItemRequest` yang Anda buat pada langkah sebelumnya.

Contoh kode #C berikut mendemonstrasikan langkah sebelumnya. Contoh tersebut memperbarui item buku dalam tabel `ProductCatalog`. Contoh tersebut menambahkan penulis baru ke koleksi `Authors`, dan menghapus atribut `ISBN` yang ada. Contoh tersebut juga mengurangi harga sebanyak satu.

```
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);
```

Menentukan parameter opsional

Anda juga dapat memberikan parameter opsional menggunakan objek `UpdateItemRequest`, seperti yang ditunjukkan dalam contoh C# berikut. Contoh tersebut menentukan parameter opsional berikut:

- `ExpressionAttributeValues` dan `ConditionExpression` untuk menentukan bahwa harga dapat diperbarui hanya jika harga yang ada adalah 20,00.
- Parameter `ReturnValues` untuk meminta item yang diperbarui dalam respons.

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);
```

Untuk informasi selengkapnya, lihat UpdateItem.

Penghitung atom

Anda dapat menggunakan `updateItem` untuk mengimplementasi penghitung atom, di mana Anda menambahkan atau mengurangi nilai atribut yang ada tanpa mengganggu permintaan tulis lainnya. Untuk memperbarui penghitung atom, gunakan `updateItem` dengan atribut jenis `Number` dalam parameter `UpdateExpression`, dan `ADD` sebagai `Action`.

Contoh berikut menunjukkan hal ini, menambahkan satu atribut `Quantity`.

```
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);
```

Menghapus item

Metode `DeleteItem` menghapus item dari tabel.

Berikut ini adalah langkah-langkah untuk menghapus item menggunakan .NET SDK API tingkat rendah.

1. Buat instans dari kelas `AmazonDynamoDBClient`.
2. Berikan parameter yang diperlukan dengan membuat instans kelas `DeleteItemRequest`.

   Untuk menghapus item, diperlukan nama tabel dan kunci primer item.

3. Jalankan metode `DeleteItem` dengan menyediakan objek `DeleteItemRequest` yang Anda buat pada langkah sebelumnya.

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);
```

Menentukan parameter opsional

Anda juga dapat memberikan parameter opsional menggunakan objek `DeleteItemRequest` seperti yang ditunjukkan dalam contoh kode C# berikut. Contoh tersebut menentukan parameter opsional berikut:

- `ExpressionAttributeValues`dan `ConditionExpression` untuk menentukan bahwa item buku dapat dihapus hanya jika tidak lagi dalam publikasi (nilai InPublication atribut salah).
- Parameter `ReturnValues` untuk meminta item yang dihapus dalam respons.

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);
```

Untuk informasi selengkapnya, lihat DeleteItem.

Penulisan batch: Menempatkan dan menghapus beberapa item

Penulisan batch merujuk pada penempatan dan penghapusan beberapa item dalam satu batch. Metode `BatchWriteItem` memungkinkan Anda untuk memasukkan dan menghapus beberapa item dari satu atau lebih tabel dalam satu panggilan. Berikut ini adalah langkah-langkah untuk mengambil beberapa item menggunakan .NET SDK API tingkat rendah.

1. Buat instans dari kelas `AmazonDynamoDBClient`.
2. Jelaskan semua operasi tempatkan dan hapus dengan membuat instans dari kelas `BatchWriteItemRequest`.
3. Jalankan metode `BatchWriteItem` dengan menyediakan objek `BatchWriteItemRequest` yang Anda buat pada langkah sebelumnya.
4. Proses responsnya. Anda harus memeriksa apakah ada item permintaan yang belum diproses yang dikembalikan sebagai respons. Hal ini dapat terjadi jika Anda mencapai kuota throughput yang disediakan atau kesalahan sementara lainnya. Selain itu, DynamoDB membatasi ukuran permintaan dan jumlah operasi yang dapat Anda tentukan dalam permintaan. Jika Anda melebihi batas ini, DynamoDB menolak permintaan tersebut. Untuk informasi selengkapnya, lihat BatchWriteItem.

Contoh kode #C berikut mendemonstrasikan langkah sebelumnya. Contoh ini membuat `BatchWriteItemRequest` untuk melakukan operasi tulis berikut:

- Tempatkan satu item dalam tabel `Forum`.
- Tempatkan dan hapus satu item dari tabel `Thread`.

Kode menjalankan `BatchWriteItem` untuk melakukan operasi batch.

```
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);
```

Untuk contoh pekerjaan, lihat Contoh: Operasi batch menggunakan API tingkat rendah AWS SDK untuk .NET.

Batch get: Mendapatkan beberapa item

Metode `BatchGetItem` memungkinkan Anda mengambil beberapa item dari satu atau lebih tabel.

> **Note**
>
> Untuk mengambil satu item, Anda dapat menggunakan metode `GetItem`.

Berikut ini adalah langkah-langkah untuk mengambil beberapa item menggunakan API AWS SDK untuk .NET tingkat rendah.

1. Buat instans dari kelas `AmazonDynamoDBClient`.
2. Berikan parameter yang diperlukan dengan membuat instans kelas `BatchGetItemRequest`.

   Untuk mengambil beberapa item, nama tabel dan daftar nilai kunci primer diperlukan.
3. Jalankan metode `BatchGetItem` dengan menyediakan objek `BatchGetItemRequest` yang Anda buat pada langkah sebelumnya.
4. Proses responsnya. Anda harus memeriksa apakah ada kunci yang belum diproses, yang dapat terjadi jika Anda mencapai kuota throughput yang disediakan atau kesalahan sementara lainnya.

Contoh kode #C berikut mendemonstrasikan langkah sebelumnya. Contoh tersebut mengambil item dari dua tabel, `Forum` dan `Thread`. Permintaan menentukan dua item dalam `Forum` dan tiga item dalam tabel `Thread`. Respons tersebut mencakup item dari kedua tabel. Kode tersebut menunjukkan bagaimana Anda dapat memproses respons.

```
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);
}
```

Menentukan parameter opsional

Anda juga dapat memberikan parameter opsional menggunakan objek `BatchGetItemRequest` seperti yang ditunjukkan dalam contoh kode C# berikut. Contoh tersebut mengambil dua item dari tabel `Forum`. Contoh tersebut menentukan parameter opsional berikut:

- Parameter `ProjectionExpression` untuk menentukan atribut yang akan diambil.

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);
```

Untuk informasi selengkapnya, lihat BatchGetItem.

Contoh: Operasi CRUD menggunakan API tingkat rendah AWS SDK untuk .NET

Contoh kode C# berikut menunjukkan operasi CRUD pada item Amazon DynamoDB. Contoh tersebut menambahkan item ke tabel `ProductCatalog`, mengambilnya, melakukan berbagai pembaruan, dan akhirnya menghapus item. Jika Anda belum membuat tabel ini, Anda juga dapat membuatnya secara terprogram. Untuk informasi selengkapnya, lihat Membuat tabel contoh dan mengunggah data menggunakan AWS SDK untuk .NET.

Untuk step-by-step instruksi untuk menguji sampel berikut, lihatContoh kode .NET.

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("************************************************");
        }
    }
}
```

Contoh: Operasi batch menggunakan API tingkat rendah AWS SDK untuk .NET

Topik

- Contoh: Operasi batch tulis menggunakan API tingkat rendah AWS SDK untuk .NET
- Contoh: Operasi batch get menggunakan API tingkat rendah AWS SDK untuk .NET

Bagian ini memberikan contoh operasi batch, batch write dan batch get, yang mendukung Amazon DynamoDB.

Contoh: Operasi batch tulis menggunakan API tingkat rendah AWS SDK untuk .NET

Contoh kode C# berikut menggunakan metode `BatchWriteItem` untuk melakukan operasi put dan delete berikut:

- Tempatkan satu item dalam tabel `Forum`.
- Tempatkan satu item dan hapus satu item dari tabel `Thread`.

Anda dapat menentukan sejumlah permintaan put dan delete terhadap satu atau beberapa tabel saat membuat permintaan batch write Anda. Namun, `BatchWriteItem` DynamoDB membatasi ukuran permintaan penulisan batch dan jumlah operasi put dan delete dalam satu operasi penulisan batch. Untuk informasi selengkapnya, lihat BatchWriteItem. Jika permintaan Anda melebihi batas ini, permintaan Anda ditolak. Jika tabel Anda tidak memiliki cukup throughput yang disediakan untuk melayani permintaan ini, item permintaan yang belum diproses akan dikembalikan sebagai respons.

Contoh berikut memeriksa respons untuk melihat apakah ada item permintaan yang belum diproses. Jika ya, contoh tersebut akan mengulang kembali dan mengirim ulang permintaan `BatchWriteItem` dengan item yang belum diproses dalam permintaan tersebut. Anda juga dapat membuat tabel sampel ini dan mengunggah data sampel secara terprogram. Untuk informasi selengkapnya, lihat Membuat tabel contoh dan mengunggah data menggunakan AWS SDK untuk .NET.

Untuk step-by-step instruksi untuk menguji sampel berikut, lihatContoh kode .NET.

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);
        }
    }
}
```

Contoh: Operasi batch get menggunakan API tingkat rendah AWS SDK untuk .NET

Contoh kode C# berikut menggunakan metode `BatchGetItem` untuk mengambil beberapa item dari tabel `Forum` dan `Thread` dalam Amazon DynamoDB. `BatchGetItemRequest` menentukan nama tabel dan daftar kunci primer untuk setiap tabel. Contoh tersebut memproses respons dengan mencetak item yang diambil.

Untuk step-by-step instruksi untuk menguji sampel berikut, lihatContoh kode .NET.

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("************************************************");
        }
    }
}
```

Contoh: Menangani atribut tipe biner menggunakan API AWS SDK untuk .NET tingkat rendah

Contoh kode C# berikut menggambarkan penanganan atribut jenis biner. Contoh tersebut menambahkan item ke tabel `Reply`. Item termasuk atribut jenis biner (`ExtendedMessage`) yang menyimpan data terkompresi. Contoh tersebut kemudian mengambil item dan mencetak semua nilai atribut. Sebagai ilustrasi, contoh menggunakan kelas `GZipStream` untuk mengompresi aliran sampel dan menetapkannya ke atribut `ExtendedMessage`, serta mendekompresinya saat mencetak nilai atribut.

Untuk step-by-step instruksi untuk menguji contoh berikut, lihatContoh kode .NET.

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();
            }
        }
    }
}
```

Meningkatkan akses data dengan indeks sekunder di DynamoDB

Amazon DynamoDB menyediakan akses cepat ke item-item dalam sebuah tabel dengan menentukan nilai-nilai kunci primer. Namun, banyak aplikasi mungkin mendapat manfaat dari tersedianya satu atau lebih kunci sekunder (atau alternatif), untuk memungkinkan akses efisien ke data dengan atribut selain kunci primer. Untuk mengatasi hal ini, Anda dapat membuat satu atau lebih indeks sekunder pada tabel dan mengeluarkan permintaan `Query` atau `Scan` terhadap indeks ini.

Indeks sekunder adalah struktur data yang berisi subset atribut dari tabel, bersama dengan kunci alternatif untuk mendukung operasi `Query`. Anda dapat mengambil data dari indeks menggunakan sebuah `Query`, dengan cara yang sama seperti saat Anda menggunakan `Query` dengan tabel. Sebuah tabel dapat memiliki beberapa indeks sekunder, yang memberikan aplikasi Anda akses ke banyak pola kueri yang berbeda.

> **Note**
>
> Anda juga dapat melakukan `Scan` pada indeks, dengan cara yang sama seperti Anda melakukan `Scan` pada tabel.
> Akses lintas akun untuk operasi pemindaian indeks sekunder saat ini tidak didukung dengan kebijakan berbasis sumber daya.

Setiap indeks sekunder dikaitkan dengan tepat satu tabel, dari mana indeks memperoleh datanya. Ini disebut tabel dasar untuk indeks. Saat Anda membuat indeks, Anda menentukan kunci alternatif

untuk indeks tersebut (kunci partisi dan kunci urutan). Anda juga menentukan atribut yang ingin Anda proyeksikan, atau salin, dari tabel dasar ke dalam indeks. DynamoDB menyalin atribut ini ke dalam indeks, bersama dengan atribut kunci primer dari tabel dasar. Anda kemudian dapat mengkueri atau memindai indeks sama seperti Anda mengkueri atau memindai tabel.

Setiap indeks sekunder dikelola secara otomatis oleh DynamoDB. Saat Anda menambahkan, mengubah, atau menghapus item di tabel dasar, indeks apa pun di tabel tersebut juga diperbarui untuk mencerminkan perubahan ini.

DynamoDB mendukung dua jenis indeks sekunder:

- Indeks sekunder global — Indeks dengan kunci partisi dan kunci urutan yang mungkin berbeda dari yang ada di tabel dasar. Indeks sekunder global dianggap "global" karena kueri pada indeks dapat menjangkau semua data di tabel dasar, di semua partisi. Indeks sekunder global disimpan dalam ruang partisinya sendiri jauh dari tabel dasar dan diskalakan secara terpisah dari tabel dasar.
- Indeks sekunder lokal – Indeks yang memiliki kunci partisi yang sama dengan tabel, namun kunci urutan berbeda. Indeks sekunder lokal bersifat "lokal" dalam arti bahwa setiap partisi indeks sekunder lokal dicakup ke partisi tabel dasar yang memiliki nilai kunci partisi yang sama.

Untuk perbandingan indeks sekunder global dan indeks sekunder lokal, lihat video ini.

Membuat pilihan yang tepat antara GSI dan LSI

Topik

- Menggunakan Indeks Sekunder Global di DynamoDB
- Indeks Sekunder Lokal di DynamoDB

Anda harus mempertimbangkan persyaratan aplikasi Anda saat menentukan jenis indeks yang akan digunakan. Tabel berikut menunjukkan perbedaan utama antara indeks sekunder global dan indeks sekunder lokal.

Karakteristik	Indeks sekunder global	Indeks sekunder lokal
Skema Kunci	Kunci primer indeks sekunder global dapat berupa sederhana (kunci partisi) atau	Kunci primer dari indeks sekunder lokal harus

Karakteristik	Indeks sekunder global	Indeks sekunder lokal
	gabungan (kunci partisi dan kunci urutan).	gabungan (kunci partisi dan kunci urutan).
Atribut Kunci	Kunci partisi indeks dan kunci urutan (jika ada) dapat berupa atribut tabel dasar apa pun berupa jenis string, angka, atau biner.	Kunci partisi indeks adalah atribut yang sama dengan kunci partisi dari tabel dasar. Kunci urutan dapat berupa atribut tabel dasar apa pun dari jenis string, angka, atau biner.
Pembatasan Ukuran Per Nilai Kunci Partisi	Tidak ada batasan ukuran untuk indeks sekunder global.	Untuk setiap nilai kunci partisi, ukuran total semua item yang diindeks harus 10 GB atau kurang.
Operasi Indeks Online	Indeks sekunder global dapat dibuat pada saat yang sama saat Anda membuat tabel. Anda juga dapat menambahk an indeks sekunder global baru ke tabel yang ada, atau menghapus indeks sekunder global yang ada. Untuk informasi selengkapnya, lihat Mengelola Indeks Sekunder Global di DynamoDB.	Indeks sekunder lokal dibuat pada saat yang sama saat Anda membuat tabel. Anda tidak dapat menambahkan indeks sekunder lokal ke tabel yang sudah ada, Anda juga tidak dapat menghapus indeks sekunder lokal yang saat ini ada.
Kueri dan Partisi	Indeks sekunder global memungkinkan Anda melakukan kueri di seluruh tabel, di semua partisi.	Indeks sekunder lokal memungkinkan Anda melakukan kueri pada satu partisi, seperti yang ditentuka n oleh nilai kunci partisi dalam kueri.

Karakteristik	Indeks sekunder global	Indeks sekunder lokal
Konsistensi Baca	Kueri pada indeks sekunder global hanya mendukung konsistensi akhir.	Saat Anda melakukan kueri indeks sekunder lokal, Anda dapat memilih konsistensi akhir atau konsistensi kuat.
Konsumsi Throughput yang Disediakan	Setiap indeks sekunder global memiliki pengaturan throughput yang disediakan sendiri untuk aktivitas baca dan tulis. Kueri atau pemindaian pada indeks sekunder global menggunakan unit kapasitas dari indeks, bukan dari tabel dasar. Hal yang sama berlaku untuk pembaruan indeks sekunder global karena penulisan tabel. Indeks sekunder global yang terkait dengan tabel global mengonsumsi unit kapasitas tulis.	Kueri atau pemindaian pada indeks sekunder lokal menggunakan unit kapasitas baca dari tabel dasar. Saat Anda menulis ke tabel, indeks sekunder lokalnya juga diperbarui, dan pembaruan ini menggunakan unit kapasitas tulis dari tabel dasar. Indeks sekunder lokal yang terkait dengan tabel global mengonsumsi unit kapasitas tulis.
Atribut yang Diproyeksikan	Dengan kueri atau pemindaian indeks sekunder global, Anda hanya dapat meminta atribut yang diproyeksikan ke dalam indeks. DynamoDB tidak mengambil atribut apa pun dari tabel.	Jika Anda membuat kueri atau memindai indeks sekunder lokal, Anda dapat meminta atribut yang tidak diproyeksikan ke dalam indeks. DynamoDB secara otomatis mengambil atribut tersebut dari tabel.

Jika ingin membuat lebih dari satu tabel dengan indeks sekunder, Anda harus melakukannya secara berurutan. Misalnya, Anda akan membuat tabel pertama dan menunggu agar tabel menjadi `ACTIVE`, buat tabel berikutnya dan tunggu agar tabel menjadi `ACTIVE`, dan seterusnya. Jika Anda

mencoba membuat lebih dari satu tabel secara bersamaan dengan indeks sekunder, DynamoDB mengembalikan `LimitExceededException`.

Setiap indeks sekunder menggunakan kelas tabel dan mode kapasitas yang sama dengan tabel dasar yang terkait dengannya. Untuk setiap indeks sekunder, Anda harus menentukan hal berikut:

- Jenis indeks yang akan dibuat – baik indeks sekunder global atau indeks sekunder lokal.
- Nama untuk indeks. Aturan penamaan untuk indeks sama dengan aturan untuk tabel, seperti yang tercantum dalam Kuota di Amazon DynamoDB. Nama harus unik untuk tabel dasar yang terkait dengannya, namun Anda dapat menggunakan nama yang sama untuk indeks yang terkait dengan tabel dasar berbeda.
- Skema kunci indeks. Setiap atribut dalam skema kunci indeks harus merupakan atribut tingkat atas dari jenis `String`, `Number`, atau `Binary`. Jenis data lain, termasuk dokumen dan set, tidak diperbolehkan. Persyaratan lain untuk skema kunci bergantung pada jenis indeks:
  - Untuk indeks sekunder global, kunci partisi dapat berupa atribut skalar apa pun dari tabel dasar. Kunci urutan bersifat opsional, dan dapat berupa atribut skalar apa pun dari tabel dasar.
  - Untuk indeks sekunder lokal, kunci partisi harus sama dengan kunci partisi tabel dasar, dan kunci urutan harus berupa atribut tabel dasar non-kunci.
- Atribut tambahan, jika ada, untuk diproyeksikan dari tabel dasar ke dalam indeks. Atribut ini merupakan tambahan dari atribut kunci tabel, yang secara otomatis diproyeksikan ke dalam setiap indeks. Anda dapat memproyeksikan atribut dari jenis data apa pun, termasuk skalar, dokumen, dan set.
- Pengaturan throughput yang disediakan untuk indeks, jika perlu:
  - Untuk indeks sekunder global, Anda harus menentukan pengaturan unit kapasitas baca dan tulis. Pengaturan throughput yang disediakan ini tidak tergantung pada pengaturan tabel dasar.
  - Untuk indeks sekunder lokal, Anda tidak perlu menentukan pengaturan unit kapasitas baca dan tulis. Setiap operasi baca dan tulis pada indeks sekunder lokal diambil dari pengaturan throughput yang disediakan dari tabel dasarnya.

Untuk fleksibilitas kueri maksimum, Anda dapat membuat hingga 20 indeks sekunder global (kuota default) dan hingga 5 indeks sekunder lokal per tabel.

Kuota indeks sekunder global per tabel adalah 20 untuk Wilayah berikut: AWS

- AWS GovCloud (AS-Timur)
- AWS GovCloud (AS-Barat)

- Eropa (Stockholm)

Untuk mendapatkan daftar rinci indeks sekunder pada tabel, gunakan operasi `DescribeTable`. `DescribeTable` menampilkan nama, ukuran penyimpanan, dan jumlah item untuk setiap indeks sekunder pada tabel. Nilai-nilai ini tidak diperbarui secara real time, tetapi diperbarui kira-kira setiap enam jam.

Anda dapat mengakses data dalam indeks sekunder menggunakan operasi `Query` atau `Scan`. Kueri harus menentukan nama tabel dasar dan nama indeks yang ingin Anda gunakan, atribut yang akan dihasilkan dalam hasil kueri, dan syarat kueri apa pun yang ingin Anda terapkan. DynamoDB dapat menampilkan hasil dalam urutan menaik atau menurun.

Saat Anda menghapus tabel, semua indeks yang terkait dengan tabel tersebut juga akan dihapus.

Untuk praktik terbaik, lihat Praktik terbaik untuk menggunakan indeks sekunder di DynamoDB.

Menggunakan Indeks Sekunder Global di DynamoDB

Beberapa aplikasi mungkin perlu melakukan banyak jenis kueri, menggunakan berbagai atribut berbeda sebagai kriteria kueri. Untuk mendukung persyaratan ini, Anda dapat membuat satu atau beberapa indeks sekunder global dan mengeluarkan permintaan `Query` terhadap indeks ini di Amazon DynamoDB.

Topik

- Skenario: Menggunakan Indeks Sekunder Global
- Proyeksi atribut
- Membaca data dari Indeks Sekunder Global
- Sinkronisasi data antara tabel dan Indeks Sekunder Global
- Kelas tabel dengan Indeks Sekunder Global
- Pertimbangan Throughput yang disediakan untuk Indeks Sekunder Global
- Pertimbangan penyimpanan untuk Indeks Sekunder Global
- Mengelola Indeks Sekunder Global di DynamoDB
- Mendeteksi dan memperbaiki pelanggaran kunci indeks di DynamoDB
- Menggunakan Indeks Sekunder Global: Java
- Menggunakan Indek Sekunder Global: .NET

- Bekerja dengan Indeks Sekunder Global di DynamoDB menggunakan AWS CLI

Skenario: Menggunakan Indeks Sekunder Global

Sebagai ilustrasi, pertimbangkan tabel bernama `GameScores` yang melacak pengguna dan skor untuk aplikasi game seluler. Setiap item dalam `GameScores` diidentifikasi dengan kunci partisi (`UserId`) dan kunci urutan (`GameTitle`). Diagram berikut menunjukkan bagaimana item dalam tabel akan diatur. (Tidak semua atribut ditampilkan.)

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	...
...	...	...	...	...	...	

Sekarang, anggaplah Anda ingin menulis aplikasi papan peringkat untuk menampilkan skor teratas untuk setiap game. Kueri yang menentukan atribut kunci (`UserId` dan `GameTitle`) akan sangat efisien. Namun, jika aplikasi perlu mengambil data dari `GameScores` berdasarkan `GameTitle` saja, aplikasi tersebut perlu menggunakan operasi `Scan`. Karena lebih banyak item ditambahkan ke tabel, pemindaian semua data akan lambat dan tidak efisien. Hal ini membuat pertanyaan-pertanyaan seperti berikut ini sulit dijawab:

- Berapa skor tertinggi yang pernah tercatat untuk game Meteor Blasters?
- Pengguna mana yang memiliki skor tertinggi untuk Galaxy Invaders?

- Berapa rasio tertinggi untuk kemenangan vs kekalahan?

Untuk mempercepat kueri pada atribut non-kunci, Anda dapat membuat indeks sekunder global. Indeks sekunder global berisi pilihan atribut dari tabel dasar, tetapi atribut tersebut diatur oleh kunci primer yang berbeda dari tabel. Kunci indeks tidak perlu memiliki atribut kunci apa pun dari tabel. Bahkan tidak perlu memiliki skema kunci yang sama dengan tabel.

Misalnya, Anda dapat membuat indeks sekunder global bernama `GameTitleIndex`, dengan kunci partisi `GameTitle` dan kunci urutan `TopScore`. Atribut kunci primer tabel dasar selalu diproyeksikan ke dalam indeks, sehingga atribut `UserId` juga ada. Diagram berikut menunjukkan seperti apa tampilan indeks `GameTitleIndex`.

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"
...	...	...

Sekarang Anda dapat meminta `GameTitleIndex` dan dengan mudah mendapatkan skor untuk Meteor Blasters. Hasilnya diurutkan berdasarkan nilai kunci urutan, `TopScore`. Jika Anda mangatur parameter `ScanIndexForward` ke false, hasilnya ditampilkan dalam urutan menurun, sehingga skor tertinggi ditampilkan terlebih dahulu.

Setiap indeks sekunder global harus memiliki kunci partisi, dan dapat memiliki kunci urutan opsional. Skema kunci indeks dapat berbeda dari skema tabel dasar. Anda dapat memiliki tabel dengan kunci primer sederhana (kunci partisi), dan membuat indeks sekunder global dengan kunci primer komposit (kunci partisi dan kunci urutan)—atau sebaliknya. Atribut kunci indeks dapat terdiri dari atribut `String`, `Number`, atau `Binary` tingkat atas dari tabel dasar. Jenis skalar, jenis dokumen, dan jenis kumpulan lain tidak diperbolehkan.

Anda dapat memproyeksikan atribut tabel dasar lain ke dalam indeks jika ingin. Saat Anda mengkueri indeks, DynamoDB dapat mengambil atribut yang diproyeksikan ini secara efisien. Namun, kueri indeks sekunder global tidak dapat mengambil atribut dari tabel dasar. Misalnya, jika Anda mengkueri `GameTitleIndex` seperti yang ditunjukkan pada diagram sebelumnya, kueri tidak dapat mengakses atribut non-kunci selain `TopScore` (meskipun atribut kunci `GameTitle` dan `UserId` akan diproyeksikan secara otomatis).

Dalam tabel DynamoDB, setiap nilai kunci harus unik. Namun, nilai kunci dalam indeks sekunder global tidak perlu unik. Sebagai ilustrasi, anggaplah sebuah game bernama Comet Quest sangat sulit, dengan banyak pengguna baru yang telah mencoba tetapi gagal mendapatkan skor di atas nol. Berikut adalah beberapa data yang dapat mewakili hal tersebut.

UserId	GameTitle	TopScore
123	Comet Quest	0
201	Comet Quest	0
301	Comet Quest	0

Ketika data ini ditambahkan ke tabel `GameScores`, DynamoDB menyebarkannya ke `GameTitleIndex`. Jika kita kemudian mengkueri indeks menggunakan Comet Quest untuk `GameTitle` dan 0 untuk `TopScore`, data berikut dikembalikan.

GameTitle	*TopScore*	*UserId*
"Comet Quest"	*0*	*"123"*
"Comet Quest"	*0*	*"201"*
"Comet Quest"	*0*	*"301"*

Hanya item dengan nilai kunci tertentu yang muncul dalam respons. Dalam kumpulan data itu, item-item tersebut tidak berada dalam urutan tertentu.

Indeks sekunder global hanya melacak item data yang atribut utamanya benar-benar ada. Misalnya, katakanlah Anda menambahkan item baru ke tabel `GameScores`, tetapi hanya menyediakan atribut kunci primer yang diperlukan.

UserId	GameTitle
400	Comet Quest

Karena Anda tidak menentukan atribut `TopScore`, DynamoDB tidak akan menyebarkan item ini ke `GameTitleIndex`. Jadi, jika mengkueri `GameScores` untuk semua item Comet Quest, Anda akan mendapatkan empat item berikut.

UserId	GameTitle	TopScore
"123"	"Comet Quest"	0
"201"	"Comet Quest"	0
"301"	"Comet Quest"	0
"400"	"Comet Quest"	

Kueri serupa pada `GameTitleIndex` masih akan mengembalikan tiga item, bukan empat. Ini karena item dengan `TopScore` yang tidak ada tidak disebarkan ke indeks.

GameTitle	*TopScore*	*UserId*
"Comet Quest"	*0*	*"123"*
"Comet Quest"	*0*	*"201"*
"Comet Quest"	*0*	*"301"*

Proyeksi atribut

Proyeksi adalah kumpulan atribut yang disalin dari tabel ke indeks sekunder. Kunci partisi dan kunci urutan tabel selalu diproyeksikan ke dalam indeks; Anda dapat memproyeksikan atribut lain untuk mendukung persyaratan kueri aplikasi Anda. Saat Anda mengkueri indeks, Amazon DynamoDB

dapat mengakses atribut apa pun dalam proyeksi seolah-olah atribut tersebut berada dalam tabelnya sendiri.

Saat membuat indeks sekunder, Anda perlu menentukan atribut yang akan diproyeksikan ke dalam indeks. DynamoDB menyediakan tiga opsi berbeda untuk ini:

- KEYS_ONLY – Setiap item dalam indeks hanya terdiri dari kunci partisi tabel dan nilai kunci urutan, ditambah nilai kunci indeks. Opsi `KEYS_ONLY` menghasilkan indeks sekunder sekecil mungkin.
- INCLUDE – Selain atribut yang dijelaskan dalam `KEYS_ONLY`, indeks sekunder akan menyertakan atribut non-kunci lain yang Anda tentukan.
- ALL – Indeks sekunder menyertakan semua atribut dari tabel sumber. Karena semua data tabel diduplikasi dalam indeks, proyeksi `ALL` menghasilkan indeks sekunder sebesar yang mungkin.

Pada diagram sebelumnya, `GameTitleIndex` hanya memiliki satu atribut yang diproyeksikan: `UserId`. Jadi, meskipun aplikasi dapat secara efisien menentukan `UserId` dari pencetak skor terbanyak untuk setiap game menggunakan `GameTitle` dan `TopScore` dalam kueri, aplikasi tidak dapat secara efisien menentukan rasio kemenangan vs. kekalahan tertinggi untuk pencetak skor terbanyak. Untuk melakukannya, aplikasi harus melakukan kueri tambahan pada tabel dasar untuk mengambil kemenangan dan kekalahan untuk masing-masing pencetak skor terbanyak. Cara yang lebih efisien untuk mendukung kueri pada data ini adalah dengan memproyeksikan atribut-atribut ini dari tabel dasar ke dalam indeks sekunder global, seperti yang ditunjukkan dalam diagram ini.

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
...	...	...	...	...

Karena atribut non-kunci `Wins` dan `Losses` diproyeksikan ke dalam indeks, aplikasi dapat menentukan rasio kemenangan vs. kekalahan untuk game apa pun, atau untuk kombinasi game dan ID pengguna mana pun.

Saat memilih atribut untuk diproyeksikan ke dalam indeks sekunder global, Anda harus mempertimbangkan tradeoff antara biaya throughput yang disediakan dan biaya penyimpanan:

- Jika Anda hanya perlu mengakses beberapa atribut dengan latensi serendah mungkin, pertimbangkan untuk hanya memproyeksikan atribut tersebut ke dalam indeks sekunder global. Semakin kecil indeks, semakin sedikit biaya penyimpanannya, dan semakin sedikit pula biaya tulis Anda.
- Jika aplikasi Anda sering mengakses beberapa atribut non-kunci, sebaiknya Anda memproyeksikan atribut tersebut ke dalam indeks sekunder global. Biaya penyimpanan tambahan untuk indeks sekunder global mengimbangi biaya melakukan pemindaian tabel yang sering dilakukan.

- Jika sering mengakses sebagian besar atribut non-kunci, Anda dapat memproyeksikan atribut ini —atau bahkan seluruh tabel dasar— ke dalam indeks sekunder global. Hal ini akan memberi Anda fleksibilitas maksimum. Namun, biaya penyimpanan Anda akan meningkat, atau bahkan berlipat ganda.
- Jika aplikasi Anda jarang mengkueri tabel, tetapi harus melakukan banyak penulisan atau pembaruan terhadap data dalam tabel, pertimbangkan untuk memproyeksikan `KEYS_ONLY`. Indeks sekunder global akan berukuran minimal, tetapi akan tetap tersedia jika diperlukan untuk aktivitas kueri.

Membaca data dari Indeks Sekunder Global

Anda dapat mengambil item dari indeks sekunder global menggunakan operasi `Query` dan `Scan`. Operasi `GetItem` dan `BatchGetItem` tidak dapat digunakan pada indeks sekunder global.

Mengkueri Indeks Sekunder Global

Anda dapat menggunakan operasi `Query` untuk mengakses satu atau beberapa item dalam indeks sekunder global. Kueri harus menentukan nama tabel dasar dan nama indeks yang ingin Anda gunakan, atribut yang akan dihasilkan dalam hasil kueri, dan syarat kueri apa pun yang ingin Anda terapkan. DynamoDB dapat menampilkan hasil dalam urutan menaik atau menurun.

Pertimbangkan data berikut yang dihasilkan dari `Query` yang meminta data game untuk aplikasi papan peringkat.

```
{
    "TableName": "GameScores",
    "IndexName": "GameTitleIndex",
    "KeyConditionExpression": "GameTitle = :v_title",
    "ExpressionAttributeValues": {
        ":v_title": {"S": "Meteor Blasters"}
    },
    "ProjectionExpression": "UserId, TopScore",
    "ScanIndexForward": false
}
```

Dalam kueri ini:

- DynamoDB GameTitleIndexmengakses, menggunakan kunci partisi untuk menemukan GameTitleitem indeks untuk Meteor Blasters. Semua item indeks dengan kunci ini disimpan berdekatan satu sama lain untuk pengambilan cepat.

- Dalam game ini, DynamoDB menggunakan indeks untuk mengakses semua IDs pengguna dan skor teratas untuk game ini.
- Hasilnya ditampilkan dalam urutan menurun karena parameter `ScanIndexForward` diatur ke false.

Memindai Indeks Sekunder Global

Anda dapat menggunakan operasi `Scan` untuk mengambil semua data dari indeks sekunder global. Anda harus memberikan nama tabel dasar dan nama indeks dalam permintaan. Dengan `Scan`, DynamoDB membaca semua data dalam indeks dan mengembalikannya ke aplikasi. Anda juga dapat meminta agar hanya sebagian data yang dikembalikan, dan data yang tersisa harus dibuang. Untuk melakukannya, gunakan parameter `FilterExpression` dari operasi `Scan`. Untuk informasi selengkapnya, lihat Ekspresi filter untuk pemindaian.

Sinkronisasi data antara tabel dan Indeks Sekunder Global

DynamoDB otomatis menyinkronkan setiap indeks sekunder global dengan tabel dasarnya. Saat aplikasi menulis atau menghapus item dalam tabel, indeks sekunder global apa pun di tabel tersebut diperbarui secara asinkron, menggunakan model yang akhirnya konsisten. Aplikasi tidak pernah menulis langsung ke indeks. Namun, Anda harus memahami implikasi dari cara DynamoDB mempertahankan indeks ini.

Indeks sekunder global mewarisi mode kapasitas baca/tulis dari tabel dasar. Untuk informasi selengkapnya, lihat Pertimbangan saat mengganti mode kapasitas di DynamoDB.

Saat membuat indeks sekunder global, Anda menentukan satu atau beberapa atribut kunci indeks dan jenis datanya. Ini berarti bahwa setiap kali Anda menulis item ke tabel dasar, jenis data untuk atribut tersebut harus cocok dengan jenis data skema kunci indeks. Dalam kasus `GameTitleIndex`, kunci partisi `GameTitle` dalam indeks didefinisikan sebagai jenis data `String`. Kunci urutan `TopScore` dalam indeks adalah jenis `Number`. Jika Anda mencoba menambahkan item ke tabel `GameScores` dan menentukan jenis data yang berbeda untuk `GameTitle` atau `TopScore`, DynamoDB mengembalikan `ValidationException` karena jenis data tidak cocok.

Saat Anda memasukkan atau menghapus item dalam tabel, indeks sekunder global pada tabel tersebut diperbarui secara konsisten. Perubahan pada data tabel disebarkan ke indeks sekunder global dalam sepersekian detik, dalam kondisi normal. Namun, dalam beberapa skenario kegagalan yang tidak terduga, penundaan penyebaran yang lebih lama mungkin terjadi. Oleh karena itu,

aplikasi Anda perlu mengantisipasi dan menangani situasi ketika kueri pada indeks sekunder global mengembalikan hasil yang tidak mutakhir.

Jika menulis item ke tabel, Anda tidak perlu menentukan atribut untuk kunci urutan indeks sekunder global. Menggunakan `GameTitleIndex`, misalnya, Anda tidak perlu menentukan nilai atribut `TopScore` untuk menulis item baru ke tabel `GameScores`. Dalam hal ini, DynamoDB tidak menulis data apa pun ke indeks untuk item khusus ini.

Tabel dengan banyak indeks sekunder global menimbulkan biaya lebih tinggi untuk aktivitas tulis dibandingkan tabel dengan lebih sedikit indeks. Untuk informasi selengkapnya, lihat Pertimbangan Throughput yang disediakan untuk Indeks Sekunder Global.

Kelas tabel dengan Indeks Sekunder Global

Indeks sekunder global akan selalu menggunakan kelas tabel yang sama dengan tabel dasarnya. Setiap kali indeks sekunder global baru ditambahkan untuk tabel, indeks baru akan menggunakan kelas tabel yang sama dengan tabel dasarnya. Ketika kelas tabel diperbarui, semua indeks sekunder global terkait juga diperbarui.

Pertimbangan Throughput yang disediakan untuk Indeks Sekunder Global

Saat membuat indeks sekunder global pada tabel mode yang disediakan, Anda harus menentukan unit kapasitas baca dan tulis untuk beban kerja yang diharapkan pada indeks tersebut. Pengaturan throughput yang disediakan pada indeks sekunder global terpisah dari pengaturan tabel dasarnya. Operasi `Query` pada indeks sekunder global menggunakan unit kapasitas baca dari indeks, bukan tabel dasar. Saat Anda memasukkan, memperbarui, atau menghapus item dalam tabel, indeks sekunder global pada tabel tersebut juga diperbarui. Pembaruan indeks ini menggunakan unit kapasitas tulis dari indeks, bukan dari tabel dasar.

Misalnya, jika Anda `Query` indeks sekunder global dan melebihi kapasitas baca yang disediakan, permintaan Anda akan mengalami throttling. Jika Anda melakukan aktivitas tulis berat pada tabel, tetapi indeks sekunder global pada tabel tersebut memiliki kapasitas tulis yang tidak memadai, aktivitas tulis pada tabel akan mengalami throttling.

> **Important**
>
> Untuk menghindari potensi throttling, kapasitas tulis yang disediakan untuk indeks sekunder global harus sama atau lebih besar dari kapasitas tulis tabel dasar karena pembaruan baru menulis ke tabel dasar dan indeks sekunder global.

Untuk melihat pengaturan throughput yang disediakan untuk indeks sekunder global, gunakan operasi `DescribeTable`. Informasi mendetail tentang semua indeks sekunder global tabel ditampilkan.

Unit kapasitas baca

Indeks sekunder global mendukung bacaan akhir konsisten, yang masing-masing menggunakan setengah dari unit kapasitas baca. Artinya, satu kueri indeks sekunder global dapat mengambil hingga 2 × 4 KB = 8 KB per unit kapasitas baca.

Untuk kueri indeks sekunder global, DynamoDB menghitung aktivitas baca yang disediakan dengan cara yang sama seperti kueri terhadap tabel. Satu-satunya hal yang membedakan adalah penghitungan didasarkan pada ukuran entri indeks, bukan ukuran item dalam tabel dasar. Jumlah unit kapasitas baca adalah jumlah dari semua ukuran atribut yang diproyeksikan di semua item yang dikembalikan. Hasilnya kemudian dibulatkan ke batas 4 KB berikutnya. Untuk informasi selengkapnya tentang cara DynamoDB menghitung penggunaan throughput yang disediakan, lihat Mode kapasitas yang disediakan DynamoDB.

Ukuran maksimum hasil yang dikembalikan oleh operasi `Query` adalah 1 MB. Ini termasuk ukuran semua nama dan nilai atribut di semua item yang dikembalikan.

Misalnya, pertimbangkan indeks sekunder global yang setiap itemnya berisi 2.000 byte data. Sekarang, anggaplah Anda `Query` indeks ini dan `KeyConditionExpression` kueri cocok dengan delapan item. Ukuran total item yang cocok adalah 2.000 byte × 8 item = 16.000 byte. Hasil ini kemudian dibulatkan ke batas 4 KB terdekat. Karena kueri indeks sekunder global pada akhirnya konsisten, biaya totalnya adalah 0,5 × (16 KB / 4 KB), atau 2 unit kapasitas baca.

Unit kapasitas tulis

Ketika item dalam tabel ditambahkan, diperbarui, atau dihapus, dan indeks sekunder global terpengaruh oleh ini, indeks sekunder global menggunakan unit kapasitas tulis yang disediakan untuk operasi. Total biaya throughput yang disediakan untuk penulisan terdiri dari jumlah unit kapasitas tulis yang digunakan dengan menulis ke tabel dasar dan yang digunakan dengan memperbarui indeks sekunder global. Jika penulisan ke tabel tidak memerlukan pembaruan indeks sekunder global, tidak ada kapasitas tulis yang digunakan dari indeks.

Agar penulisan tabel berhasil, pengaturan throughput yang disediakan untuk tabel dan semua indeks sekunder globalnya harus memiliki kapasitas tulis yang memadai untuk mengakomodasi penulisan. Jika tidak, penulisan ke tabel akan mengalami throttling.

Biaya tulis item ke indeks sekunder global tergantung pada beberapa faktor:

- Jika Anda menulis item baru ke tabel yang menentukan atribut yang diindeks, atau Anda memperbarui item yang ada untuk menentukan atribut terindeks yang sebelumnya tidak ditentukan, satu operasi tulis diperlukan untuk memasukkan item ke dalam indeks.
- Jika pembaruan pada tabel mengubah nilai atribut kunci yang diindeks (dari A menjadi B), diperlukan dua penulisan, satu untuk menghapus item sebelumnya dari indeks dan satu lagi untuk memasukkan item baru ke dalam indeks.
- Jika suatu item ada dalam indeks, tetapi penulisan ke tabel menyebabkan atribut yang diindeks terhapus, satu penulisan diperlukan untuk menghapus proyeksi item lama dari indeks.
- Jika item tidak ada dalam indeks sebelum atau setelah item diperbarui, tidak ada biaya penulisan tambahan untuk indeks tersebut.
- Jika pembaruan pada tabel hanya mengubah nilai atribut yang diproyeksikan dalam skema kunci indeks, tetapi tidak mengubah nilai atribut kunci yang diindeks, satu penulisan diperlukan untuk memperbarui nilai atribut yang diproyeksikan ke dalam indeks.

Semua faktor ini mengasumsikan bahwa ukuran setiap item dalam indeks kurang dari atau sama dengan ukuran item 1 KB untuk menghitung unit kapasitas tulis. Entri indeks yang lebih besar memerlukan unit kapasitas tulis tambahan. Anda dapat meminimalkan biaya penulisan dengan mempertimbangkan atribut yang perlu dikembalikan oleh kueri dan hanya memproyeksikan atribut tersebut ke dalam indeks.

Pertimbangan penyimpanan untuk Indeks Sekunder Global

Saat aplikasi menulis item ke tabel, DynamoDB otomatis menyalin subset atribut yang benar ke indeks sekunder global tempat atribut tersebut akan muncul. AWS Akun Anda dikenakan biaya untuk penyimpanan item di tabel dasar dan juga untuk penyimpanan atribut dalam indeks sekunder global apa pun di tabel itu.

Jumlah ruang yang digunakan oleh item indeks adalah jumlah dari berikut ini:

- Ukuran kunci primer tabel dasar dalam byte (kunci partisi dan kunci urutan)
- Ukuran atribut kunci indeks dalam byte
- Ukuran atribut yang diproyeksikan dalam byte (jika ada)
- 100 byte dari overhead per item indeks

Untuk memperkirakan kebutuhan penyimpanan indeks sekunder global, Anda dapat memperkirakan ukuran rata-rata item dalam indeks lalu mengalikannya dengan jumlah item dalam tabel dasar yang memiliki atribut kunci indeks sekunder global.

Jika tabel berisi item dengan atribut tertentu tidak ditentukan, tetapi atribut tersebut ditentukan sebagai kunci partisi indeks atau kunci urutan, DynamoDB tidak menulis data apa pun untuk item tersebut ke indeks.

Mengelola Indeks Sekunder Global di DynamoDB

Bagian ini menjelaskan cara membuat, mengubah, dan menghapus indeks sekunder global di Amazon DynamoDB.

Topik

- Membuat tabel dengan Indeks Sekunder Global
- Mendeskripsikan Indeks Sekunder Global pada tabel
- Menambahkan Indeks Sekunder Global ke tabel yang sudah ada
- Menghapus Indeks Sekunder Global
- Mengubah Indeks Sekunder Global selama pembuatan

Membuat tabel dengan Indeks Sekunder Global

Untuk membuat tabel dengan satu atau beberapa indeks sekunder global, gunakan operasi `CreateTable` dengan parameter `GlobalSecondaryIndexes`. Untuk fleksibilitas kueri maksimum, Anda dapat membuat hingga 20 indeks sekunder global (kuota default) per tabel.

Anda harus menentukan satu atribut untuk bertindak sebagai kunci partisi indeks. Anda dapat secara opsional menentukan atribut lain untuk kunci urutan indeks. Atribut kunci ini tidak perlu sama dengan atribut kunci dalam tabel. Misalnya, dalam GameScorestabel (lihatMenggunakan Indeks Sekunder Global di DynamoDB), tidak `TopScore` `TopScoreDateTime` juga atribut kunci. Anda dapat membuat indeks sekunder global dengan kunci partisi `TopScore` dan kunci urutan `TopScoreDateTime`. Anda dapat menggunakan indeks semacam itu untuk menentukan apakah ada korelasi antara skor tinggi dan waktu game dimainkan.

Setiap atribut kunci indeks harus berupa skalar berjenis `String`, `Number`, atau `Binary`. (Tidak boleh berupa dokumen atau kumpulan.) Anda dapat memproyeksikan atribut jenis data apa pun ke dalam indeks sekunder global. Ini termasuk skalar, dokumen, dan kumpulan. Untuk daftar lengkap jenis data, lihat Jenis Data.

Jika menggunakan mode yang disediakan, Anda harus menyediakan pengaturan `ProvisionedThroughput` untuk indeks, yang terdiri dari `ReadCapacityUnits` dan `WriteCapacityUnits`. Pengaturan throughput yang disediakan ini terpisah dari pengaturan tabel, tetapi berperilaku dengan cara yang serupa. Untuk informasi selengkapnya, lihat Pertimbangan Throughput yang disediakan untuk Indeks Sekunder Global.

Indeks sekunder global mewarisi mode kapasitas baca/tulis dari tabel dasar. Untuk informasi selengkapnya, lihat Pertimbangan saat mengganti mode kapasitas di DynamoDB.

> **Note**
>
> Operasi backfill dan operasi tulis yang sedang berlangsung berbagi throughput tulis dalam indeks sekunder global. Saat membuat GSI baru, penting untuk memeriksa apakah pilihan kunci partisi Anda menghasilkan distribusi data atau lalu lintas yang tidak merata atau menyempit di seluruh nilai kunci partisi indeks baru. Jika ini terjadi, Anda mungkin melihat operasi backfill dan tulis terjadi secara bersamaan dan membatasi penulisan ke tabel dasar. Layanan mengambil tindakan untuk meminimalkan potensi skenario ini, tetapi tidak memiliki wawasan tentang bentuk data pelanggan sehubungan dengan kunci partisi indeks, proyeksi yang dipilih, atau ketersebaran kunci primer indeks.
> Jika Anda mencurigai bahwa indeks sekunder global Anda yang baru mungkin memiliki data atau distribusi lalu lintas yang sempit atau tidak merata di seluruh nilai kunci partisi, pertimbangkan hal berikut sebelum menambahkan indeks baru ke tabel yang penting secara operasional.
>
> - Mungkin yang paling aman adalah menambahkan indeks pada saat aplikasi Anda menghasilkan jumlah lalu lintas paling sedikit.
> - Pertimbangkan untuk mengaktifkan CloudWatch Contributor Insights pada tabel dasar dan indeks Anda. Ini akan memberi Anda wawasan berharga mengenai distribusi lalu lintas Anda.
> - Untuk indeks dan tabel dasar mode kapasitas yang disediakan, atur kapasitas tulis yang disediakan dari indeks baru Anda setidaknya dua kali lipat dari tabel dasar Anda. TontonWriteThrottleEvents,ThrottledRequests,OnlineIndexPercentageProgress, `OnlineIndexConsumedWriteCapacity` dan `OnlineIndexThrottleEvents` CloudWatch metrik selama proses berlangsung. Sesuaikan kapasitas tulis yang disediakan sesuai kebutuhan untuk menyelesaikan backfill dalam waktu yang wajar tanpa efek throttling yang signifikan pada operasi Anda yang sedang berlangsung.

- Bersiaplah untuk membatalkan pembuatan indeks jika Anda mengalami dampak operasional karena throttling tulis, dan peningkatan kapasitas tulis yang disediakan di GSI baru Anda tidak menyelesaikan masalah tersebut.

Mendeskripsikan Indeks Sekunder Global pada tabel

Untuk melihat status semua indeks sekunder global pada tabel, gunakan operasi `DescribeTable`. Bagian `GlobalSecondaryIndexes` dari respons menunjukkan semua indeks pada tabel, beserta status masing-masing indeks saat ini (`IndexStatus`).

`IndexStatus` untuk indeks sekunder global akan menjadi salah satu dari berikut ini:

- `CREATING` — Indeks sedang dibuat, dan belum tersedia untuk digunakan.
- `ACTIVE` — Indeks siap digunakan, dan aplikasi dapat melakukan operasi `Query` pada indeks.
- `UPDATING` — Pengaturan throughput indeks yang tersedia sedang diubah.
- `DELETING` — Indeks saat ini sedang dihapus, dan tidak dapat lagi digunakan.

Setelah DynamoDB selesai membuat indeks sekunder global, status indeks berubah dari `CREATING` menjadi `ACTIVE`.

Menambahkan Indeks Sekunder Global ke tabel yang sudah ada

Untuk menambahkan indeks sekunder global ke tabel yang sudah ada, gunakan operasi `UpdateTable` dengan parameter `GlobalSecondaryIndexUpdates`. Anda harus memasukkan berikut ini:

- Nama indeks. Nama harus berbeda dari semua indeks pada tabel.
- Skema kunci indeks. Anda harus menentukan satu atribut untuk kunci partisi indeks; Anda dapat secara opsional menentukan atribut lain untuk kunci urutan indeks. Atribut kunci ini tidak perlu sama dengan atribut kunci dalam tabel. Jenis data untuk setiap atribut skema harus skalar: `String`, `Number`, atau `Binary`.
- Atribut yang akan diproyeksikan dari tabel ke dalam indeks:
  - `KEYS_ONLY` — Setiap item dalam indeks hanya terdiri dari kunci partisi tabel dan nilai kunci urutan, ditambah nilai kunci indeks.
  - `INCLUDE` — Selain atribut yang dijelaskan dalam `KEYS_ONLY`, indeks sekunder menyertakan atribut non-kunci lain yang Anda tentukan.

  - `ALL` — Indeks menyertakan semua atribut dari tabel sumber.
- Setelan throughput yang disediakan untuk indeks, yang terdiri dari `ReadCapacityUnits` dan `WriteCapacityUnits`. Pengaturan throughput yang disediakan ini terpisah dari pengaturan tabel.

Anda hanya dapat membuat satu indeks sekunder global per operasi `UpdateTable`.

Fase pembuatan indeks

Saat Anda menambahkan indeks sekunder global baru ke tabel yang sudah ada, tabel akan terus tersedia meskipun indeks sedang dibuat. Namun, indeks baru tidak tersedia untuk operasi Kueri hingga statusnya berubah dari `CREATING` menjadi `ACTIVE`.

> **Note**
>
> Pembuatan indeks sekunder global tidak menggunakan Application Auto Scaling. Meningkatkan kapasitas Application Auto Scaling `MIN` tidak akan mengurangi waktu pembuatan indeks sekunder global.

Di balik layar, DynamoDB membuat indeks dalam dua fase:

Alokasi Sumber Daya

DynamoDB mengalokasikan sumber daya komputasi dan penyimpanan yang diperlukan untuk membuat indeks.

Selama fase alokasi sumber daya, atribut `IndexStatus` adalah `CREATING` dan atribut `Backfilling` adalah false. Gunakan operasi `DescribeTable` untuk mengambil status tabel dan semua indeks sekundernya.

Saat indeks berada dalam fase alokasi sumber daya, Anda tidak dapat menghapus indeks atau menghapus tabel induknya. Anda juga tidak dapat mengubah throughput indeks atau tabel yang disediakan. Anda tidak dapat menambahkan atau menghapus indeks lain pada tabel. Namun, Anda dapat mengubah throughput yang disediakan dari indeks lain tersebut.

Backfill

Untuk setiap item dalam tabel, DynamoDB menentukan kumpulan atribut yang akan ditulis ke indeks berdasarkan proyeksinya (`KEYS_ONLY`, `INCLUDE`, atau `ALL`). Kemudian atribut ini ditulis

ke indeks. Selama fase backfill, DynamoDB melacak item yang ditambahkan, dihapus, atau diperbarui dalam tabel. Atribut dari item ini juga ditambahkan, dihapus, atau diperbarui dalam indeks yang sesuai.

Selama fase backfill, atribut `IndexStatus` diatur ke `CREATING`, dan atribut `Backfilling` adalah true. Gunakan operasi `DescribeTable` untuk mengambil status tabel dan semua indeks sekundernya.

Saat indeks sedang melakukan backfill, Anda tidak dapat menghapus tabel induknya. Namun, Anda masih dapat menghapus indeks atau mengubah throughput tabel yang disediakan dan indeks sekunder globalnya.

> **Note**
>
> Selama fase backfill, beberapa penulisan item yang melanggar mungkin berhasil, sementara yang lain ditolak. Setelah backfill, semua penulisan ke item yang melanggar skema kunci indeks baru akan ditolak. Sebaiknya Anda menjalankan alat Detektor Pelanggaran setelah fase backfill selesai untuk mendeteksi dan mengatasi pelanggaran utama yang mungkin terjadi. Untuk informasi selengkapnya, lihat Mendeteksi dan memperbaiki pelanggaran kunci indeks di DynamoDB.

Saat fase alokasi sumber daya dan backfill sedang berlangsung, indeks berada dalam status `CREATING`. Selama waktu ini, DynamoDB melakukan operasi baca pada tabel. Operasi baca dari tabel dasar untuk mengisi indeks sekunder global tidak dikenakan biaya. Namun, operasi tulis untuk mengisi indeks sekunder global yang baru dibuat dikenakan biaya.

Setelah pembuatan indeks selesai, statusnya berubah menjadi `ACTIVE`. Anda tidak dapat `Query` atau `Scan` indeks sampai indeks itu `ACTIVE`.

> **Note**
>
> Dalam beberapa kasus, DynamoDB tidak dapat menulis data dari tabel ke indeks karena pelanggaran kunci indeks. Hal ini dapat terjadi jika:
>
> - Jenis data nilai atribut tidak cocok dengan jenis data skema kunci indeks.
> - Ukuran atribut melebihi panjang maksimum atribut kunci indeks.
> - Atribut kunci indeks memiliki String kosong atau nilai atribut Binari kosong.

Pelanggaran kunci indeks tidak mengganggu pembuatan indeks sekunder global. Namun, ketika indeks menjadi `ACTIVE`, kunci yang melanggar tidak ada dalam indeks. DynamoDB menyediakan alat mandiri untuk menemukan dan menyelesaikan masalah ini. Untuk informasi selengkapnya, lihat Mendeteksi dan memperbaiki pelanggaran kunci indeks di DynamoDB.

Menambahkan Indeks Sekunder Global ke tabel besar

Waktu yang dibutuhkan untuk membuat indeks sekunder global tergantung beberapa faktor, seperti berikut ini:

- Ukuran ruang tabel
- Jumlah item dalam tabel yang memenuhi syarat untuk dimasukkan dalam indeks
- Jumlah atribut yang diproyeksikan ke dalam indeks
- Kapasitas tulis yang disediakan pada indeks
- Aktivitas tulis di tabel utama selama pembuatan indeks

Jika Anda menambahkan indeks sekunder global ke tabel yang sangat besar, mungkin perlu waktu lama untuk menyelesaikan proses pembuatan. Untuk memantau kemajuan dan menentukan apakah indeks memiliki kapasitas tulis yang memadai, lihat CloudWatch metrik Amazon berikut:

- `OnlineIndexPercentageProgress`
- `OnlineIndexConsumedWriteCapacity`
- `OnlineIndexThrottleEvents`

Untuk informasi selengkapnya tentang CloudWatch metrik yang terkait dengan DynamoDB, lihat. Metrik DynamoDB

Important

Anda mungkin perlu mengizinkan daftar tabel yang sangat besar sebelum membuat atau memperbarui Indeks Sekunder Global. Silakan hubungi AWS Support untuk mengizinkan daftar tabel Anda.

Jika pengaturan throughput tulis yang disediakan pada indeks terlalu rendah, pembuatan indeks akan memakan waktu lebih lama untuk diselesaikan. Untuk mempersingkat waktu yang diperlukan dalam membuat indeks sekunder global baru, Anda dapat meningkatkan kapasitas tulis yang disediakan untuk sementara.

> **Note**
>
> Sebagai aturan umum, sebaiknya atur kapasitas tulis yang disediakan pada indeks menjadi 1,5 kali kapasitas tulis tabel. Ini adalah pengaturan yang bagus untuk banyak kasus penggunaan. Namun, kebutuhan Anda yang sebenarnya mungkin lebih tinggi atau lebih rendah.

Saat indeks sedang melakukan backfill, DynamoDB menggunakan kapasitas sistem internal untuk membaca dari tabel. Hal ini bertujuan untuk meminimalkan dampak pembuatan indeks dan untuk memastikan bahwa tabel Anda tidak kehabisan kapasitas baca.

Namun, ada kemungkinan volume aktivitas tulis yang masuk melebihi kapasitas tulis yang disediakan pada indeks. Ini adalah skenario kemacetan, yaitu pembuatan indeks memerlukan waktu lebih lama karena aktivitas penulisan ke indeks mengalami throttling. Selama pembuatan indeks, sebaiknya Anda memantau CloudWatch metrik Amazon untuk indeks guna menentukan apakah kapasitas tulis yang dikonsumsi melebihi kapasitas yang disediakannya. Dalam skenario kemacetan, Anda harus meningkatkan kapasitas tulis yang disediakan pada indeks untuk menghindari throttling tulis selama fase backfill.

Setelah indeks dibuat, Anda harus mengatur kapasitas tulis yang disediakan untuk mencerminkan penggunaan aplikasi yang normal.

Menghapus Indeks Sekunder Global

Jika tidak lagi memerlukan indeks sekunder global, Anda dapat menghapusnya menggunakan operasi `UpdateTable`.

Anda hanya dapat menghapus satu indeks sekunder global per operasi `UpdateTable`.

Saat indeks sekunder global sedang dihapus, aktivitas baca atau tulis apa pun di tabel induk tidak akan terpengaruh. Saat penghapusan sedang berlangsung, Anda masih dapat mengubah throughput yang disediakan pada indeks lain.

> **Note**
>
> - Saat Anda menghapus tabel menggunakan tindakan `DeleteTable`, semua indeks sekunder global pada tabel tersebut juga dihapus.
> - Akun Anda tidak akan dikenakan biaya untuk operasi penghapusan indeks sekunder global.

Mengubah Indeks Sekunder Global selama pembuatan

Saat indeks sedang dibuat, Anda dapat menggunakan operasi `DescribeTable` untuk mengetahui fasenya. Deskripsi untuk indeks menyertakan atribut Boolean, `Backfilling`, untuk menunjukkan apakah DynamoDB saat ini memuat indeks dengan item dari tabel. Jika nilai `Backfilling` adalah true, artinya fase alokasi sumber daya sudah selesai dan indeks sekarang melakukan backfill.

Saat backfill sedang berlangsung, Anda dapat memperbarui parameter throughput yang disediakan untuk indeks. Sebaiknya Anda melakukan ini untuk mempercepat pembuatan indeks: Anda dapat meningkatkan kapasitas tulis indeks saat sedang dibuat, lalu menurunkannya setelahnya. Untuk mengubah pengaturan throughput indeks yang disediakan, gunakan operasi `UpdateTable`. Status indeks berubah menjadi `UPDATING`, dan `Backfilling` adalah true hingga indeks siap digunakan.

Selama fase backfill, Anda dapat menghapus indeks yang sedang dibuat. Selama fase ini, Anda tidak dapat menambahkan atau menghapus indeks lain di tabel.

> **Note**
>
> Untuk indeks yang dibuat sebagai bagian dari operasi `CreateTable`, atribut `Backfilling` tidak muncul di output `DescribeTable`. Untuk informasi selengkapnya, lihat Fase pembuatan indeks.

Mendeteksi dan memperbaiki pelanggaran kunci indeks di DynamoDB

Selama fase backfill pembuatan indeks sekunder global, Amazon DynamoDB memeriksa setiap item dalam tabel untuk menentukan apakah item tersebut memenuhi syarat untuk dimasukkan dalam indeks. Beberapa item mungkin tidak memenuhi syarat karena akan menyebabkan pelanggaran kunci indeks. Dalam kasus ini, item tetap berada di tabel, tetapi indeks tidak memiliki entri yang sesuai untuk item tersebut.

Pelanggaran kunci indeks terjadi dalam situasi berikut:

- Ada ketidakcocokan jenis data antara nilai atribut dan jenis data skema kunci indeks. Misalnya, katakanlah salah satu item dalam tabel `GameScores` memiliki nilai `TopScore` jenis `String`. Jika Anda menambahkan indeks sekunder global dengan kunci partisi `TopScore`, jenis `Number`, item dari tabel tersebut akan melanggar kunci indeks.
- Nilai atribut dari tabel melebihi panjang maksimum untuk atribut kunci indeks. Panjang maksimum kunci partisi adalah 2048 byte, dan panjang maksimum kunci urutan adalah 1024 byte. Jika salah satu nilai atribut yang sesuai dalam tabel melebihi batas ini, item dari tabel akan melanggar kunci indeks.

> **Note**
>
> Jika nilai atribut Binari atau String ditetapkan untuk atribut yang digunakan sebagai kunci indeks, maka nilai atribut harus memiliki panjang lebih besar dari nol;, jika tidak, item dari tabel akan melanggar kunci indeks.
> Alat ini tidak menandai pelanggaran kunci indeks ini, saat ini.

Jika pelanggaran kunci indeks terjadi, fase backfill berlanjut tanpa gangguan. Namun, item yang melanggar tidak termasuk dalam indeks. Setelah fase backfill selesai, semua penulisan ke item yang melanggar skema kunci indeks baru akan ditolak.

Untuk mengidentifikasi dan memperbaiki nilai atribut dalam tabel yang melanggar kunci indeks, gunakan alat Detektor Pelanggaran. Untuk menjalankan Detektor Pelanggaran, buat file konfigurasi yang menentukan nama tabel yang akan dipindai, nama dan jenis data kunci partisi indeks sekunder global dan kunci urutan, serta tindakan yang harus diambil jika ditemukan pelanggaran kunci indeks. Detektor Pelanggaran dapat dijalankan dalam salah satu dari dua mode berbeda:

- Mode deteksi — Mendeteksi pelanggaran kunci indeks. Gunakan mode deteksi untuk melaporkan item dalam tabel yang akan menyebabkan pelanggaran kunci dalam indeks sekunder global. (Anda juga dapat meminta agar item tabel yang melanggar ini segera dihapus ketika ditemukan.) Output dari mode deteksi ditulis ke file, yang dapat Anda gunakan untuk analisis lebih lanjut.
- Mode koreksi — Memperbaiki pelanggaran kunci indeks. Dalam mode koreksi, Detektor Pelanggaran membaca file input dengan format yang sama dengan file output dari mode deteksi. Mode koreksi membaca catatan dari file input dan, untuk setiap catatan, mode ini menghapus atau memperbarui item yang sesuai dalam tabel. (Perhatikan bahwa jika Anda memilih untuk

memperbarui item, Anda harus mengedit file input dan menetapkan nilai yang sesuai untuk pembaruan ini.)

Mengunduh dan menjalankan Detektor Pelanggaran

Detektor Pelanggaran tersedia sebagai Java Archive (file `.jar`) yang dapat dieksekusi, dan dapat dijalankan di komputer Windows, macOS, atau Linux. Detektor Pelanggaran membutuhkan Java 1.7 (atau yang lebih baru) dan Apache Maven.

- Unduh detektor pelanggaran dari GitHub

Ikuti petunjuk dalam file `README.md` untuk mengunduh dan menginstal Detektor Pelanggaran menggunakan Maven.

Untuk memulai Detektor Pelanggaran, buka direktori tempat Anda membuat `ViolationDetector.java` dan masukkan perintah berikut.

```
java -jar ViolationDetector.jar [options]
```

Baris perintah Detektor Pelanggaran menerima opsi berikut:

- `-h | --help` — Mencetak ringkasan penggunaan dan opsi untuk Detektor Pelanggaran.
- `-p | --configFilePath value` — Nama yang sepenuhnya memenuhi syarat untuk file konfigurasi Detektor Pelanggaran. Untuk informasi selengkapnya, lihat File konfigurasi Detektor Pelanggaran.
- `-t | --detect value` — Mendeteksi pelanggaran kunci indeks di tabel, dan menuliskannya ke file output Detektor Pelanggaran. Jika nilai parameter ini diatur ke `keep`, item dengan pelanggaran kunci tidak diubah. Jika nilai diatur ke `delete`, item dengan pelanggaran kunci dihapus dari tabel.
- `-c | --correct value` — Membaca pelanggaran kunci indeks dari file input, dan mengambil tindakan perbaikan pada item dalam tabel. Jika nilai parameter ini disetel ke `update`, item dengan pelanggaran kunci diperbarui dengan nilai baru yang tidak melanggar. Jika nilai diatur ke `delete`, item dengan pelanggaran kunci dihapus dari tabel.

File konfigurasi Detektor Pelanggaran

Saat runtime, alat Detektor Pelanggaran memerlukan file konfigurasi. Parameter dalam file ini menentukan sumber daya DynamoDB yang dapat diakses oleh Detektor Pelanggaran, dan jumlah

throughput yang disediakan yang dapat digunakan oleh sumber daya tersebut. Tabel berikut mendeskripsikan parameter ini.

Nama parameter	Deskripsi	Diperlukan?

`awsCredentialsFile`	Nama yang sepenuhnya memenuhi syarat untuk file yang berisi kredensial AWS Anda. File kredensial harus dalam format berikut:
	`accessKey = access_key_id_goes_here`
	`secretKey = secret_key_goes_here`
		Ya

`dynamoDBRegion`	AWS Wilayah di mana tabel berada. Sebagai contoh: `us-west-2` .	Ya
`tableName`	Nama tabel DynamoDB yang akan dipindai.	Ya
`gsiHashKeyName`	Nama kunci partisi indeks.	Ya
`gsiHashKeyType`	Jenis data kunci partisi indeks —`String`, `Number`, atau `Binary`: `S | N | B`	Ya
`gsiRangeKeyName`	Nama kunci urutan indeks. Jangan tentukan parameter ini jika indeks hanya memiliki kunci primer sederhana (kunci partisi).	Tidak

Nama parameter	Deskripsi	Diperlukan?
`gsiRangeKeyType`	Jenis data untuk kunci urutan indeks—`String`, `Number`, atau `Binary`: `S | N | B` Jangan tentukan parameter ini jika indeks hanya memiliki kunci primer sederhana (kunci partisi).	Tidak
`recordDetails`	Apakah akan menulis detail lengkap pelanggaran kunci indeks ke file keluaran. Jika diatur ke `true` (default), informasi lengkap tentang item yang melanggar akan dilaporkan. Jika diatur ke `false`, hanya jumlah pelanggaran yang dilaporkan.	Tidak
`recordGsiValueInViolationRecord`	Apakah akan menulis nilai untuk kunci indeks yang melanggar ke file output. Jika diatur ke `true` (default), nilai kunci dilaporkan. Jika diatur ke `false`, nilai kunci tidak dilaporkan.	Tidak

Nama parameter	Deskripsi	Diperlukan?

`detectionOutputPath`	Jalur lengkap file output Detektor Pelanggaran. Parameter ini mendukung penulisan ke direktori lokal atau ke Amazon Simple Storage Service (Amazon S3). Berikut ini adalah beberapa contohnya:
	
	`detectionOutputPath = //local/path/filename.csv`
	
	`detectionOutputPath = s3://bucket/filename.csv`
	
	Informasi dalam file output muncul dalam format nilai yang dipisahkan koma (CSV). Jika Anda tidak mengatur `detectionOutputPath` , file outputnya bernama `violation_detection.csv` dan ditulis ke direktori kerja saat ini.
		Tidak

Nama parameter	Deskripsi	Diperlukan?
`numOfSegments`	Jumlah segmen pemindaian paralel yang akan digunakan saat Detektor Pelanggaran memindai tabel. Nilai defaultnya adalah 1, artinya tabel dipindai secara berurutan. Jika nilainya 2 atau lebih tinggi, Detektor Pelanggaran membagi tabel menjadi banyak segmen logis dan jumlah utas pemindaian yang sama. Pengaturan maksimum untuk `numOfSegments` adalah 4096. Untuk tabel yang lebih besar, pemindaian paralel umumnya lebih cepat daripada pemindaian berurutan. Selain itu, jika tabel cukup besar untuk menjangkau beberapa partisi, pemindaian paralel mendistribusikan aktivitas bacanya secara merata di beberapa partisi. Untuk informasi selengkapnya tentang pemindaian paralel di DynamoDB, lihat Pemindaian paralel.	Tidak

Nama parameter	Deskripsi	Diperlukan?
`numOfViolations`	Batas atas pelanggaran kunci indeks untuk menulis ke file output. Jika diatur ke `-1` (default), seluruh tabel dipindai. Jika diatur ke angka bulat positif, Detektor Pelanggaran akan berhenti setelah menemukan jumlah pelanggaran tersebut.	Tidak
`numOfRecords`	Jumlah item dalam tabel yang akan dipindai. Jika diatur ke -1 (default), seluruh tabel dipindai. Jika diatur ke angka bulat positif, Detektor Pelanggaran akan berhenti setelah memindai jumlah item tersebut dalam tabel.	Tidak
`readWriteIOPSPercent`	Mengatur persentase unit kapasitas baca yang disediakan yang digunakan selama pemindaian tabel. Nilai yang valid berkisar dari `1` sampai `100`. Nilai default (`25`) berarti bahwa Detektor Pelanggaran akan menggunakan tidak lebih dari 25% dari throughput baca yang disediakan pada tabel.	Tidak

Nama parameter	Deskripsi	Diperlukan?
`correctionInputPath`	Jalur lengkap file input koreksi Detektor Pelanggaran. Jika Anda menjalankan Detektor Pelanggaran dalam mode koreksi, konten file ini digunakan untuk mengubah atau menghapus item data dalam tabel yang melanggar indeks sekunder global. Format file `correctionInputPath` sama dengan format file `detectionOutputPath`. Ini memungkinkan Anda memproses output dari mode deteksi sebagai input dalam mode koreksi.	Tidak

Nama parameter	Deskripsi	Diperlukan?

`correctionOutputPath`	Jalur lengkap file output koreksi Detektor Pelanggaran. File ini dibuat hanya jika ada kesalahan pembaruan.
	
	Parameter ini mendukung penulisan ke direktori lokal atau ke Amazon S3. Berikut ini adalah beberapa contohnya:
	
	`correctionOutputPath = //local/path/filename.csv`
	
	`correctionOutputPath = s3://bucket/filename.csv`
	
	Informasi dalam file output ditampilkan dalam format CSV. Jika Anda tidak mengatur `correctionOutputPath`, file outputnya bernama `violation_update_errors.csv` dan ditulis ke direktori kerja saat ini.
		Tidak

Deteksi

Untuk mendeteksi pelanggaran kunci indeks, gunakan Detektor Pelanggaran dengan opsi baris perintah `--detect`. Untuk menunjukkan cara kerja opsi ini, pertimbangkan `ProductCatalog` tabel. Berikut ini adalah daftar item dalam tabel. Hanya kunci primer (`Id`) dan atribut `Price` yang ditampilkan.

Id (kunci primer)	Harga
101	5
102	20
103	200
201	100
202	200
203	300
204	400
205	500

Semua nilai untuk `Price` berjenis `Number`. Namun, karena DynamoDB tidak memiliki skema, Anda dapat menambahkan item dengan `Price` non-numerik. Misalnya, anggaplah Anda menambahkan item lain ke tabel `ProductCatalog`.

Id (kunci primer)	Harga
999	"Hello"

Tabel sekarang memiliki total sembilan item.

Sekarang Anda menambahkan indeks sekunder global baru ke tabel: `PriceIndex`. Kunci primer untuk indeks ini adalah kunci partisi, `Price`, yang berjenis `Number`. Setelah indeks dibuat, indeks akan berisi delapan item—tetapi tabel `ProductCatalog` memiliki sembilan item. Alasan perbedaan ini adalah bahwa nilai "`Hello`" adalah jenis `String`, tetapi `PriceIndex` memiliki kunci primer jenis `Number`. Nilai `String` melanggar kunci indeks sekunder global, sehingga tidak ada dalam indeks.

Untuk menggunakan Detektor Pelanggaran dalam skenario ini, Anda akan membuat file konfigurasi seperti berikut ini terlebih dahulu.

```
# 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
```

Selanjutnya, Anda menjalankan Detektor Pelanggaran seperti dalam contoh berikut.

```
$  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
```

Jika parameter konfigurasi `recordDetails` diatur ke `true`, Detektor Pelanggaran menulis detail setiap pelanggaran ke file output, seperti dalam contoh berikut.

```
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,,
```

File output dalam format CSV. Baris pertama dalam file tersebut adalah header, diikuti oleh satu catatan per item yang melanggar kunci indeks. Bidang catatan pelanggaran tersebut adalah sebagai berikut:

- Kunci hash tabel — Nilai kunci partisi item dalam tabel.
- Kunci rentang tabel — Nilai kunci urutan item dalam tabel.

- Nilai kunci hash GSI — Nilai kunci partisi indeks sekunder global.
- Jenis pelanggaran kunci hash GSI — `Type Violation` atau `Size Violation`.
- Deskripsi pelanggaran kunci hash GSI — Penyebab pelanggaran.
- Nilai pembaruan kunci hash GSI (UNTUK PENGGUNA) — Dalam mode koreksi, nilai baru yang diberikan pengguna untuk atribut tersebut.
- Nilai kunci rentang GSI — Nilai kunci urutan indeks sekunder global.
- Jenis pelanggaran kunci rentang GSI — `Type Violation` atau `Size Violation`.
- Deskripsi pelanggaran kunci rentang GSI — Penyebab pelanggaran.
- Nilai pembaruan kunci rentang GSI (UNTUK PENGGUNA) — Dalam mode koreksi, nilai baru yang diberikan pengguna untuk atribut tersebut.
- Hapus atribut kosong saat memperbarui (Y/T) — Dalam mode koreksi, menentukan apakah akan menghapus (Y) atau menyimpan (T) item yang melanggar dalam tabel—tetapi hanya jika salah satu bidang berikut ini kosong:
  - `GSI Hash Key Update Value(FOR USER)`
  - `GSI Range Key Update Value(FOR USER)`

  Jika salah satu dari bidang ini tidak kosong, `Delete Blank Attribute When Updating(Y/N)` tidak berpengaruh.

> **Note**
>
> Format output mungkin berbeda, tergantung file konfigurasi dan opsi baris perintah. Misalnya, jika tabel memiliki kunci primer sederhana (tanpa kunci urutan), tidak ada bidang kunci urutan yang akan ditampilkan dalam output.
> Catatan pelanggaran dalam file mungkin tidak diurutkan.

Koreksi

Untuk memperbaiki pelanggaran kunci indeks, gunakan Detektor Pelanggaran dengan opsi baris perintah `--correct`. Dalam mode koreksi, Detektor Pelanggaran membaca file input yang ditentukan oleh parameter `correctionInputPath`. File ini memiliki format yang sama dengan file `detectionOutputPath`, sehingga Anda dapat menggunakan output dari deteksi sebagai input untuk koreksi.

Detektor Pelanggaran menyediakan dua cara berbeda untuk memperbaiki pelanggaran kunci indeks:

- Hapus pelanggaran — Menghapus item tabel yang memiliki nilai atribut yang melanggar.
- Perbarui pelanggaran — Memperbarui item tabel, mengganti atribut yang melanggar dengan nilai yang tidak melanggar.

Dalam kasus tersebut, Anda dapat menggunakan file output dari mode deteksi sebagai input untuk mode koreksi.

Melanjutkan contoh `ProductCatalog`, misalkan Anda ingin menghapus item yang melanggar dari tabel. Untuk melakukannya, gunakan perintah berikut.

```
$  java -jar ViolationDetector.jar --configFilePath config.txt --correct delete
```

Pada tahap ini, Anda akan diminta untuk mengonfirmasi apakah Anda ingin menghapus item yang melanggar.

```
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
```

Sekarang, `ProductCatalog` dan `PriceIndex` memiliki jumlah item yang sama.

Menggunakan Indeks Sekunder Global: Java

Anda dapat menggunakan AWS SDK untuk Java Document API untuk membuat tabel Amazon DynamoDB dengan satu atau beberapa indeks sekunder global, menjelaskan indeks pada tabel, dan melakukan kueri menggunakan indeks.

Berikut ini adalah langkah-langkah umum untuk operasi tabel.

1. Buat instans dari kelas `DynamoDB`.
2. Berikan parameter wajib dan opsional untuk operasi dengan membuat objek permintaan yang sesuai.
3. Panggil metode sesuai yang ditentukan oleh klien yang Anda buat pada langkah sebelumnya.

Topik

- Membuat tabel dengan Indeks Sekunder Global
- Mendeskripsikan tabel dengan Indeks Sekunder Global
- Mengkueri Indeks Sekunder Global
- Contoh: Indeks Sekunder Global menggunakan API dokumen AWS SDK untuk Java

Membuat tabel dengan Indeks Sekunder Global

Anda dapat membuat indeks sekunder global pada saat membuat tabel. Untuk melakukannya, gunakan `CreateTable` dan berikan spesifikasi Anda untuk satu atau beberapa indeks sekunder global. Contoh kode Java berikut membuat tabel untuk menyimpan informasi tentang data cuaca. Kunci partisinya adalah `Location` dan kunci urutannya adalah `Date`. Indeks sekunder global bernama `PrecipIndex` memungkinkan akses cepat ke data curah hujan untuk berbagai lokasi.

Berikut adalah langkah-langkah untuk membuat tabel dengan indeks sekunder global, menggunakan API dokumen DynamoDB.

1. Buat instans dari kelas `DynamoDB`.
2. Buat instans dari kelas `CreateTableRequest` untuk memberikan informasi permintaan.

   Anda harus memberikan nama tabel, kunci primernya, dan nilai throughput yang ditentukan. Untuk indeks sekunder global, Anda harus memberikan nama indeks, pengaturan throughput yang ditentukan, definisi atribut untuk kunci urutan indeks, skema kunci untuk indeks, dan proyeksi atribut.
3. Panggil metode `createTable` dengan menentukan objek permintaan sebagai parameter.

Contoh kode Java berikut mendemonstrasikan langkah sebelumnya. Kode tersebut membuat tabel (`WeatherData`) dengan indeks sekunder global (`PrecipIndex`). Kunci partisi indeksnya adalah `Date` dan kunci urutannya adalah `Precipitation`. Semua atribut tabel diproyeksikan ke dalam indeks. Pengguna dapat mengkueri indeks ini untuk mendapatkan data cuaca untuk tanggal tertentu, yang secara opsional mengurutkan data berdasarkan jumlah curah hujan.

Karena `Precipitation` bukan atribut kunci untuk tabel tersebut, maka tidak diperlukan. Namun, item `WeatherData` tanpa `Precipitation` tidak muncul di `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());
```

Anda harus menunggu hingga DynamoDB membuat tabel dan menetapkan status tabel menjadi `ACTIVE`. Setelah itu, Anda bisa mulai memasukkan item data ke dalam tabel.

Mendeskripsikan tabel dengan Indeks Sekunder Global

Untuk mendapatkan informasi tentang indeks sekunder global pada sebuah tabel, gunakan `DescribeTable`. Untuk setiap indeks, Anda dapat mengakses namanya, skema kunci, dan atribut yang diproyeksikan.

Berikut ini adalah langkah-langkah untuk mengakses informasi indeks sekunder global sebuah tabel.

1. Buat instans dari kelas `DynamoDB`.
2. Buat instans dari kelas `Table` untuk mewakili indeks yang ingin Anda gunakan.
3. Panggil metode `describe` pada objek `Table`.

Contoh kode Java berikut mendemonstrasikan langkah sebelumnya.

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());
    }
}
```

Mengkueri Indeks Sekunder Global

Anda dapat menggunakan `Query` pada indeks sekunder global, sama seperti Anda `Query` tabel. Anda perlu menentukan nama indeks, kriteria kueri untuk kunci partisi indeks dan kunci urutan (jika ada), dan atribut yang ingin Anda kembalikan. Dalam contoh ini, indeks adalah `PrecipIndex`, yang memiliki kunci partisi `Date` dan kunci urutan `Precipitation`. Kueri indeks mengembalikan semua data cuaca untuk tanggal tertentu, dengan curah hujan lebih besar dari nol.

Berikut ini adalah langkah-langkah untuk query indeks sekunder global menggunakan AWS SDK untuk Java Document API.

1. Buat instans dari kelas `DynamoDB`.
2. Buat instans dari kelas `Table` untuk mewakili indeks yang ingin Anda gunakan.
3. Buat instans dari kelas `Index` untuk indeks yang ingin Anda kueri.
4. Panggil metode `query` pada objek `Index`.

Nama atribut `Date` adalah kata yang disimpan DynamoDB. Oleh karena itu, Anda harus menggunakan nama atribut ekspresi sebagai placeholder di `KeyConditionExpression`.

Contoh kode Java berikut mendemonstrasikan langkah sebelumnya.

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());
}
```

Contoh: Indeks Sekunder Global menggunakan API dokumen AWS SDK untuk Java

Contoh kode Java berikut menunjukkan cara menggunakan indeks sekunder global. Contoh tersebut membuat tabel bernama `Issues`, yang dapat digunakan dalam sistem pelacakan bug sederhana untuk pengembangan perangkat lunak. Kunci partisinya adalah `IssueId` dan kunci urutannya adalah `Title`. Ada tiga indeks sekunder global pada tabel ini:

- `CreateDateIndex` — Kunci partisinya adalah `CreateDate` dan kunci urutannya adalah `IssueId`. Selain kunci tabel, atribut `Description` dan `Status` diproyeksikan ke dalam indeks.
- `TitleIndex` — Kunci partisinya adalah `Title` dan kunci urutannya adalah `IssueId`. Tidak ada atribut selain kunci tabel yang diproyeksikan ke dalam indeks.
- `DueDateIndex` — Kunci partisinya adalah `DueDate`, dan tidak ada kunci urutan. Semua atribut tabel diproyeksikan ke dalam indeks.

Setelah tabel `Issues` dibuat, program memuat tabel dengan data yang mewakili laporan bug perangkat lunak. Kemudian, data dikueri menggunakan indeks sekunder global. Terakhir, program menghapus tabel `Issues`.

Untuk step-by-step instruksi untuk menguji contoh berikut, lihatContoh kode Java.

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);
    }

}
```

Menggunakan Indek Sekunder Global: .NET

Anda dapat menggunakan API AWS SDK untuk .NET tingkat rendah untuk membuat tabel Amazon DynamoDB dengan satu atau beberapa indeks sekunder global, menjelaskan indeks pada tabel, dan

melakukan kueri menggunakan indeks. Operasi ini dipetakan ke operasi DynamoDB yang sesuai. Untuk informasi selengkapnya, lihat Referensi Amazon DynamoDB API.

Berikut ini adalah langkah-langkah umum untuk operasi tabel menggunakan API tingkat rendah .NET.

1. Buat instans dari kelas `AmazonDynamoDBClient`.
2. Berikan parameter wajib dan opsional untuk operasi dengan membuat objek permintaan yang sesuai.

   Misalnya, buat objek `CreateTableRequest` untuk membuat tabel dan objek `QueryRequest` untuk mengkueri tabel atau indeks.
3. Jalankan metode sesuai yang ditentukan oleh klien yang Anda buat pada langkah sebelumnya.

Topik

- Membuat tabel dengan Indeks Sekunder Global
- Mendeskripsikan tabel dengan Indeks Sekunder Global
- Mengkueri Indeks Sekunder Global
- Contoh: Indeks Sekunder Global menggunakan API tingkat rendah AWS SDK untuk .NET

Membuat tabel dengan Indeks Sekunder Global

Anda dapat membuat indeks sekunder global pada saat membuat tabel. Untuk melakukannya, gunakan `CreateTable` dan berikan spesifikasi Anda untuk satu atau beberapa indeks sekunder global. Contoh kode C# berikut membuat tabel untuk menyimpan informasi tentang data cuaca. Kunci partisinya adalah `Location` dan kunci urutannya adalah `Date`. Indeks sekunder global bernama `PrecipIndex` memungkinkan akses cepat ke data curah hujan untuk berbagai lokasi.

Berikut ini adalah langkah-langkah untuk membuat tabel dengan indeks sekunder global, menggunakan API tingkat rendah .NET.

1. Buat instans dari kelas `AmazonDynamoDBClient`.
2. Buat instans dari kelas `CreateTableRequest` untuk memberikan informasi permintaan.

   Anda harus memberikan nama tabel, kunci primernya, dan nilai throughput yang ditentukan. Untuk indeks sekunder global, Anda harus memberikan nama indeks, pengaturan throughput yang ditentukan, definisi atribut untuk kunci urutan indeks, skema kunci untuk indeks, dan proyeksi atribut.

3. Jalankan metode `CreateTable` dengan menentukan objek permintaan sebagai parameter.

Contoh kode #C berikut mendemonstrasikan langkah sebelumnya. Kode tersebut membuat tabel (`WeatherData`) dengan indeks sekunder global (`PrecipIndex`). Kunci partisi indeksnya adalah `Date` dan kunci urutannya adalah `Precipitation`. Semua atribut tabel diproyeksikan ke dalam indeks. Pengguna dapat mengkueri indeks ini untuk mendapatkan data cuaca untuk tanggal tertentu, yang secara opsional mengurutkan data berdasarkan jumlah curah hujan.

Karena `Precipitation` bukan atribut kunci untuk tabel tersebut, maka tidak diperlukan. Namun, item `WeatherData` tanpa `Precipitation` tidak muncul di `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);
```

Anda harus menunggu hingga DynamoDB membuat tabel dan menetapkan status tabel menjadi `ACTIVE`. Setelah itu, Anda bisa mulai memasukkan item data ke dalam tabel.

Mendeskripsikan tabel dengan Indeks Sekunder Global

Untuk mendapatkan informasi tentang indeks sekunder global pada sebuah tabel, gunakan `DescribeTable`. Untuk setiap indeks, Anda dapat mengakses namanya, skema kunci, dan atribut yang diproyeksikan.

Berikut ini adalah langkah-langkah untuk mengakses informasi indeks sekunder global untuk tabel menggunakan API tingkat rendah .NET.

1. Buat instans dari kelas `AmazonDynamoDBClient`.
2. Jalankan metode `describeTable` dengan menyediakan objek permintaan sebagai parameter.

   Buat instans dari kelas `DescribeTableRequest` untuk memberikan informasi permintaan. Anda harus memberikan nama tabel.
3.

Contoh kode #C berikut mendemonstrasikan langkah sebelumnya.

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);
      }
}
```

Mengkueri Indeks Sekunder Global

Anda dapat menggunakan `Query` pada indeks sekunder global, sama seperti Anda `Query` tabel. Anda perlu menentukan nama indeks, kriteria kueri untuk kunci partisi indeks dan kunci urutan (jika ada), dan atribut yang ingin Anda kembalikan. Dalam contoh ini, indeks adalah `PrecipIndex`, yang memiliki kunci partisi `Date` dan kunci urutan `Precipitation`. Kueri indeks mengembalikan semua data cuaca untuk tanggal tertentu, dengan curah hujan lebih besar dari nol.

Berikut ini adalah langkah-langkah untuk mengkueri indeks sekunder global menggunakan API tingkat rendah .NET.

1. Buat instans dari kelas `AmazonDynamoDBClient`.
2. Buat instans dari kelas `QueryRequest` untuk memberikan informasi permintaan.
3. Jalankan metode `query` dengan menentukan objek permintaan sebagai parameter.

Nama atribut `Date` adalah kata yang disimpan DynamoDB. Oleh karena itu, Anda harus menggunakan nama atribut ekspresi sebagai placeholder di `KeyConditionExpression`.

Contoh kode #C berikut mendemonstrasikan langkah sebelumnya.

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();
}
```

Contoh: Indeks Sekunder Global menggunakan API tingkat rendah AWS SDK untuk .NET

Contoh kode C# berikut menunjukkan cara menggunakan indeks sekunder global. Contoh tersebut membuat tabel bernama `Issues`, yang dapat digunakan dalam sistem pelacakan bug sederhana untuk pengembangan perangkat lunak. Kunci partisinya adalah `IssueId` dan kunci urutannya adalah `Title`. Ada tiga indeks sekunder global pada tabel ini:

- `CreateDateIndex` — Kunci partisinya adalah `CreateDate` dan kunci urutannya adalah `IssueId`. Selain kunci tabel, atribut `Description` dan `Status` diproyeksikan ke dalam indeks.
- `TitleIndex` — Kunci partisinya adalah `Title` dan kunci urutannya adalah `IssueId`. Tidak ada atribut selain kunci tabel yang diproyeksikan ke dalam indeks.
- `DueDateIndex` — Kunci partisinya adalah `DueDate`, dan tidak ada kunci urutan. Semua atribut tabel diproyeksikan ke dalam indeks.

Setelah tabel `Issues` dibuat, program memuat tabel dengan data yang mewakili laporan bug perangkat lunak. Kemudian, data dikueri menggunakan indeks sekunder global. Terakhir, program menghapus tabel `Issues`.

Untuk step-by-step petunjuk pengujian sampel berikut, lihatContoh kode .NET.

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;
                }
            }
        }
    }
}
```

Bekerja dengan Indeks Sekunder Global di DynamoDB menggunakan AWS CLI

Anda dapat menggunakan AWS CLI untuk membuat tabel Amazon DynamoDB dengan satu atau lebih indeks sekunder global, menjelaskan indeks pada tabel, dan melakukan kueri menggunakan indeks.

Topik

- Membuat tabel dengan Indeks Sekunder Global
- Menambahkan Indeks Sekunder Global ke tabel yang sudah ada
- Mendeskripsikan tabel dengan Indeks Sekunder Global
- Mengkueri Indeks Sekunder Global

Membuat tabel dengan Indeks Sekunder Global

Indeks sekunder global dapat dibuat pada saat Anda membuat tabel. Untuk melakukannya, gunakan parameter `create-table` dan berikan spesifikasi Anda untuk satu atau beberapa indeks sekunder global. Contoh berikut membuat tabel bernama `GameScores` dengan indeks sekunder global yang disebut `GameTitleIndex`. Tabel dasar memiliki kunci partisi `UserId` dan kunci urutan `GameTitle`, sehingga Anda dapat menemukan skor terbaik pengguna individu untuk game tertentu secara efisien, sedangkan GSI memiliki kunci partisi `GameTitle` dan kunci urutan `TopScore`, memungkinkan Anda untuk cepat menemukan skor tertinggi secara keseluruhan untuk game tertentu.

```
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
                }
            }
        ]"
```

Anda harus menunggu hingga DynamoDB membuat tabel dan menetapkan status tabel menjadi `ACTIVE`. Setelah itu, Anda bisa mulai memasukkan item data ke dalam tabel. Anda dapat menggunakan describe-table untuk menentukan status pembuatan tabel.

Menambahkan Indeks Sekunder Global ke tabel yang sudah ada

Indeks sekunder global juga dapat ditambahkan atau diubah setelah pembuatan tabel. Untuk melakukannya, gunakan parameter `update-table` dan berikan spesifikasi Anda untuk satu atau beberapa indeks sekunder global. Contoh berikut menggunakan skema yang sama seperti contoh sebelumnya, tetapi mengasumsikan bahwa tabel telah dibuat dan GSI akan ditambahkan nanti.

```
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\"]
                    }
                }
            }
        ]"
```

Mendeskripsikan tabel dengan Indeks Sekunder Global

Untuk mendapatkan informasi tentang Indeks Sekunder Global pada sebuah tabel, gunakan parameter `describe-table`. Untuk setiap indeks, Anda dapat mengakses namanya, skema kunci, dan atribut yang diproyeksikan.

```
aws dynamodb describe-table --table-name GameScores
```

Mengkueri Indeks Sekunder Global

Anda dapat menggunakan operasi `query` pada indeks sekunder global dengan cara yang hampir sama seperti Anda `query` tabel. Anda harus menentukan nama indeks, kriteria kueri untuk kunci urutan indeks, dan atribut yang ingin Anda kembalikan. Dalam contoh ini, indeks adalah `GameTitleIndex` dan kunci urutan indeks adalah `GameTitle`.

Satu-satunya atribut yang dikembalikan adalah atribut yang telah diproyeksikan ke dalam indeks. Anda dapat memodifikasi kueri ini untuk memilih atribut non-kunci juga, tetapi ini akan memerlukan aktivitas pengambilan tabel yang relatif mahal. Untuk informasi selengkapnya tentang pengambilan tabel, lihat Proyeksi atribut.

```
aws dynamodb query --table-name GameScores\
    --index-name GameTitleIndex \
    --key-condition-expression "GameTitle = :v_game" \
    --expression-attribute-values '{":v_game":{"S":"Alien Adventure"} }'
```

Indeks Sekunder Lokal di DynamoDB

Beberapa aplikasi hanya perlu mengkueri data menggunakan kunci primer tabel dasar. Namun, mungkin ada situasi ketika kunci urutan alternatif akan berguna. Untuk memberi aplikasi Anda pilihan kunci urutan, Anda dapat membuat satu atau beberapa indeks sekunder lokal pada tabel Amazon DynamoDB dan mengeluarkan permintaan `Query` atau `Scan` terhadap indeks ini.

Topik

- Skenario: Menggunakan Indeks Sekunder Lokal
- Proyeksi atribut
- Membuat Indeks Sekunder Lokal
- Membaca data dari Indeks Sekunder Lokal
- Penulisan item dan Indeks Sekunder Lokal
- Pertimbangan throughput yang disediakan untuk Indeks Sekunder Lokal
- Pertimbangan penyimpanan untuk Indeks Sekunder Lokal
- Kumpulan item dalam Indeks Sekunder Lokal
- Menggunakan Indeks Sekunder Lokal: Java
- Menggunakan Indeks Sekunder Lokal: .NET
- Bekerja dengan Indeks Sekunder Lokal di DynamoDB AWS CLI

Skenario: Menggunakan Indeks Sekunder Lokal

Sebagai contoh, perhatikan `Thread` tabel. Tabel ini berguna untuk aplikasi seperti forum diskusi AWS. Diagram berikut menunjukkan bagaimana item dalam tabel akan diatur. (Tidak semua atribut ditampilkan.)

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 menyimpan semua item dengan nilai kunci partisi yang sama secara berkelanjutan. Dalam contoh ini, dengan `ForumName` tertentu, operasi `Query` dapat segera menemukan semua utas untuk forum itu. Dalam grup item dengan nilai kunci partisi yang sama, item diurutkan berdasarkan nilai kunci urutan. Jika kunci urutan (`Subject`) juga disediakan dalam kueri, DynamoDB dapat mempersempit hasil yang dikembalikan—misalnya, mengembalikan semua utas di forum "S3" yang memiliki `Subject` yang diawali dengan huruf "a".

Beberapa permintaan mungkin memerlukan pola akses data yang lebih kompleks. Misalnya:

- Utas forum mana yang paling banyak dilihat dan mendapatkan balasan?
- Utas mana di forum tertentu yang memiliki jumlah pesan terbanyak?
- Berapa banyak utas yang diposting di forum tertentu dalam jangka waktu tertentu?

Untuk menjawab pertanyaan-pertanyaan ini, tindakan `Query` tidak akan cukup. Sebaliknya, Anda harus `Scan` seluruh tabel. Untuk tabel dengan jutaan item, ini akan menghabiskan banyak throughput baca yang disediakan dan membutuhkan waktu lama untuk selesai.

Namun, Anda dapat menentukan satu atau beberapa indeks sekunder lokal pada atribut non-kunci, seperti `Replies` atau `LastPostDateTime`.

Indeks sekunder lokal mempertahankan kunci urutan alternatif untuk nilai kunci partisi yang diberikan. Indeks sekunder lokal juga berisi salinan beberapa atau semua atribut dari tabel dasarnya. Anda akan menentukan atribut yang diproyeksikan ke indeks sekunder lokal saat membuat tabel. Data dalam indeks sekunder lokal diatur oleh kunci partisi yang sama dengan tabel dasar, tetapi dengan

kunci urutan yang berbeda. Ini memungkinkan Anda mengakses item data secara efisien di seluruh dimensi yang berbeda ini. Untuk fleksibilitas kueri atau pemindaian yang lebih baik, Anda dapat membuat hingga lima indeks sekunder lokal per tabel.

Misalkan sebuah aplikasi perlu menemukan semua utas yang telah diposting dalam tiga bulan terakhir di forum tertentu. Tanpa indeks sekunder lokal, aplikasi harus `Scan` seluruh tabel `Thread` dan membuang postingan apa pun yang tidak berada dalam periode waktu yang ditentukan. Dengan indeks sekunder lokal, operasi `Query` dapat menggunakan `LastPostDateTime` sebagai kunci urutan dan menemukan data dengan cepat.

Diagram berikut menunjukkan indeks sekunder lokal bernama `LastPostIndex`. Perhatikan bahwa kunci partisi sama dengan tabel `Thread`, tetapi kunci urutannya adalah `LastPostDateTime`.

LastPostIndex

ForumName	*LastPostDateTime*	*Subject*
"S3"	*"2015-01-03:09:21:11"*	*"ddd"*
"S3"	*"2015-01-22:23:18:01"*	*"bbb"*
"S3"	*"2015-02-31:13:14:21"*	*"ccc"*
"S3"	*"2015-03-15:17:24:31"*	*"aaa"*
"EC2"	*"2015-01-18:07:33:42"*	*"zzz"*
"EC2"	*"2015-02-12:11:07:56"*	*"yyy"*
"RDS"	*"2015-01-19:01:13:24"*	*"rrr"*
"RDS"	*"2015-02-22:12:19:44"*	*"ttt"*
"RDS"	*"2015-03-11:06:53:00"*	*"sss"*
...	...	...

Setiap indeks sekunder lokal harus memenuhi syarat berikut:

- Kunci partisi sama dengan tabel dasarnya.
- Kunci urutan terdiri dari satu atribut skalar.

- Kunci urutan tabel dasar diproyeksikan ke dalam indeks, tempat kunci tersebut berfungsi sebagai atribut non-kunci.

Dalam contoh ini, kunci partisi adalah `ForumName` dan kunci urutan indeks sekunder lokal adalah `LastPostDateTime`. Selain itu, nilai kunci urutan dari tabel dasar (dalam contoh ini, `Subject`) diproyeksikan ke dalam indeks, tetapi bukan merupakan bagian dari kunci indeks. Jika aplikasi membutuhkan daftar yang didasarkan pada `ForumName` dan `LastPostDateTime`, aplikasi dapat mengeluarkan permintaan `Query` terhadap `LastPostIndex`. Hasil kueri diurutkan menurut `LastPostDateTime`, dan dapat dikembalikan dalam urutan menaik atau menurun. Kueri juga dapat menerapkan syarat kunci, seperti hanya mengembalikan item yang memiliki `LastPostDateTime` dalam rentang waktu tertentu.

Setiap indeks sekunder lokal otomatis berisi kunci partisi dan kunci urutan dari tabel dasarnya; Anda secara opsional dapat memproyeksikan atribut non-kunci ke dalam indeks. Saat Anda mengkueri indeks, DynamoDB dapat mengambil atribut yang diproyeksikan ini secara efisien. Saat Anda mengkueri indeks sekunder lokal, kueri juga dapat mengambil atribut yang tidak diproyeksikan ke dalam indeks. DynamoDB otomatis mengambil atribut ini dari tabel dasar, tetapi dengan latensi lebih besar dan dengan biaya throughput yang disediakan lebih tinggi.

Untuk indeks sekunder lokal apa pun, Anda dapat menyimpan hingga 10 GB data per nilai kunci partisi yang berbeda. Angka ini mencakup semua item dalam tabel dasar, ditambah semua item dalam indeks, yang memiliki nilai kunci partisi yang sama. Untuk informasi selengkapnya, lihat Kumpulan item dalam Indeks Sekunder Lokal.

Proyeksi atribut

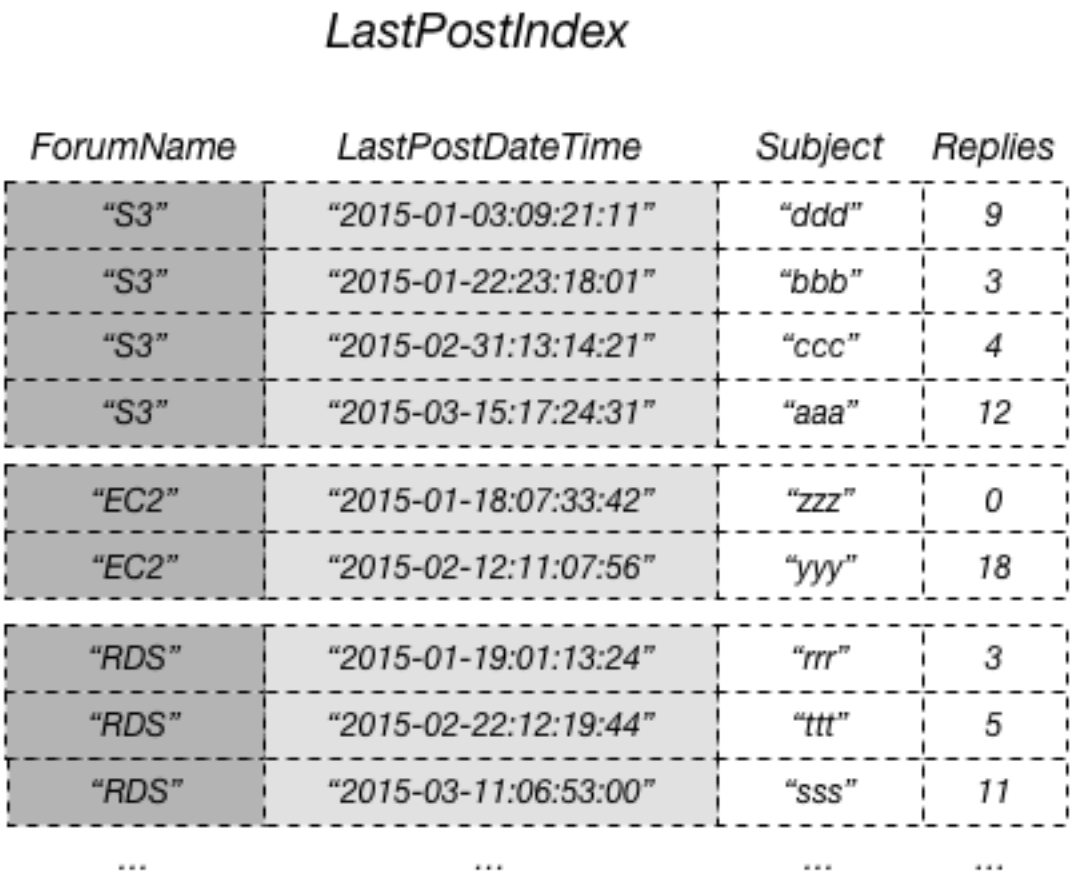
Dengan `LastPostIndex`, sebuah aplikasi dapat menggunakan `ForumName` dan `LastPostDateTime` sebagai kriteria kueri. Namun, untuk mengambil atribut tambahan apa pun, DynamoDB harus melakukan operasi baca tambahan terhadap tabel `Thread`. Pembacaan tambahan ini dikenal sebagai pengambilan, dan pembacaan tersebut dapat meningkatkan jumlah total throughput yang disediakan yang diperlukan untuk kueri.

Misalkan Anda ingin mengisi halaman web dengan daftar semua utas di "S3" dan jumlah balasan untuk setiap utas, diurutkan berdasarkan tanggal/waktu balasan terakhir yang dimulai dengan balasan terbaru. Untuk mengisi daftar ini, Anda memerlukan atribut berikut:

- `Subject`
- `Replies`

- `LastPostDateTime`

Cara paling efisien untuk mengkueri data ini dan untuk menghindari operasi pengambilan adalah dengan memproyeksikan atribut `Replies` dari tabel ke dalam indeks sekunder lokal, seperti yang ditunjukkan dalam diagram ini.



Proyeksi adalah kumpulan atribut yang disalin dari tabel ke indeks sekunder. Kunci partisi dan kunci urutan tabel selalu diproyeksikan ke dalam indeks; Anda dapat memproyeksikan atribut lain untuk mendukung persyaratan kueri aplikasi Anda. Saat Anda mengkueri indeks, Amazon DynamoDB dapat mengakses atribut apa pun dalam proyeksi seolah-olah atribut tersebut berada dalam tabelnya sendiri.

Saat membuat indeks sekunder, Anda perlu menentukan atribut yang akan diproyeksikan ke dalam indeks. DynamoDB menyediakan tiga opsi berbeda untuk ini:

- KEYS_ONLY – Setiap item dalam indeks hanya terdiri dari kunci partisi tabel dan nilai kunci urutan, ditambah nilai kunci indeks. Opsi `KEYS_0NLY` menghasilkan indeks sekunder sekecil mungkin.

- INCLUDE – Selain atribut yang dijelaskan dalam KEYS_ONLY, indeks sekunder akan menyertakan atribut non-kunci lain yang Anda tentukan.
- ALL – Indeks sekunder menyertakan semua atribut dari tabel sumber. Karena semua data tabel diduplikasi dalam indeks, proyeksi ALL menghasilkan indeks sekunder sebesar yang mungkin.

Pada diagram sebelumnya, atribut non-kunci Replies diproyeksikan ke dalam LastPostIndex. Aplikasi dapat meminta LastPostIndex sebagai ganti dari tabel Thread lengkap untuk mengisi halaman web dengan Subject, Replies, dan LastPostDateTime. Jika ada atribut non-kunci lain yang diminta, DynamoDB perlu mengambil atribut tersebut dari tabel Thread.

Dari sudut pandang aplikasi, pengambilan atribut tambahan dari tabel dasar dilakukan secara otomatis dan transparan, jadi tidak perlu menulis ulang logika aplikasi apa pun. Namun, pengambilan tersebut dapat sangat mengurangi keuntungan performa menggunakan indeks sekunder lokal.

Saat memilih atribut untuk diproyeksikan ke dalam indeks sekunder lokal, Anda harus mempertimbangkan tradeoff antara biaya throughput yang disediakan dan biaya penyimpanan:

- Jika Anda hanya perlu mengakses beberapa atribut dengan latensi serendah mungkin, pertimbangkan untuk hanya memproyeksikan atribut tersebut ke dalam indeks sekunder lokal. Semakin kecil indeks, semakin sedikit biaya penyimpanannya, dan semakin sedikit pula biaya tulis Anda. Jika ada atribut yang terkadang perlu Anda ambil, biaya untuk throughput yang disediakan mungkin lebih besar daripada biaya jangka panjang untuk menyimpan atribut tersebut.
- Jika aplikasi Anda sering mengakses beberapa atribut non-kunci, sebaiknya Anda memproyeksikan atribut tersebut ke dalam indeks sekunder lokal. Biaya penyimpanan tambahan untuk indeks sekunder lokal mengimbangi biaya melakukan pemindaian tabel yang sering dilakukan.
- Jika sering mengakses sebagian besar atribut non-kunci, Anda dapat memproyeksikan atribut ini—atau bahkan seluruh tabel dasar— ke dalam indeks sekunder lokal. Ini memberi Anda fleksibilitas maksimum dan penggunaan persediaan throughput terendah, karena pengambilan tidak diperlukan. Namun, biaya penyimpanan Anda akan meningkat, atau bahkan dua kali lipat jika Anda memproyeksikan semua atribut.
- Jika aplikasi Anda jarang mengkueri tabel, tetapi harus melakukan banyak penulisan atau pembaruan terhadap data dalam tabel, pertimbangkan untuk memproyeksikan KEYS_ONLY. Indeks sekunder lokal akan berukuran minimal, tetapi akan tetap tersedia jika diperlukan untuk aktivitas kueri.

Membuat Indeks Sekunder Lokal

Untuk membuat satu atau beberapa indeks sekunder lokal pada tabel, gunakan parameter `LocalSecondaryIndexes` operasi `CreateTable`. Indeks sekunder lokal pada tabel dibuat saat tabel dibuat. Saat Anda menghapus tabel, indeks sekunder lokal apa pun di tabel itu juga akan dihapus.

Anda harus menentukan satu atribut non-kunci untuk berfungsi sebagai kunci urutan indeks sekunder lokal. Atribut yang Anda pilih harus berupa skalar `String`, `Number`, atau `Binary`. Jenis skalar, jenis dokumen, dan jenis kumpulan lain tidak diperbolehkan. Untuk daftar lengkap jenis data, lihat Jenis Data.

> **Important**
>
> Untuk tabel dengan indeks sekunder lokal, ada batas ukuran 10 GB per nilai kunci partisi. Tabel dengan indeks sekunder lokal dapat menyimpan berapa pun jumlah item, asalkan ukuran total untuk nilai kunci satu partisi tidak melebihi 10 GB. Untuk informasi selengkapnya, lihat Batas ukuran kumpulan item.

Anda dapat memproyeksikan atribut jenis data apa pun ke dalam indeks sekunder lokal. Ini termasuk skalar, dokumen, dan kumpulan. Untuk daftar lengkap jenis data, lihat Jenis Data.

Membaca data dari Indeks Sekunder Lokal

Anda dapat mengambil item dari indeks sekunder lokal menggunakan operasi `Query` dan `Scan`. Operasi `GetItem` dan `BatchGetItem` tidak dapat digunakan pada indeks sekunder lokal.

Mengkueri Indeks Sekunder Lokal

Dalam tabel DynamoDB, nilai kunci partisi gabungan dan nilai kunci urutan untuk setiap item harus unik. Namun, dalam indeks sekunder lokal, nilai kunci urutan tidak perlu unik untuk nilai kunci partisi yang diberikan. Jika ada beberapa item dalam indeks sekunder lokal yang memiliki nilai kunci urutan yang sama, operasi `Query` mengembalikan semua item yang memiliki nilai kunci partisi yang sama. Sebagai respons, item yang cocok tidak dikembalikan dalam urutan tertentu.

Anda dapat mengkueri indeks sekunder lokal menggunakan bacaan akhir konsisten atau bacaan sangat konsisten. Untuk menentukan jenis konsistensi yang Anda inginkan, gunakan parameter `ConsistentRead` operasi `Query`. Bacaan sangat konsisten dari indeks sekunder lokal selalu

mengembalikan nilai terbaru yang diperbarui. Jika kueri perlu mengambil atribut tambahan dari tabel dasar, atribut tersebut akan konsisten dengan indeks.

Example

Pertimbangkan data berikut yang dikembalikan dari `Query` yang meminta data dari utas diskusi di forum tertentu.

```
{
    "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"}
    }
}
```

Dalam kueri ini:

- DynamoDB `LastPostIndex` mengakses, menggunakan kunci `ForumName` partisi untuk menemukan item indeks untuk "". EC2 Semua item indeks dengan kunci ini disimpan berdekatan satu sama lain untuk pengambilan cepat.
- Dalam forum ini, DynamoDB menggunakan indeks untuk mencari kunci yang cocok dengan syarat `LastPostDateTime` yang ditentukan.
- Karena atribut `Replies` diproyeksikan ke dalam indeks, DynamoDB dapat mengambil atribut ini tanpa menggunakan throughput tambahan yang disediakan.
- Atribut `Tags` tidak diproyeksikan ke dalam indeks, jadi DynamoDB harus mengakses tabel `Thread` dan mengambil atribut ini.
- Hasilnya ditampilkan, diurutkan berdasarkan `LastPostDateTime`. Entri indeks diurutkan menurut nilai kunci partisi dan kemudian menurut nilai kunci urutan, dan `Query` mengembalikannya sesuai urutan penyimpanannya. (Anda dapat menggunakan parameter `ScanIndexForward` untuk mengembalikan hasil dalam urutan menurun.)

Karena atribut `Tags` tidak diproyeksikan ke indeks sekunder lokal, DynamoDB harus menggunakan unit kapasitas baca tambahan untuk mengambil atribut ini dari tabel dasar. Jika sering menjalankan kueri ini, Anda harus memproyeksikan `Tags` ke `LastPostIndex` untuk menghindari pengambilan dari tabel dasar. Namun, jika Anda hanya perlu mengakses `Tags` sesekali, biaya penyimpanan tambahan untuk memproyeksikan `Tags` ke dalam indeks mungkin tidak bermanfaat.

Memindai Indeks Sekunder Lokal

Anda dapat menggunakan `Scan` untuk mengambil semua data dari indeks sekunder lokal. Anda harus memberikan nama tabel dasar dan nama indeks dalam permintaan. Dengan `Scan`, DynamoDB membaca semua data dalam indeks dan mengembalikannya ke aplikasi. Anda juga dapat meminta agar hanya sebagian data yang dikembalikan, dan data yang tersisa harus dibuang. Untuk melakukannya, gunakan parameter `FilterExpression` API `Scan`. Untuk informasi selengkapnya, lihat Ekspresi filter untuk pemindaian.

Penulisan item dan Indeks Sekunder Lokal

DynamoDB otomatis membuat semua indeks sekunder lokal tersinkronisasi dengan tabel dasar masing-masing. Aplikasi tidak pernah menulis langsung ke indeks. Namun, Anda harus memahami implikasi dari cara DynamoDB mempertahankan indeks ini.

Saat membuat indeks sekunder lokal, Anda menentukan atribut untuk dijadikan sebagai kunci urutan untuk indeks. Anda juga menentukan jenis data untuk atribut tersebut. Artinya, setiap kali Anda menulis item ke tabel dasar, jika item tersebut mendefinisikan atribut kunci indeks, jenisnya harus cocok dengan jenis data skema kunci indeks. Dalam kasus `LastPostIndex`, kunci urutan `LastPostDateTime` dalam indeks didefinisikan sebagai jenis data `String`. Jika Anda mencoba menambahkan item ke tabel `Thread` dan menentukan jenis data yang berbeda untuk `LastPostDateTime` (seperti `Number`), DynamoDB mengembalikan `ValidationException` karena jenis data tidak cocok.

Tidak ada persyaratan untuk one-to-one hubungan antara item dalam tabel dasar dan item dalam indeks sekunder lokal. Bahkan, perilaku ini dapat menguntungkan untuk banyak aplikasi.

Tabel dengan banyak indeks sekunder lokal menimbulkan biaya lebih tinggi untuk aktivitas tulis dibandingkan tabel dengan lebih sedikit indeks. Untuk informasi selengkapnya, lihat Pertimbangan throughput yang disediakan untuk Indeks Sekunder Lokal.

> **Important**
>
> Untuk tabel dengan indeks sekunder lokal, ada batas ukuran 10 GB per nilai kunci partisi. Tabel dengan indeks sekunder lokal dapat menyimpan berapa pun jumlah item, asalkan ukuran total untuk nilai kunci satu partisi tidak melebihi 10 GB. Untuk informasi selengkapnya, lihat Batas ukuran kumpulan item.

Pertimbangan throughput yang disediakan untuk Indeks Sekunder Lokal

Saat membuat tabel di DynamoDB, Anda menyediakan unit kapasitas baca dan tulis untuk beban kerja tabel yang diharapkan. Beban kerja tersebut mencakup aktivitas baca dan tulis pada indeks sekunder lokal tabel.

Untuk melihat tarif saat ini untuk kapasitas throughput yang disediakan, lihat Harga Amazon DynamoDB.

Unit kapasitas baca

Saat Anda mengkueri indeks sekunder lokal, jumlah unit kapasitas baca yang digunakan bergantung pada cara data diakses.

Seperti kueri tabel, kueri indeks dapat menggunakan bacaan akhir konsisten atau bacaan sangat konsisten, tergantung pada nilai `ConsistentRead`. Satu bacaan sangat konsisten menghabiskan satu unit kapasitas baca; bacaan akhir konsisten hanya menghabiskan setengahnya. Jadi, dengan memilih bacaan akhir konsisten, Anda dapat mengurangi biaya unit kapasitas baca Anda.

Untuk kueri indeks yang hanya meminta kunci indeks dan atribut yang diproyeksikan, DynamoDB menghitung aktivitas baca yang disediakan dengan cara yang sama seperti kueri terhadap tabel. Satu-satunya hal yang membedakan adalah penghitungan didasarkan pada ukuran entri indeks, bukan ukuran item dalam tabel dasar. Jumlah unit kapasitas baca adalah jumlah dari semua ukuran atribut yang diproyeksikan di semua item yang dikembalikan; hasilnya kemudian dibulatkan ke batas 4 KB berikutnya. Untuk informasi selengkapnya tentang cara DynamoDB menghitung penggunaan throughput yang disediakan, lihat Mode kapasitas yang disediakan DynamoDB.

Untuk kueri indeks yang membaca atribut yang tidak diproyeksikan ke indeks sekunder lokal, DynamoDB perlu mengambil atribut tersebut dari tabel dasar, selain membaca atribut yang diproyeksikan dari indeks. Pengambilan ini terjadi saat Anda menyertakan atribut yang tidak diproyeksikan dalam parameter `Select` atau `ProjectionExpression` operasi `Query`. Pengambilan menyebabkan latensi tambahan dalam respons kueri, dan juga menimbulkan biaya

throughput yang disediakan lebih tinggi: Selain pembacaan dari indeks sekunder lokal yang dijelaskan sebelumnya, Anda dikenakan biaya untuk unit kapasitas baca untuk setiap item tabel dasar yang diambil. Biaya ini untuk pembacaan setiap item dari tabel, bukan hanya atribut yang diminta.

Ukuran maksimum hasil yang dikembalikan oleh operasi `Query` adalah 1 MB. Ini termasuk ukuran semua nama dan nilai atribut di semua item yang dikembalikan. Namun, jika Kueri terhadap indeks sekunder lokal menyebabkan DynamoDB mengambil atribut item dari tabel dasar, ukuran maksimum data dalam hasil mungkin lebih rendah. Dalam hal ini, ukuran hasil adalah jumlah dari:

- Ukuran item yang cocok dalam indeks, dibulatkan ke atas hingga 4 KB berikutnya.
- Ukuran setiap item yang cocok di tabel dasar, dengan setiap item dibulatkan satu per satu ke 4 KB berikutnya.

Menggunakan rumus ini, ukuran maksimum hasil yang dikembalikan oleh operasi Kueri masih 1 MB.

Misalnya, pertimbangkan tabel tempat ukuran setiap item adalah 300 byte. Ada indeks sekunder lokal di tabel itu, tetapi hanya 200 byte dari setiap item yang diproyeksikan ke dalam indeks. Sekarang anggaplah Anda `Query` indeks ini, bahwa kueri memerlukan pengambilan tabel untuk setiap item, dan kueri mengembalikan 4 item. DynamoDB merangkum berikut ini:

- Ukuran item yang cocok dalam indeks: 200 byte × 4 item = 800 byte; ini kemudian dibulatkan menjadi 4 KB.
- Ukuran setiap item yang cocok di tabel dasar: (300 byte, dibulatkan hingga 4 KB) × 4 item = 16 KB.

Oleh karena itu, ukuran total data dalam hasil adalah 20 KB.

Unit kapasitas tulis

Saat item dalam tabel ditambahkan, diperbarui, atau dihapus, memperbarui indeks sekunder lokal akan menggunakan unit kapasitas tulis yang disediakan untuk tabel tersebut. Total biaya throughput yang disediakan untuk penulisan adalah jumlah unit kapasitas tulis yang digunakan dengan menulis ke tabel dan yang digunakan dengan memperbarui indeks sekunder lokal.

Biaya tulis item ke indeks sekunder lokal tergantung pada beberapa faktor:

- Jika Anda menulis item baru ke tabel yang menentukan atribut yang diindeks, atau Anda memperbarui item yang ada untuk menentukan atribut terindeks yang sebelumnya tidak ditentukan, satu operasi tulis diperlukan untuk memasukkan item ke dalam indeks.

- Jika pembaruan pada tabel mengubah nilai atribut kunci yang diindeks (dari A menjadi B), diperlukan dua penulisan: satu untuk menghapus item sebelumnya dari indeks dan satu lagi untuk memasukkan item baru ke dalam indeks.
- Jika suatu item ada dalam indeks, tetapi penulisan ke tabel menyebabkan atribut yang diindeks terhapus, satu penulisan diperlukan untuk menghapus proyeksi item lama dari indeks.
- Jika item tidak ada dalam indeks sebelum atau setelah item diperbarui, tidak ada biaya penulisan tambahan untuk indeks tersebut.

Semua faktor ini mengasumsikan bahwa ukuran setiap item dalam indeks kurang dari atau sama dengan ukuran item 1 KB untuk menghitung unit kapasitas tulis. Entri indeks yang lebih besar memerlukan unit kapasitas tulis tambahan. Anda dapat meminimalkan biaya penulisan dengan mempertimbangkan atribut yang perlu dikembalikan oleh kueri dan hanya memproyeksikan atribut tersebut ke dalam indeks.

Pertimbangan penyimpanan untuk Indeks Sekunder Lokal

Saat aplikasi menulis item ke tabel, DynamoDB otomatis menyalin subset atribut yang benar ke indeks sekunder lokal tempat atribut tersebut akan muncul. AWS Akun Anda dikenakan biaya untuk penyimpanan item di tabel dasar dan juga untuk penyimpanan atribut dalam indeks sekunder lokal apa pun di tabel itu.

Jumlah ruang yang digunakan oleh item indeks adalah jumlah dari berikut ini:

- Ukuran kunci primer tabel dasar dalam byte (kunci partisi dan kunci urutan)
- Ukuran atribut kunci indeks dalam byte
- Ukuran atribut yang diproyeksikan dalam byte (jika ada)
- 100 byte dari overhead per item indeks

Untuk memperkirakan kebutuhan penyimpanan indeks sekunder lokal, Anda dapat memperkirakan ukuran rata-rata item dalam indeks lalu mengalikannya dengan jumlah item di index.

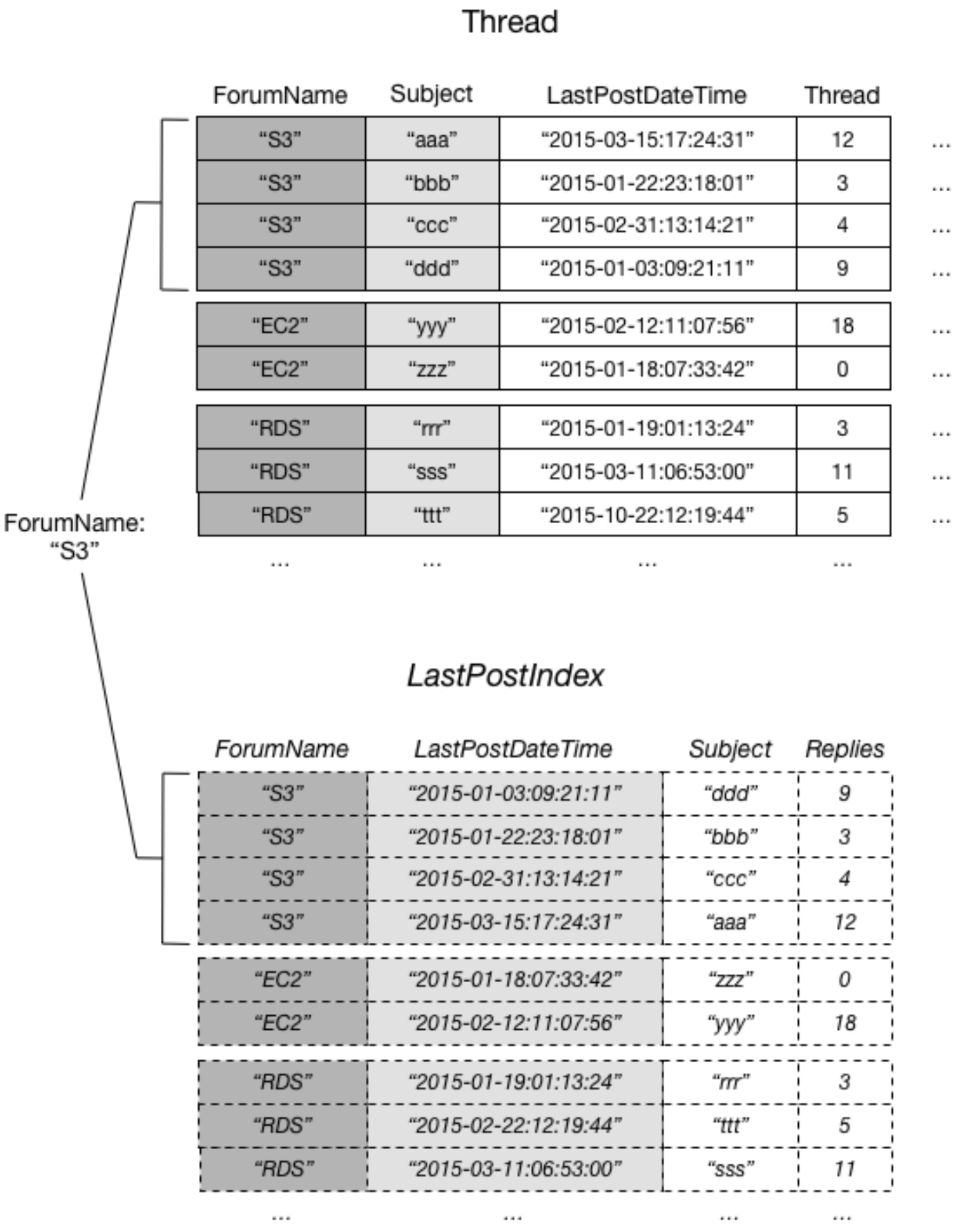
Jika tabel berisi item dengan atribut tertentu tidak ditentukan, tetapi atribut tersebut didefinisikan sebagai kunci urutan indeks, maka DynamoDB tidak menulis data apa pun untuk item tersebut ke indeks.

Kumpulan item dalam Indeks Sekunder Lokal

> **Note**
>
> Bagian ini hanya berkaitan dengan tabel yang memiliki indeks sekunder lokal.

Di DynamoDB, kumpulan item adalah grup item yang memiliki nilai kunci partisi yang sama dalam tabel dan semua indeks sekunder lokalnya. Dalam contoh yang digunakan di seluruh bagian ini, kunci partisi untuk tabel `Thread` adalah `ForumName`, dan kunci partisi untuk `LastPostIndex` juga `ForumName`. Semua item tabel dan indeks dengan `ForumName` yang sama adalah bagian dari kumpulan item yang sama. Misalnya, di tabel `Thread` dan indeks sekunder lokal `LastPostIndex`, terdapat kumpulan item untuk forum `EC2` dan kumpulan item yang berbeda untuk forum `RDS`.

Diagram berikut menunjukkan kumpulan item untuk forum `S3`.


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

Dalam diagram ini, kumpulan item terdiri dari semua item di `Thread` dan `LastPostIndex` dengan nilai kunci partisi `ForumName`-nya adalah "S3". Jika terdapat indeks sekunder lokal lainnya di tabel, item apa pun dalam indeks tersebut dengan `ForumName` yang sama dengan "S3" juga akan menjadi bagian dari kumpulan item.

Anda dapat menggunakan salah satu operasi berikut di DynamoDB untuk menampilkan informasi tentang kumpulan item:

- `BatchWriteItem`
- `DeleteItem`
- `PutItem`
- `UpdateItem`
- `TransactWriteItems`

Masing-masing operasi ini mendukung parameter `ReturnItemCollectionMetrics`. Jika mengatur parameter ini ke `SIZE`, Anda dapat melihat informasi tentang ukuran setiap kumpulan item dalam indeks.

Example

Berikut ini adalah contoh dari output operasi `UpdateItem` pada tabel `Thread`, dengan `ReturnItemCollectionMetrics` diatur ke `SIZE`. Item yang diperbarui memiliki `ForumName` nilai "EC2“, jadi outputnya mencakup informasi tentang koleksi item tersebut.

```
{
    ItemCollectionMetrics: {
        ItemCollectionKey: {
            ForumName: "EC2"
        },
        SizeEstimateRangeGB: [0.0, 1.0]
    }
}
```

Objek `SizeEstimateRangeGB` menunjukkan bahwa ukuran kumpulan item ini antara 0 dan 1 GB. DynamoDB memperbarui perkiraan ukuran ini secara berkala, sehingga jumlahnya mungkin berbeda saat item diubah lagi.

Batas ukuran kumpulan item

Ukuran maksimum kumpulan item apa pun untuk tabel yang memiliki satu atau beberapa indeks sekunder lokal adalah 10 GB. Ini tidak berlaku untuk kumpulan item dalam tabel tanpa indeks sekunder lokal, dan juga tidak berlaku untuk kumpulan item dalam indeks sekunder global. Hanya tabel yang memiliki satu atau beberapa indeks sekunder lokal yang terpengaruh.

Jika kumpulan item melebihi batas 10 GB, DynamoDB mengembalikan `ItemCollectionSizeLimitExceededException`, dan Anda tidak akan bisa menambahkan lebih banyak item ke kumpulan item atau menambah ukuran item yang ada di kumpulan item. (Operasi baca dan tulis yang mengecilkan ukuran kumpulan item masih diperbolehkan.) Anda masih dapat menambahkan item ke kumpulan item lain.

Untuk mengurangi ukuran kumpulan item, Anda dapat melakukan salah satu berikut ini:

- Hapus item yang tidak diperlukan dengan nilai kunci partisi terkait. Jika Anda menghapus item ini dari tabel dasar, DynamoDB juga menghapus entri indeks yang memiliki nilai kunci partisi yang sama.
- Perbarui item dengan menghapus atribut atau dengan mengurangi ukuran atribut. Jika atribut ini diproyeksikan ke indeks sekunder lokal, DynamoDB juga mengurangi ukuran entri indeks terkait.
- Buat tabel baru dengan kunci partisi dan kunci urutan yang sama, lalu pindahkan item dari tabel lama ke tabel baru. Ini mungkin merupakan pendekatan yang tepat jika tabel memiliki data historis yang jarang diakses. Anda juga dapat mempertimbangkan untuk mengarsipkan data historis ini ke Amazon Simple Storage Service (Amazon S3).

Ketika ukuran total kumpulan item turun di bawah 10 GB, Anda dapat menambahkan item dengan nilai kunci partisi yang sama sekali lagi.

Sebagai praktik terbaik, sebaiknya Anda melengkapi aplikasi Anda untuk memantau ukuran kumpulan item Anda. Salah satu cara untuk melakukannya adalah dengan mengatur parameter `ReturnItemCollectionMetrics` ke `SIZE` setiap kali Anda menggunakan `BatchWriteItem`, `DeleteItem`, `PutItem`, atau `UpdateItem`. Aplikasi Anda akan memeriksa objek `ReturnItemCollectionMetrics` di output dan mencatat pesan kesalahan setiap kali kumpulan item melebihi batas yang ditentukan pengguna (8 GB, misalnya). Menetapkan batas kurang dari 10 GB akan memberi sistem peringatan dini sehingga Anda mengetahui bahwa kumpulan item mendekati batas, dan Anda dapat melakukan tindakan terhadap hal tersebut.

Kumpulan dan partisi item

Di tabel dengan satu atau beberapa indeks sekunder lokal, masing-masing kumpulan item disimpan dalam satu partisi. Ukuran total kumpulan item tersebut terbatas pada kemampuan partisi itu: 10 GB. Untuk aplikasi yang model datanya menyertakan kumpulan item yang ukurannya tidak dibatasi, atau jika Anda memperkirakan beberapa kumpulan item akan bertambah melebihi 10 GB di masa mendatang, Anda sebaiknya mempertimbangkan untuk menggunakan indeks sekunder global.

Anda harus merancang aplikasi Anda sehingga data tabel didistribusikan secara merata ke seluruh nilai kunci partisi yang berbeda. Untuk tabel dengan indeks sekunder lokal, aplikasi Anda tidak boleh membuat "hot spot" aktivitas baca dan tulis dalam satu kumpulan item pada satu partisi.

Menggunakan Indeks Sekunder Lokal: Java

Anda dapat menggunakan AWS SDK untuk Java Document API untuk membuat tabel Amazon DynamoDB dengan satu atau beberapa indeks sekunder lokal, menjelaskan indeks pada tabel, dan melakukan kueri menggunakan indeks.

Berikut ini adalah langkah-langkah umum untuk operasi tabel menggunakan AWS SDK untuk Java Document API.

1. Buat instans dari kelas `DynamoDB`.
2. Berikan parameter wajib dan opsional untuk operasi dengan membuat objek permintaan yang sesuai.
3. Panggil metode sesuai yang ditentukan oleh klien yang Anda buat pada langkah sebelumnya.

Topik

- Membuat tabel dengan Indeks Sekunder Lokal
- Mendeskripsikan tabel dengan Indeks Sekunder Lokal
- Mengkueri Indeks Sekunder Lokal
- Contoh: Indeks Sekunder Lokal menggunakan API dokumen Java

Membuat tabel dengan Indeks Sekunder Lokal

Indeks sekunder lokal harus dibuat pada saat Anda membuat tabel. Untuk melakukannya, gunakan metode `createTable` dan berikan spesifikasi Anda untuk satu atau beberapa indeks sekunder lokal. Contoh kode Java berikut membuat tabel untuk menyimpan informasi tentang lagu dalam koleksi

musik. Kunci partisinya adalah `Artist` dan kunci urutannya adalah `SongTitle`. Indeks sekunder, `AlbumTitleIndex`, memfasilitasi kueri berdasarkan judul album.

Berikut adalah langkah-langkah untuk membuat tabel dengan indeks sekunder lokal, menggunakan API dokumen DynamoDB.

1. Buat instans dari kelas `DynamoDB`.
2. Buat instans dari kelas `CreateTableRequest` untuk memberikan informasi permintaan.

   Anda harus memberikan nama tabel, kunci primernya, dan nilai throughput yang ditentukan. Untuk indeks sekunder lokal, Anda harus memberikan nama indeks, nama dan jenis data untuk kunci urutan indeks, skema kunci untuk indeks, dan proyeksi atribut.
3. Panggil metode `createTable` dengan menentukan objek permintaan sebagai parameter.

Contoh kode Java berikut mendemonstrasikan langkah sebelumnya. Kode ini membuat tabel (`Music`) dengan indeks sekunder pada atribut `AlbumTitle`. Kunci urutan dan kunci partisi tabel, ditambah kunci urutan indeks, adalah satu-satunya atribut yang diproyeksikan ke dalam indeks.

```
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());
```

Anda harus menunggu hingga DynamoDB membuat tabel dan menetapkan status tabel menjadi `ACTIVE`. Setelah itu, Anda bisa mulai memasukkan item data ke dalam tabel.

Mendeskripsikan tabel dengan Indeks Sekunder Lokal

Untuk mendapatkan informasi tentang indeks sekunder lokal pada tabel, gunakan metode `describeTable`. Untuk setiap indeks, Anda dapat mengakses namanya, skema kunci, dan atribut yang diproyeksikan.

Berikut ini adalah langkah-langkah untuk mengakses informasi indeks sekunder lokal suatu tabel menggunakan API Dokumen AWS SDK untuk Java .

1. Buat instans dari kelas `DynamoDB`.
2. Buat instans dari kelas `Table`. Anda harus memberikan nama tabel.
3. Panggil metode `describeTable` pada objek `Table`.

Contoh kode Java berikut mendemonstrasikan langkah sebelumnya.

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());
    }
}
```

Mengkueri Indeks Sekunder Lokal

Anda dapat menggunakan operasi `Query` pada indeks sekunder lokal dengan cara yang hampir sama seperti Anda `Query` tabel. Anda harus menentukan nama indeks, kriteria kueri untuk kunci urutan indeks, dan atribut yang ingin Anda kembalikan. Dalam contoh ini, indeks adalah `AlbumTitleIndex` dan kunci urutan indeks adalah `AlbumTitle`.

Satu-satunya atribut yang dikembalikan adalah atribut yang telah diproyeksikan ke dalam indeks. Anda dapat memodifikasi kueri ini untuk memilih atribut non-kunci juga, tetapi ini akan memerlukan aktivitas pengambilan tabel yang relatif mahal. Untuk informasi selengkapnya tentang pengambilan tabel, lihat Proyeksi atribut.

Berikut ini adalah langkah-langkah untuk query indeks sekunder lokal menggunakan AWS SDK untuk Java Document API.

1. Buat instans dari kelas `DynamoDB`.
2. Buat instans dari kelas `Table`. Anda harus memberikan nama tabel.
3. Buat instans dari kelas `Index`. Anda harus memberikan nama indeks.
4. Panggil metode `query` dari kelas `Index`.

Contoh kode Java berikut mendemonstrasikan langkah sebelumnya.

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());
}
```

Contoh: Indeks Sekunder Lokal menggunakan API dokumen Java

Contoh kode Java berikut menunjukkan cara menggunakan indeks sekunder lokal di Amazon DynamoDB. Contoh membuat tabel bernama `CustomerOrders` dengan kunci partisi `CustomerId` dan kunci urutan `OrderId`. Ada dua indeks sekunder lokal di tabel ini:

- `OrderCreationDateIndex` — Kunci urutannya adalah `OrderCreationDate`, dan atribut berikut diproyeksikan ke dalam indeks:
  - `ProductCategory`
  - `ProductName`
  - `OrderStatus`
  - `ShipmentTrackingId`
- `IsOpenIndex` — Kunci urutannya adalah `IsOpen`, dan semua atribut tabel diproyeksikan ke dalam indeks.

Setelah tabel `CustomerOrders` dibuat, program memuat tabel dengan data yang mewakili pesanan pelanggan. Kemudian mengkueri data menggunakan indeks sekunder lokal. Terakhir, program menghapus tabel `CustomerOrders`.

Untuk step-by-step petunjuk pengujian sampel berikut, lihat[Contoh kode Java](#).

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.PutItemOutcome;
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.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;
import com.amazonaws.services.dynamodbv2.model.ReturnConsumedCapacity;
import com.amazonaws.services.dynamodbv2.model.Select;

public class DocumentAPILocalSecondaryIndexExample {

        static AmazonDynamoDB client = AmazonDynamoDBClientBuilder.standard().build();
        static DynamoDB dynamoDB = new DynamoDB(client);

        public static String tableName = "CustomerOrders";

        public static void main(String[] args) throws Exception {

                createTable();
                loadData();

                query(null);
                query("IsOpenIndex");
                query("OrderCreationDateIndex");

                deleteTable(tableName);
```

```
        }

        public static void createTable() {

                CreateTableRequest createTableRequest = new
CreateTableRequest().withTableName(tableName)
                                .withProvisionedThroughput(
                                                new
ProvisionedThroughput().withReadCapacityUnits((long) 1)

.withWriteCapacityUnits((long) 1));

                // Attribute definitions for table partition and sort keys
                ArrayList<AttributeDefinition> attributeDefinitions = new
ArrayList<AttributeDefinition>();
                attributeDefinitions
                                .add(new
AttributeDefinition().withAttributeName("CustomerId").withAttributeType("S"));
                attributeDefinitions.add(new
AttributeDefinition().withAttributeName("OrderId").withAttributeType("N"));

                // Attribute definition for index primary key attributes
                attributeDefinitions
                                .add(new
AttributeDefinition().withAttributeName("OrderCreationDate")
                                                .withAttributeType("N"));
                attributeDefinitions.add(new
AttributeDefinition().withAttributeName("IsOpen").withAttributeType("N"));

                createTableRequest.setAttributeDefinitions(attributeDefinitions);

                // Key schema for table
                ArrayList<KeySchemaElement> tableKeySchema = new
ArrayList<KeySchemaElement>();
                tableKeySchema.add(new
KeySchemaElement().withAttributeName("CustomerId").withKeyType(KeyType.HASH)); //
Partition

                                // key
                tableKeySchema.add(new
KeySchemaElement().withAttributeName("OrderId").withKeyType(KeyType.RANGE)); // Sort

                               // key
```

```
                createTableRequest.setKeySchema(tableKeySchema);

                ArrayList<LocalSecondaryIndex> localSecondaryIndexes = new
ArrayList<LocalSecondaryIndex>();

                // OrderCreationDateIndex
                LocalSecondaryIndex orderCreationDateIndex = new LocalSecondaryIndex()
                                .withIndexName("OrderCreationDateIndex");

                // Key schema for OrderCreationDateIndex
                ArrayList<KeySchemaElement> indexKeySchema = new
ArrayList<KeySchemaElement>();
                indexKeySchema.add(new
KeySchemaElement().withAttributeName("CustomerId").withKeyType(KeyType.HASH)); //
Partition

                                // key
                indexKeySchema.add(new
KeySchemaElement().withAttributeName("OrderCreationDate")
                                .withKeyType(KeyType.RANGE)); // Sort
                                                              // key

                orderCreationDateIndex.setKeySchema(indexKeySchema);

                // Projection (with list of projected attributes) for
                // OrderCreationDateIndex
                Projection projection = new
Projection().withProjectionType(ProjectionType.INCLUDE);
                ArrayList<String> nonKeyAttributes = new ArrayList<String>();
                nonKeyAttributes.add("ProductCategory");
                nonKeyAttributes.add("ProductName");
                projection.setNonKeyAttributes(nonKeyAttributes);

                orderCreationDateIndex.setProjection(projection);

                localSecondaryIndexes.add(orderCreationDateIndex);

                // IsOpenIndex
                LocalSecondaryIndex isOpenIndex = new
LocalSecondaryIndex().withIndexName("IsOpenIndex");

                // Key schema for IsOpenIndex
                indexKeySchema = new ArrayList<KeySchemaElement>();
```

```
                indexKeySchema.add(new
KeySchemaElement().withAttributeName("CustomerId").withKeyType(KeyType.HASH)); //
Partition

                                // key
                indexKeySchema.add(new
KeySchemaElement().withAttributeName("IsOpen").withKeyType(KeyType.RANGE)); // Sort

                            // key

                // Projection (all attributes) for IsOpenIndex
                projection = new Projection().withProjectionType(ProjectionType.ALL);

                isOpenIndex.setKeySchema(indexKeySchema);
                isOpenIndex.setProjection(projection);

                localSecondaryIndexes.add(isOpenIndex);

                // Add index definitions to CreateTable request
                createTableRequest.setLocalSecondaryIndexes(localSecondaryIndexes);

                System.out.println("Creating table " + tableName + "...");
                System.out.println(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 query(String indexName) {

                Table table = dynamoDB.getTable(tableName);


System.out.println("\n***********************************************************\n");
                System.out.println("Querying table " + tableName + "...");
```

```
                QuerySpec querySpec = new
 QuerySpec().withConsistentRead(true).withScanIndexForward(true)

 .withReturnConsumedCapacity(ReturnConsumedCapacity.TOTAL);

                if (indexName == "IsOpenIndex") {

                        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");
                        Index index = table.getIndex(indexName);

                        querySpec.withKeyConditionExpression("CustomerId = :v_custid
 and IsOpen = :v_isopen")
                                        .withValueMap(new
 ValueMap().withString(":v_custid", "bob@example.com")
                                                        .withNumber(":v_isopen", 1));

                        querySpec.withProjectionExpression(
                                        "OrderCreationDate, ProductCategory,
 ProductName, OrderStatus");

                        ItemCollection<QueryOutcome> items = index.query(querySpec);
                        Iterator<Item> iterator = items.iterator();

                        System.out.println("Query: printing results...");

                        while (iterator.hasNext()) {
                                System.out.println(iterator.next().toJSONPretty());
                        }

                } else if (indexName == "OrderCreationDateIndex") {
                        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");
                        Index index = table.getIndex(indexName);

                        querySpec.withKeyConditionExpression(
                                        "CustomerId = :v_custid and OrderCreationDate
 >= :v_orddate")
                                        .withValueMap(
```

```
                                                        new
ValueMap().withString(":v_custid", "bob@example.com")

.withNumber(":v_orddate",

20150131));

                        querySpec.withSelect(Select.ALL_PROJECTED_ATTRIBUTES);

                        ItemCollection<QueryOutcome> items = index.query(querySpec);
                        Iterator<Item> iterator = items.iterator();

                        System.out.println("Query: printing results...");

                        while (iterator.hasNext()) {
                                System.out.println(iterator.next().toJSONPretty());
                        }

                } else {
                        System.out.println("\nNo index: All of Bob's orders, by
OrderId:\n");

                        querySpec.withKeyConditionExpression("CustomerId = :v_custid")
                                        .withValueMap(new
ValueMap().withString(":v_custid", "bob@example.com"));

                        ItemCollection<QueryOutcome> items = table.query(querySpec);
                        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) {

                Table table = dynamoDB.getTable(tableName);
                System.out.println("Deleting table " + 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() {

                Table table = dynamoDB.getTable(tableName);

                System.out.println("Loading data into table " + tableName + "...");

                Item item = new Item().withPrimaryKey("CustomerId",
"alice@example.com").withNumber("OrderId", 1)
                                .withNumber("IsOpen",
1).withNumber("OrderCreationDate", 20150101)
                                .withString("ProductCategory", "Book")
                                .withString("ProductName", "The Great Outdoors")
                                .withString("OrderStatus", "PACKING ITEMS");
                // no ShipmentTrackingId attribute

                PutItemOutcome putItemOutcome = table.putItem(item);

                item = new Item().withPrimaryKey("CustomerId",
"alice@example.com").withNumber("OrderId", 2)
                                .withNumber("IsOpen",
1).withNumber("OrderCreationDate", 20150221)
                                .withString("ProductCategory", "Bike")
                                .withString("ProductName", "Super Mountain")
                                .withString("OrderStatus", "ORDER RECEIVED");
                // no ShipmentTrackingId attribute

                putItemOutcome = table.putItem(item);

                item = new Item().withPrimaryKey("CustomerId",
"alice@example.com").withNumber("OrderId", 3)
                                // no IsOpen attribute
                                .withNumber("OrderCreationDate",
20150304).withString("ProductCategory", "Music")
```

```
                                .withString("ProductName", "A Quiet
Interlude").withString("OrderStatus", "IN TRANSIT")
                                .withString("ShipmentTrackingId", "176493");

                putItemOutcome = table.putItem(item);

                item = new Item().withPrimaryKey("CustomerId",
"bob@example.com").withNumber("OrderId", 1)
                                // no IsOpen attribute
                                .withNumber("OrderCreationDate",
20150111).withString("ProductCategory", "Movie")
                                .withString("ProductName", "Calm Before The Storm")
                                .withString("OrderStatus", "SHIPPING DELAY")
                                .withString("ShipmentTrackingId", "859323");

                putItemOutcome = table.putItem(item);

                item = new Item().withPrimaryKey("CustomerId",
"bob@example.com").withNumber("OrderId", 2)
                                // no IsOpen attribute
                                .withNumber("OrderCreationDate",
20150124).withString("ProductCategory", "Music")
                                .withString("ProductName", "E-Z
Listening").withString("OrderStatus", "DELIVERED")
                                .withString("ShipmentTrackingId", "756943");

                putItemOutcome = table.putItem(item);

                item = new Item().withPrimaryKey("CustomerId",
"bob@example.com").withNumber("OrderId", 3)
                                // no IsOpen attribute
                                .withNumber("OrderCreationDate",
20150221).withString("ProductCategory", "Music")
                                .withString("ProductName", "Symphony
9").withString("OrderStatus", "DELIVERED")
                                .withString("ShipmentTrackingId", "645193");

                putItemOutcome = table.putItem(item);

                item = new Item().withPrimaryKey("CustomerId",
"bob@example.com").withNumber("OrderId", 4)
                                .withNumber("IsOpen",
1).withNumber("OrderCreationDate", 20150222)
                                .withString("ProductCategory", "Hardware")
```

```
                                .withString("ProductName", "Extra Heavy Hammer")
                                .withString("OrderStatus", "PACKING ITEMS");
                // no ShipmentTrackingId attribute

                putItemOutcome = table.putItem(item);

                item = new Item().withPrimaryKey("CustomerId",
 "bob@example.com").withNumber("OrderId", 5)
                                /* no IsOpen attribute */
                                .withNumber("OrderCreationDate",
 20150309).withString("ProductCategory", "Book")
                                .withString("ProductName", "How To
 Cook").withString("OrderStatus", "IN TRANSIT")
                                .withString("ShipmentTrackingId", "440185");

                putItemOutcome = table.putItem(item);

                item = new Item().withPrimaryKey("CustomerId",
 "bob@example.com").withNumber("OrderId", 6)
                                // no IsOpen attribute
                                .withNumber("OrderCreationDate",
 20150318).withString("ProductCategory", "Luggage")
                                .withString("ProductName", "Really Big
 Suitcase").withString("OrderStatus", "DELIVERED")
                                .withString("ShipmentTrackingId", "893927");

                putItemOutcome = table.putItem(item);

                item = new Item().withPrimaryKey("CustomerId",
 "bob@example.com").withNumber("OrderId", 7)
                                /* no IsOpen attribute */
                                .withNumber("OrderCreationDate",
 20150324).withString("ProductCategory", "Golf")
                                .withString("ProductName", "PGA Pro
 II").withString("OrderStatus", "OUT FOR DELIVERY")
                                .withString("ShipmentTrackingId", "383283");

                putItemOutcome = table.putItem(item);
                assert putItemOutcome != null;
        }

}
```

Menggunakan Indeks Sekunder Lokal: .NET

Topik

- Membuat tabel dengan Indeks Sekunder Lokal
- Mendeskripsikan tabel dengan Indeks Sekunder Lokal
- Mengkueri Indeks Sekunder Lokal
- Contoh: Indeks Sekunder Lokal menggunakan API tingkat rendah AWS SDK untuk .NET

Anda dapat menggunakan API AWS SDK untuk .NET tingkat rendah untuk membuat tabel Amazon DynamoDB dengan satu atau beberapa indeks sekunder lokal, menjelaskan indeks pada tabel, dan melakukan kueri menggunakan indeks. Operasi ini dipetakan ke tindakan DynamoDB API tingkat rendah yang sesuai. Untuk informasi selengkapnya, lihat Contoh kode .NET.

Berikut ini adalah langkah-langkah umum untuk operasi tabel menggunakan API tingkat rendah .NET.

1. Buat instans dari kelas `AmazonDynamoDBClient`.
2. Berikan parameter wajib dan opsional untuk operasi dengan membuat objek permintaan yang sesuai.

   Misalnya, buat objek `CreateTableRequest` untuk membuat tabel dan objek `QueryRequest` untuk mengkueri tabel atau indeks.
3. Jalankan metode sesuai yang ditentukan oleh klien yang Anda buat pada langkah sebelumnya.

Membuat tabel dengan Indeks Sekunder Lokal

Indeks sekunder lokal harus dibuat pada saat Anda membuat tabel. Untuk melakukannya, gunakan `CreateTable` dan berikan spesifikasi Anda untuk satu atau beberapa indeks sekunder lokal. Contoh kode C# berikut membuat tabel untuk menyimpan informasi tentang lagu dalam koleksi musik. Kunci partisinya adalah `Artist` dan kunci urutannya adalah `SongTitle`. Indeks sekunder, `AlbumTitleIndex`, memfasilitasi kueri berdasarkan judul album.

Berikut ini adalah langkah-langkah untuk membuat tabel dengan indeks sekunder lokal, menggunakan API tingkat rendah .NET.

1. Buat instans dari kelas `AmazonDynamoDBClient`.
2. Buat instans dari kelas `CreateTableRequest` untuk memberikan informasi permintaan.

Anda harus memberikan nama tabel, kunci primernya, dan nilai throughput yang ditentukan. Untuk indeks sekunder lokal, Anda harus memberikan nama indeks, nama dan jenis data kunci urutan indeks, skema kunci untuk indeks, dan proyeksi atribut.

3. Jalankan metode `CreateTable` dengan menentukan objek permintaan sebagai parameter.

Contoh kode #C berikut mendemonstrasikan langkah sebelumnya. Kode ini membuat tabel (`Music`) dengan indeks sekunder pada atribut `AlbumTitle`. Kunci urutan dan kunci partisi tabel, ditambah kunci urutan indeks, adalah satu-satunya atribut yang diproyeksikan ke dalam indeks.

```
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);
```

Anda harus menunggu hingga DynamoDB membuat tabel dan menetapkan status tabel menjadi `ACTIVE`. Setelah itu, Anda bisa mulai memasukkan item data ke dalam tabel.

Mendeskripsikan tabel dengan Indeks Sekunder Lokal

Untuk mendapatkan informasi tentang indeks sekunder lokal pada tabel, gunakan API `DescribeTable`. Untuk setiap indeks, Anda dapat mengakses namanya, skema kunci, dan atribut yang diproyeksikan.

Berikut ini adalah langkah-langkah untuk mengakses informasi indeks sekunder lokal suatu tabel menggunakan API tingkat rendah .NET.

1. Buat instans dari kelas `AmazonDynamoDBClient`.
2. Buat instans dari kelas `DescribeTableRequest` untuk memberikan informasi permintaan. Anda harus memberikan nama tabel.
3. Jalankan metode `describeTable` dengan menentukan objek permintaan sebagai parameter.
4.

Contoh kode #C berikut mendemonstrasikan langkah sebelumnya.

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);
        }

    }
}
```

Mengkueri Indeks Sekunder Lokal

Anda dapat menggunakan `Query` pada indeks sekunder lokal, sama seperti Anda `Query` tabel. Anda harus menentukan nama indeks, kriteria kueri untuk kunci urutan indeks, dan atribut yang ingin Anda kembalikan. Dalam contoh ini, indeks adalah `AlbumTitleIndex`, dan kunci urutan indeks adalah `AlbumTitle`.

Satu-satunya atribut yang dikembalikan adalah atribut yang telah diproyeksikan ke dalam indeks. Anda dapat memodifikasi kueri ini untuk memilih atribut non-kunci juga, tetapi ini akan memerlukan aktivitas pengambilan tabel yang relatif mahal. Untuk informasi selengkapnya tentang pengambilan tabel, lihat Proyeksi atribut

Berikut ini adalah langkah-langkah untuk mengkueri indeks sekunder lokal menggunakan API tingkat rendah .NET.

1. Buat instans dari kelas `AmazonDynamoDBClient`.
2. Buat instans dari kelas `QueryRequest` untuk memberikan informasi permintaan.
3. Jalankan metode `query` dengan menentukan objek permintaan sebagai parameter.

Contoh kode #C berikut mendemonstrasikan langkah sebelumnya.

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();
}
```

Contoh: Indeks Sekunder Lokal menggunakan API tingkat rendah AWS SDK untuk .NET

Contoh kode C# berikut menunjukkan cara menggunakan indeks sekunder lokal di Amazon DynamoDB. Contoh membuat tabel bernama `CustomerOrders` dengan kunci partisi `CustomerId` dan kunci urutan `OrderId`. Ada dua indeks sekunder lokal di tabel ini:

- `OrderCreationDateIndex` — Kunci urutannya adalah `OrderCreationDate`, dan atribut berikut diproyeksikan ke dalam indeks:
  - `ProductCategory`
  - `ProductName`
  - `OrderStatus`
  - `ShipmentTrackingId`
- `IsOpenIndex` — Kunci urutannya adalah `IsOpen`, dan semua atribut tabel diproyeksikan ke dalam indeks.

Setelah tabel `CustomerOrders` dibuat, program memuat tabel dengan data yang mewakili pesanan pelanggan. Kemudian mengkueri data menggunakan indeks sekunder lokal. Terakhir, program menghapus tabel `CustomerOrders`.

Untuk step-by-step instruksi untuk menguji contoh berikut, lihatContoh kode .NET.

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;
            }
        }
```

```
        }
    }
}
```

Bekerja dengan Indeks Sekunder Lokal di DynamoDB AWS CLI

Anda dapat menggunakan AWS CLI untuk membuat tabel Amazon DynamoDB dengan satu atau beberapa Indeks Sekunder Lokal, menjelaskan indeks pada tabel, dan melakukan kueri menggunakan indeks.

Topik

- Membuat tabel dengan Indeks Sekunder Lokal
- Mendeskripsikan tabel dengan Indeks Sekunder Lokal
- Mengkueri Indeks Sekunder Lokal

Membuat tabel dengan Indeks Sekunder Lokal

Indeks Sekunder Lokal harus dibuat pada saat Anda membuat tabel. Untuk melakukannya, gunakan parameter `create-table` dan berikan spesifikasi Anda untuk satu atau beberapa Indeks Sekunder Lokal. Contoh berikut membuat tabel (`Music`) untuk menyimpan informasi tentang lagu dalam koleksi musik. Kunci partisinya adalah `Artist` dan kunci urutannya adalah `SongTitle`. Indeks sekunder, `AlbumTitleIndex` pada atribut `AlbumTitle` memfasilitasi kueri berdasarkan judul album.

```
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\"]}}]"
```

Anda harus menunggu hingga DynamoDB membuat tabel dan menetapkan status tabel menjadi `ACTIVE`. Setelah itu, Anda bisa mulai memasukkan item data ke dalam tabel. Anda dapat menggunakan describe-table untuk menentukan status pembuatan tabel.

Mendeskripsikan tabel dengan Indeks Sekunder Lokal

Untuk mendapatkan informasi tentang indeks sekunder lokal pada tabel, gunakan parameter `describe-table`. Untuk setiap indeks, Anda dapat mengakses namanya, skema kunci, dan atribut yang diproyeksikan.

```
aws dynamodb describe-table --table-name Music
```

Mengkueri Indeks Sekunder Lokal

Anda dapat menggunakan operasi `query` pada Indeks Sekunder Lokal dengan cara yang hampir sama seperti Anda `query` tabel. Anda harus menentukan nama indeks, kriteria kueri untuk kunci urutan indeks, dan atribut yang ingin Anda kembalikan. Dalam contoh ini, indeks adalah `AlbumTitleIndex` dan kunci urutan indeks adalah `AlbumTitle`.

Satu-satunya atribut yang dikembalikan adalah atribut yang telah diproyeksikan ke dalam indeks. Anda dapat memodifikasi kueri ini untuk memilih atribut non-kunci juga, tetapi ini akan memerlukan aktivitas pengambilan tabel yang relatif mahal. Untuk informasi selengkapnya tentang pengambilan tabel, lihat Proyeksi atribut.

```
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"} }'
```

Mengelola alur kerja kompleks dengan DynamoDB Transactions

Transaksi Amazon DynamoDB menyederhanakan pengalaman pengembang dalam membuat terkoordinasi all-or-nothing, perubahan ke beberapa item baik di dalam maupun di seluruh tabel. Transaksi memberikan atomisitas, konsistensi, isolasi, dan daya tahan (ACID) di DynamoDB, yang memungkinkan Anda menjaga kebenaran data dalam aplikasi Anda dengan lebih mudah.

Anda dapat menggunakan DynamoDB transaksional read and APIs write untuk mengelola alur kerja bisnis kompleks yang memerlukan penambahan, pembaruan, atau penghapusan beberapa item

sebagai operasi tunggal. all-or-nothing Misalnya, developer video game dapat memastikan bahwa profil pemain diperbarui dengan benar saat mereka bertukar item dalam game atau melakukan pembelian dalam game.

Dengan transaksi API tulis, Anda dapat mengelompokkan beberapa tindakan `Put`, `Update`, `Delete`, dan `ConditionCheck`. Anda kemudian dapat mengirimkan tindakan sebagai operasi `TransactWriteItems` tunggal yang baik berhasil atau gagal sebagai unit. Hal yang sama berlaku untuk beberapa tindakan `Get`, yang dapat Anda kelompokkan dan kirimkan sebagai operasi `TransactGetItems` tunggal.

Tidak ada biaya tambahan untuk mengaktifkan transaksi untuk tabel DynamoDB Anda. Anda hanya membayar untuk baca atau tulis yang merupakan bagian dari transaksi Anda. DynamoDB melakukan dua baca atau tulis mendasar dari setiap item dalam transaksi: satu untuk mempersiapkan transaksi dan satu untuk melakukan transaksi. Dua operasi baca/tulis yang mendasari ini terlihat di metrik Amazon CloudWatch Anda.

Untuk memulai transaksi DynamoDB, unduh SDK AWS terbaru atau (). AWS Command Line Interface AWS CLI Kemudian ikuti Contoh DynamoDB transactions.

Bagian berikut memberikan gambaran rinci tentang transaksi APIs dan bagaimana Anda dapat menggunakannya di DynamoDB.

Topik

- Amazon DynamoDB Transactions: Cara kerjanya
- Menggunakan IAM dengan DynamoDB transactions
- Contoh DynamoDB transactions

Amazon DynamoDB Transactions: Cara kerjanya

Dengan transaksi Amazon DynamoDB, Anda dapat mengelompokkan beberapa tindakan bersama-sama dan mengirimkannya sebagai satu atau all-or-nothing `TransactWriteItems` operasi. `TransactGetItems` Bagian berikut menjelaskan operasi API, manajemen kapasitas, praktik terbaik, dan detail lainnya tentang menggunakan operasi transaksional di DynamoDB.

Topik

- TransactWriteItems API
- TransactGetItems API

- Tingkat isolasi untuk DynamoDB transactions
- Penanganan konflik transaksi di DynamoDB
- Menggunakan transaksional di APIs DynamoDB Accelerator (DAX)
- Manajemen kapasitas untuk transaksi
- Praktik terbaik untuk transactions
- Menggunakan transaksional APIs dengan tabel global
- Transaksi DynamoDB AWSLabs vs. perpustakaan klien transaksi

TransactWriteItems API

`TransactWriteItems`adalah operasi penulisan sinkron dan idempoten yang mengelompokkan hingga 100 tindakan penulisan dalam satu operasi. all-or-nothing Tindakan ini dapat menargetkan hingga 100 item berbeda dalam satu atau lebih tabel DynamoDB dalam akun AWS yang sama dan di Wilayah yang sama. Ukuran agregat item dalam transaksi tidak dapat melebihi 4 MB. Tindakan tersebut diselesaikan secara atomik, sehingga semuanya berhasil atau tidak satu pun yang berhasil.

> **Note**
>
> - Operasi `TransactWriteItems` berbeda dari operasi `BatchWriteItem` dalam hal semua tindakan yang di dalamnya harus berhasil diselesaikan, atau tidak ada perubahan yang dibuat sama sekali. Dengan operasi `BatchWriteItem`, mungkin bahwa hanya beberapa tindakan dalam batch yang berhasil sementara yang lain tidak.
> - Transaksi tidak dapat dilakukan dengan menggunakan indeks.

Anda tidak dapat menargetkan item yang sama dengan beberapa operasi dalam transaksi yang sama. Misalnya, Anda tidak dapat melakukan `ConditionCheck` dan juga tindakan `Update` pada item yang sama dalam transaksi yang sama.

Anda dapat menambahkan jenis tindakan berikut pada transaksi:

- `Put` — Memulai operasi `PutItem` untuk membuat item baru atau mengganti item lama dengan item baru, kondisional atau tanpa menentukan syarat apa pun.
- `Update` — Memulai operasi `UpdateItem` untuk mengedit atribut item yang ada atau menambahkan item baru ke tabel jika belum ada. Gunakan tindakan ini untuk menambah,

menghapus, atau memperbarui atribut pada item yang ada secara dengan syarat maupun tanpa syarat.

- `Delete` — Memulai operasi `DeleteItem` untuk menghapus satu item dalam tabel yang diidentifikasi oleh kunci primernya.
- `ConditionCheck` — Memeriksa bahwa item ada atau periksa kondisi atribut tertentu dari item.

Ketika transaksi selesai di DynamoDB, perubahannya mulai menyebar ke indeks sekunder global GSIs (), aliran, dan cadangan. Propagasi ini terjadi secara bertahap: catatan aliran dari transaksi yang sama mungkin muncul pada waktu yang berbeda dan dapat disisipkan dengan catatan dari transaksi lain. Konsumen streaming tidak boleh menganggap atomisitas transaksi atau jaminan pemesanan.

Untuk memastikan snapshot atom dari item yang dimodifikasi dalam transaksi, gunakan TransactGetItems operasi untuk membaca semua item yang relevan bersama-sama. Operasi ini memberikan tampilan data yang konsisten, memastikan Anda melihat semua perubahan dari transaksi yang diselesaikan atau tidak sama sekali.

Karena propagasi tidak langsung, jika tabel dipulihkan dari backup (RestoreTableFromBackup) atau diekspor ke titik waktu (ExportTableToPointInTime) pertengahan propagasi, itu hanya dapat berisi beberapa perubahan yang dibuat selama transaksi baru-baru ini.

Idempotensi

Anda dapat menyertakan token klien ketika Anda membuat panggilan `TransactWriteItems` untuk memastikan bahwa permintaan tersebut idempoten. Menjadikan transaksi Anda idempoten membantu mencegah kesalahan aplikasi jika operasi yang sama diajukan beberapa kali karena waktu habis koneksi atau masalah konektivitas lainnya.

Jika panggilan `TransactWriteItems` asli berhasil, panggilan `TransactWriteItems` berikutnya dengan token klien yang sama berhasil kembali tanpa membuat perubahan apa pun. Jika parameter `ReturnConsumedCapacity` diatur, panggilan `TransactWriteItems` awal mengembalikan jumlah unit kapasitas tulis yang dikonsumsi dalam membuat perubahan. Panggilan `TransactWriteItems` selanjutnya dengan token klien yang sama mengembalikan jumlah unit kapasitas baca yang dikonsumsi dalam membaca item.

Poin penting tentang idempotensi

- Token klien berlaku selama 10 menit setelah permintaan yang menggunakannya selesai. Setelah 10 menit, setiap permintaan yang menggunakan token klien yang sama diperlakukan sebagai

permintaan baru. Anda tidak boleh menggunakan kembali token klien yang sama untuk permintaan yang sama setelah 10 menit.

- Jika Anda mengulangi permintaan dengan token klien yang sama dalam jendela idempotensi 10 menit tetapi mengubah beberapa parameter permintaan lainnya, DynamoDB mengembalikan pengecualian `IdempotentParameterMismatch`.

Penanganan kesalahan untuk penulisan

Transaksi tulis tidak berhasil dalam situasi berikut:

- Ketika syarat di salah satu ekspresisyarat tidak terpenuhi.
- Ketika kesalahan validasi transaksi terjadi karena lebih dari satu tindakan dalam operasi `TransactWriteItems` yang sama menargetkan item yang sama.
- Saat permintaan `TransactWriteItems` bertentangan dengan operasi `TransactWriteItems` yang sedang berlangsung pada satu atau lebih item dalam permintaan `TransactWriteItems`. Dalam kasus ini, permintaan gagal dengan `TransactionCanceledException`.
- Ketika ada kapasitas yang ditetapkan tidak mencukupi untuk penyelesaian transaksi.
- Ketika ukuran item menjadi terlalu besar (lebih besar dari 400 KB), atau indeks sekunder lokal (LSI) menjadi terlalu besar, atau kesalahan validasi serupa terjadi karena perubahan yang dilakukan oleh transaksi.
- Ketika ada kesalahan pengguna, seperti format data yang tidak valid.

Untuk informasi selengkapnya tentang bagaimana pertentangan dengan operasi `TransactWriteItems` ditangani, lihat Penanganan konflik transaksi di DynamoDB.

TransactGetItems API

`TransactGetItems` adalah operasi baca sinkron yang mengelompokkan hingga 100 tindakan `Get` menjadi satu. Tindakan ini dapat menargetkan hingga 100 item berbeda dalam satu atau lebih tabel DynamoDB dalam akun dan Wilayah yang AWS sama. Ukuran agregat item dalam transaksi tidak dapat melebihi 4 MB.

Tindakan `Get` dilakukan secara atom sehingga semuanya berhasil atau semuanya gagal:

- `Get` — Memulai operasi `GetItem` untuk mengambil satu set atribut untuk item dengan kunci primer yang diberikan. Jika tidak ditemukan item yang cocok, `Get` tidak mengembalikan data apa pun.

Penanganan kesalahan untuk pembacaan

Transaksi baca tidak berhasil dalam situasi berikut:

- Saat permintaan `TransactGetItems` bertentangan dengan operasi `TransactWriteItems` yang sedang berlangsung pada satu atau lebih item dalam permintaan `TransactGetItems`. Dalam kasus ini, permintaan gagal dengan `TransactionCanceledException`.
- Ketika ada kapasitas yang ditetapkan tidak mencukupi untuk penyelesaian transaksi.
- Ketika ada kesalahan pengguna, seperti format data yang tidak valid.

Untuk informasi selengkapnya tentang bagaimana pertentangan dengan operasi `TransactGetItems` ditangani, lihat Penanganan konflik transaksi di DynamoDB.

Tingkat isolasi untuk DynamoDB transactions

Tingkat isolasi operasi transaksional (`TransactWriteItems` atau `TransactGetItems`) dan operasi lainnya adalah sebagai berikut.

DAPAT DISERIALKAN

Isolasi yang dapat diserialkan memastikan bahwa hasil dari beberapa operasi bersamaan sama seperti jika tidak ada operasi dimulai hingga operasi sebelumnya telah selesai.

Ada isolasi serialisasi antara jenis operasi berikut:

- Antara operasi transaksional dan operasi tulis standar (`PutItem`, `UpdateItem`, atau `DeleteItem`).
- Antara operasi transaksional dan operasi baca standar (`GetItem`).
- Antara operasi `TransactWriteItems` dan operasi `TransactGetItems`.

Meskipun ada isolasi serialisasi antara operasi transaksional, dan setiap standar individu menulis dalam suatu `BatchWriteItem` operasi, tidak ada isolasi serial antara transaksi dan operasi sebagai satu unit. `BatchWriteItem`

Demikian pula, tingkat isolasi antara operasi transaksional dan individu `GetItems` dalam operasi `BatchGetItem` dapat diserialisasikan. Tetapi tahap isolasi antara transaksi dan operasi `BatchGetItem` sebagai unit adalah read-committed.

Permintaan `GetItem` tunggal dapat diserialisasikan sehubungan dengan permintaan `TransactWriteItems` dalam salah satu dari dua cara, baik sebelum atau setelah permintaan `TransactWriteItems`. Beberapa permintaan `GetItem`, terhadap kunci dalam permintaan `TransactWriteItems` bersamaan dapat dijalankan dalam urutan apa pun, dan karena itu hasilnya read-committed.

Misalnya, jika permintaan `GetItem` untuk item A dan item B dijalankan bersamaan dengan permintaan `TransactWriteItems` yang memodifikasi item A dan item B, ada empat kemungkinan:

- Kedua permintaan `GetItem` dijalankan sebelum permintaan `TransactWriteItems`.
- Kedua permintaan `GetItem` dijalankan setelah permintaan `TransactWriteItems`.
- Permintaan `GetItem` untuk item A dijalankan sebelum permintaan `TransactWriteItems`. Untuk item B, `GetItem` dijalankan setelah `TransactWriteItems`.
- Permintaan `GetItem` untuk item B dijalankan sebelum permintaan `TransactWriteItems`. Untuk item A, `GetItem` dijalankan setelah `TransactWriteItems`.

Anda harus menggunakan `TransactGetItems` jika Anda lebih suka tingkat isolasi serializable untuk beberapa permintaan. `GetItem`

Jika pembacaan non-transaksional dilakukan pada beberapa item yang merupakan bagian dari permintaan penulisan transaksi yang sama dalam penerbangan, Anda mungkin dapat membaca status baru dari beberapa item dan status lama item lainnya. Anda akan dapat membaca status baru dari semua item yang merupakan bagian dari permintaan tulis transaksi hanya ketika respons berhasil diterima untuk penulisan transaksional, yang menunjukkan bahwa transaksi telah selesai.

Setelah transaksi berhasil diselesaikan dan respons diterima, operasi baca yang konsisten selanjutnya mungkin masih mengembalikan status lama untuk waktu yang singkat karena model konsistensi DynamoDB akhirnya. Untuk menjamin membaca sebagian besar up-to-date data segera setelah transaksi, Anda harus menggunakan pembacaan yang sangat konsisten dengan menyetel `ConsistentRead` ke true.

READ-COMMITTED

Isolasi read-commit memastikan bahwa operasi baca selalu mengembalikan komitmen untuk suatu item - pembacaan tidak akan pernah menampilkan tampilan ke item yang mewakili keadaan dari penulisan transaksional yang pada akhirnya tidak berhasil. Isolasi komitmen baca tidak mencegah modifikasi item segera setelah operasi baca.

Tingkat isolasi read-committed antara operasi transaksional dan operasi baca yang melibatkan beberapa baca standar (`BatchGetItem`, `Query`, atau `Scan`). Jika transaksional tulis memperbarui item di tengah operasi `BatchGetItem`, `Query`, atau `Scan`, bagian selanjutnya dari operasi baca mengembalikan nilai berkomitmen baru (dengan `ConsistentRead`) atau mungkin nilai berkomitmen sebelumnya (bacaan akhir konsisten).

Ringkasan operasi

Untuk meringkas, tabel berikut menunjukkan tingkat isolasi antara operasi transaksi (`TransactWriteItems` atau `TransactGetItems`) dan operasi lainnya.

Operasi	Tingkat Isolasi
`DeleteItem`	Dapat diserialkan
`PutItem`	Dapat diserialkan
`UpdateItem`	Dapat diserialkan
`GetItem`	Dapat diserialkan
`BatchGetItem`	Read-committed*
`BatchWriteItem`	Tidak Dapat Diserialkan*
`Query`	Read-committed
`Scan`	Read-committed
Operasi transaksional lainnya	Dapat diserialkan

Tingkat yang ditandai dengan tanda bintang (*) berlaku untuk operasi sebagai sebuah unit. Namun, tindakan individu dalam operasi tersebut memiliki tingkat isolasi yang dapat diserialkan.

Penanganan konflik transaksi di DynamoDB

Konflik transaksional dapat terjadi selama permintaan tingkat item bersamaan pada item dalam transaksi. Konflik transaksi dapat terjadi dalam skenario berikut:

- Permintaan `PutItem`, `UpdateItem`, atau `DeleteItem` untuk item yang bertentangan dengan permintaan `TransactWriteItems` yang sedang berlangsung yang mencakup item yang sama.
- Item dalam permintaan `TransactWriteItems` adalah bagian dari permintaan `TransactWriteItems` lain yang sedang berlangsung.
- Item dalam permintaan `TransactGetItems` adalah bagian dari permintaan `TransactWriteItems`, `BatchWriteItem`, `PutItem`, `UpdateItem`, atau `DeleteItem` yang sedang berlangsung.

> **Note**
>
> - Saat permintaan `PutItem`, `UpdateItem`, atau `DeleteItem` ditolak, permintaan gagal dengan `TransactionConflictException`.
> - Jika ada permintaan tingkat item dalam `TransactWriteItems` atau `TransactGetItems` ditolak, permintaan gagal dengan `TransactionCanceledException`. Jika permintaan itu gagal, AWS SDKs jangan coba lagi permintaan tersebut.
>
> Jika Anda menggunakan AWS SDK untuk Java, pengecualian berisi daftar CancellationReasons, diurutkan sesuai dengan daftar item dalam parameter `TransactItems` permintaan. Untuk bahasa lain, representasi string daftar disertakan dalam pesan kesalahan pengecualian.
> - Jika operasi `TransactWriteItems` atau `TransactGetItems` yang sedang berlangsung bertentangan dengan permintaan `GetItem` bersamaan, kedua operasi dapat berhasil.

TransactionConflict CloudWatch Metrik ditambahkan untuk setiap permintaan tingkat item yang gagal.

Menggunakan transaksional di APIs DynamoDB Accelerator (DAX)

`TransactWriteItems` dan `TransactGetItems` keduanya didukung di DynamoDB Accelerator (DAX) dengan tingkat isolasi yang sama seperti di DynamoDB.

`TransactWriteItems` menulis melalui DAX. DAX menyampaikan panggilan `TransactWriteItems` ke DynamoDB dan mengembalikan respons. Untuk mempopulasi cache setelah menulis, DAX memanggil `TransactGetItems` di latar belakang untuk setiap item dalam

operasi `TransactWriteItems`, yang mengonsumsi unit kapasitas baca tambahan. (Untuk informasi selengkapnya, lihat Manajemen kapasitas untuk transaksi.) Fungsionalitas ini memungkinkan Anda untuk menjaga logika aplikasi Anda sederhana dan menggunakan DAX untuk kedua operasi transaksional dan nontransaksional.

Panggilan `TransactGetItems` disampaikan melalui DAX tanpa item di-cache secara lokal. Ini adalah perilaku yang sama dengan pembacaan yang sangat konsisten APIs di DAX.

Manajemen kapasitas untuk transaksi

Tidak ada biaya tambahan untuk mengaktifkan transaksi untuk tabel DynamoDB Anda. Anda hanya membayar untuk baca atau tulis yang merupakan bagian dari transaksi Anda. DynamoDB melakukan dua baca atau tulis mendasar dari setiap item dalam transaksi: satu untuk mempersiapkan transaksi dan satu untuk melakukan transaksi. Dua operasi baca/tulis yang mendasarinya terlihat di metrik Amazon CloudWatch Anda.

Rencanakan untuk membaca dan menulis tambahan yang diperlukan oleh transaksional APIs ketika Anda menyediakan kapasitas ke tabel Anda. Misalnya, anggaplah aplikasi Anda menjalankan satu transaksi per detik, dan setiap transaksi menulis tiga item 500-byte dalam tabel Anda. Setiap item membutuhkan dua unit kapasitas tulis (WCUs): satu untuk mempersiapkan transaksi dan satu untuk melakukan transaksi. Oleh karena itu, Anda perlu menyediakan enam WCUs ke meja.

Jika Anda menggunakan DynamoDB Accelerator (DAX) pada contoh sebelumnya, Anda juga akan menggunakan dua unit kapasitas baca RCUs () untuk setiap item dalam panggilan. `TransactWriteItems` Jadi, Anda perlu memberikan enam tambahan RCUs ke tabel.

Demikian pula, jika aplikasi Anda menjalankan satu transaksi baca per detik, dan setiap transaksi membaca tiga item 500 byte dalam tabel Anda, Anda perlu menyediakan enam unit kapasitas baca (RCUs) ke tabel. Membaca setiap item membutuhkan dua RCUs: satu untuk mempersiapkan transaksi dan satu untuk melakukan transaksi.

Selain itu, perilaku SDK default adalah untuk mencoba lagi transaksi dalam kasus pengecualian `TransactionInProgressException`. Rencanakan unit kapasitas baca tambahan (RCUs) yang dikonsumsi oleh percobaan ulang ini. Hal yang sama juga berlaku jika Anda mencoba ulang transaksi dalam kode Anda sendiri menggunakan `ClientRequestToken`.

Praktik terbaik untuk transactions

Pertimbangkan praktik yang disarankan berikut saat menggunakan DynamoDB Transactions.

- Aktifkan penskalaan otomatis pada tabel Anda, atau pastikan bahwa Anda telah menyediakan kapasitas throughput yang cukup untuk melakukan dua operasi baca atau tulis untuk setiap item dalam transaksi Anda.
- Jika Anda tidak menggunakan SDK yang AWS disediakan, sertakan `ClientRequestToken` atribut saat Anda melakukan `TransactWriteItems` panggilan untuk memastikan bahwa permintaan tersebut idempoten.
- Jangan mengelompokkan operasi bersama-sama dalam sebuah transaksi jika tidak diperlukan. Misalnya, jika satu transaksi dengan 10 operasi dapat dipecah menjadi beberapa transaksi tanpa mengurangi kebenaran aplikasi, sebaiknya pisahkan transaksi tersebut. Transaksi yang lebih sederhana meningkatkan throughput dan lebih mungkin untuk berhasil.
- Beberapa transaksi memperbarui item yang sama secara bersamaan dapat menyebabkan konflik yang membatalkan transaksi. Kami merekomendasikan praktik terbaik DynamoDB berikut untuk pemodelan data untuk meminimalkan konflik tersebut.
- Jika satu set atribut sering diperbarui di beberapa item sebagai bagian dari transaksi tunggal, pertimbangkan pengelompokan atribut menjadi satu item untuk mengurangi lingkup transaksi.
- Hindari penggunaan transaksi untuk menyerap data secara massal. Untuk tulis massal, lebih baik menggunakan `BatchWriteItem`.

Menggunakan transaksional APIs dengan tabel global

Operasi transaksional memberikan jaminan atomisitas, konsistensi, isolasi, dan daya tahan (ACID) hanya di AWS Wilayah tempat API tulis dipanggil. Transaksi tidak didukung di seluruh Wilayah dalam tabel global. Misalnya, Anda memiliki tabel global dengan replika di Wilayah AS Timur (Ohio) dan AS Barat (Oregon) dan Anda melakukan operasi `TransactWriteItems` di Wilayah AS Timur (Virginia Utara). Anda dapat mengamati transaksi yang diselesaikan sebagian di Wilayah Barat AS (Oregon) saat perubahan direplikasi. Perubahan direplikasi ke Wilayah lain hanya setelah dilakukan di Wilayah sumber.

Transaksi DynamoDB AWSLabs vs. perpustakaan klien transaksi

Transaksi DynamoDB menyediakan penggantian yang lebih hemat biaya, kuat, dan berkinerja untuk pustaka klien transaksi. AWSLabs Kami menyarankan agar Anda memperbarui aplikasi Anda untuk menggunakan transaksi asli, sisi server. APIs

Menggunakan IAM dengan DynamoDB transactions

Anda dapat menggunakan AWS Identity and Access Management (IAM) untuk membatasi tindakan yang dapat dilakukan operasi transaksional di Amazon DynamoDB. Untuk informasi selengkapnya tentang menggunakan kebijakan IAM dalam DynamoDB, lihat Kebijakan berbasis identitas untuk DynamoDB.

Izin untuk tindakan `Put`, `Update`, `Delete`, dan `Get` diatur oleh izin yang digunakan untuk operasi `PutItem`, `UpdateItem`, `DeleteItem`, dan `GetItem` yang mendasarinya. Untuk tindakan `ConditionCheck`, Anda dapat menggunakan izin `dynamodb:ConditionCheckItem` dalam kebijakan IAM.

Berikut ini adalah contoh kebijakan IAM yang dapat Anda gunakan untuk mengonfigurasi DynamoDB transactions.

Contoh 1: Mengizinkan operasi transaksional

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "dynamodb:ConditionCheckItem",
                "dynamodb:PutItem",
                "dynamodb:UpdateItem",
                "dynamodb:DeleteItem",
                "dynamodb:GetItem"
            ],
            "Resource": [
                "arn:aws:dynamodb:*:*:table/table04"
            ]
        }
    ]
}
```

Contoh 2: Hanya mengizinkan operasi transaksional

```
{
    "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"
                    ]
                }
            }
        }
    ]
}
```

Contoh 3: Mengizinkan baca dan tulis nontransaksional, dan memblokir baca dan tulis transaksional

```
{
    "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"
             ]
         }
    ]
}
```

Contoh 4: Mencegah informasi dikembalikan pada ConditionCheck kegagalan

```
{
    "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"
                }
            }
```

```
        }
    ]
}
```

Contoh DynamoDB transactions

Sebagai contoh situasi di mana Amazon DynamoDB transactions dapat berguna, pertimbangkan contoh aplikasi Java untuk marketplace online ini.

Aplikasi ini memiliki tiga tabel DynamoDB di backend:

- `Customers` — Tabel ini menyimpan detail tentang pelanggan marketplace. Kunci primernya adalah pengidentifikasi unik `CustomerId`.
- `ProductCatalog` — Tabel ini menyimpan detail seperti harga dan ketersediaan produk yang dijual di marketplace. Kunci primernya adalah pengidentifikasi unik `ProductId`.
- `Orders` — Tabel ini menyimpan detail pesanan dari marketplace. Kunci primernya adalah pengidentifikasi unik `OrderId`.

Membuat pesanan

Potongan kode berikut menggambarkan cara menggunakan DynamoDB transaction untuk mengoordinasikan beberapa langkah yang diperlukan untuk membuat dan memproses pesanan. Menggunakan satu all-or-nothing operasi memastikan bahwa jika ada bagian dari transaksi yang gagal, tidak ada tindakan dalam transaksi yang dijalankan dan tidak ada perubahan yang dilakukan.

Dalam contoh ini, Anda menyiapkan pesanan dari pelanggan yang `customerId` adalah `09e8e9c8-ec48`. Anda kemudian menjalankannya sebagai satu transaksi menggunakan alur kerja pemrosesan pesanan sederhana berikut:

1. Tentukan apakah ID pelanggan valid.
2. Pastikan produk `IN_STOCK`, dan perbarui status produk menjadi `SOLD`.
3. Pastikan pesanan belum ada, lalu buat pesanan.

Memvalidasi pelanggan

Pertama, tentukan tindakan untuk memverifikasi bahwa pelanggan dengan `customerId` sama dengan `09e8e9c8-ec48` ada di tabel pelanggan.

```
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 + ")");
```

Memperbarui status produk

Selanjutnya, tentukan tindakan untuk memperbarui status produk menjadi `SOLD` jika kondisi status produk saat ini diatur ke `IN_STOCK` `true`. Mengatur parameter `ReturnValuesOnConditionCheckFailure` akan mengembalikan item jika atribut status produk item tidak sama dengan `IN_STOCK`.

```
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);
```

Membuat pesanan

Terakhir, buat pesanan selama pesanan dengan `OrderId` tersebut belum ada.

```
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 + ")");
```

Menjalankan transaksi

Contoh berikut menggambarkan bagaimana menjalankan tindakan yang didefinisikan sebelumnya sebagai all-or-nothing operasi tunggal.

```
    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());
    }
```

Membaca detail pesanan

Contoh berikut menunjukkan cara membaca pesanan yang telah selesai secara transaksional di seluruh tabel `Orders` dan `ProductCatalog`.

```
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());
}
```

Tangkapan data perubahan dengan Amazon DynamoDB

Banyak aplikasi mendapatkan manfaat dari menangkap perubahan ke item yang disimpan dalam tabel DynamoDB pada titik waktu ketika perubahan terjadi. Berikut adalah beberapa contoh kasus penggunaan:

- Sebuah aplikasi seluler populer memodifikasi data dalam tabel DynamoDB, pada tingkat ribuan pembaruan per detik. Aplikasi lain menangkap dan menyimpan data tentang pembaruan ini, menyediakan metrik near-real-time penggunaan untuk aplikasi seluler.
- Sebuah aplikasi keuangan memodifikasi data pasar saham dalam tabel DynamoDB. Berbagai aplikasi yang berjalan secara paralel melacak perubahan ini secara real time, menghitung value-at-risk, dan secara otomatis menyeimbangkan portofolio berdasarkan pergerakan harga saham.
- Sensor dalam kendaraan transportasi dan peralatan industri mengirim data ke meja DynamoDB. Aplikasi yang berbeda memantau kinerja dan mengirim pesan peringatan ketika masalah terdeteksi, memprediksi potensi cacat dengan menerapkan algoritma machine learning, dan mengompres serta mengarsip data ke Amazon Simple Storage Service (Amazon S3).
- Sebuah aplikasi secara otomatis mengirimkan pemberitahuan ke perangkat seluler dari semua teman dalam grup segera setelah satu teman mengunggah gambar baru.
- Seorang pelanggan baru menambahkan data ke tabel DynamoDB. Acara ini memanggil aplikasi lain yang mengirimkan email selamat datang ke pelanggan baru.

DynamoDB mendukung streaming catatan tangkapan data perubahan tingkat item yang dalam hampir waktu nyata. Anda dapat membangun aplikasi yang mengkonsumsi stream ini dan mengambil tindakan berdasarkan isinya.

> **Note**
>
> Menambahkan tag ke DynamoDB Streams dan menggunakan kontrol akses berbasis atribut (ABAC) dengan DynamoDB Streams tidak didukung.

Video berikut akan memberi Anda tampilan pengantar tentang konsep pengambilan data perubahan.

Mode kapasitas tabel

Topik

- Opsi streaming untuk tangkapan data perubahan

- Menggunakan Kinesis Data Streams untuk menangkap perubahan pada DynamoDB
- Tangkapan data perubahan DynamoDB Streams

Opsi streaming untuk tangkapan data perubahan

DynamoDB menawarkan dua model streaming untuk penangkapan data perubahan: Kinesis Data Streams untuk DynamoDB dan DynamoDB Streams.

Untuk membantu Anda memilih solusi yang tepat untuk aplikasi Anda, tabel berikut merangkum fitur dari setiap model streaming.

Properti	Aliran Data Kinesis untuk DynamoDB	DynamoDB Streams
Retensi data	Hingga 1 tahun.	24 jam.
Dukungan Perpustakaan Klien Kinesis (KCL)	Mendukung KCL versi 1.X, 2.X, dan 3.X.	Mendukung KCL versi 1.X dan 2.X.
Jumlah konsumen	Hingga 5 konsumen simultan per serpihan, atau hingga 20 konsumen simultan per serpihan dengan fan-out yang ditingkatkan.	Hingga 2 konsumen simultan per serpihan.
Kuota throughput	Tidak terbatas.	Tunduk pada kuota throughput oleh tabel DynamoDB dan Wilayah. AWS
Model pengiriman catatan	Tarik model melalui HTTP menggunakan GetRecords dan dengan fan-out yang disempurnakan, Kinesis Data Streams mendorong catatan melalui HTTP/2 dengan menggunakan. SubscribeToShard	Tarik model melalui HTTP menggunakan GetRecords.

Properti	Aliran Data Kinesis untuk DynamoDB	DynamoDB Streams
Pengurutan catatan	Atribut stempel waktu pada setiap catatan stream dapat digunakan untuk mengident ifikasi urutan aktual di mana perubahan terjadi dalam tabel DynamoDB.	Untuk setiap item yang diubah dalam tabel DynamoDB, catatan stream muncul dalam urutan yang sama seperti modifikasi aktual untuk item.
Catatan duplikat	Catatan duplikat terkadang muncul di stream.	Tidak ada catatan duplikat muncul di stream.
Opsi pemrosesan stream	Memproses rekaman aliran menggunakan AWS Lambda, Layanan Terkelola Amazon untuk Apache Flink, Kinesis data firehose, atau streaming ETL AWS Glue.	Proses catatan stream menggunakan AWS Lambda atau Adaptor Kinesis DynamoDB Streams.
Tingkat daya tahan	Zone ketersediaan untuk menyediakan failover otomatis tanpa gangguan.	Zone ketersediaan untuk menyediakan failover otomatis tanpa gangguan.

Anda dapat mengaktifkan kedua model streaming pada tabel DynamoDB yang sama.

Video berikut berbicara lebih banyak tentang perbedaan antara dua opsi tersebut.

DynamoDB Streams vs Kinesis Data Streams

Menggunakan Kinesis Data Streams untuk menangkap perubahan pada DynamoDB

Gunakan Amazon Kinesis Data Streams untuk menangkap perubahan pada Amazon DynamoDB.

Kinesis Data Streams menangkap modifikasi tingkat item di tabel DynamoDB apa pun dan mereplikasikannya ke Kinesis data stream. Aplikasi Anda dapat mengakses stream ini dan melihat perubahan tingkat item dalam hampir waktu nyata. Anda dapat terus menangkap dan menyimpan

data berukuran terabyte per jam. Anda dapat memanfaatkan waktu retensi data yang lebih lama—dan dengan kemampuan fan-out yang ditingkatkan, Anda dapat menjangkau dua atau lebih aplikasi hilir secara bersamaan. Manfaat lainnya termasuk audit tambahan dan transparansi keamanan.

Kinesis Data Streams juga memberi Anda akses ke Amazon Data Firehose dan Amazon Managed Service untuk Apache Flink. Layanan ini dapat membantu Anda membangun aplikasi yang mendukung dasbor real-time, menghasilkan peringatan, menerapkan penetapan harga dan periklanan dinamis, serta menerapkan analitik data dan algoritma machine learning yang canggih.

> **Note**
>
> Penggunaan Kinesis Data Streams untuk DynamoDB tunduk pada harga Kinesis Data Streams untuk aliran data dan harga DynamoDB untuk tabel sumber.

Cara kerja Kinesis Data Streams dengan DynamoDB

Ketika aliran data Kinesis diaktifkan untuk tabel DynamoDB, tabel mengirimkan rekaman data yang mencatat perubahan apa pun pada data tabel tersebut. Catatan data ini meliputi:

- Waktu spesifik item mana pun baru saja dibuat, diperbarui, atau dihapus
- Kunci utama item itu
- Cuplikan catatan sebelum modifikasi
- Cuplikan catatan setelah modifikasi

Catatan data ini diambil dan dipublikasikan hampir secara waktu nyata. Setelah ditulis ke aliran data Kinesis, data tersebut dapat dibaca seperti catatan lainnya. Anda dapat menggunakan Kinesis Client Library, menggunakan AWS Lambda, memanggil Kinesis Data Streams API, dan menggunakan layanan terhubung lainnya. Untuk informasi selengkapnya, lihat Membaca Data dari Amazon Kinesis Data Streams dalam Panduan Developer Amazon Kinesis Data Streams.

Perubahan pada data ini juga ditangkap secara asinkron. Kinesis tidak memiliki dampak kinerja pada tabel asal streamingnya. Catatan aliran yang disimpan dalam aliran data Kinesis Anda juga dienkripsi saat tidak digunakan. Untuk informasi selengkapnya, lihat Perlindungan Data di Amazon Kinesis Data Streams.

Catatan aliran data Kinesis mungkin muncul dalam urutan yang berbeda dari saat perubahan item terjadi. Notifikasi item yang sama mungkin juga muncul lebih dari satu kali di aliran. Anda dapat

memeriksa `ApproximateCreationDateTime` atribut untuk mengidentifikasi urutan modifikasi item terjadi, dan untuk mengidentifikasi catatan duplikat.

Saat Anda mengaktifkan aliran data Kinesis sebagai tujuan streaming tabel DynamoDB, Anda dapat mengonfigurasi presisi `ApproximateCreationDateTime` nilai dalam milidetik atau mikrodetik. Secara default, `ApproximateCreationDateTime` menunjukkan waktu perubahan dalam milidetik. Selain itu, Anda dapat mengubah nilai ini di tujuan streaming aktif. Setelah pembaruan seperti itu, catatan aliran yang ditulis ke Kinesis akan memiliki `ApproximateCreationDateTime` nilai presisi yang diinginkan.

Nilai biner yang ditulis ke DynamoDB harus dikodekan dalam format berenkode base64 . Namun, ketika catatan data ditulis ke aliran data Kinesis, nilai biner yang dikodekan ini dikodekan dengan pengodean base64 untuk kedua kalinya. Saat membaca catatan ini dari aliran data Kinesis, untuk mengambil nilai biner mentah, aplikasi harus mendekode nilai ini dua kali.

DynamoDB mengenakan biaya untuk penggunaan Kinesis Data Streams dalam unit pengambilan data perubahan. Perubahan 1 KB per item dihitung sebagai satu unit pengambilan data perubahan. KB perubahan di setiap item dihitung berdasarkan gambar "sebelum" dan "sesudah" yang lebih besar dari item yang ditulis ke aliran, menggunakan logika yang sama dengan konsumsi unit kapasitas untuk operasi tulis. Mirip dengan bagaimana mode DynamoDB sesuai permintaan bekerja, Anda tidak perlu menyediakan throughput kapasitas untuk unit penangkapan data perubahan.

Mengaktifkan aliran data Kinesis untuk tabel DynamoDB Anda

Anda dapat mengaktifkan atau menonaktifkan streaming ke Kinesis dari tabel DynamoDB yang ada dengan menggunakan, AWS SDK AWS Management Console, atau (). AWS Command Line Interface AWS CLI

- Anda hanya dapat mengalirkan data dari DynamoDB ke Kinesis Data Streams di AWS akun dan Wilayah yang sama dengan tabel Anda. AWS
- Anda hanya dapat melakukan streaming data dari tabel DynamoDB ke satu Kinesis data stream.

Membuat perubahan pada tujuan Kinesis Data Streams pada tabel DynamoDB

Secara default, semua catatan aliran data Kinesis menyertakan atribut`ApproximateCreationDateTime`. Atribut ini mewakili stempel waktu dalam milidetik dari perkiraan waktu ketika setiap rekaman dibuat. Anda dapat mengubah presisi nilai-nilai ini dengan menggunakan https://console.aws.amazon.com/kinesis, SDK atau AWS CLI

Memulai Kinesis Data Streams untuk Amazon DynamoDB

Bagian ini menjelaskan cara menggunakan Kinesis Data Streams untuk tabel Amazon DynamoDB dengan konsol Amazon AWS Command Line Interface AWS CLI DynamoDB, (), dan API.

Membuat aliran data Amazon Kinesis yang aktif

Semua contoh ini menggunakan tabel `Music` DynamoDB yang dibuat sebagai bagian dari tutorial Memulai dengan DynamoDB.

Untuk mempelajari lebih lanjut tentang cara membangun konsumen dan menghubungkan aliran data Kinesis Anda ke layanan AWS lain, lihat Membaca data dari Kinesis Data Streams dalam panduan pengembang Amazon Kinesis Data Streams.

> **Note**
>
> Saat Anda pertama kali menggunakan serpihan KDS, sebaiknya atur serpihan Anda untuk ditingkatkan dan diturunkan skalanya sesuai dengan pola penggunaan. Setelah mengumpulkan lebih banyak data tentang pola penggunaan, Anda dapat menyesuaikan serpihan di aliran agar sesuai.

Console

1. Masuk ke AWS Management Console dan buka konsol Kinesis di. https://console.aws.amazon.com/kinesis/
2. Pilih Buat aliran data dan ikuti petunjuk untuk membuat stream yang disebut `samplestream`.
3. Buka konsol DynamoDB di. https://console.aws.amazon.com/dynamodb/
4. Di panel navigasi di sisi kiri konsol, pilih Tabel.
5. Pilih tabel Musik.
6. Pilih tab Ekspor dan aliran.
7. (Opsional) Di bawah detail aliran data Amazon Kinesis, Anda dapat mengubah presisi stempel waktu rekaman dari mikrodetik (default) menjadi milidetik.
8. Pilih samplestream dari daftar menurun.
9. Pilih tombol Hidupkan.

AWS CLI

1. Buat Kinesis Data Streams bernama `samplestream` dengan menggunakan perintah create-stream.

   ```
   aws kinesis create-stream --stream-name samplestream --shard-count 3
   ```

   Lihat Pertimbangan manajemen pecahan untuk Kinesis Data Streams sebelum menetapkan jumlah serpihan untuk Kinesis data stream.

2. Periksa apakah stream Kinesis aktif dan siap digunakan dengan menggunakan perintah describe-stream.

   ```
   aws kinesis describe-stream --stream-name samplestream
   ```

3. Aktifkan Kinesis streaming pada tabel DynamoDB dengan menggunakan perintah `enable-kinesis-streaming-destination` DynamoDB. Ganti `stream-arn` nilai dengan yang ditampilkan `describe-stream` pada langkah sebelumnya. Secara opsional, aktifkan streaming dengan presisi nilai stempel waktu yang lebih granular (mikrodetik) yang dikembalikan pada setiap catatan.

   Aktifkan streaming dengan presisi stempel waktu mikrodetik:

   ```
   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
   ```

   Atau aktifkan streaming dengan presisi stempel waktu default (milidetik):

   ```
   aws dynamodb enable-kinesis-streaming-destination \
     --table-name Music \
     --stream-arn arn:aws:kinesis:us-west-2:12345678901:stream/samplestream
   ```

4. Periksa apakah streaming Kinesis aktif pada tabel DynamoDB dengan menggunakan perintah `describe-kinesis-streaming-destination` DynamoDB.

   ```
   aws dynamodb describe-kinesis-streaming-destination --table-name Music
   ```

5. Menulis data ke Daftar Tabel DynamoDB dengan menggunakan perintah `put-item`, seperti yang dijelaskan dalam Panduan Developer DynamoDB.

```
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. Gunakan perintah CLI Kinesis get-records untuk mengambil konten stream Kinesis. Kemudian gunakan potongan kode berikut untuk melakukan deserialisasi konten stream.

```
/**
 * 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. Ikuti petunjuk dalam panduan developer Kinesis Data Streams untuk menciptakan Kinesis data stream bernama `samplestream` menggunakan Java.

   Lihat Pertimbangan manajemen pecahan untuk Kinesis Data Streams sebelum menetapkan jumlah serpihan untuk Kinesis data stream.

2. Gunakan cuplikan kode berikut untuk mengaktifkan Kinesis streaming pada tabel DynamoDB. Secara opsional, aktifkan streaming dengan presisi nilai stempel waktu yang lebih granular (mikrodetik) yang dikembalikan pada setiap catatan.

   Aktifkan streaming dengan presisi stempel waktu mikrodetik:

```
EnableKinesisStreamingConfiguration enableKdsConfig =
 EnableKinesisStreamingConfiguration.builder()

 .approximateCreationDateTimePrecision(ApproximateCreationDateTimePrecision.MICROSECOND)
  .build();

EnableKinesisStreamingDestinationRequest enableKdsRequest =
 EnableKinesisStreamingDestinationRequest.builder()
  .tableName(tableName)
  .streamArn(kdsArn)
  .enableKinesisStreamingConfiguration(enableKdsConfig)
  .build();

EnableKinesisStreamingDestinationResponse enableKdsResponse =
 ddbClient.enableKinesisStreamingDestination(enableKdsRequest);
```

Atau aktifkan streaming dengan presisi stempel waktu default (milidetik):

```
EnableKinesisStreamingDestinationRequest enableKdsRequest =
 EnableKinesisStreamingDestinationRequest.builder()
  .tableName(tableName)
  .streamArn(kdsArn)
  .build();

EnableKinesisStreamingDestinationResponse enableKdsResponse =
 ddbClient.enableKinesisStreamingDestination(enableKdsRequest);
```

3. Ikuti instruksi dalam panduan pengembang Kinesis Data Streams untuk membaca dari aliran data yang dibuat.
4. Gunakan potongan kode berikut untuk melakukan deserialisasi konten stream

```
/**
 * 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());
}
```

Membuat perubahan pada aliran data Amazon Kinesis yang aktif

Bagian ini menjelaskan cara membuat perubahan pada Kinesis Data Streams aktif untuk penyiapan DynamoDB dengan menggunakan konsol, dan API. AWS CLI

AWS Management Console

1. Buka konsol DynamoDB di https://console.aws.amazon.com/dynamodb/
2. Pergi ke meja Anda.
3. Pilih Ekspor dan Aliran.

AWS CLI

1. Hubungi `describe-kinesis-streaming-destination` untuk mengonfirmasi bahwa streaming tersebut`ACTIVE`.
2. Panggil`UpdateKinesisStreamingDestination`, seperti dalam contoh ini:

```
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. Hubungi `describe-kinesis-streaming-destination` untuk mengonfirmasi bahwa streaming tersebut`UPDATING`.
4. Panggil `describe-kinesis-streaming-destination` secara berkala hingga status streaming `ACTIVE` kembali. Biasanya diperlukan waktu hingga 5 menit agar pembaruan presisi stempel waktu diterapkan. Setelah status ini diperbarui, itu menunjukkan bahwa pembaruan selesai dan nilai presisi baru akan diterapkan pada catatan masa depan.
5. Tulis ke meja menggunakan`putItem`.

6. Gunakan `get-records` perintah Kinesis untuk mendapatkan konten aliran.
7. Konfirmasikan bahwa penulisan memiliki presisi yang diinginkan. `ApproximateCreationDateTime`

API Java

1. Berikan cuplikan kode yang membuat `UpdateKinesisStreamingDestination` permintaan dan respons. `UpdateKinesisStreamingDestination`
2. Berikan cuplikan kode yang membuat `DescribeKinesisStreamingDestination` permintaan dan file. `DescribeKinesisStreamingDestination response`
3. Panggil `describe-kinesis-streaming-destination` secara berkala hingga status streaming `ACTIVE` kembali, menunjukkan bahwa pembaruan selesai dan nilai presisi baru akan diterapkan pada catatan masa depan.
4. Lakukan menulis ke meja.
5. Baca dari aliran dan deserialisasi konten streaming.
6. Konfirmasikan bahwa penulisan memiliki presisi yang diinginkan. `ApproximateCreationDateTime`

Menggunakan pecahan dan metrik dengan DynamoDB Streams dan Kinesis Data Streams

Pertimbangan manajemen pecahan untuk Kinesis Data Streams

Kinesis data strea, menghitung throughputnya dalam pecahan. Di Amazon Kinesis Data stream, Anda dapat memilih antara mode sesuai permintaan dan mode yang disediakan untuk aliran data Anda.

Sebaiknya gunakan mode sesuai permintaan untuk Kinesis Data Stream Anda jika beban kerja penulisan DynamoDB Anda sangat bervariasi dan tidak dapat diprediksi. Dengan mode on-demand, tidak ada perencanaan kapasitas yang diperlukan karena Kinesis Data Streams secara otomatis mengelola pecahan untuk menyediakan throughput yang diperlukan.

Untuk beban kerja yang dapat diprediksi, Anda dapat menggunakan mode yang disediakan untuk Kinesis Data Stream Anda. Dengan mode yang disediakan, Anda harus menentukan jumlah pecahan untuk aliran data untuk mengakomodasi catatan pengambilan data perubahan dari DynamoDB.

Untuk menentukan jumlah pecahan yang dibutuhkan aliran data Kinesis untuk mendukung tabel DynamoDB Anda, Anda memerlukan nilai input berikut:

- Ukuran rata-rata catatan tabel DynamoDB Anda dalam byte (`average_record_size_in_bytes`).
- Jumlah maksimum operasi tulis yang tabel DynamoDB Anda akan melakukan per detik. Ini termasuk membuat, menghapus, dan memperbarui operasi yang dilakukan oleh aplikasi Anda, serta operasi yang dihasilkan secara otomatis seperti Time to Live generated delete operations (`write_throughput`).
- Persentase operasi pembaruan dan penimpaan yang Anda lakukan pada tabel Anda, dibandingkan dengan membuat atau menghapus operasi (`percentage_of_updates`). Perlu diingat bahwa operasi pembaruan dan penimpaan mereplikasi gambar lama dan baru dari item yang dimodifikasi ke stream. Ini menghasilkan dua kali ukuran item DynamoDB.

Anda dapat menghitung jumlah pecahan (`number_of_shards`) yang dibutuhkan aliran data Kinesis Anda dengan menggunakan nilai input dalam rumus berikut:

```
number_of_shards = ceiling( max( ((write_throughput * (4+percentage_of_updates) *
 average_record_size_in_bytes) / 1024 / 1024), (write_throughput/1000)), 1)
```

Misalnya, Anda mungkin memiliki throughput maksimum 1040 operasi tulis per detik (`write_throughput`) dengan ukuran catatan rata-rata 800 byte (. `average_record_size_in_bytes`) Jika 25 persen dari operasi penulisan tersebut adalah operasi pembaruan (`percentage_of_updates`), maka Anda akan memerlukan dua pecahan (`number_of_shards`) untuk mengakomodasi throughput streaming DynamoDB Anda:

```
ceiling( max( ((1040 * (4+25/100) * 800)/ 1024 / 1024), (1040/1000)), 1).
```

Pertimbangkan hal berikut sebelum menggunakan rumus untuk menghitung jumlah pecahan yang diperlukan dengan mode yang disediakan untuk aliran data Kinesis:

- Rumus ini membantu memperkirakan jumlah pecahan yang akan diperlukan untuk mengakomodasi catatan data perubahan DynamoDB Anda. Ini tidak mewakili jumlah total pecahan yang dibutuhkan dalam aliran data Kinesis Anda, seperti jumlah pecahan yang diperlukan untuk mendukung konsumen aliran data Kinesis tambahan.
- Anda mungkin masih mengalami pengecualian throughput baca dan tulis dalam mode yang disediakan jika Anda tidak mengonfigurasi aliran data untuk menangani throughput puncak Anda.

Dalam hal ini, Anda harus menskalakan aliran data Anda secara manual untuk mengakomodasi lalu lintas data Anda.

- Rumus ini mempertimbangkan bloat tambahan yang dihasilkan oleh DynamoDB sebelum mengalirkan catatan data log perubahan ke Kinesis Data Stream.

Untuk mempelajari lebih lanjut tentang mode kapasitas pada Kinesis Data Stream lihat Memilih Mode Kapasitas Aliran Data. Untuk mempelajari selengkapnya tentang perbedaan harga antara mode kapasitas yang berbeda, lihat harga Amazon Kinesis Data Streams.

Memantau pengambilan data perubahan dengan Kinesis Data Streams

DynamoDB menyediakan beberapa metrik CloudWatch Amazon untuk membantu Anda memantau replikasi pengambilan data perubahan ke Kinesis. Untuk daftar lengkap CloudWatch metrik, lihatDimensi dan Metrik DynamoDB.

Untuk menentukan apakah streaming Anda memiliki kapasitas yang memadai, sebaiknya pantau item berikut selama pengaktifan streaming dan produksi:

- `ThrottledPutRecordCount`: Jumlah catatan yang dibatasi oleh aliran data Kinesis Anda karena kapasitas aliran data Kinesis tidak mencukupi. Anda mungkin mengalami beberapa throttling selama puncak penggunaan yang luar biasa, namun `ThrottledPutRecordCount` harus tetap serendah mungkin. DynamoDB mencoba lagi mengirimkan catatan yang dibatasi ke aliran data Kinesis, namun hal ini mungkin mengakibatkan latensi replikasi yang lebih tinggi.

  Jika Anda mengalami throttling yang berlebihan dan secara rutin, maka Anda mungkin perlu meningkatkan jumlah serpihan aliran Kinesis secara proporsional dengan throughput tulis yang diamati pada tabel Anda. Untuk mempelajari selengkapnya tentang menentukan ukuran Kinesis data stream, lihat Menentukan Ukuran Awal Kinesis Data Stream.
- `AgeOfOldestUnreplicatedRecord`: Waktu yang berlalu sejak perubahan tingkat item terlama yang belum direplikasi ke aliran data Kinesis muncul di tabel DynamoDB. Dalam operasi normal, `AgeOfOldestUnreplicatedRecord` harus dalam urutan milidetik. Jumlah ini bertambah berdasarkan upaya replikasi yang gagal karena pilihan konfigurasi yang dikontrol pelanggan.

  Jika `AgeOfOldestUnreplicatedRecord` metrik melebihi 168 jam, replikasi perubahan tingkat item dari tabel DynamoDB ke aliran data Kinesis akan dinonaktifkan secara otomatis.

  Contoh konfigurasi yang dikontrol pelanggan yang menyebabkan upaya replikasi gagal adalah kapasitas aliran data Kinesis yang kurang tersedia sehingga menyebabkan throttling berlebihan,

atau pembaruan manual pada kebijakan akses aliran data Kinesis Anda yang mencegah DynamoDB menambahkan data ke aliran data Anda. Untuk menjaga metrik ini serendah mungkin, Anda mungkin perlu memastikan penyediaan kapasitas aliran data Kinesis yang tepat, dan memastikan bahwa izin DynamoDB tidak berubah.

- `FailedToReplicateRecordCount`: Jumlah catatan yang gagal direplikasi DynamoDB ke Kinesis data stream Anda. Item tertentu yang lebih besar dari 34 KB mungkin bertambah besar ukurannya untuk mengubah catatan data yang lebih besar dari batas ukuran item 1 MB Kinesis Data Streams. Perluasan ukuran ini terjadi ketika item yang lebih besar dari 34 KB ini menyertakan sejumlah besar nilai atribut Boolean atau kosong. Nilai atribut Boolean dan kosong disimpan sebagai 1 byte di DynamoDB, namun diperluas hingga 5 byte saat diserialkan menggunakan JSON standar untuk replikasi Kinesis Data Streams. DynamoDB tidak dapat mereplikasi catatan perubahan tersebut ke aliran data Kinesis Anda. DynamoDB akan melewatkan catatan data perubahan ini, dan secara otomatis akan tetap melakukan replikasi atas catatan-catatan berikutnya.

Anda dapat membuat CloudWatch alarm Amazon yang mengirim pesan Amazon Simple Notification Service (Amazon SNS) untuk pemberitahuan ketika salah satu metrik sebelumnya melebihi ambang batas tertentu.

Menggunakan kebijakan IAM untuk Amazon Kinesis Data Streams dan Amazon DynamoDB

Saat pertama kali Anda mengaktifkan Amazon Kinesis Data Streams untuk Amazon DynamoDB, DynamoDB secara otomatis membuat peran terkait layanan (IAM) untuk Anda. AWS Identity and Access Management Peran ini, `AWSServiceRoleForDynamoDBKinesisDataStreamsReplication`, memungkinkan DynamoDB untuk mengelola replikasi perubahan tingkat item ke Kinesis Data Streams atas nama Anda. Jangan hapus peran tertaut layanan ini.

Untuk informasi selengkapnya tentang peran tertaut layanan, lihat Menggunakan peran tertaut layanan di Panduan Pengguna IAM.

> **Note**
>
> DynamoDB tidak mendukung kondisi berbasis tag untuk kebijakan IAM.

Untuk mengaktifkan Amazon Kinesis Data Streams untuk Amazon DynamoDB, Anda harus memiliki izin berikut di tabel:

- `dynamodb:EnableKinesisStreamingDestination`
- `kinesis:ListStreams`
- `kinesis:PutRecords`
- `kinesis:DescribeStream`

Untuk menjelaskan Amazon Kinesis Data Streams untuk Amazon DynamoDB untuk tabel DynamoDB tertentu, Anda harus memiliki izin berikut pada tabel.

- `dynamodb:DescribeKinesisStreamingDestination`
- `kinesis:DescribeStreamSummary`
- `kinesis:DescribeStream`

Untuk menonaktifkan Amazon Kinesis Data Streams untuk Amazon DynamoDB, Anda harus memiliki izin berikut pada tabel.

- `dynamodb:DisableKinesisStreamingDestination`

Untuk memperbarui Amazon Kinesis Data Streams untuk Amazon DynamoDB, Anda harus memiliki izin berikut di atas tabel.

- `dynamodb:UpdateKinesisStreamingDestination`

Contoh berikut menunjukkan cara menggunakan kebijakan IAM untuk memberikan izin untuk Amazon Kinesis Data Streams untuk Amazon DynamoDB.

Contoh: Mengaktifkan Amazon Kinesis Data Streams untuk Amazon DynamoDB

Kebijakan IAM berikut memberikan izin untuk mengaktifkan Amazon Kinesis Data Streams untuk Amazon DynamoDB untuk tabel. `Music` Itu tidak memberikan izin untuk menonaktifkan, memperbarui atau menjelaskan Kinesis Data Streams untuk DynamoDB untuk tabel. `Music`

```
{
    "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-2:12345678901:table/Music"
        }
    ]
}
```

Contoh: Perbarui Amazon Kinesis Data Streams untuk Amazon DynamoDB

Kebijakan IAM berikut memberikan izin untuk memperbarui Amazon Kinesis Data Streams untuk Amazon DynamoDB untuk tabel. `Music` Itu tidak memberikan izin untuk mengaktifkan, menonaktifkan, atau menjelaskan Amazon Kinesis Data Streams untuk Amazon DynamoDB untuk tabel. `Music`

```
{
"Version": "2012-10-17",
    "Statement": [
        {
"Effect": "Allow",
            "Action": [
                "dynamodb:UpdateKinesisStreamingDestination"
            ],
            "Resource": "arn:aws:dynamodb:us-west-2:12345678901:table/Music"
        }
    ]
}
```

Contoh: Menonaktifkan Amazon Kinesis Data Streams untuk Amazon DynamoDB

Kebijakan IAM berikut memberikan izin untuk menonaktifkan Amazon Kinesis Data Streams untuk Amazon DynamoDB untuk tabel. `Music` Itu tidak memberikan izin untuk mengaktifkan, memperbarui, atau menjelaskan Amazon Kinesis Data Streams untuk Amazon DynamoDB untuk tabel. `Music`

```
{
"Version": "2012-10-17",
    "Statement": [
        {
"Effect": "Allow",
            "Action": [
                "dynamodb:DisableKinesisStreamingDestination"
            ],
            "Resource": "arn:aws:dynamodb:us-west-2:12345678901:table/Music"
        }
    ]
}
```

Contoh: Selektif menerapkan izin untuk Amazon Kinesis Data Streams untuk Amazon DynamoDB berdasarkan sumber daya

Kebijakan IAM berikut memberikan izin untuk mengaktifkan dan menjelaskan Amazon Kinesis Data Streams untuk Amazon DynamoDB untuk tabel, dan menolak izin untuk menonaktifkan Amazon Kinesis Data Streams `Music` untuk Amazon DynamoDB untuk tabel. `Orders`

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "dynamodb:EnableKinesisStreamingDestination",
                "dynamodb:DescribeKinesisStreamingDestination"
            ],
            "Resource": "arn:aws:dynamodb:us-west-2:12345678901:table/Music"
        },
        {
            "Effect": "Deny",
            "Action": [
                "dynamodb:DisableKinesisStreamingDestination"
            ],
```

```
            "Resource": "arn:aws:dynamodb:us-west-2:12345678901:table/Orders"
        }

    ]
}
```

Menggunakan peran tertaut layanan untuk Kinesis Data Streams untuk DynamoDB

Amazon Kinesis Data Streams untuk Amazon DynamoDB menggunakan peran terkait layanan (AWS Identity and Access Management IAM). Peran tertaut layanan adalah jenis IAM role unik yang ditautkan langsung ke Kinesis Data Streams untuk DynamoDB. Peran terkait layanan telah ditentukan sebelumnya oleh Kinesis Data Streams untuk DynamoDB dan menyertakan semua izin yang diperlukan layanan untuk memanggil layanan lain atas nama Anda. AWS

Peran tertaut layanan membuat pengaturan Kinesis Data Streams untuk DynamoDB lebih mudah karena Anda tidak perlu menambahkan izin yang diperlukan secara manual. Kinesis Data Streams untuk DynamoDB menentukan izin peran terkait layanannya, dan kecuali ditentukan lain, hanya Kinesis Data Streams untuk DynamoDB yang dapat mengambil peran tersebut. Izin yang ditentukan mencakup kebijakan kepercayaan dan kebijakan izin, serta bahwa kebijakan izin tidak dapat dilampirkan ke entitas IAM lainnya.

Untuk informasi tentang layanan lain yang mendukung peran terkait layanan, lihat Layanan AWS yang Berfungsi dengan IAM dan cari layanan yang memiliki Ya di kolom Peran Terkait Layanan. Pilih Ya dengan tautan untuk melihat dokumentasi peran terkait layanan untuk layanan tersebut.

Izin peran tertaut layanan untuk Kinesis Data Streams untuk DynamoDB

Kinesis Data Streams untuk DynamoDB menggunakan peran terkait layanan bernama. AWSServiceRoleForDynamoDBKinesisDataStreamsReplication Tujuan dari peran tertaut layanan adalah untuk memungkinkan Amazon DynamoDB mengelola replikasi perubahan tingkat item pada Kinesis Data Streams, atas nama Anda.

Peran tertaut layanan `AWSServiceRoleForDynamoDBKinesisDataStreamsReplication` memercayai layanan berikut untuk mengambil peran tersebut:

- `kinesisreplication.dynamodb.amazonaws.com`

Kebijakan izin peran memungkinkan Kinesis Data Streams for DynamoDB menyelesaikan tindakan berikut pada sumber daya yang ditentukan:

- Tindakan: `Put records and describe` pada `Kinesis stream`
- Tindakan: `Generate data keys` aktif `AWS KMS` untuk menempatkan data pada aliran Kinesis yang dienkripsi menggunakan kunci Buatan Pengguna. AWS KMS

Untuk isi dokumen kebijakan yang tepat, lihat Dynamo DBKinesis ReplicationServiceRolePolicy.

Anda harus mengonfigurasi izin untuk mengizinkan entitas IAM (seperti pengguna, grup, atau peran) untuk membuat, mengedit, atau menghapus peran terkait layanan. Untuk informasi selengkapnya, silakan lihat Izin Peran Tertaut Layanan di Panduan Pengguna IAM.

Membuat peran tertaut layanan untuk Kinesis Data Streams untuk DynamoDB

Anda tidak perlu membuat peran terkait layanan secara manual. Saat Anda mengaktifkan Kinesis Data Streams untuk AWS Management Console DynamoDB di AWS CLI, atau AWS API, Kinesis Data Streams untuk DynamoDB akan membuat peran terkait layanan untuk Anda.

Jika Anda menghapus peran terkait layanan ini, dan ingin membuatnya lagi, Anda dapat mengulangi proses yang sama untuk membuat kembali peran tersebut di akun Anda. Saat Anda mengaktifkan Kinesis Data Streams untuk DynamoDB, Kinesis Data Streams untuk DynamoDB menciptakan peran terkait layanan untuk Anda lagi.

Mengedit peran tertaut layanan untuk Kinesis Data Streams untuk DynamoDB

Kinesis Data Streams untuk DynamoDB tidak mengizinkan Anda mengedit peran tertaut layanan `AWSServiceRoleForDynamoDBKinesisDataStreamsReplication`. Setelah membuat peran terkait layanan, Anda tidak dapat mengubah nama peran karena berbagai entitas mungkin mereferensikan peran tersebut. Namun, Anda dapat mengedit penjelasan peran menggunakan IAM. Untuk informasi selengkapnya, lihat Mengedit Peran terkait Layanan di Panduan Pengguna IAM.

Menghapus peran tertaut layanan untuk Kinesis Data Streams untuk DynamoDB

Anda juga dapat menggunakan konsol IAM, AWS CLI atau AWS API untuk menghapus peran terkait layanan secara manual. Untuk melakukannya, Anda harus membersihkan sumber daya untuk peran tertaut layanan terlebih dahulu, lalu Anda dapat menghapusnya secara manual.

> **Note**
>
> Jika layanan Kinesis Data Streams for DynamoDB menggunakan peran tersebut saat Anda mencoba menghapus sumber daya, maka penghapusan mungkin gagal. Jika hal itu terjadi, tunggu beberapa menit dan coba lagi.

Untuk menghapus peran tertaut layanan secara manual menggunakan IAM

Gunakan konsol IAM, the AWS CLI, atau AWS API untuk menghapus peran `AWSServiceRoleForDynamoDBKinesisDataStreamsReplication` terkait layanan. Untuk informasi selengkapnya, lihat Menghapus Peran Terkait Layanan di Panduan Pengguna IAM.

Tangkapan data perubahan DynamoDB Streams

DynamoDB Streams menangkap urutan waktu modifikasi tingkat item di tabel DynamoDB mana pun dan menyimpan informasi ini dalam log hingga 24 jam. Aplikasi dapat mengakses log ini dan melihat item data seperti yang muncul sebelum dan setelah diubah, hampir secara waktu nyata.

Enkripsi saat istirahat mengenkripsi data dalam DynamoDB streams. Untuk informasi selengkapnya, lihat Enkripsi DynamoDB saat diam.

Streami DynamoDB adalah aliran berurutan informasi tentang perubahan item dalam tabel DynamoDB. Saat Anda mengaktifkan aliran pada tabel, DynamoDB menangkap informasi tentang setiap modifikasi pada item data dalam tabel.

Setiap kali aplikasi membuat, memperbarui, atau menghapus item dalam tabel, DynamoDB Streams menulis rekaman aliran dengan atribut kunci primer dari item yang diubah. Catatan stream berisi informasi tentang modifikasi data pada satu item dalam tabel DynamoDB. Anda dapat mengonfigurasi aliran sehingga rekaman aliran menangkap informasi tambahan, seperti gambar "sebelum" dan "sesudah" dari item yang diubah.

DynamoDB Streams membantu memastikan hal berikut:

- Setiap catatan stream muncul tepat sekali dalam stream.
- Untuk setiap item yang diubah dalam tabel DynamoDB, catatan stream muncul dalam urutan yang sama seperti modifikasi aktual untuk item.

DynamoDB Streams menulis catatan aliran hampir secara real-time sehingga Anda dapat membangun aplikasi yang menggunakan aliran ini dan mengambil tindakan berdasarkan kontennya.

Topik

- Titik akhir untuk DynamoDB Streams
- Mengaktifkan aliran
- Membaca dan memproses aliran
- DynamoDB Streams dan Waktu untuk Tayang
- Menggunakan adaptor DynamoDB Streams Kinesis untuk memproses catatan aliran
- API tingkat rendah DynamoDB Streams: Contoh Java
- DynamoDB Streams dan pemicu AWS Lambda
- DynamoDB Streams dan Apache Flink

Titik akhir untuk DynamoDB Streams

AWS mempertahankan endpoint terpisah untuk DynamoDB dan DynamoDB Streams. Untuk bekerja dengan tabel dan indeks basis data, aplikasi Anda harus mengakses titik akhir DynamoDB. Untuk membaca dan memproses catatan DynamoDB Streams, aplikasi Anda harus mengakses titik akhir DynamoDB Streams di Wilayah yang sama.

Konvensi penamaan untuk titik akhir DynamoDB Streams adalah `streams.dynamodb.<region>.amazonaws.com`. Misalnya, jika Anda menggunakan titik akhir `dynamodb.us-west-2.amazonaws.com` untuk mengakses DynamoDB, Anda akan menggunakan titik akhir `streams.dynamodb.us-west-2.amazonaws.com` untuk mengakses DynamoDB Streams.

> **Note**
>
> Untuk daftar lengkap Wilayah dan titik akhir DynamoDB dan DynamoDB Streams, lihat Wilayah dan titik akhir di Referensi Umum AWS.

AWS SDKs Menyediakan klien terpisah untuk DynamoDB dan DynamoDB Streams. Bergantung pada kebutuhan Anda, aplikasi Anda dapat mengakses titik akhir DynamoDB, titik akhir DynamoDB Streams, atau keduanya secara bersamaan. Untuk terhubung ke kedua titik akhir, aplikasi Anda harus membuat instance dua klien—satu untuk DynamoDB dan satu lagi untuk DynamoDB Streams.

Mengaktifkan aliran

Anda dapat mengaktifkan aliran pada tabel baru saat Anda membuatnya menggunakan AWS CLI atau salah satu tabel AWS SDKs. Anda juga dapat mengaktifkan atau menonaktifkan aliran pada tabel yang sudah ada, atau mengubah pengaturan aliran. DynamoDB Streams beroperasi secara asinkron, sehingga tidak ada dampak kinerja pada tabel jika Anda mengaktifkan aliran.

Cara termudah untuk mengelola DynamoDB Streams adalah dengan menggunakan AWS Management Console.

1. Masuk ke AWS Management Console dan buka konsol DynamoDB di. https://console.aws.amazon.com/dynamodb/
2. Pada dasbor konsol DynamoDB, pilih Tabel dan pilih tabel yang ada.
3. Pilih tab Ekspor dan aliran.
4. Di bagian detail aliran DynamoDB, pilih Aktifkan.
5. Pada halaman Aliran DynamoDB, pilih informasi yang akan ditulis ke aliran setiap kali data dalam tabel diubah:

   - Atribut kunci saja — Hanya atribut kunci dari item yang dimodifikasi.
   - Gambar baru — Seluruh item, saat muncul setelah diubah.
   - Gambar lama — Seluruh item, saat muncul sebelum diubah.
   - Gambar baru dan lama — Baik gambar baru dan lama dari item.

   Ketika pengaturan seperti yang Anda inginkan, pilih Aktifkan aliran.
6. (Opsional) Untuk menonaktifkan aliran yang ada, pilih Matikan di bawah detail aliran DynamoDB.

Anda juga dapat menggunakan operasi `CreateTable` atau `UpdateTable` API untuk mengaktifkan atau memodifikasi aliran. Parameter `StreamSpecification` menentukan bagaimana aliran dikonfigurasi:

- `StreamEnabled` — Menentukan apakah stream diaktifkan (`true`) atau dinonaktifkan (`false`) untuk tabel.
- `StreamViewType` — Menentukan informasi yang akan ditulis ke stream setiap kali data dalam tabel dimodifikasi:
  - `KEYS_ONLY` — Hanya atribut kunci dari item yang dimodifikasi.

- `NEW_IMAGE` — Keseluruhan item, seperti yang muncul setelah diubah.
- `OLD_IMAGE` — Keseluruhan item, seperti yang terlihat sebelum diubah.
- `NEW_AND_OLD_IMAGES` — Baik gambar baru dan lama dari item.

Anda dapat mengaktifkan atau menonaktifkan stream kapan saja. Namun, Anda menerima `ValidationException` jika Anda mencoba untuk mengaktifkan stream pada tabel yang sudah memiliki stream. Anda juga menerima `ValidationException` jika Anda mencoba menonaktifkan aliran pada tabel yang tidak memiliki aliran.

Ketika Anda mengatur `StreamEnabled` ke `true`, DynamoDB menciptakan stream baru dengan deskriptor stream unik yang ditugaskan untuk itu. Jika Anda menonaktifkan dan kemudian mengaktifkan kembali stream pada tabel, stream baru dibuat dengan deskriptor stream yang berbeda.

Setiap stream diidentifikasi secara unik oleh Amazon Resource Name (ARN). Berikut ini adalah contoh ARN untuk stream pada tabel DynamoDB bernama `TestTable`.

```
arn:aws:dynamodb:us-west-2:111122223333:table/TestTable/stream/2015-05-11T21:21:33.291
```

Untuk menentukan deskriptor stream terbaru untuk tabel, terbitkan permintaan `DescribeTable` DynamoDB dan cari elemen `LatestStreamArn` dalam respons.

> **Note**
>
> Hal ini tidak mungkin untuk mengedit `StreamViewType` setelah stream telah diatur. Jika Anda perlu melakukan perubahan pada stream setelah pengaturan, Anda harus menonaktifkan stream saat ini dan membuat yang baru.

Membaca dan memproses aliran

Untuk membaca dan memproses aliran, aplikasi Anda harus terhubung ke titik akhir DynamoDB Streams dan menerbitkan permintaan API.

Stream terdiri dari catatan aliran. Setiap catatan stream mewakili modifikasi data tunggal dalam tabel DynamoDB di mana aliran berada. Setiap catatan stream ditugaskan nomor urut, mencerminkan urutan catatan diterbitkan ke aliran.

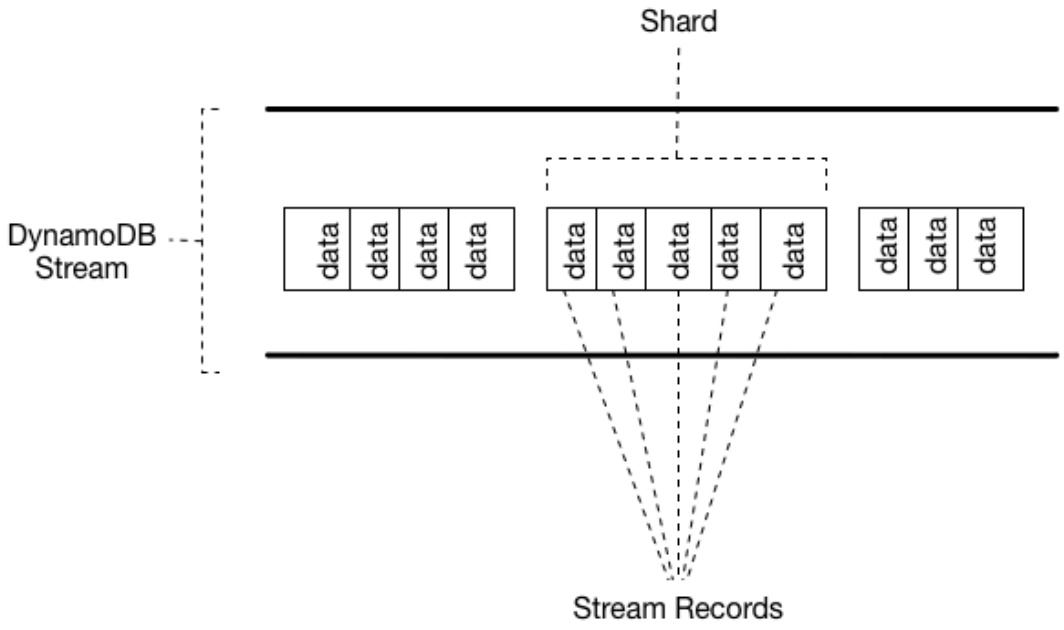
Rekaman streaming disusun ke dalam grup, atau serpihan. Setiap serpihan bertindak sebagai kontainer untuk beberapa catatan aliran, dan berisi informasi yang diperlukan untuk mengakses dan iterasi melalui catatan-catatan ini. Catatan stream dalam serpihan dihapus secara otomatis setelah 24 jam.

Serpihan bersifat fana: Mereka dibuat dan dihapus secara otomatis, sesuai kebutuhan. Setiap serpihan juga dapat dibagi menjadi beberapa serpihan baru; ini juga terjadi secara otomatis. (Pecahan induk juga mungkin hanya memiliki satu pecahan turunan.) Sebuah pecahan mungkin terpecah sebagai respons terhadap aktivitas tulis tingkat tinggi pada tabel induknya, sehingga aplikasi dapat memproses rekaman dari beberapa pecahan secara paralel.

Jika Anda menonaktifkan aliran, semua pecahan yang terbuka akan ditutup. Data dalam aliran akan terus dapat dibaca selama 24 jam.

Karena serpihan memiliki garis keturunan (induk dan turunan), aplikasi harus selalu memproses serpihan induk sebelum memproses serpihan anak. Hal ini membantu memastikan bahwa rekaman aliran juga diproses dalam urutan yang benar. (Jika Anda menggunakan DynamoDB Streams Kinesis Adapter, hal ini akan ditangani untuk Anda. Aplikasi Anda memproses pecahan dan catatan aliran dalam urutan yang benar. Secara otomatis menangani pecahan baru atau yang sudah habis masa berlakunya, selain pecahan yang terpecah saat aplikasi berjalan. Misalnya informasi lebih lanjut, lihat Menggunakan adaptor DynamoDB Streams Kinesis untuk memproses catatan aliran.)

Diagram berikut menunjukkan hubungan antara aliran, serpihan dalam aliran, dan rekaman aliran dalam serpihan.



> **Note**
>
> Jika Anda melakukan operasi `PutItem` atau `UpdateItem` yang tidak mengubah data dalam item, DynamoDB Streams tidak menulis catatan stream untuk operasi itu.

Untuk mengakses aliran dan memproses rekaman aliran di dalamnya, Anda harus melakukan hal berikut:

- Tentukan ARN unik aliran yang ingin Anda akses.
- Tentukan pecahan mana dalam aliran yang berisi rekaman aliran yang Anda minati.
- Akses pecahan dan ambil rekaman aliran yang Anda inginkan.

> **Note**
>
> Maksimum dua proses harus membaca dari pecahan aliran yang sama pada waktu yang sama. Memiliki lebih dari dua pembaca per shard dapat mengakibatkan throttling.

DynamoDB Streams API menyediakan tindakan berikut untuk digunakan oleh program aplikasi:

- `ListStreams` — Mengembalikan daftar deskriptor stream untuk akun dan titik akhir saat ini. Anda secara opsional dapat meminta hanya deskriptor stream untuk nama tabel tertentu.
- `DescribeStream` — Mengembalikan informasi mendetail tentang stream tertentu. Outputnya mencakup daftar pecahan yang terkait dengan aliran, termasuk pecahan IDs.
- `GetShardIterator` — Mengembalikan iterator serpihan, yang menggambarkan lokasi di dalam serpihan. Anda dapat meminta iterator yang menyediakan akses ke titik terlama, titik terbaru, atau titik tertentu dalam serpihan.
- `GetRecords` - Mengembalikan catatan aliran dari dalam serpihan yang diberikan. Anda harus memberikan iterator serpihan yang dikembalikan dari permintaan `GetShardIterator`.

Untuk deskripsi lengkap operasi API ini, termasuk contoh permintaan dan tanggapan, lihat Referensi API Amazon DynamoDB Streams.

Batas retensi data untuk DynamoDB Streams

Semua data dalam DynamoDB Streams tunduk pada masa hidup 24 jam. Anda dapat mengambil dan menganalisis 24 jam terakhir aktivitas untuk setiap tabel yang ditentukan. Namun, data yang lebih lama dari 24 jam rentan terhadap pemangkasan (penghapusan) setiap saat.

Jika Anda menonaktifkan stream pada tabel, data di stream terus dapat dibaca selama 24 jam. Setelah waktu ini, data kedaluwarsa dan catatan stream secara otomatis dihapus. Tidak ada mekanisme untuk menghapus stream yang ada secara manual. Anda harus menunggu hingga batas retensi berakhir (24 jam), dan semua catatan stream akan dihapus.

DynamoDB Streams dan Waktu untuk Tayang

Anda dapat mencadangkan, atau memproses, item yang dihapus oleh Waktu untuk Tayang (TTL) dengan mengaktifkan Amazon DynamoDB Stream pada tabel dan memproses catatan stream dari item yang kedaluwarsa. Untuk informasi selengkapnya, lihat Membaca dan memproses aliran.

Catatan stream berisi bidang identitas pengguna `Records[<index>].userIdentity`.

Item yang dihapus oleh proses Waktu untuk Tayang setelah kedaluwarsa memiliki bidang berikut:

- `Records[<index>].userIdentity.type`

  `"Service"`
- `Records[<index>].userIdentity.principalId`

  `"dynamodb.amazonaws.com"`

> **Note**
>
> Saat Anda menggunakan TTL dalam tabel global, wilayah tempat TTL dilakukan akan memiliki `userIdentity` bidang yang ditetapkan. Bidang ini tidak akan disetel di wilayah lain saat penghapusan direplikasi.

JSON berikut menunjukkan bagian yang relevan dari catatan stream tunggal.

```
"Records": [
    {
```

```
        ...

        "userIdentity": {
            "type": "Service",
            "principalId": "dynamodb.amazonaws.com"
        }

        ...

    }
]
```

Menggunakan DynamoDB Streams dan Lambda untuk mengarsipkan item TTL yang dihapus

Menggabungkan Waktu untuk Tayang (TTL) DynamoDB, DynamoDB Streams, dan AWS Lambda dapat membantu menyederhanakan pengarsipan data, mengurangi biaya penyimpanan DynamoDB, dan mengurangi kompleksitas kode. Menggunakan Lambda sebagai konsumen aliran memberikan banyak keuntungan, terutama pengurangan biaya dibandingkan dengan konsumen lain seperti Kinesis Client Library (KCL). Anda tidak dikenakan biaya untuk panggilan API `GetRecords` di aliran DynamoDB saat menggunakan Lambda untuk mengkonsumsi peristiwa, dan Lambda dapat menyediakan pemfilteran peristiwa dengan mengidentifikasi pola JSON dalam peristiwa streaming. Dengan pemfilteran konten pola peristiwa, Anda dapat menentukan hingga lima filter berbeda untuk mengontrol peristiwa mana yang dikirim ke Lambda untuk diproses. Hal ini membantu mengurangi pemanggilan fungsi Lambda Anda, menyederhanakan kode, dan mengurangi biaya keseluruhan.

Meskipun DynamoDB Streams berisi semua modifikasi data, seperti tindakan `Create`, `Modify`, dan `Remove`, hal ini dapat mengakibatkan pemanggilan yang tidak diinginkan pada fungsi Lambda arsip Anda. Misalnya, Anda memiliki tabel dengan 2 juta modifikasi data per jam yang mengalir ke aliran, namun kurang dari 5 persen di antaranya adalah penghapusan item yang akan kedaluwarsa melalui proses TTL dan perlu diarsipkan. Dengan filter sumber acara Lambda, fungsi Lambda hanya akan dipanggil 100.000 kali per jam. Hasil dari pemfilteran peristiwa adalah Anda hanya dikenakan biaya untuk pemanggilan yang diperlukan, bukan 2 juta pemanggilan yang akan Anda dapatkan tanpa pemfilteran peristiwa.

Pemfilteran peristiwa diterapkan pada pemetaan sumber peristiwa Lambda, yang merupakan sumber daya yang membaca dari peristiwa yang dipilih—aliran DynamoDB—dan memanggil fungsi Lambda. Dalam diagram berikut, Anda dapat melihat bagaimana item Time to Live yang dihapus digunakan oleh fungsi Lambda menggunakan aliran dan filter peristiwa.

Pola filter peristiwa DynamoDB Waktu untuk Tayang

Menambahkan JSON berikut ke kriteria filter pemetaan sumber peristiwa Anda memungkinkan invokasi fungsi Lambda Anda hanya untuk item yang dihapus TTL:

```
{
    "Filters": [
        {
            "Pattern": { "userIdentity": { "type": ["Service"], "principalId":
 ["dynamodb.amazonaws.com"] } }
        }
    ]
}
```

Buat pemetaan sumber AWS Lambda acara

Gunakan cuplikan kode berikut untuk membuat pemetaan sumber peristiwa terfilter yang dapat Anda sambungkan ke aliran DynamoDB tabel. Setiap blok kode menyertakan pola filter peristiwa.

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"]
    }
}
```

Menggunakan adaptor DynamoDB Streams Kinesis untuk memproses catatan aliran

Menggunakan Adaptor Amazon Kinesis adalah cara yang disarankan untuk menggunakan aliran dari Amazon DynamoDB. DynamoDB Streams API sengaja mirip dengan Kinesis Data Streams. Di kedua layanan, aliran data terdiri dari pecahan, yang merupakan wadah untuk rekaman aliran. Kedua layanan APIs berisi`ListStreams`,, `DescribeStream`, dan `GetShardIterator` operasi. (Meskipun tindakan DynamoDB Streams ini serupa dengan tindakan serupa di Kinesis Data Streams, tindakan tersebut tidak 100 persen identik.)

Sebagai pengguna DynamoDB Streams, Anda dapat menggunakan pola desain yang ditemukan dalam KCL untuk memproses serpihan dan rekaman aliran DynamoDB Streams. Untuk melakukan ini, Anda menggunakan Adaptor DynamoDB Streams Kinesis. Adaptor Kinesis mengimplementasikan antarmuka Kinesis Data Streams sehingga KCL dapat digunakan untuk menggunakan dan memproses catatan dari DynamoDB Streams. Untuk petunjuk tentang cara mengatur dan menginstal Adaptor Kinesis DynamoDB Streams, lihat repositori. GitHub

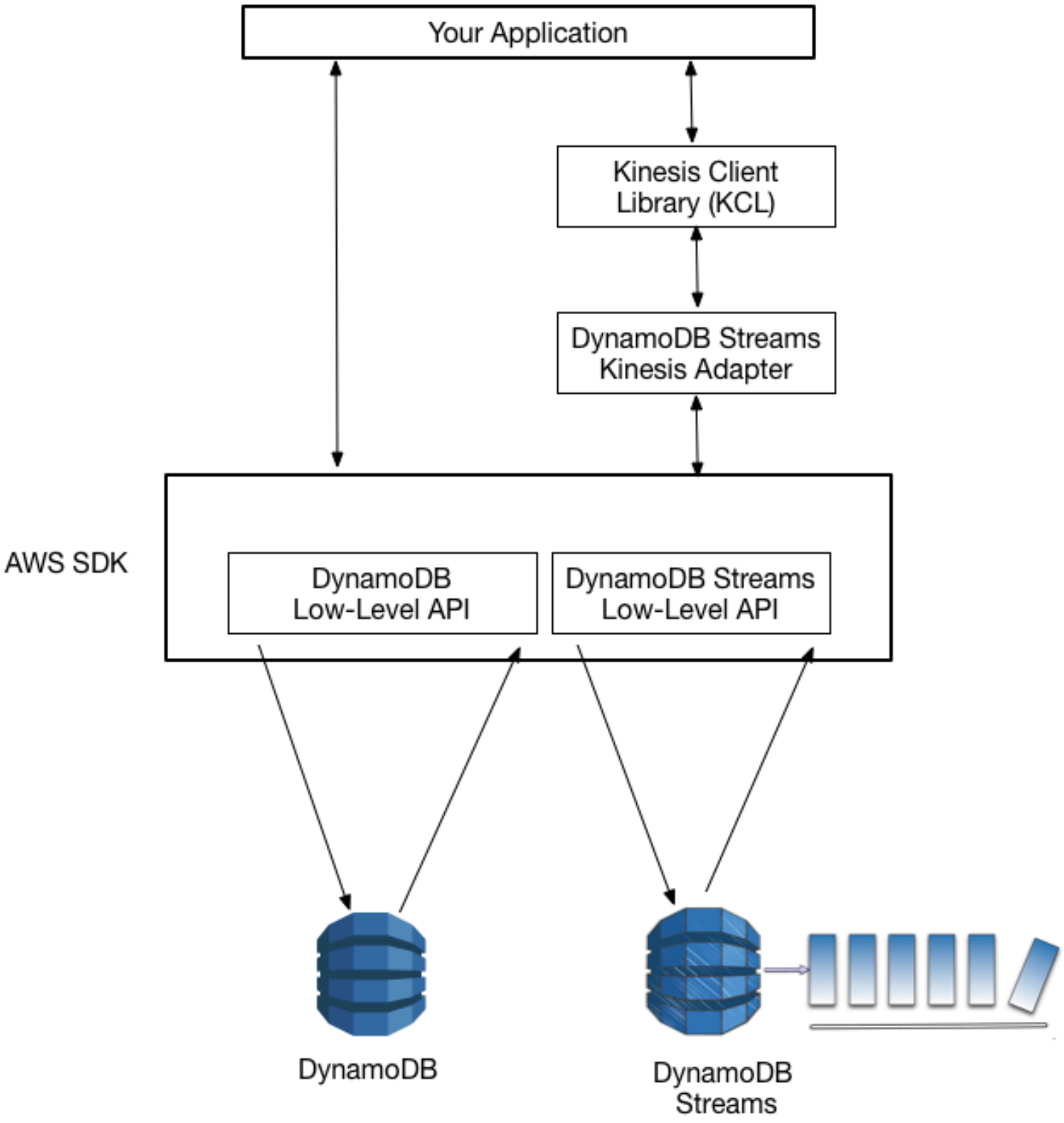
Anda dapat menulis aplikasi untuk Kinesis Data Streams menggunakan Kinesis Client Library (KCL). KCL menyederhanakan pengodean dengan menyediakan abstraksi yang berguna di atas Kinesis Data Streams API tingkat rendah. Untuk informasi selengkapnya tentang KCL, lihat Mengembangkan konsumen menggunakan Kinesis client library dalam Panduan Developer Amazon Kinesis Data Streams.

DynamoDB merekomendasikan penggunaan KCL versi 3.x dengan SDK AWS for Java v2.x. KCL versi 1.x saat ini dengan AWS SDK untuk Java v1.x akan terus didukung sepenuhnya sepanjang siklus hidupnya sebagaimana dimaksud selama periode transisi sesuai dengan dan kebijakan pemeliharaan Alat.AWS SDKs

> **Note**
>
> Perpustakaan Klien Amazon Kinesis (KCL) versi 1.x dan 2.x sudah usang. KCL 1.x akan mencapai end-of-support pada 30 Januari 2026. Kami sangat menyarankan Anda memigrasikan aplikasi KCL Anda menggunakan versi 1.x ke versi KCL terbaru sebelum 30 Januari 2026. Untuk menemukan versi KCL terbaru, lihat halaman Perpustakaan

Klien Amazon Kinesis. Untuk informasi tentang versi KCL terbaru, lihat Menggunakan Perpustakaan Klien Kinesis. Untuk informasi tentang migrasi dari KCL 1.x ke KCL 3.x, lihat. the section called “Migrasi dari KCL 1.x ke KCL 3.x”

Diagram berikut menunjukkan bagaimana perpustakaan ini berinteraksi satu sama lain.



Dengan Adaptor Kinesis DynamoDB Streams, Anda dapat mulai mengembangkan antarmuka KCL, dengan panggilan API diarahkan secara mulus ke titik akhir DynamoDB Streams.

Saat aplikasi Anda dimulai, aplikasi akan memanggil KCL untuk membuat instance pekerja. Anda harus memberi pekerja informasi konfigurasi untuk aplikasi, seperti deskriptor aliran dan AWS kredensil, dan nama kelas prosesor rekaman yang Anda berikan. Saat menjalankan kode di pemroses rekaman, pekerja melakukan tugas-tugas berikut:

- Menghubungkan ke aliran
- Menghitung pecahan dalam aliran
- Mengkoordinasikan asosiasi serpihan dengan pekerja lain (jika ada)
- Membuat instance pemroses rekaman untuk setiap pecahan yang dikelolanya
- Menarik catatan dari aliran
- Mendorong rekaman ke pemroses rekaman yang sesuai
- Catatan yang diproses di pos pemeriksaan
- Menyeimbangkan asosiasi pekerja pecahan ketika jumlah instans pekerja berubah
- Menyeimbangkan asosiasi pekerja pecahan saat pecahan dipisahkan

> **Note**
>
> Untuk deskripsi konsep KCL yang tercantum di sini, lihat Mengembangkan konsumen menggunakan Kinesis client library di Panduan Pengembang Amazon Kinesis Data Streams. Untuk informasi lebih lanjut tentang menggunakan stream dengan lihat AWS Lambda DynamoDB Streams dan pemicu AWS Lambda

Migrasi dari KCL 1.x ke KCL 3.x

Gambaran Umum

Panduan ini memberikan petunjuk untuk memigrasikan aplikasi konsumen Anda dari KCL 1.x ke KCL 3.x. Karena perbedaan arsitektur antara KCL 1.x dan KCL 3.x, migrasi memerlukan pembaruan beberapa komponen untuk memastikan kompatibilitas.

KCL 1.x menggunakan kelas dan antarmuka yang berbeda dibandingkan dengan KCL 3.x. KCL 1.x menggunakan kelas dan antarmuka yang berbeda dibandingkan dengan KCL 3.x. Anda harus memigrasikan prosesor rekaman, pabrik prosesor rekaman, dan kelas pekerja ke format yang kompatibel dengan KCL 3.x terlebih dahulu, dan ikuti langkah-langkah migrasi untuk migrasi KCL 1.x ke KCL 3.x.

Langkah migrasi

Topik

- Langkah 1: Migrasikan prosesor rekaman
- Langkah 2: Migrasikan pabrik prosesor rekaman
- Langkah 3: Migrasikan pekerja
- Langkah 4: Ikhtisar dan rekomendasi konfigurasi KCL 3.x
- Langkah 5: Migrasi dari KCL 2.x ke KCL 3.x

Langkah 1: Migrasikan prosesor rekaman

Contoh berikut menunjukkan prosesor rekaman diimplementasikan untuk adaptor KCL 1.x DynamoDB Streams Kinesis:

```
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();
        }
    }
}
```

Untuk memigrasikan kelas RecordProcessor

1. Ubah antarmuka dari `com.amazonaws.services.kinesis.clientlibrary.interfaces.v2.IRecordProcessor` dan `com.amazonaws.services.kinesis.clientlibrary.interfaces.v2.IShutdownNotificat` menjadi `com.amazonaws.services.dynamodbv2.streamsadapter.processor.DynamoDBStreamsSha` sebagai berikut:

```
// 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. Perbarui pernyataan impor untuk `initialize` dan `processRecords` metode:

```
// 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. Ganti `shutdown` metode dengan metode baru berikut:`leaseLost`,`shardEnded`, dan`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();
        }
    }
```

Berikut ini adalah versi terbaru dari kelas prosesor rekaman:

```
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
 com.amazonaws.services.dynamodbv2.streamsadapter.processor.DynamoDBStreamsShardRecordProcessor
import
 com.amazonaws.services.dynamodbv2.streamsadapter.adapter.DynamoDBStreamsKinesisClientRecord;
import software.amazon.awssdk.services.dynamodb.model.Record;

public class StreamsRecordProcessor implements DynamoDBStreamsShardRecordProcessor {
```

```
    @Override
    public void initialize(InitializationInput initializationInput) {

    }

    @Override
    public void processRecords(ProcessRecordsInput 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 sekarang menggunakan model Record. SDKv2 Dalam SDKv2, `AttributeValue` objek kompleks (`BS`,`NS`, `ML`,`SS`) tidak pernah mengembalikan null. Gunakan`hasBs()`,`hasNs()`,`hasM()`,`hasL()`, `hasSs()` metode untuk memverifikasi apakah nilai-nilai ini ada.

Langkah 2: Migrasikan pabrik prosesor rekaman

Pabrik prosesor rekaman bertanggung jawab untuk membuat prosesor rekaman ketika sewa diperoleh. Berikut ini adalah contoh pabrik KCL 1.x:

```
package com.amazonaws.codesamples;

import com.amazonaws.services.dynamodbv2.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);
    }
}
```

Untuk memigrasikan **`RecordProcessorFactory`**

- Ubah antarmuka yang diimplementasikan dari `com.amazonaws.services.kinesis.clientlibrary.interfaces.v2.IRecordProcessorFa` `kesoftware.amazon.kinesis.processor.ShardRecordProcessorFactory`, sebagai berikut:

  ```
  com.amazonaws.services.kinesis.clientlibrary.interfaces.v2.IRecordProcessorFactory
   to software.amazon.kinesis.processor.ShardRecordProcessorFactory, as follows.

  // 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() {
```

Berikut ini adalah contoh pabrik prosesor rekaman di 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();
    }
}
```

Langkah 3: Migrasikan pekerja

Dalam versi 3.0 dari KCL, kelas baru, yang disebut Scheduler, menggantikan kelas Worker. Berikut ini adalah contoh pekerja KCL 1.x:

```
final KinesisClientLibConfiguration config = new KinesisClientLibConfiguration(...)
final IRecordProcessorFactory recordProcessorFactory = new RecordProcessorFactory();
final Worker worker = StreamsWorkerFactory.createDynamoDbStreamsWorker(
        recordProcessorFactory,
        workerConfig,
        adapterClient,
        amazonDynamoDB,
        amazonCloudWatchClient);
```

Untuk memigrasikan pekerja

1. Ubah `import` pernyataan untuk `Worker` kelas ke pernyataan impor untuk `Scheduler` dan `ConfigsBuilder` kelas.

```
// import com.amazonaws.services.kinesis.clientlibrary.lib.worker.Worker;
import software.amazon.kinesis.coordinator.Scheduler;
import software.amazon.kinesis.common.ConfigsBuilder;
```

2. Impor `StreamTracker` dan ubah impor `StreamsWorkerFactory` ke`StreamsSchedulerFactory`.

```
import software.amazon.kinesis.processor.StreamTracker;
// import com.amazonaws.services.dynamodbv2.streamsadapter.StreamsWorkerFactory;
import com.amazonaws.services.dynamodbv2.streamsadapter.StreamsSchedulerFactory;
```

3. Pilih posisi untuk memulai aplikasi. Bisa jadi `TRIM_HORIZON` atau`LATEST`.

```
import software.amazon.kinesis.common.InitialPositionInStream;
import software.amazon.kinesis.common.InitialPositionInStreamExtended;
```

4. Buat sebuah `StreamTracker` instance.

```
StreamTracker streamTracker = StreamsSchedulerFactory.createSingleStreamTracker(
        streamArn,

 InitialPositionInStreamExtended.newInitialPosition(InitialPositionInStream.TRIM_HORIZON)
);
```

5. Buat `AmazonDynamoDBStreamsAdapterClient` objek.

```
import
 com.amazonaws.services.dynamodbv2.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. Buat `ConfigsBuilder` objek.

```
import software.amazon.kinesis.common.ConfigsBuilder;

...
ConfigsBuilder configsBuilder = new ConfigsBuilder(
                streamTracker,
                applicationName,
                adapterClient,
                dynamoDbAsyncClient,
                cloudWatchAsyncClient,
                UUID.randomUUID().toString(),
                new StreamsRecordProcessorFactory());
```

7. Buat `Scheduler` seperti yang ditunjukkan pada contoh berikut:

```
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);

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

`CLIENT_VERSION_CONFIG_COMPATIBLE_WITH_2X`Pengaturan mempertahankan kompatibilitas antara DynamoDB Streams Kinesis Adapter untuk KCL v3 dan KCL v1, bukan antara KCL v2 dan v3.

Langkah 4: Ikhtisar dan rekomendasi konfigurasi KCL 3.x

Untuk penjelasan rinci tentang konfigurasi yang diperkenalkan setelah KCL 1.x yang relevan di KCL 3.x lihat konfigurasi KCL dan konfigurasi klien migrasi KCL.

Konfigurasi dengan nilai default pembaruan di KCL 3.x

`billingMode`

Dalam KCL versi 1.x, nilai default untuk `billingMode` diatur ke. `PROVISIONED` Namun, dengan KCL versi 3.x, defaultnya `billingMode` adalah `PAY_PER_REQUEST` (mode on-demand). Kami menyarankan Anda menggunakan mode kapasitas sesuai permintaan untuk tabel sewa Anda untuk secara otomatis menyesuaikan kapasitas berdasarkan penggunaan Anda. Untuk panduan tentang penggunaan kapasitas yang disediakan untuk tabel sewa Anda, lihat Praktik terbaik untuk tabel sewa dengan mode kapasitas yang disediakan.

`idleTimeBetweenReadsInMillis`

Dalam KCL versi 1.x, nilai default untuk diatur ke `idleTimeBetweenReadsInMillis` adalah 1.000 (atau 1 detik). KCL versi 3.x menetapkan nilai default `dleTimeBetweenReadsInMillis` untuk i menjadi 1.500 (atau 1,5 detik), tetapi Amazon DynamoDB Streams Kinesis Adapter mengganti nilai default menjadi 1.000 (atau 1 detik).

Konfigurasi baru di KCL 3.x

`leaseAssignmentIntervalMillis`

Konfigurasi ini mendefinisikan interval waktu sebelum pecahan yang baru ditemukan mulai diproses, dan dihitung sebagai 1,5 ×. `leaseAssignmentIntervalMillis` Jika pengaturan ini tidak dikonfigurasi secara eksplisit, interval waktu default menjadi 1,5 ×. `failoverTimeMillis` Memproses pecahan baru melibatkan pemindaian tabel sewa dan menanyakan indeks sekunder global (GSI) pada tabel sewa. Menurunkan `leaseAssignmentIntervalMillis` peningkatan frekuensi operasi pemindaian dan kueri ini, menghasilkan biaya DynamoDB yang lebih tinggi. Kami merekomendasikan pengaturan nilai ini ke 2000 untuk meminimalkan keterlambatan dalam memproses pecahan baru.

`shardConsumerDispatchPollIntervalMillis`

Konfigurasi ini mendefinisikan interval antara jajak pendapat berturut-turut oleh konsumen shard untuk memicu transisi status. Di KCL versi 1.x, perilaku ini dikendalikan oleh `idleTimeInMillis` parameter, yang tidak diekspos sebagai pengaturan yang dapat dikonfigurasi. Dengan KCL versi 3.x, kami sarankan untuk mengatur konfigurasi ini agar sesuai dengan nilai yang digunakan `idleTimeInMillis` dalam pengaturan KCL versi 1.x Anda.

Konfigurasi yang tidak diganti oleh Adaptor Kinesis Amazon DynamoDB Streams

`shardSyncIntervalMillis`

Adaptor Kinesis DynamoDB Streams yang kompatibel dengan KCL versi 1.x secara eksplisit disetel ke 0. `shardSyncIntervalMillis` Sebagai perbandingan, Adaptor Kinesis DynamoDB Streams yang kompatibel dengan KCL versi 3.x tidak lagi menetapkan nilai untuk konfigurasi ini. Untuk mempertahankan perilaku adaptor yang sama seperti versi 1.x, setel nilai konfigurasi ini ke 0.

`leasesRecoveryAuditorExecutionFrequencyMillis`

Adaptor Kinesis DynamoDB Streams yang kompatibel dengan KCL versi 1.x secara eksplisit disetel ke 1000. `leasesRecoveryAuditorExecutionFrequencyMillis` Sebagai perbandingan, Adaptor Kinesis DynamoDB Streams yang kompatibel dengan KCL versi 3.x tidak lagi menetapkan nilai default untuk konfigurasi ini. Untuk mempertahankan perilaku adaptor yang sama seperti versi 1.x, setel nilai konfigurasi ini ke 1000.

Langkah 5: Migrasi dari KCL 2.x ke KCL 3.x

Untuk memastikan kelancaran transisi dan kompatibilitas dengan versi Kinesis Client Library (KCL) terbaru, ikuti langkah 5-8 dalam petunjuk panduan migrasi untuk meningkatkan dari KCL 2.x ke KCL 3.x.

Kembali ke versi KCL sebelumnya

Topik ini menjelaskan cara mengembalikan aplikasi konsumen Anda ke versi KCL sebelumnya. Proses roll-back terdiri dari dua langkah:

1. Jalankan Alat Migrasi KCL.
2. Menerapkan ulang kode versi KCL sebelumnya.

Langkah 1: Jalankan Alat Migrasi KCL

Ketika Anda perlu memutar kembali ke versi KCL sebelumnya, Anda harus menjalankan Alat Migrasi KCL. Alat ini melakukan dua tugas penting:

- Ini menghapus tabel metadata yang disebut tabel metrik pekerja dan indeks sekunder global pada tabel sewa di DynamoDB. Artefak ini dibuat oleh KCL 3.x tetapi tidak diperlukan saat Anda memutar kembali ke versi sebelumnya.
- Itu membuat semua pekerja berjalan dalam mode yang kompatibel dengan KCL 1.x dan mulai menggunakan algoritma load balancing yang digunakan dalam versi KCL sebelumnya. Jika Anda memiliki masalah dengan algoritme penyeimbangan beban baru di KCL 3.x, ini akan segera mengurangi masalah.

> **Important**
>
> Tabel status koordinator di DynamoDB harus ada dan tidak boleh dihapus selama proses migrasi, rollback, dan rollforward.

> **Note**
>
> Sangat penting bahwa semua pekerja dalam aplikasi konsumen Anda menggunakan algoritma load balancing yang sama pada waktu tertentu. Alat Migrasi KCL memastikan bahwa semua pekerja di aplikasi konsumen KCL 3.x Anda beralih ke mode yang kompatibel dengan KCL 1.x sehingga semua pekerja menjalankan algoritma load balancing yang sama selama rollback aplikasi ke versi KCL sebelumnya.

Anda dapat mengunduh Alat Migrasi KCL di direktori skrip repositori GitHubKCL. Jalankan skrip dari pekerja atau host dengan izin yang sesuai untuk menulis ke tabel status koordinator, tabel metrik pekerja, dan tabel sewa. Pastikan izin IAM yang sesuai dikonfigurasi untuk aplikasi konsumen KCL. Jalankan skrip hanya sekali per aplikasi KCL menggunakan perintah yang ditentukan:

```
python3 ./KclMigrationTool.py --region region --mode rollback [--
application_name applicationName] [--lease_table_name leaseTableName]
 [--coordinator_state_table_name coordinatorStateTableName] [--
worker_metrics_table_name workerMetricsTableName]
```

Parameter

`--region`

Ganti *`region`* dengan Anda Wilayah AWS.

`--application_name`

Parameter ini diperlukan jika Anda menggunakan nama default untuk tabel metadata DynamoDB Anda (tabel sewa, tabel status koordinator, dan tabel metrik pekerja). Jika Anda telah menentukan nama kustom untuk tabel ini, Anda dapat menghilangkan parameter ini. Ganti *`applicationName`* dengan nama aplikasi KCL Anda yang sebenarnya. Alat ini menggunakan nama ini untuk mendapatkan nama tabel default jika nama kustom tidak disediakan.

`--lease_table_name`

Parameter ini diperlukan ketika Anda telah menetapkan nama kustom untuk tabel sewa dalam konfigurasi KCL Anda. Jika Anda menggunakan nama tabel default, Anda dapat menghilangkan parameter ini. Ganti *`leaseTableName`* dengan nama tabel kustom yang Anda tentukan untuk tabel sewa Anda.

`--coordinator_state_table_name`

Parameter ini diperlukan ketika Anda telah menetapkan nama kustom untuk tabel status koordinator dalam konfigurasi KCL Anda. Jika Anda menggunakan nama tabel default, Anda dapat menghilangkan parameter ini. Ganti *`coordinatorStateTableName`* dengan nama tabel kustom yang Anda tentukan untuk tabel status koordinator Anda.

`--worker_metrics_table_name`

Parameter ini diperlukan ketika Anda telah menetapkan nama khusus untuk tabel metrik pekerja dalam konfigurasi KCL Anda. Jika Anda menggunakan nama tabel default, Anda dapat menghilangkan parameter ini. Ganti *`workerMetricsTableName`* dengan nama tabel kustom yang Anda tentukan untuk tabel metrik pekerja Anda.

Langkah 2: Menerapkan ulang kode dengan versi KCL sebelumnya

> **Important**
>
> Setiap penyebutan versi 2.x dalam output yang dihasilkan oleh Alat Migrasi KCL harus ditafsirkan sebagai mengacu pada KCL versi 1.x. Menjalankan skrip tidak melakukan rollback lengkap, itu hanya mengalihkan algoritma load balancing ke yang digunakan dalam KCL versi 1.x.

Setelah menjalankan KCL Migration Tool untuk rollback, Anda akan melihat salah satu pesan berikut:

Pesan 1

"Rollback selesai. Aplikasi Anda menjalankan fungsionalitas yang kompatibel dengan 2x. Harap kembalikan ke binari aplikasi Anda sebelumnya dengan menerapkan kode dengan versi KCL Anda sebelumnya.

Tindakan yang diperlukan: Ini berarti pekerja Anda berjalan dalam mode kompatibel KCL 1.x. Menerapkan ulang kode dengan versi KCL sebelumnya ke pekerja Anda.

Pesan 2

> “Rollback selesai. Aplikasi KCL Anda menjalankan fungsionalitas 3x dan akan mengembalikan ke fungsionalitas yang kompatibel 2x. Jika Anda tidak melihat mitigasi setelah periode waktu yang singkat, harap kembalikan ke binari aplikasi Anda sebelumnya dengan menerapkan kode dengan versi KCL Anda sebelumnya.
>
> Tindakan yang diperlukan: Ini berarti pekerja Anda berjalan dalam mode KCL 3.x dan Alat Migrasi KCL mengalihkan semua pekerja ke mode yang kompatibel dengan KCL 1.x. Menerapkan ulang kode dengan versi KCL sebelumnya ke pekerja Anda.

Pesan 3

> “Aplikasi sudah digulung kembali. KCLv3 Sumber daya apa pun yang dapat dihapus dibersihkan untuk menghindari biaya hingga aplikasi dapat digulirkan dengan migrasi.
>
> Tindakan yang diperlukan: Ini berarti bahwa pekerja Anda sudah diputar kembali untuk berjalan dalam mode yang kompatibel dengan KCL 1.x. Menerapkan ulang kode dengan versi KCL sebelumnya ke pekerja Anda.

Gulung maju ke KCL 3.x setelah rollback

Topik ini menjelaskan cara meneruskan aplikasi konsumen Anda ke KCL 3.x setelah rollback. Ketika Anda perlu maju, Anda harus menyelesaikan proses dua langkah:

1. Jalankan Alat Migrasi KCL.
2. Terapkan kode dengan KCL 3.x.

Langkah 1: Jalankan Alat Migrasi KCL

Jalankan KCL Migration Tool dengan perintah berikut untuk maju ke KCL 3.x:

```
python3 ./KclMigrationTool.py --region region --mode rollforward [--
application_name applicationName] [--
coordinator_state_table_name coordinatorStateTableName]
```

Parameter

`--region`

> Ganti *region* dengan Anda Wilayah AWS.

`--application_name`

Parameter ini diperlukan jika Anda menggunakan nama default untuk tabel status koordinator Anda. Jika Anda telah menentukan nama kustom untuk tabel status koordinator, Anda dapat menghilangkan parameter ini. Ganti *`applicationName`* dengan nama aplikasi KCL Anda yang sebenarnya. Alat ini menggunakan nama ini untuk mendapatkan nama tabel default jika nama kustom tidak disediakan.

`--coordinator_state_table_name`

Parameter ini diperlukan ketika Anda telah menetapkan nama kustom untuk tabel status koordinator dalam konfigurasi KCL Anda. Jika Anda menggunakan nama tabel default, Anda dapat menghilangkan parameter ini. Ganti *`coordinatorStateTableName`* dengan nama tabel kustom yang Anda tentukan untuk tabel status koordinator Anda.

Setelah Anda menjalankan alat migrasi dalam mode roll-forward, KCL membuat sumber daya DynamoDB berikut yang diperlukan untuk KCL 3.x:

- Indeks Sekunder Global pada tabel sewa
- Tabel metrik pekerja

Langkah 2: Menyebarkan kode dengan KCL 3.x

Setelah menjalankan Alat Migrasi KCL untuk maju, terapkan kode Anda dengan KCL 3.x ke pekerja Anda. Untuk menyelesaikan migrasi, lihat Langkah 8: Selesaikan migrasi.

Panduan: Adaptor DynamoDB Streams Kinesis

Bagian ini adalah panduan aplikasi Java yang menggunakan Perpustakaan Klien Amazon Kinesis dan Adaptor Amazon DynamoDB Streams Kinesis. Aplikasi ini memperlihatkan contoh replikasi data, di mana aktivitas penulisan dari satu tabel diterapkan ke tabel kedua, dengan konten kedua tabel tetap sinkron. Untuk kode sumber, lihat Program lengkap: Adaptor DynamoDB Streams Kinesis.

Program ini melakukan hal berikut:

1. Menciptakan dua tabel DynamoDB bernama `KCL-Demo-src` dan `KCL-Demo-dst`. Masing-masing tabel ini memiliki stream yang diaktifkan.
2. Menghasilkan aktivitas pembaruan dalam tabel sumber dengan menambahkan, memperbarui, dan menghapus item. Hal ini menyebabkan data akan ditulis ke stream tabel.

3. Membaca catatan dari stream, merekonstruksinya sebagai permintaan DynamoDB, dan menerapkan permintaan ke tabel tujuan.
4. Memindai tabel sumber dan tujuan untuk memastikan bahwa isinya identik.
5. Membersihkan dengan menghapus tabel.

Langkah-langkah ini dijelaskan di bagian berikut, dan aplikasi lengkap ditampilkan di akhir panduan.

Topik

- Langkah 1: Buat tabel DynamoDB
- Langkah 2: Hasilkan aktivitas pembaruan di tabel sumber
- Langkah 3: Proses alirannya
- Langkah 4: Pastikan bahwa kedua tabel memiliki isi identik
- Langkah 5: Bersihkan
- Program lengkap: Adaptor DynamoDB Streams Kinesis

Langkah 1: Buat tabel DynamoDB

Langkah pertama adalah membuat dua tabel DynamoDB—tabel sumber dan tabel tujuan. `StreamViewType` pada aliran tabel sumber adalah `NEW_IMAGE`. Ini berarti bahwa setiap kali item diubah dalam tabel ini, gambar "setelah" item tersebut ditulis ke aliran. Dengan cara ini, aliran melacak semua aktivitas penulisan di tabel.

Contoh berikut menunjukkan kode yang digunakan untuk membuat kedua tabel.

```
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)
```

Langkah 2: Hasilkan aktivitas pembaruan di tabel sumber

Langkah selanjutnya adalah menghasilkan beberapa aktivitas menulis pada tabel sumber. Saat aktivitas ini berlangsung, aliran tabel sumber juga diperbarui hampir secara waktu nyata.

Aplikasi ini mendefinisikan kelas pembantu dengan metode yang memanggil operasi `PutItem`, `UpdateItem`, dan API `DeleteItem` untuk menulis data. Contoh kode berikut menunjukkan bagaimana metode ini digunakan.

```
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");
```

Langkah 3: Proses alirannya

Sekarang program mulai memproses aliran. Adaptor Kinesis DynamoDB Streams bertindak sebagai lapisan transparan antara KCL dan titik akhir DynamoDB Streams, sehingga kode dapat sepenuhnya menggunakan KCL daripada harus melakukan panggilan DynamoDB Streams tingkat rendah. Program ini melakukan tugas-tugas berikut:

- Ini mendefinisikan kelas prosesor catatan, `StreamsRecordProcessor`, dengan metode yang sesuai dengan definisi antarmuka KCL: `initialize`, `processRecords`, dan `shutdown`. Metode `processRecords` berisi logika yang diperlukan untuk membaca dari stream tabel sumber dan menulis ke tabel tujuan.

- Ini mendefinisikan sebuah pabrik kelas untuk kelas prosesor catatan (`StreamsRecordProcessorFactory`). Hal ini diperlukan untuk program Java yang menggunakan KCL.
- Ini menginstanskan KCL `Worker` baru, yang terkait dengan pabrik kelas.
- Ini mematikan `Worker` saat pemrosesan catatan selesai.

Untuk mempelajari lebih lanjut tentang definisi antarmuka KCL, lihat Mengembangkan konsumen menggunakan Kinesis client library di Panduan Pengembang Amazon Kinesis Data Streams.

Contoh kode berikut menunjukkan loop utama dalam `StreamsRecordProcessor`. Pernyataan `case` menentukan apa tindakan apa yang harus dilakukan, berdasarkan `OperationType` yang muncul dalam catatan stream.

```
for (Record record : records) {
    String data = new String(record.getData().array(), Charset.forName("UTF-8"));
    System.out.println(data);
    if (record instanceof RecordAdapter) {
                com.amazonaws.services.dynamodbv2.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();
        }
```

```
    }
}
```

Langkah 4: Pastikan bahwa kedua tabel memiliki isi identik

Pada titik ini, isi sumber dan tujuan tabel tersinkronisasi. Aplikasi menerbitkan permintaan `Scan` terhadap kedua tabel untuk memverifikasi bahwa isinya, pada kenyataannya, identik.

Kelas `DemoHelper` berisi metode `ScanTable` yang memanggil API `Scan` tingkat rendah. Contoh berikut menunjukkan cara penggunaannya.

```
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!");
}
```

Langkah 5: Bersihkan

Demo selesai, sehingga aplikasi menghapus tabel sumber dan tujuan. Lihat contoh kode berikut. Bahkan setelah tabel dihapus, alirannya tetap tersedia hingga 24 jam, setelah itu tabel akan dihapus secara otomatis.

```
dynamoDBClient.deleteTable(new DeleteTableRequest().withTableName(srcTable));
dynamoDBClient.deleteTable(new DeleteTableRequest().withTableName(destTable));
```

Program lengkap: Adaptor DynamoDB Streams Kinesis

Berikut ini adalah program Java lengkap yang melakukan tugas yang dijelaskan dalam Panduan: Adaptor DynamoDB Streams Kinesis. Saat Anda menjalankannya, Anda akan melihat output yang serupa dengan yang seperti berikut.

```
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**
>
> Untuk menjalankan program ini, pastikan bahwa aplikasi klien memiliki akses ke DynamoDB dan CloudWatch Amazon menggunakan kebijakan. Untuk informasi selengkapnya, lihat Kebijakan berbasis identitas untuk DynamoDB.

Kode sumber terdiri dari empat file `.java`:

- `StreamsAdapterDemo.java`
- `StreamsRecordProcessor.java`
- `StreamsRecordProcessorFactory.java`
- `StreamsAdapterDemoHelper.java`

StreamsAdapterDemo.jawa

```
package com.amazonaws.codesamples;

import com.amazonaws.auth.AWSCredentialsProvider;
import com.amazonaws.auth.DefaultAWSCredentialsProviderChain;
import com.amazonaws.regions.Regions;
import com.amazonaws.services.cloudwatch.AmazonCloudWatch;
import com.amazonaws.services.cloudwatch.AmazonCloudWatchClientBuilder;
import com.amazonaws.services.dynamodbv2.AmazonDynamoDB;
import com.amazonaws.services.dynamodbv2.AmazonDynamoDBClientBuilder;
import com.amazonaws.services.dynamodbv2.AmazonDynamoDBStreams;
import com.amazonaws.services.dynamodbv2.AmazonDynamoDBStreamsClientBuilder;
import com.amazonaws.services.dynamodbv2.model.DeleteTableRequest;
import com.amazonaws.services.dynamodbv2.model.DescribeTableResult;
import
 com.amazonaws.services.dynamodbv2.streamsadapter.AmazonDynamoDBStreamsAdapterClient;
import com.amazonaws.services.dynamodbv2.streamsadapter.StreamsWorkerFactory;
import
 com.amazonaws.services.kinesis.clientlibrary.interfaces.v2.IRecordProcessorFactory;
import com.amazonaws.services.kinesis.clientlibrary.lib.worker.InitialPositionInStream;
```

```
import
 com.amazonaws.services.kinesis.clientlibrary.lib.worker.KinesisClientLibConfiguration;
import com.amazonaws.services.kinesis.clientlibrary.lib.worker.Worker;

public class StreamsAdapterDemo {
    private static Worker worker;
    private static KinesisClientLibConfiguration workerConfig;
    private static IRecordProcessorFactory recordProcessorFactory;

    private static AmazonDynamoDB dynamoDBClient;
    private static AmazonCloudWatch cloudWatchClient;
    private static AmazonDynamoDBStreams dynamoDBStreamsClient;
    private static AmazonDynamoDBStreamsAdapterClient adapterClient;

    private static String tablePrefix = "KCL-Demo";
    private static String streamArn;

    private static Regions awsRegion = Regions.US_EAST_2;

    private static AWSCredentialsProvider awsCredentialsProvider =
 DefaultAWSCredentialsProviderChain.getInstance();

    /**
     * @param args
     */
    public static void main(String[] args) throws Exception {
        System.out.println("Starting demo...");

        dynamoDBClient = AmazonDynamoDBClientBuilder.standard()
                .withRegion(awsRegion)
                .build();
        cloudWatchClient = AmazonCloudWatchClientBuilder.standard()
                .withRegion(awsRegion)
                .build();
        dynamoDBStreamsClient = AmazonDynamoDBStreamsClientBuilder.standard()
                .withRegion(awsRegion)
                .build();
        adapterClient = new AmazonDynamoDBStreamsAdapterClient(dynamoDBStreamsClient);
        String srcTable = tablePrefix + "-src";
        String destTable = tablePrefix + "-dest";
        recordProcessorFactory = new StreamsRecordProcessorFactory(dynamoDBClient,
 destTable);

        setUpTables();
```

```
        workerConfig = new KinesisClientLibConfiguration("streams-adapter-demo",
                streamArn,
                awsCredentialsProvider,
                "streams-demo-worker")
                .withMaxRecords(1000)
                .withIdleTimeBetweenReadsInMillis(500)
                .withInitialPositionInStream(InitialPositionInStream.TRIM_HORIZON);

        System.out.println("Creating worker for stream: " + streamArn);
        worker =
StreamsWorkerFactory.createDynamoDbStreamsWorker(recordProcessorFactory, workerConfig,
adapterClient,
                dynamoDBClient, cloudWatchClient);
        System.out.println("Starting worker...");
        Thread t = new Thread(worker);
        t.start();

        Thread.sleep(25000);
        worker.shutdown();
        t.join();

        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!");
        }

        System.out.println("Done.");
        cleanupAndExit(0);
    }

    private static void setUpTables() {
        String srcTable = tablePrefix + "-src";
        String destTable = tablePrefix + "-dest";
        streamArn = StreamsAdapterDemoHelper.createTable(dynamoDBClient, srcTable);
        StreamsAdapterDemoHelper.createTable(dynamoDBClient, destTable);

        awaitTableCreation(srcTable);

        performOps(srcTable);
    }
```

```
    private static void awaitTableCreation(String tableName) {
        Integer retries = 0;
        Boolean created = false;
        while (!created && retries < 100) {
            DescribeTableResult result =
 StreamsAdapterDemoHelper.describeTable(dynamoDBClient, tableName);
            created = result.getTable().getTableStatus().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(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");
    }

    private static void cleanupAndExit(Integer returnValue) {
        String srcTable = tablePrefix + "-src";
        String destTable = tablePrefix + "-dest";
        dynamoDBClient.deleteTable(new DeleteTableRequest().withTableName(srcTable));
        dynamoDBClient.deleteTable(new DeleteTableRequest().withTableName(destTable));
        System.exit(returnValue);
    }
}
```

StreamsRecordProcessor.jawa

```
package com.amazonaws.codesamples;

import com.amazonaws.services.dynamodbv2.AmazonDynamoDB;
import com.amazonaws.services.dynamodbv2.streamsadapter.model.RecordAdapter;
import com.amazonaws.services.kinesis.clientlibrary.interfaces.v2.IRecordProcessor;
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;
import com.amazonaws.services.kinesis.model.Record;

import java.nio.charset.Charset;

public class StreamsRecordProcessor implements IRecordProcessor {
    private Integer checkpointCounter;

    private final AmazonDynamoDB dynamoDBClient;
    private final String tableName;

    public StreamsRecordProcessor(AmazonDynamoDB dynamoDBClient2, String tableName) {
        this.dynamoDBClient = dynamoDBClient2;
        this.tableName = tableName;
    }

    @Override
    public void initialize(InitializationInput initializationInput) {
        checkpointCounter = 0;
    }

    @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) {
                com.amazonaws.services.dynamodbv2.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 {
                    processRecordsInput.getCheckpointer().checkpoint();
                } catch (Exception e) {
                    e.printStackTrace();
                }
            }
        }

    }

    @Override
    public void shutdown(ShutdownInput shutdownInput) {
        if (shutdownInput.getShutdownReason() == ShutdownReason.TERMINATE) {
            try {
                shutdownInput.getCheckpointer().checkpoint();
            } catch (Exception e) {
                e.printStackTrace();
            }
        }

    }
}
```

StreamsRecordProcessorFactory.jawa

```
package com.amazonaws.codesamples;

import com.amazonaws.services.dynamodbv2.AmazonDynamoDB;
import com.amazonaws.services.kinesis.clientlibrary.interfaces.v2.IRecordProcessor;
```

```
import
 com.amazonaws.services.kinesis.clientlibrary.interfaces.v2.IRecordProcessorFactory;

public class StreamsRecordProcessorFactory implements IRecordProcessorFactory {
    private final String tableName;
    private final AmazonDynamoDB dynamoDBClient;

    public StreamsRecordProcessorFactory(AmazonDynamoDB dynamoDBClient, String
 tableName) {
        this.tableName = tableName;
        this.dynamoDBClient = dynamoDBClient;
    }

    @Override
    public IRecordProcessor createProcessor() {
        return new StreamsRecordProcessor(dynamoDBClient, tableName);
    }
}
```

StreamsAdapterDemoHelper.jawa

```
package com.amazonaws.codesamples;

import java.util.ArrayList;
import java.util.HashMap;
import java.util.Map;

import com.amazonaws.services.dynamodbv2.AmazonDynamoDB;
import com.amazonaws.services.dynamodbv2.model.AttributeAction;
import com.amazonaws.services.dynamodbv2.model.AttributeDefinition;
import com.amazonaws.services.dynamodbv2.model.AttributeValue;
import com.amazonaws.services.dynamodbv2.model.AttributeValueUpdate;
import com.amazonaws.services.dynamodbv2.model.CreateTableRequest;
import com.amazonaws.services.dynamodbv2.model.CreateTableResult;
import com.amazonaws.services.dynamodbv2.model.DeleteItemRequest;
import com.amazonaws.services.dynamodbv2.model.DescribeTableRequest;
import com.amazonaws.services.dynamodbv2.model.DescribeTableResult;
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.PutItemRequest;
import com.amazonaws.services.dynamodbv2.model.ResourceInUseException;
```

```
import com.amazonaws.services.dynamodbv2.model.ScanRequest;
import com.amazonaws.services.dynamodbv2.model.ScanResult;
import com.amazonaws.services.dynamodbv2.model.StreamSpecification;
import com.amazonaws.services.dynamodbv2.model.StreamViewType;
import com.amazonaws.services.dynamodbv2.model.UpdateItemRequest;

public class StreamsAdapterDemoHelper {

    /**
     * @return StreamArn
     */
    public static String createTable(AmazonDynamoDB client, String tableName) {
        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)

        try {
            System.out.println("Creating table " + tableName);
            CreateTableResult result = client.createTable(createTableRequest);
            return result.getTableDescription().getLatestStreamArn();
        } catch (ResourceInUseException e) {
            System.out.println("Table already exists.");
            return describeTable(client, tableName).getTable().getLatestStreamArn();
```

```
        }
    }

    public static DescribeTableResult describeTable(AmazonDynamoDB client, String
tableName) {
        return client.describeTable(new
DescribeTableRequest().withTableName(tableName));
    }

    public static ScanResult scanTable(AmazonDynamoDB dynamoDBClient, String tableName)
{
        return dynamoDBClient.scan(new ScanRequest().withTableName(tableName));
    }

    public static void putItem(AmazonDynamoDB dynamoDBClient, String tableName, String
id, String val) {
        java.util.Map<String, AttributeValue> item = new HashMap<String,
AttributeValue>();
        item.put("Id", new AttributeValue().withN(id));
        item.put("attribute-1", new AttributeValue().withS(val));

        PutItemRequest putItemRequest = new
PutItemRequest().withTableName(tableName).withItem(item);
        dynamoDBClient.putItem(putItemRequest);
    }

    public static void putItem(AmazonDynamoDB dynamoDBClient, String tableName,
            java.util.Map<String, AttributeValue> items) {
        PutItemRequest putItemRequest = new
PutItemRequest().withTableName(tableName).withItem(items);
        dynamoDBClient.putItem(putItemRequest);
    }

    public static void updateItem(AmazonDynamoDB dynamoDBClient, String tableName,
String id, String val) {
        java.util.Map<String, AttributeValue> key = new HashMap<String,
AttributeValue>();
        key.put("Id", new AttributeValue().withN(id));

        Map<String, AttributeValueUpdate> attributeUpdates = new HashMap<String,
AttributeValueUpdate>();
        AttributeValueUpdate update = new
AttributeValueUpdate().withAction(AttributeAction.PUT)
                .withValue(new AttributeValue().withS(val));
```

```
        attributeUpdates.put("attribute-2", update);

        UpdateItemRequest updateItemRequest = new
 UpdateItemRequest().withTableName(tableName).withKey(key)
                .withAttributeUpdates(attributeUpdates);
        dynamoDBClient.updateItem(updateItemRequest);
    }

    public static void deleteItem(AmazonDynamoDB dynamoDBClient, String tableName,
 String id) {
        java.util.Map<String, AttributeValue> key = new HashMap<String,
 AttributeValue>();
        key.put("Id", new AttributeValue().withN(id));

        DeleteItemRequest deleteItemRequest = new
 DeleteItemRequest().withTableName(tableName).withKey(key);
        dynamoDBClient.deleteItem(deleteItemRequest);
    }

}
```

API tingkat rendah DynamoDB Streams: Contoh Java

> **Note**
>
> Kode di halaman ini tidak lengkap dan tidak menangani semua skenario untuk menggunakan Amazon DynamoDB Streams. Cara yang disarankan untuk menggunakan catatan aliran dari DynamoDB adalah melalui Adaptor Amazon Kinesis menggunakan Kinesis Client Library (KCL), seperti yang dijelaskan dalam Menggunakan adaptor DynamoDB Streams Kinesis untuk memproses catatan aliran.

Bagian ini berisi program Java yang menunjukkan aksi DynamoDB Streams. Program ini melakukan hal berikut. Program ini melakukan hal berikut:

1. Membuat tabel DynamoDB dengan aliran diaktifkan.
2. Menjelaskan pengaturan aliran untuk tabel ini.
3. Memodifikasi data dalam tabel.
4. Menjelaskan pecahan di aliran.

5. Membaca catatan aliran dari serpihan.
6. Membersihkan.

Saat Anda menjalankan program, Anda akan melihat output seperti berikut.

```
Issuing CreateTable request for TestTableForStreams
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:
 01234567890...,},}
    Shard iterator: EjYFEkX2a26eVTWe...
        ApproximateCreationDateTime: Tue Mar 20 09:50:00 PDT 2018,Keys:
 {Id={N: 1,}},NewImage: {Message={S: New item!,}, Id={N: 1,}},SequenceNumber:
 100000000003218256368,SizeBytes: 24,StreamViewType: NEW_AND_OLD_IMAGES}
         {ApproximateCreationDateTime: Tue Mar 20 09:50:00 PDT 2018,Keys: {Id={N:
 1,}},NewImage: {Message={S: This item has changed,}, Id={N: 1,}},OldImage:
 {Message={S: New item!,}, Id={N: 1,}},SequenceNumber: 200000000003218256412,SizeBytes:
 56,StreamViewType: NEW_AND_OLD_IMAGES}
         {ApproximateCreationDateTime: Tue Mar 20 09:50:00 PDT 2018,Keys: {Id={N:
 1,}},OldImage: {Message={S: This item has changed,}, Id={N: 1,}},SequenceNumber:
 300000000003218256413,SizeBytes: 36,StreamViewType: NEW_AND_OLD_IMAGES}
...
Deleting the table...
Demo complete
```

Example

```
package com.amazon.codesamples;
```

```
import java.util.ArrayList;
import java.util.Arrays;
import java.util.HashMap;
import java.util.List;
import java.util.Map;

import com.amazonaws.AmazonClientException;
import com.amazonaws.auth.DefaultAWSCredentialsProviderChain;
import com.amazonaws.regions.Regions;
import com.amazonaws.services.dynamodbv2.AmazonDynamoDB;
import com.amazonaws.services.dynamodbv2.AmazonDynamoDBClientBuilder;
import com.amazonaws.services.dynamodbv2.AmazonDynamoDBStreams;
import com.amazonaws.services.dynamodbv2.AmazonDynamoDBStreamsClientBuilder;
import com.amazonaws.services.dynamodbv2.model.AttributeAction;
import com.amazonaws.services.dynamodbv2.model.AttributeDefinition;
import com.amazonaws.services.dynamodbv2.model.AttributeValue;
import com.amazonaws.services.dynamodbv2.model.AttributeValueUpdate;
import com.amazonaws.services.dynamodbv2.model.CreateTableRequest;
import com.amazonaws.services.dynamodbv2.model.DescribeStreamRequest;
import com.amazonaws.services.dynamodbv2.model.DescribeStreamResult;
import com.amazonaws.services.dynamodbv2.model.DescribeTableResult;
import com.amazonaws.services.dynamodbv2.model.GetRecordsRequest;
import com.amazonaws.services.dynamodbv2.model.GetRecordsResult;
import com.amazonaws.services.dynamodbv2.model.GetShardIteratorRequest;
import com.amazonaws.services.dynamodbv2.model.GetShardIteratorResult;
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.Record;
import com.amazonaws.services.dynamodbv2.model.Shard;
import com.amazonaws.services.dynamodbv2.model.ShardIteratorType;
import com.amazonaws.services.dynamodbv2.model.StreamSpecification;
import com.amazonaws.services.dynamodbv2.model.StreamViewType;
import com.amazonaws.services.dynamodbv2.util.TableUtils;

public class StreamsLowLevelDemo {

        public static void main(String args[]) throws InterruptedException {

                AmazonDynamoDB dynamoDBClient = AmazonDynamoDBClientBuilder
                                .standard()
                                .withRegion(Regions.US_EAST_2)
                                .withCredentials(new
 DefaultAWSCredentialsProviderChain())
```

```
                                .build();

                AmazonDynamoDBStreams streamsClient =
AmazonDynamoDBStreamsClientBuilder
                                .standard()
                                .withRegion(Regions.US_EAST_2)
                                .withCredentials(new
DefaultAWSCredentialsProviderChain())
                                .build();

                // Create a table, with a stream enabled
                String tableName = "TestTableForStreams";

                ArrayList<AttributeDefinition> attributeDefinitions = new ArrayList<>(
                                Arrays.asList(new AttributeDefinition()
                                                .withAttributeName("Id")
                                                .withAttributeType("N")));

                ArrayList<KeySchemaElement> keySchema = new ArrayList<>(
                                Arrays.asList(new KeySchemaElement()
                                                .withAttributeName("Id")
                                                .withKeyType(KeyType.HASH))); //
Partition key

                StreamSpecification streamSpecification = new StreamSpecification()
                                .withStreamEnabled(true)
                                .withStreamViewType(StreamViewType.NEW_AND_OLD_IMAGES);

                CreateTableRequest createTableRequest = new
CreateTableRequest().withTableName(tableName)

.withKeySchema(keySchema).withAttributeDefinitions(attributeDefinitions)
                                .withProvisionedThroughput(new ProvisionedThroughput()
                                                .withReadCapacityUnits(10L)
                                                .withWriteCapacityUnits(10L))
                                .withStreamSpecification(streamSpecification);

                System.out.println("Issuing CreateTable request for " + tableName);
                dynamoDBClient.createTable(createTableRequest);
                System.out.println("Waiting for " + tableName + " to be created...");

                try {
                        TableUtils.waitUntilActive(dynamoDBClient, tableName);
                } catch (AmazonClientException e) {
```

```
                e.printStackTrace();
            }

            // Print the stream settings for the table
            DescribeTableResult describeTableResult =
dynamoDBClient.describeTable(tableName);
            String streamArn = describeTableResult.getTable().getLatestStreamArn();
            System.out.println("Current stream ARN for " + tableName + ": " +
                               describeTableResult.getTable().getLatestStreamArn());
            StreamSpecification streamSpec =
describeTableResult.getTable().getStreamSpecification();
            System.out.println("Stream enabled: " + streamSpec.getStreamEnabled());
            System.out.println("Update view type: " +
streamSpec.getStreamViewType());
            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
                Map<String, AttributeValue> item = new HashMap<>();
                item.put("Id", new AttributeValue().withN(i.toString()));
                item.put("Message", new AttributeValue().withS("New item!"));
                dynamoDBClient.putItem(tableName, item);

                // Update the item
                Map<String, AttributeValue> key = new HashMap<>();
                key.put("Id", new AttributeValue().withN(i.toString()));
                Map<String, AttributeValueUpdate> attributeUpdates = new
HashMap<>();
                attributeUpdates.put("Message", new AttributeValueUpdate()
                                .withAction(AttributeAction.PUT)
                                .withValue(new AttributeValue()
                                                .withS("This item has
changed")));
                dynamoDBClient.updateItem(tableName, key, attributeUpdates);

                // Delete the item
                dynamoDBClient.deleteItem(tableName, key);
```

```
                }

                // Get all the shard IDs from the stream. Note that DescribeStream
returns
                // the shard IDs one page at a time.
                String lastEvaluatedShardId = null;

                do {
                        DescribeStreamResult describeStreamResult =
streamsClient.describeStream(
                                        new DescribeStreamRequest()
                                                        .withStreamArn(streamArn)

.withExclusiveStartShardId(lastEvaluatedShardId));
                        List<Shard> shards =
describeStreamResult.getStreamDescription().getShards();

                        // Process each shard on this page

                        for (Shard shard : shards) {
                                String shardId = shard.getShardId();
                                System.out.println("Shard: " + shard);

                                // Get an iterator for the current shard

                                GetShardIteratorRequest getShardIteratorRequest = new
GetShardIteratorRequest()
                                                        .withStreamArn(streamArn)
                                                        .withShardId(shardId)

.withShardIteratorType(ShardIteratorType.TRIM_HORIZON);
                                GetShardIteratorResult getShardIteratorResult =
streamsClient

.getShardIterator(getShardIteratorRequest);
                                String currentShardIter =
getShardIteratorResult.getShardIterator();

                                // Shard iterator is not null until the Shard is sealed
(marked as READ_ONLY).
                                // To prevent running the loop until the Shard is
sealed, which will be on
                                // average
```

```
                                // 4 hours, we process only the items that were written
into DynamoDB and then
                                // exit.
                                int processedRecordCount = 0;
                                while (currentShardIter != null && processedRecordCount
< maxItemCount) {
                                        System.out.println("    Shard iterator: " +
currentShardIter.substring(380));

                                        // Use the shard iterator to read the stream
records

                                        GetRecordsResult getRecordsResult =
streamsClient
                                                        .getRecords(new
GetRecordsRequest()

.withShardIterator(currentShardIter));
                                        List<Record> records =
getRecordsResult.getRecords();
                                        for (Record record : records) {
                                                System.out.println("        " +
record.getDynamodb());
                                        }
                                        processedRecordCount += records.size();
                                        currentShardIter =
getRecordsResult.getNextShardIterator();
                                }
                        }

                        // If LastEvaluatedShardId is set, then there is
                        // at least one more page of shard IDs to retrieve
                        lastEvaluatedShardId =
describeStreamResult.getStreamDescription().getLastEvaluatedShardId();

                } while (lastEvaluatedShardId != null);

                // Delete the table
                System.out.println("Deleting the table...");
                dynamoDBClient.deleteTable(tableName);

                System.out.println("Demo complete");

        }
```

```
}
```

DynamoDB Streams dan pemicu AWS Lambda

Amazon DynamoDB terintegrasi AWS Lambda sehingga Anda dapat membuat pemicu —potongan kode yang secara otomatis merespons peristiwa di DynamoDB Streams. Dengan pemicu, Anda dapat membangun aplikasi yang bereaksi terhadap modifikasi data di tabel DynamoDB.

Topik

- Tutorial #1: Menggunakan filter untuk memproses semua peristiwa dengan Amazon AWS Lambda DynamoDB dan menggunakan AWS CLI
- Tutorial #2: Menggunakan filter untuk memproses beberapa peristiwa dengan DynamoDB dan Lambda
- Praktik terbaik menggunakan DynamoDB Streams dengan Lambda

Jika Anda mengaktifkan DynamoDB Streams pada tabel, Anda dapat mengaitkan aliran Amazon Resource Name (ARN) dengan fungsi yang Anda tulis. AWS Lambda Semua tindakan mutasi pada tabel DynamoDB tersebut kemudian dapat ditangkap sebagai item di aliran. Misalnya, Anda dapat menyetel pemicu sehingga ketika item dalam tabel diubah, rekaman baru segera muncul di aliran tabel tersebut.

> **Note**
>
> Jika Anda berlangganan lebih dari dua fungsi Lambda ke satu aliran DynamoDB, pelambatan baca mungkin terjadi.

Layanan AWS Lambda ini melakukan polling aliran untuk catatan baru empat kali per detik. Ketika rekaman aliran baru tersedia, fungsi Lambda Anda dipanggil secara sinkron. Anda dapat berlangganan hingga dua fungsi Lambda ke aliran DynamoDB yang sama. Jika Anda berlangganan lebih dari dua fungsi Lambda ke aliran DynamoDB yang sama, pelambatan baca mungkin terjadi.

Fungsi Lambda dapat mengirimkan pemberitahuan, memulai alur kerja, atau melakukan banyak tindakan lain yang Anda tentukan. Anda dapat menulis fungsi Lambda dengan mudah menyalin setiap catatan aliran ke penyimpanan persisten, seperti Amazon S3 File Gateway (Amazon S3), dan membuat jejak audit permanen dari aktivitas penulisan di tabel Anda. Selain itu, misalkan Anda mempunyai aplikasi game seluler yang menulis ke tabel `GameScores`. Setiap kali atribut `TopScore`

dari tabel `GameScores` diperbarui, catatan stream yang sesuai ditulis ke stream tabel ini. Peristiwa ini kemudian dapat memicu fungsi Lambda yang memposting pesan ucapan selamat di jaringan media sosial. Fungsi ini juga dapat ditulis untuk mengabaikan catatan stream yang tidak memperbarui `GameScores`, atau yang tidak memodifikasi atribut `TopScore`.

Jika fungsi Anda mengembalikan kesalahan, Lambda akan mencoba ulang batch tersebut hingga berhasil diproses atau datanya habis masa berlakunya. Anda juga dapat mengonfigurasi Lambda untuk mencoba ulang dengan batch yang lebih kecil, membatasi jumlah percobaan ulang, membuang catatan setelah terlalu lama, dan opsi lainnya.

Sebagai praktik terbaik kinerja, fungsi Lambda harus berumur pendek. Untuk menghindari penundaan pemrosesan yang tidak perlu, logika yang rumit juga tidak boleh dijalankan. Khususnya untuk aliran kecepatan tinggi, lebih baik memicu alur kerja fungsi langkah pasca-pemrosesan asinkron daripada Lambdas yang berjalan lama secara sinkron.

Anda tidak dapat menggunakan pemicu Lambda yang sama di berbagai AWS akun. Baik tabel DynamoDB dan fungsi Lambda harus milik akun yang sama. AWS

Untuk informasi selengkapnya AWS Lambda, lihat Panduan AWS Lambda Pengembang.

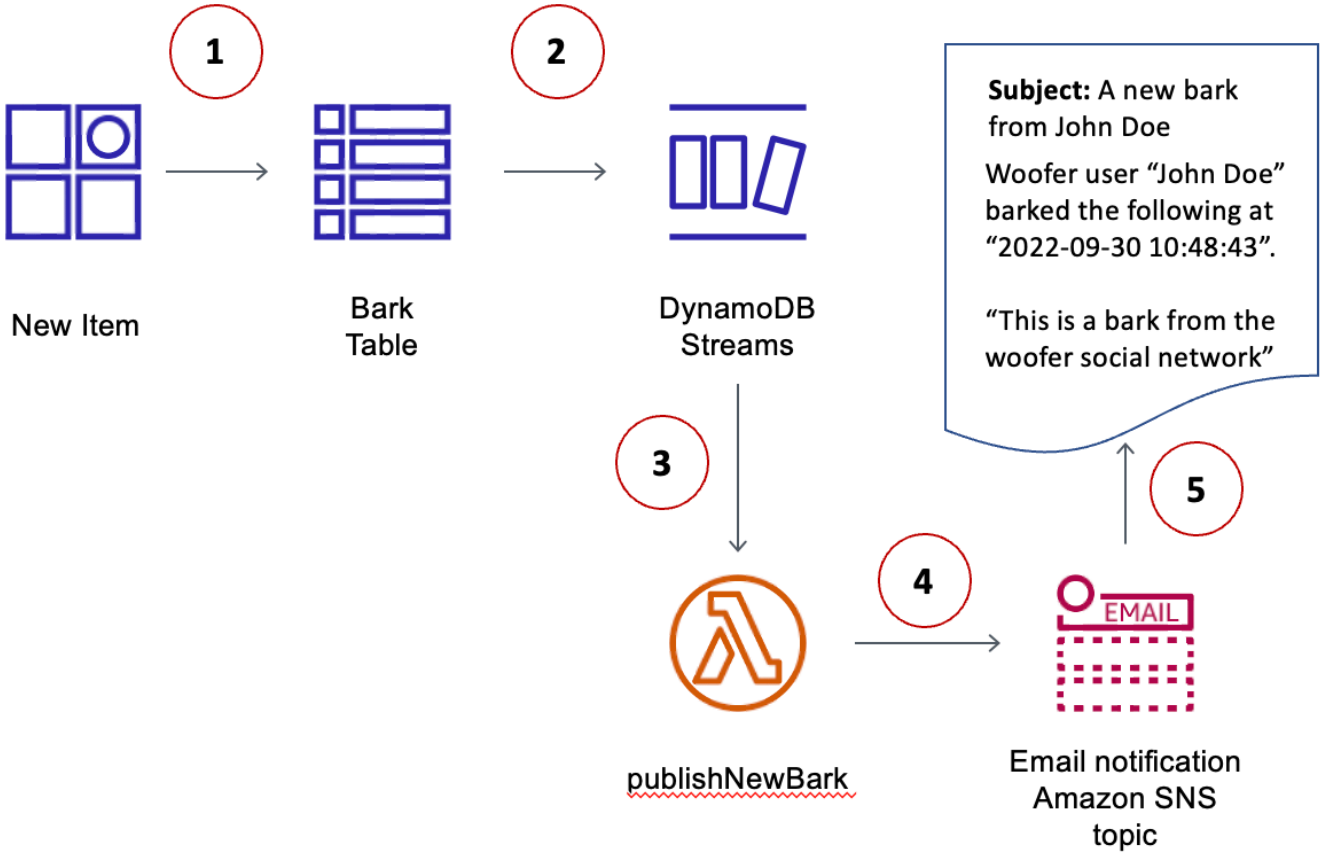
Tutorial #1: Menggunakan filter untuk memproses semua peristiwa dengan Amazon AWS Lambda DynamoDB dan menggunakan AWS CLI

Dalam tutorial ini, Anda akan membuat AWS Lambda pemicu untuk memproses aliran dari tabel DynamoDB.

Topik

- Langkah 1: Buat tabel DynamoDB dengan aliran diaktifkan
- Langkah 2: Buat peran eksekusi Lambda
- Langkah 3: Buat Topik Amazon SNS
- Langkah 4: Buat dan uji fungsi Lambda
- Langkah 5: Buat dan uji pemicu

Skenario untuk tutorial ini adalah Woofer, sebuah jejaring sosial sederhana. Pengguna Woofer berkomunikasi menggunakan bark (pesan teks singkat) yang dikirim ke pengguna Woofer lainnya. Diagram berikut menunjukkan komponen dan alur kerja untuk aplikasi ini.



1. Seorang pengguna menulis item ke tabel DynamoDB (`BarkTable`). Setiap item dalam tabel mewakili bark.
2. Sebuah catatan stream baru ditulis untuk mencerminkan bahwa item baru telah ditambahkan ke `BarkTable`.
3. Rekaman aliran baru memicu AWS Lambda fungsi (`publishNewBark`).
4. Jika catatan stream menunjukkan bahwa item baru ditambahkan ke `BarkTable`, fungsi Lambda membaca data dari catatan stream dan menerbitkan pesan ke topik di Amazon Simple Notification Service (Amazon SNS).
5. Pesan diterima oleh pelanggan untuk topik Amazon SNS. (Dalam tutorial ini, satu-satunya pelanggan adalah alamat email.)

Sebelum Anda Memulai

Tutorial ini menggunakan AWS Command Line Interface AWS CLI. Jika Anda belum melakukannya, ikuti petunjuk di Panduan Pengguna AWS Command Line Interface untuk menginstal dan mengkonfigurasi AWS CLI.

Langkah 1: Buat tabel DynamoDB dengan aliran diaktifkan

Pada langkah ini, Anda membuat tabel DynamoDB (`BarkTable`) untuk menyimpan semua bark dari pengguna Woofer. Kunci primer terdiri dari `Username` (kunci partisi) dan `Timestamp` (kunci urutan). Kedua atribut ini berjenis string.

`BarkTable` memiliki aliran yang diaktifkan. Kemudian dalam tutorial ini, Anda membuat pemicu dengan mengaitkan AWS Lambda fungsi dengan aliran.

1. Masukkan perintah berikut untuk membuat tabel.

```
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. Dalam output, cari `LatestStreamArn`.

```
...
"LatestStreamArn": "arn:aws:dynamodb:region:accountID:table/BarkTable/
stream/timestamp
...
```

   Buat catatan tentang *`region`* dan *`accountID`*, karena Anda membutuhkannya untuk langkah-langkah lain dalam tutorial ini.

Langkah 2: Buat peran eksekusi Lambda

Pada langkah ini, Anda membuat peran AWS Identity and Access Management (IAM) (`WooferLambdaRole`) dan menetapkan izin untuk itu. Peran ini digunakan oleh fungsi Lambda yang Anda buat di Langkah 4: Buat dan uji fungsi Lambda.

Anda juga membuat kebijakan untuk peran. Kebijakan ini berisi semua izin yang dibutuhkan fungsi Lambda pada saat waktu aktif.

1. Buat file bernama `trust-relationship.json` dengan isi berikut ini.

```
{
   "Version": "2012-10-17",
   "Statement": [
     {
       "Effect": "Allow",
       "Principal": {
         "Service": "lambda.amazonaws.com"
       },
       "Action": "sts:AssumeRole"
     }
   ]
 }
```

2. Masukkan perintah berikut untuk membuat `WooferLambdaRole`.

```
aws iam create-role --role-name WooferLambdaRole \
    --path "/service-role/" \
    --assume-role-policy-document file://trust-relationship.json
```

3. Buat file bernama `role-policy.json` dengan isi berikut ini. (Ganti *`region`* dan *`accountID`* dengan AWS Wilayah dan ID akun Anda.)

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "logs:CreateLogGroup",
                "logs:CreateLogStream",
                "logs:PutLogEvents"
            ],
            "Resource": "arn:aws:logs:region:accountID:*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "dynamodb:DescribeStream",
                "dynamodb:GetRecords",
                "dynamodb:GetShardIterator",
                "dynamodb:ListStreams"
            ],
```

```
            "Resource": "arn:aws:dynamodb:region:accountID:table/BarkTable/stream/
*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "sns:Publish"
            ],
            "Resource": [
                "*"
            ]
        }
    ]
}
```

Kebijakan ini memiliki empat pernyataan yang memungkinkan `WooferLambdaRole` untuk melakukan hal berikut:

- Menjalankan fungsi Lambda (`publishNewBark`). Anda membuat fungsi nanti dalam tutorial ini.
- Akses CloudWatch Log Amazon. Fungsi Lambda menulis diagnostik ke CloudWatch Log saat runtime.
- Membaca data dari stream DynamoDB untuk `BarkTable`.
- Memublikasikan pesan ke Amazon SNS.

4. Masukkan perintah berikut untuk melampirkan kebijakan ke `WooferLambdaRole`.

```
aws iam put-role-policy --role-name WooferLambdaRole \
    --policy-name WooferLambdaRolePolicy \
    --policy-document file://role-policy.json
```

Langkah 3: Buat Topik Amazon SNS

Pada langkah ini, Anda membuat topik Amazon SNS (`wooferTopic`) dan mendaftarkan langganan alamat email untuk itu. Fungsi Lambda Anda menggunakan topik ini untuk memublikasikan bark baru dari pengguna Woofer.

1. Masukkan perintah berikut untuk membuat topik Amazon SNS.

```
aws sns create-topic --name wooferTopic
```

2. Masukkan perintah berikut untuk mendaftarkan langganan alamat email ke `wooferTopic`. (Ganti *`region`* dan *`accountID`* dengan Wilayah AWS dan ID akun Anda, dan ganti *`example@example.com`* dengan alamat email yang valid.)

```
aws sns subscribe \
    --topic-arn arn:aws:sns:region:accountID:wooferTopic \
    --protocol email \
    --notification-endpoint example@example.com
```

3. Amazon SNS mengirimkan pesan konfirmasi ke alamat email Anda. Pilih tautan Konfirmasi langganan di pesan tersebut untuk menyelesaikan proses berlangganan.

Langkah 4: Buat dan uji fungsi Lambda

Pada langkah ini, Anda membuat AWS Lambda fungsi (`publishNewBark`) untuk memproses catatan aliran dari`BarkTable`.

Fungsi `publishNewBark` hanya memproses hanya peristiwa stream yang sesuai dengan item baru di `BarkTable`. Fungsi membaca data dari peristiwa tersebut, dan kemudian memanggil Amazon SNS untuk memublikasikannya.

1. Buat file bernama `publishNewBark.js` dengan isi berikut ini. Ganti *`region`* dan *`accountID`* dengan AWS Wilayah dan ID akun Anda.

```
'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. Buat file zip untuk menampung `publishNewBark.js`. Jika Anda memiliki utilitas baris perintah zip, Anda dapat memasukkan perintah berikut untuk melakukan hal ini.

```
zip publishNewBark.zip publishNewBark.js
```

3. Ketika Anda membuat fungsi Lambda, Anda menentukan Amazon Resource Name (ARN) untuk `WooferLambdaRole`, yang Anda buat di Langkah 2: Buat peran eksekusi Lambda. Masukkan perintah berikut untuk mengambil ARN ini.

```
aws iam get-role --role-name WooferLambdaRole
```

Dalam output, cari ARN untuk `WooferLambdaRole`.

```
...
"Arn": "arn:aws:iam::region:role/service-role/WooferLambdaRole"
...
```

Masukkan perintah berikut untuk membuat fungsi Lambda. Ganti *roleARN* dengan ARN untuk. `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. Sekarang uji `publishNewBark` untuk memverifikasi bahwa ia bekerja. Untuk melakukannya, Anda memberikan input yang menyerupai catatan nyata dari DynamoDB Streams.

   Buat file bernama `payload.json` dengan isi berikut ini. Ganti *region* dan *accountID* dengan ID akun Wilayah AWS dan Anda.

```
{
    "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": "130216000000000001596893679",
                "SizeBytes": 112,
                "StreamViewType": "NEW_IMAGE"
            },
            "eventSourceARN": "arn:aws:dynamodb:region:account ID:table/BarkTable/
stream/2016-11-16T20:42:48.104"
        }
    ]
}
```

Masukkan perintah berikut untuk menguji fungsi `publishNewBark`.

```
aws lambda invoke --function-name publishNewBark --payload file://payload.json --
cli-binary-format raw-in-base64-out output.txt
```

Jika tes berhasil, Anda akan melihat output sebagai berikut.

```
{
    "StatusCode": 200,
    "ExecutedVersion": "$LATEST"
}
```

Selain itu, file `output.txt` akan berisi teks berikut.

```
"Successfully processed 1 records."
```

Anda juga akan menerima pesan email baru dalam beberapa menit.

> **Note**
>
> AWS Lambda menulis informasi diagnostik ke Amazon CloudWatch Logs. Jika Anda mengalami kesalahan dengan fungsi Lambda Anda, Anda dapat menggunakan diagnostik ini untuk tujuan pemecahan masalah:
>
> 1. Buka CloudWatch konsol di https://console.aws.amazon.com/cloudwatch/.
> 2. Pilih Log di panel navigasi.
> 3. Pilih grup log berikut ini: `/aws/lambda/publishNewBark`
> 4. Pilih stream log terbaru untuk melihat output (dan kesalahan) dari fungsi.

Langkah 5: Buat dan uji pemicu

Dalam Langkah 4: Buat dan uji fungsi Lambda, Anda menguji fungsi Lambda untuk memastikan bahwa itu berjalan dengan benar. Pada langkah ini, Anda membuat pemicu dengan mengaitkan fungsi Lambda (`publishNewBark`) dengan sumber peristiwa (aliran `BarkTable`).

1. Saat Anda membuat pemicu, Anda perlu menentukan ARN untuk stream `BarkTable`. Masukkan perintah berikut untuk mengambil ARN ini.

   ```
   aws dynamodb describe-table --table-name BarkTable
   ```

   Dalam output, cari `LatestStreamArn`.

   ```
   ...
    "LatestStreamArn": "arn:aws:dynamodb:region:accountID:table/BarkTable/
   stream/timestamp
   ...
   ```

2. Masukkan perintah berikut untuk membuat pemicu. Ganti *`streamARN`* dengan ARN stream yang sebenarnya.

   ```
   aws lambda create-event-source-mapping \
       --region region \
       --function-name publishNewBark \
       --event-source streamARN  \
       --batch-size 1 \
       --starting-position TRIM_HORIZON
   ```

3. Uji pemicu. Masukkan perintah berikut untuk menambahkan item ke `BarkTable`.

   ```
   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"}
   ```

   Anda akan menerima pesan email baru dalam beberapa menit.

4. Buka konsol DynamoDB dan tambahkan beberapa item lagi ke `BarkTable`. Anda harus menentukan nilai untuk atribut `Username` dan `Timestamp`. (Anda juga harus menentukan nilai untuk `Message`, meskipun tidak diperlukan.) Anda akan menerima pesan email baru untuk setiap item yang ditambahkan ke `BarkTable`.

Fungsi Lambda memproses hanya item baru yang Anda tambahkan ke `BarkTable`. Jika Anda memperbarui atau menghapus item dalam tabel, fungsi tidak melakukan apa pun.

> **Note**
>
> AWS Lambda menulis informasi diagnostik ke Amazon CloudWatch Logs. Jika Anda mengalami kesalahan dengan fungsi Lambda Anda, Anda dapat menggunakan diagnostik ini untuk tujuan pemecahan masalah.
>
> 1. Buka CloudWatch konsol di https://console.aws.amazon.com/cloudwatch/.
> 2. Pilih Log di panel navigasi.
> 3. Pilih grup log berikut ini: `/aws/lambda/publishNewBark`
> 4. Pilih stream log terbaru untuk melihat output (dan kesalahan) dari fungsi.

Tutorial #2: Menggunakan filter untuk memproses beberapa peristiwa dengan DynamoDB dan Lambda

Dalam tutorial ini, Anda akan membuat AWS Lambda pemicu untuk memproses hanya beberapa peristiwa dalam aliran dari tabel DynamoDB.

Topik

- Menyatukan semuanya - AWS CloudFormation
- Menyatukan semuanya - CDK

Dengan pemfilteran peristiwa Lambda, Anda dapat menggunakan ekspresi filter untuk mengontrol peristiwa mana yang dikirim Lambda ke fungsi Anda untuk diproses. Anda dapat mengonfigurasi hingga 5 filter berbeda per aliran DynamoDB. Jika Anda menggunakan jendela batching, Lambda menerapkan kriteria filter untuk setiap acara baru untuk melihat apakah itu harus disertakan dalam batch saat ini.

Filter diterapkan melalui struktur yang disebut `FilterCriteria`. 3 atribut utama `FilterCriteria` adalah`metadata properties`, `data properties`, dan `filter patterns`.

Berikut adalah contoh struktur dari acara DynamoDB Streams:

```
{
```

```
  "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` adalah bidang objek peristiwa. Dalam kasus DynamoDB Streams, `metadata properties` adalah bidang seperti dynamodb atau `eventName`.

`data properties` adalah bidang badan peristiwa. Untuk memfilter `data properties`, pastikan untuk memasukkannya ke `FilterCriteria` dalam kunci yang tepat. Untuk sumber peristiwa DynamoDB, kunci data adalah `NewImage` atau `OldImage`.

Akhirnya, aturan filter akan menentukan ekspresi filter yang ingin Anda terapkan ke properti tertentu. Berikut ini adalah beberapa contohnya:

Operator perbandingan	Contoh	Sintaks aturan (Sebagian)
Null	Jenis Produk adalah null	`{ "product_type": { "S": null } }`
Kosong	Nama produk kosong	`{ "product_name": { "S": [ ""] } }`
Setara	Negara bagian sama dengan Florida	`{ "state": { "S": ["FL"] } }`
Dan	Negara bagian produk sama dengan Florida dan kategori produk Cokelat	`{ "state": { "S": ["FL"] } , "category": { "S": [ "CHOCOLATE"] } }`
Atau	Negara bagian produk adalah Florida atau California	`{ "state": { "S": ["FL","CA"] } }`
Bukan	Negara bagian produk bukan Florida	`{"state": {"S": [{"anything-but": ["FL"]}]}}`
Exists	Produk Rumahan ada	`{"homemade": {"S": [{"exists": true}]}}`
Tidak ada	Produk Rumahan tidak ada	`{"homemade": {"S": [{"exists": false}]}}`
Dimulai dengan	PK dimulai dengan PERUSAHAAN	`{"PK": {"S": [{"prefix": "COMPANY"}]}}`

Anda dapat menentukan hingga 5 pola penyaringan peristiwa untuk fungsi Lambda. Perhatikan bahwa masing-masing dari 5 peristiwa tersebut akan dievaluasi sebagai OR logis. Jadi jika Anda mengkonfigurasi dua filter bernama `Filter_One` dan`Filter_Two`, fungsi Lambda akan mengeksekusi `Filter_One` OR `Filter_Two`.

> **Note**
>
> Di halaman pemfilteran acara Lambda ada beberapa opsi untuk memfilter dan membandingkan nilai numerik, namun dalam kasus peristiwa filter DynamoDB itu tidak berlaku karena angka di DynamoDB disimpan sebagai string. Misalnya `"quantity": { "N": "50" }`, kita tahu itu nomor karena properti `"N"`.

Menyatukan semuanya - AWS CloudFormation

Untuk menampilkan fungsionalitas pemfilteran acara dalam praktiknya, berikut adalah contoh CloudFormation template. Templat ini akan menghasilkan tabel DynamoDB Sederhana dengan PK Kunci Partisi dan SK Kunci Urutan dengan Amazon DynamoDB Streams diaktifkan. Ini akan membuat fungsi lambda dan peran Eksekusi Lambda sederhana yang memungkinkan penulisan log ke Amazon Cloudwatch, dan membaca peristiwa dari Amazon DynamoDB Stream. Ini juga akan menambahkan pemetaan sumber peristiwa antara DynamoDB Streams dan fungsi Lambda, sehingga fungsi tersebut dapat dijalankan setiap kali ada kejadian di Amazon DynamoDB Stream.

```
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
```

Setelah Anda menerapkan templat pembentukan cloud ini, Anda dapat memasukkan Item Amazon DynamoDB berikut:

```
{
 "PK": "COMPANY#1000",
 "SK": "PRODUCT#CHOCOLATE#DARK",
 "company_id": "1000",
 "type": "",
 "state": "FL",
 "stores": 5,
 "price": 15,
 "quantity": 50,
 "fabric": "Florida Chocolates"
}
```

Berkat fungsi lambda sederhana yang disertakan sebaris dalam template pembentukan cloud ini, Anda akan melihat peristiwa di grup CloudWatch log Amazon untuk fungsi lambda sebagai berikut:

```
{
  "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"
}
```


Contoh Filter

- Hanya produk yang cocok dengan status tertentu

Contoh ini memodifikasi CloudFormation template untuk menyertakan filter untuk mencocokkan semua produk yang berasal dari Florida, dengan singkatan “FL”.

```
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
```

Setelah Anda menerapkan kembali tumpukan, Anda dapat menambahkan item DynamoDB berikut ke tabel. Perhatikan bahwa itu tidak akan muncul di log fungsi Lambda, karena produk dalam contoh ini berasal dari California.

```
{
 "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": ""
}
```

- Hanya item yang dimulai dengan beberapa nilai di PK dan SK

Contoh ini memodifikasi CloudFormation template untuk menyertakan kondisi berikut:

```
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
```

Perhatikan kondisi AND mengharuskan kondisi berada di dalam pola, di mana PK dan SK Kunci berada dalam ekspresi yang sama dipisahkan oleh koma.

Baik mulai dengan beberapa nilai pada PK dan SK atau dari keadaan tertentu.

Contoh ini memodifikasi CloudFormation template untuk menyertakan kondisi berikut:

```
 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
```

Perhatikan kondisi OR ditambahkan dengan memperkenalkan pola baru di bagian filter.

Menyatukan semuanya - CDK

Contoh templat pembentukan proyek CDK berikut berjalan melalui fungsionalitas penyaringan acara. Sebelum bekerja dengan proyek CDK ini, Anda perlu menginstal prasyarat termasuk menjalankan skrip persiapan.

Buat proyek CDK

Pertama buat AWS CDK proyek baru, dengan memanggil `cdk init` dalam direktori kosong.

```
mkdir ddb_filters
cd ddb_filters
cdk init app --language python
```

Perintah `cdk init` menggunakan nama folder proyek untuk memberi nama berbagai elemen proyek, termasuk kelas, subfolder, dan file. Tanda hubung apa pun dalam nama folder diubah

menjadi garis bawah. Nama tersebut harus mengikuti bentuk pengenal Python. Misalnya, seharusnya tidak dimulai dengan angka atau berisi spasi.

Untuk bekerja dengan proyek baru, aktifkan lingkungan virtualnya. Ini memungkinkan dependensi proyek diinstal secara lokal di folder proyek, bukan secara global.

```
source .venv/bin/activate
python -m pip install -r requirements.txt
```

> **Note**
>
> Anda mungkin mengenali ini sebagai Mac/Linux perintah untuk mengaktifkan lingkungan virtual. Templat Python menyertakan file batch, `source.bat`, yang memungkinkan perintah yang sama untuk digunakan pada Windows. Perintah Windows tradisional `.venv\Scripts\activate.bat` juga berfungsi. Jika Anda menginisialisasi AWS CDK proyek Anda menggunakan AWS CDK Toolkit v1.70.0 atau yang lebih lama, lingkungan virtual Anda ada di direktori, bukan. `.env` `.venv`

Infrastruktur Dasar

Buka file `./ddb_filters/ddb_filters_stack.py` dengan editor teks pilihan Anda. File ini dibuat secara otomatis saat Anda membuat AWS CDK proyek.

Selanjutnya, tambahkan fungsi `_create_ddb_table` dan `_set_ddb_trigger_function`. Fungsi-fungsi ini akan membuat tabel DynamoDB dengan kunci partisi PK dan mengurutkan kunci SK dalam mode penyediaan mode sesuai permintaan, dengan Amazon DynamoDB Streams diaktifkan secara default untuk menampilkan gambar Baru dan Lama.

Fungsi Lambda akan disimpan di folder `lambda` di bagian file `app.py`. File ini akan dibuat nanti. Ini akan mencakup variabel lingkungan `APP_TABLE_NAME`, yang akan menjadi nama Tabel Amazon DynamoDB yang dibuat oleh tumpukan ini. Dalam fungsi yang sama kami akan memberikan izin baca aliran ke fungsi Lambda. Akhirnya, hal tersebut akan berlangganan DynamoDB Streams sebagai sumber acara untuk fungsi lambda.

Di akhir file dalam metode `__init__`, Anda akan memanggil konstruksi masing-masing untuk menginisialisasi mereka dalam tumpukan. Untuk proyek yang lebih besar yang memerlukan komponen dan layanan tambahan, mungkin yang terbaik adalah mendefinisikan konstruksi ini di luar tumpukan dasar.

```
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)
```

Sekarang kita akan membuat fungsi lambda yang sangat sederhana yang akan mencetak log ke Amazon CloudWatch. Untuk melakukannya, buat folder baru bernama `lambda`.

```
mkdir lambda
touch app.py
```

Menggunakan editor teks favorit Anda, tambahkan konten berikut ke file `app.py`:

```
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)
```

Memastikan Anda berada di folder `/ddb_filters/`, ketikkan perintah berikut untuk membuat aplikasi sampel:

```
cdk deploy
```

Pada titik tertentu Anda akan diminta untuk mengonfirmasi apakah Anda ingin menerapkan solusi tersebut. Terima perubahan dengan mengetik 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
```

Setelah perubahan diterapkan, buka AWS konsol Anda dan tambahkan satu item ke tabel Anda.

```
{
 "PK": "COMPANY#1000",
 "SK": "PRODUCT#CHOCOLATE#DARK",
 "company_id": "1000",
 "type": "",
 "state": "FL",
 "stores": 5,
 "price": 15,
 "quantity": 50,
 "fabric": "Florida Chocolates"
}
```

CloudWatch Log sekarang harus berisi semua informasi dari entri ini.

Contoh Filter

- Hanya produk yang cocok dengan status tertentu

Buka file `ddb_filters/ddb_filters/ddb_filters_stack.py`, dan modifikasi untuk menyertakan filter yang cocok dengan semua produk yang setara dengan “FL”. Ini dapat direvisi tepat di bawah `event_subscription` di baris 45.

```
event_subscription.add_property_override(
    property_path="FilterCriteria",
    value={
        "Filters": [
            {
                "Pattern": json.dumps(
                    {"dynamodb": {"NewImage": {"state": {"S": ["FL"]}}}}
                )
            },
        ]
    },
)
```

- Hanya item yang dimulai dengan beberapa nilai di PK dan SK

Ubah skrip python untuk menyertakan kondisi berikut:

```
event_subscription.add_property_override(
    property_path="FilterCriteria",
    value={
        "Filters": [
            {
                "Pattern": json.dumps(
                    {
                        {
                            "dynamodb": {
                                "Keys": {
                                    "PK": {"S": [{"prefix": "COMPANY"}]},
                                    "SK": {"S": [{"prefix": "PRODUCT"}]},
                                }
                            }
                        }
                    }
                )
            },
        ]
    },
```

- Baik mulai dengan beberapa nilai pada PK dan SK atau dari keadaan tertentu.

Ubah skrip python untuk menyertakan kondisi berikut:

```
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"]}}}}
                )
            },
        ]
    },
)
```

Perhatikan bahwa kondisi OR ditambahkan dengan menambahkan lebih banyak elemen ke array Filter.

Pembersihan

Temukan tumpukan filter di dasar direktori kerja Anda, dan jalankan`cdk destroy`. Anda akan diminta untuk mengonfirmasi penghapusan sumber daya:

```
cdk destroy
Are you sure you want to delete: DdbFiltersStack (y/n)? y
```

Praktik terbaik menggunakan DynamoDB Streams dengan Lambda

AWS Lambda Fungsi berjalan di dalam kontainer —lingkungan eksekusi yang diisolasi dari fungsi lain. Ketika Anda menjalankan fungsi untuk pertama kalinya, AWS Lambda membuat wadah baru dan mulai mengeksekusi kode fungsi.

Fungsi Lambda memiliki handler yang dijalankan sekali per permohonan. Handler berisi logika bisnis utama untuk fungsi. Misalnya, fungsi Lambda yang ditampilkan dalam Langkah 4: Buat dan uji fungsi Lambda memiliki handler yang dapat memproses catatan dalam DynamoDB stream.

Anda juga dapat memberikan kode inisialisasi yang hanya berjalan satu kali—setelah penampung dibuat, tetapi sebelumnya AWS Lambda menjalankan handler untuk pertama kalinya. Fungsi Lambda yang ditampilkan Langkah 4: Buat dan uji fungsi Lambda memiliki kode inisialisasi yang mengimpor SDK untuk JavaScript di Node.js, dan membuat klien untuk Amazon SNS. Objek-objek ini hanya boleh didefinisikan sekali, di luar handler.

Setelah fungsi berjalan, AWS Lambda mungkin memilih untuk menggunakan kembali wadah untuk pemanggilan fungsi berikutnya. Dalam kasus ini, handler fungsi Anda mungkin dapat menggunakan kembali sumber daya yang Anda tetapkan dalam kode inisialisasi Anda. (Anda tidak dapat mengontrol berapa lama AWS Lambda akan mempertahankan kontainer, atau apakah kontainer akan digunakan kembali.)

Untuk pemicu DynamoDB AWS Lambda menggunakan, kami merekomendasikan hal berikut:

- AWS klien layanan harus dipakai dalam kode inisialisasi, bukan di handler. Ini memungkinkan AWS Lambda untuk menggunakan kembali koneksi yang ada, selama masa pakai kontainer.
- Secara umum, Anda tidak perlu secara eksplisit mengelola koneksi atau menerapkan penyatuan koneksi karena AWS Lambda mengelola ini untuk Anda.

Konsumen Lambda untuk aliran DynamoDB tidak menjamin persis sekali pengiriman dan dapat menyebabkan duplikat sesekali. Pastikan kode fungsi Lambda Anda idempoten untuk mencegah timbulnya masalah tak terduga karena pemrosesan duplikat.

Untuk informasi selengkapnya, lihat Praktik terbaik untuk bekerja dengan AWS Lambda fungsi di Panduan AWS Lambda Pengembang.

DynamoDB Streams dan Apache Flink

Anda dapat menggunakan catatan Amazon DynamoDB Streams dengan Apache Flink. Dengan Amazon Managed Service untuk Apache Flink, Anda dapat mengubah dan menganalisis data

streaming secara real time menggunakan Apache Flink. Apache Flink adalah kerangka pemrosesan aliran sumber terbuka untuk memproses data real-time. Konektor Amazon DynamoDB Streams untuk Apache Flink menyederhanakan pembuatan dan pengelolaan beban kerja Apache Flink dan memungkinkan Anda mengintegrasikan aplikasi dengan aplikasi lain. Layanan AWS

Amazon Managed Service untuk Apache Flink membantu Anda membangun aplikasi pemrosesan end-to-end streaming dengan cepat untuk analitik log, analitik clickstream, Internet of Things (IoT), teknologi iklan, game, dan banyak lagi. Empat kasus penggunaan yang paling umum adalah streaming extract-transform-load (ETL), aplikasi berbasis peristiwa, analitik real-time responsif, dan kueri interaktif aliran data. Untuk informasi selengkapnya tentang menulis ke Apache Flink dari Amazon DynamoDB Streams, lihat Amazon DynamoDB Streams Connector.

Akselerasi dalam memori dengan DynamoDB Accelerator (DAX)

Amazon DynamoDB dirancang untuk skala dan performa. Dalam sebagian besar kasus, waktu respons DynamoDB dapat diukur dalam milidetik satu digit. Namun, ada kasus penggunaan tertentu yang memerlukan waktu respons dalam mikrodetik. Untuk kasus penggunaan ini, DynamoDB Accelerator (DAX) memberikan waktu respons yang cepat guna mengakses data akhir konsisten.

DAX adalah layanan caching yang kompatibel dengan DynamoDB yang memungkinkan Anda mendapatkan manfaat dari performa dalam memori yang cepat untuk aplikasi yang menuntut. DAX membahas tiga skenario inti:

1. Sebagai cache dalam memori, DAX mengurangi waktu respons beban kerja bacaan akhir konsisten berdasarkan urutan besarnya dari milidetik satu digit hingga mikrodetik.
2. DAX mengurangi kompleksitas operasional dan aplikasi dengan menyediakan layanan terkelola yang kompatibel dengan API DynamoDB. Oleh karena itu, DAX hanya memerlukan perubahan fungsional minimal untuk digunakan dengan aplikasi yang ada.
3. Untuk beban kerja baca berat atau melonjak, DAX menyediakan peningkatan throughput dan potensi penghematan biaya operasional dengan mengurangi kebutuhan penyediaan unit kapasitas baca yang berlebihan. Hal ini bermanfaat untuk aplikasi yang memerlukan pembacaan berulang untuk kunci individu.

DAX mendukung enkripsi sisi server. Dengan enkripsi saat diam, data yang disimpan oleh DAX pada disk akan dienkripsi. DAX menulis data ke disk sebagai bagian dari penyebaran perubahan dari simpul primer ke replika baca. Untuk informasi selengkapnya, lihat Enkripsi DAX saat diam.

DAX juga mendukung enkripsi saat bergerak untuk memastikan bahwa semua permintaan dan respons antara aplikasi dan klaster dienkripsi oleh keamanan lapisan pengangkutan (TLS), dan koneksi ke klaster dapat diautentikasi dengan verifikasi sertifikat klaster x509. Untuk informasi selengkapnya, lihat Enkripsi bergerak DAX.

Topik

- Kasus penggunaan untuk DAX
- Catatan penggunaan DAX
- DAX: Cara kerjanya

- Komponen klaster DAX
- Membuat klaster DAX
- Model konsistensi DAX dan DynamoDB
- Melakukan pengembangan dengan klien DynamoDB Accelerator (DAX)
- Mengelola klaster DAX
- Memantau DynamoDB Accelerator
- Instans DAX T3/T2 yang dapat melonjak
- Kontrol akses DAX
- Enkripsi DAX saat diam
- Enkripsi bergerak DAX
- Menggunakan peran IAM tertaut layanan untuk DAX
- Mengakses DAX di seluruh akun AWS
- Panduan pengukuran klaster DAX

Kasus penggunaan untuk DAX

DAX menyediakan akses ke data akhir konsisten dari tabel DynamoDB dengan latensi mikrodetik. Klaster DAX Multi-AZ dapat melayani jutaan permintaan per detik.

DAX ideal untuk jenis aplikasi berikut:

- Aplikasi yang membutuhkan waktu respons secepat mungkin untuk pembacaan. Beberapa contohnya termasuk penawaran waktu nyata, game sosial, dan aplikasi perdagangan. DAX memberikan performa baca dalam memori yang cepat untuk kasus penggunaan tersebut.
- Aplikasi yang membaca sejumlah kecil item yang lebih sering dibanding yang lain. Misalnya, pertimbangkan sistem e-commerce yang mengadakan obral satu hari untuk produk populer. Selama obral, permintaan untuk produk tersebut (dan datanya di DynamoDB) akan meningkat tajam dibandingkan dengan semua produk lainnya. Untuk mengurangi dampak kunci "hot" dan distribusi lalu lintas yang tidak seragam, Anda dapat memindahkan aktivitas baca ke cache DAX hingga obral satu hari berakhir.
- Aplikasi dengan pembacaan intensif, tetapi juga sensitif terhadap biaya. Dengan DynamoDB, Anda menyediakan jumlah pembacaan per detik yang dibutuhkan oleh aplikasi Anda. Jika aktivitas baca meningkat, Anda dapat meningkatkan throughput baca yang disediakan tabel (dengan biaya

tambahan). Selain itu, Anda juga dapat memindahkan aktivitas dari aplikasi Anda ke klaster DAX dan mengurangi jumlah unit kapasitas baca yang perlu dibeli.

- Aplikasi yang memerlukan pembacaan berulang untuk satu set besar data. Aplikasi semacam itu berpotensi mengalihkan sumber daya basis data dari aplikasi lain. Sebagai contoh, analisis jangka panjang cuaca regional data dapat menggunakan sementara semua kapasitas baca dalam tabel DynamoDB. Situasi ini akan berdampak negatif terhadap aplikasi lain yang perlu mengakses data yang sama. Dengan DAX, analisis cuaca dapat dilakukan terhadap data cache sebagai gantinya.

DAX tidak ideal untuk jenis aplikasi berikut:

- Aplikasi yang memerlukan bacaan sangat konsisten (atau yang tidak dapat menoleransi bacaan akhir konsisten).
- Aplikasi yang tidak memerlukan waktu respons mikrodetik untuk pembacaan atau yang tidak memerlukan pemindahan aktivitas pembacaan berulang dari tabel dasar.
- Aplikasi yang intensif menulis. Volume penulisan yang tinggi menyebabkan peningkatan replikasi di seluruh node DAX dalam sebuah cluster. Hal ini menyebabkan peningkatan konsumsi sumber daya dan risiko masalah ketersediaan.
- Aplikasi tanpa banyak pembacaan berulang. DAX berkinerja terbaik ketika tingkat hit cache melebihi 90%. Tingkat hit cache yang lebih rendah meningkatkan kesalahan cache, yang menghabiskan lebih banyak sumber daya di seluruh cluster DAX.

Catatan penggunaan DAX

- Untuk daftar AWS Wilayah di mana DAX tersedia, lihat harga Amazon DynamoDB.
- DAX mendukung aplikasi yang ditulis dalam Go, Java, Node.js, Python, dan .NET, AWS menggunakan klien yang disediakan untuk bahasa pemrograman tersebut.
- DAX hanya tersedia untuk platform EC2 -VPC.
- Kebijakan peran layanan klaster DAX harus mengizinkan tindakan `dynamodb:DescribeTable` untuk mempertahankan metadata tentang tabel DynamoDB.
- Klaster DAX mempertahankan metadata tentang nama atribut item yang disimpan. Metadata tersebut dipertahankan tanpa batas waktu (bahkan setelah item kedaluwarsa atau telah dihapus dari cache). Aplikasi yang menggunakan sejumlah nama atribut tidak terbatas, dapat menyebabkan kehabisan memori di klaster DAX dari waktu ke waktu. Batasan ini hanya berlaku

untuk nama atribut tingkat atas, bukan nama atribut bersarang. Contoh nama atribut tingkat atas yang bermasalah termasuk cap waktu, UUIDs, dan sesi. IDs

Keterbatasan ini hanya berlaku untuk nama atribut, bukan nilainya. Item seperti berikut bukan masalah.

```
{
    "Id": 123,
    "Title": "Bicycle 123",
    "CreationDate": "2017-10-24T01:02:03+00:00"
}
```

Namun, item seperti berikut menjadi masalah jika jumlahnya cukup dan masing-masing memiliki timestamp yang berbeda.

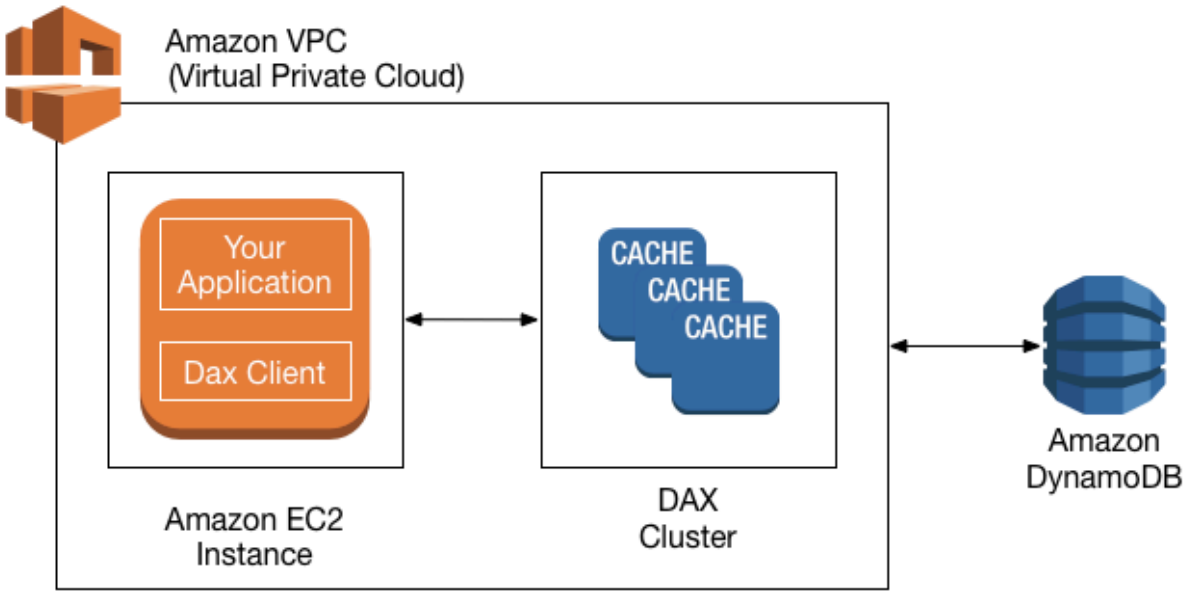
```
{
    "Id": 123,
    "Title": "Bicycle 123",
    "2017-10-24T01:02:03+00:00": "created"
}
```

DAX: Cara kerjanya

Amazon DynamoDB Accelerator (DAX) dirancang untuk berjalan di lingkungan Amazon Virtual Private Cloud (Amazon VPC). Layanan Amazon VPC menentukan jaringan virtual yang mirip dengan pusat data tradisional. Dengan VPC, Anda dapat mengontrol rentang alamat IP, subnet, tabel rute, gateway jaringan, dan pengaturan keamanan. Anda dapat meluncurkan klaster DAX di jaringan virtual dan mengontrol akses ke klaster menggunakan grup keamanan Amazon VPC.

> **Note**
>
> Jika Anda membuat AWS akun setelah 4 Desember 2013, Anda sudah memiliki VPC default di setiap AWS Wilayah. VPC siap untuk langsung digunakan tanpa harus melakukan langkah konfigurasi tambahan.
> Untuk informasi selengkapnya, lihat VPC default dan subnet default di Panduan Pengguna Amazon VPC.

Diagram berikut menunjukkan gambaran umum tingkat tinggi DAX.



Untuk membuat klaster DAX, Anda menggunakan AWS Management Console. Kecuali Anda menentukan lain, klaster DAX berjalan dalam VPC default Anda. Untuk menjalankan aplikasi Anda, Anda meluncurkan EC2 instans Amazon ke VPC Amazon Anda. Anda kemudian menerapkan aplikasi Anda (dengan klien DAX) pada instance. EC2

Saat runtime, klien DAX mengarahkan semua permintaan API DynamoDB aplikasi Anda ke klaster DAX. Jika DAX dapat memproses salah satu permintaan API ini secara langsung, tindakan tersebut akan dilakukan. Jika tidak, DAX akan meneruskan permintaan ke DynamoDB.

Terakhir, klaster DAX mengembalikan hasil ke aplikasi Anda.

Topik

- Cara DAX memproses permintaan
- Cache item
- Cache kueri

Cara DAX memproses permintaan

Klaster DAX terdiri atas satu atau beberapa simpul. Setiap simpul menjalankan instans perangkat lunak caching DAX sendiri. Salah satu simpul berfungsi sebagai simpul primer untuk klaster. Simpul tambahan (jika ada) berfungsi sebagai replika baca. Untuk informasi selengkapnya, lihat Simpul.

Aplikasi Anda dapat mengakses DAX dengan menentukan titik akhir untuk klaster DAX. Perangkat lunak klien DAX bekerja dengan titik akhir klaster untuk melakukan penyeimbangan beban dan perutean cerdas.

Operasi baca

DAX dapat merespons panggilan API berikut:

- `GetItem`
- `BatchGetItem`
- `Query`
- `Scan`

Jika permintaan menentukan bacaan akhir konsisten (perilaku default), permintaan akan mencoba membaca item dari DAX:

- Jika DAX memiliki item yang tersedia (hit cache), DAX mengembalikan item ke aplikasi tanpa mengakses DynamoDB.
- Jika DAX tidak memiliki item yang tersedia (cache miss), DAX meneruskan permintaan ke DynamoDB. Ketika menerima respons dari DynamoDB, DAX mengembalikan hasil ke aplikasi. Namun, DAX juga menulis hasil ke cache pada simpul primer.

> **Note**
>
> Jika ada replika baca di klaster, DAX otomatis menyinkronkan replika dengan simpul primer. Untuk informasi selengkapnya, lihat Klaster.

Jika permintaan menentukan bacaan sangat konsisten, DAX meneruskan permintaan ke DynamoDB. Hasil dari DynamoDB tidak di-cache di DAX. Sebaliknya, hasil akan kembali ke aplikasi.

Operasi tulis

Operasi API DAX berikut dianggap "write-through":

- `BatchWriteItem`
- `UpdateItem`
- `DeleteItem`
- `PutItem`

Dengan operasi tersebut, data terlebih dahulu ditulis ke tabel DynamoDB, lalu ke klaster DAX. Operasi ini hanya akan berhasil jika data berhasil ditulis di tabel dan DAX.

Operasi lainnya

DAX tidak mengenali operasi DynamoDB apa pun untuk mengelola tabel (seperti `CreateTable`, `UpdateTable`, dan sebagainya). Jika perlu melakukan operasi tersebut, aplikasi Anda harus mengakses DynamoDB langsung, bukan menggunakan DAX.

Untuk informasi detail tentang konsistensi DAX dan DynamoDB, lihat Model konsistensi DAX dan DynamoDB.

Untuk informasi tentang cara kerja transaksi di DAX, lihat Menggunakan transaksional di APIs DynamoDB Accelerator (DAX).

Pembatasan laju permintaan

Jika jumlah permintaan yang dikirim ke DAX melebihi kapasitas node, DAX membatasi tingkat penerimaan permintaan tambahan dengan mengembalikan file. ThrottlingException DAX terus mengevaluasi pemanfaatan CPU Anda untuk menentukan volume permintaan yang dapat diproses sekaligus mempertahankan keadaan klaster yang sehat.

Anda dapat memantau ThrottledRequestCount metrik yang diterbitkan DAX ke Amazon. CloudWatch Jika melihat pengecualian ini secara rutin, Anda harus mempertimbangkan untuk menaikkan skala klaster.

Cache item

DAX mengelola cache item untuk menyimpan hasil dari operasi `GetItem` dan `BatchGetItem`. Item dalam cache mewakili data akhir konsisten dari DynamoDB dan disimpan berdasarkan nilai kunci primer.

Ketika aplikasi mengirimkan permintaan `GetItem` atau `BatchGetItem`, DAX mencoba membaca item langsung dari cache item menggunakan nilai kunci yang ditentukan. Jika item ditemukan (hit cache), DAX segera mengembalikannya ke aplikasi. Jika item tidak ditemukan (cache miss), DAX mengirimkan permintaan ke DynamoDB. DynamoDB memproses permintaan menggunakan bacaan akhir konsisten dan mengembalikan item ke DAX. DAX menyimpannya dalam cache item dan kemudian mengembalikannya ke aplikasi.

Cache item memiliki pengaturan Time to Live (TTL) default, yaitu 5 menit. DAX memberikan timestamp untuk setiap item yang ditulis ke cache item. Item akan kedaluwarsa jika berada dalam cache lebih lama dari pengaturan TTL. Jika Anda mengeluarkan permintaan `GetItem` pada item kedaluwarsa, hal ini dianggap sebagai cache miss dan DAX mengirimkan permintaan `GetItem` ke DynamoDB.

> **Note**
>
> Anda dapat menentukan pengaturan TTL untuk cache item ketika membuat klaster DAX baru. Untuk informasi selengkapnya, lihat Mengelola klaster DAX .

DAX juga menyimpan daftar cache item yang paling lama tidak digunakan (LRU). Daftar LRU melacak saat item pertama kali ditulis ke cache dan saat item terakhir dibaca dari cache. Jika cache item penuh, DAX menghapus item lama (meskipun belum kedaluwarsa) untuk menyediakan ruang bagi item baru. Algoritma LRU selalu diaktifkan untuk cache item dan tidak dapat dikonfigurasi pengguna.

Jika Anda menentukan nol sebagai pengaturan TTL cache item, item di dalam cache item hanya akan diperbarui karena pengosongan LRU atau operasi "write-through".

Untuk informasi selengkapnya tentang konsistensi cache item di DAX, lihat Perilaku cache item DAX.

Cache kueri

DAX juga mengelola cache kueri untuk menyimpan hasil dari operasi `Query` dan `Scan`. Item dalam cache ini mewakili set hasil dari kueri dan pemindaian pada tabel DynamoDB. Set hasil ini disimpan berdasarkan nilai parameter.

Ketika aplikasi mengirimkan permintaan `Query` atau `Scan`, DAX mencoba untuk membaca set hasil pencocokan dari cache kueri menggunakan nilai parameter tertentu. Jika set hasil ditemukan (hit cache), DAX segera mengembalikannya ke aplikasi. Jika set hasil tidak ditemukan (cache miss), DAX

mengirimkan permintaan ke DynamoDB. DynamoDB memproses permintaan menggunakan bacaan akhir konsisten dan mengembalikan set hasil ke DAX. DAX menyimpannya dalam cache kueri dan kemudian mengembalikannya ke aplikasi.

> **Note**
>
> Anda dapat menentukan pengaturan TTL untuk cache kueri ketika membuat klaster DAX baru. Untuk informasi selengkapnya, lihat Mengelola klaster DAX .

DAX juga mengelola daftar LRU untuk cache kueri. Daftar tersebut melacak saat set hasil pertama kali ditulis ke cache dan saat hasil terakhir dibaca dari cache. Jika cache kueri penuh, DAX menghapus set hasil lama (meskipun belum kedaluwarsa) untuk menyediakan ruang bagi set hasil baru. Algoritma LRU selalu diaktifkan untuk cache kueri dan tidak dapat dikonfigurasi pengguna.

Jika Anda menentukan nol sebagai pengaturan TTL cache kueri, respons kueri tidak akan di-cache.

Untuk informasi selengkapnya tentang konsistensi cache kueri di DAX, lihat Perilaku cache kueri DAX.

Komponen klaster DAX

Cluster Amazon DynamoDB Accelerator (DAX) terdiri dari komponen infrastruktur. AWS Bagian ini menjelaskan komponen-komponen tersebut dan caranya bekerja sama.

Topik

- Simpul
- Klaster
- Wilayah dan zona ketersediaan
- Grup parameter
- Grup keamanan
- ARN klaster
- Titik akhir klaster
- Titik akhir simpul
- Grup subnet

- Peristiwa
- Periode pemeliharaan

Simpul

Simpul adalah blok pembangun klaster DAX terkecil. Setiap simpul menjalankan instans dari perangkat lunak DAX dan mempertahankan replika tunggal data cache.

Anda dapat menskalakan klaster DAX dengan salah satu dari dua cara berikut:

- Dengan menambahkan lebih banyak simpul ke klaster. Hal ini meningkatkan throughput pembacaan keseluruhan klaster.
- Dengan menggunakan jenis simpul yang lebih besar. Jenis simpul yang lebih besar memberikan kapasitas lebih banyak dan dapat meningkatkan throughput. (Anda harus membuat klaster baru dengan jenis simpul baru).

Setiap simpul dalam klaster adalah jenis simpul yang sama dan menjalankan perangkat lunak caching DAX yang sama. Untuk daftar jenis simpul yang tersedia, lihat Harga Amazon DynamoDB.

Klaster

Klaster adalah pengelompokan logis dari satu atau beberapa simpul yang dikelola DAX sebagai satu unit. Salah satu simpul dalam klaster ditetapkan sebagai simpul primer, dan simpul lainnya (jika ada) adalah replika baca.

Simpul primer bertanggung jawab untuk hal berikut:

- Memenuhi permintaan aplikasi untuk data cache.
- Menangani operasi penulisan ke DynamoDB.
- Menghapus data dari cache sesuai dengan kebijakan pengosongan klaster.

Ketika perubahan dilakukan pada data cache di simpul primer, DAX menyebarkan perubahan untuk semua simpul replika baca menggunakan log replikasi. Setelah konfirmasi diterima dari semua replika baca, DynamoDB menghapus log replikasi dari simpul primer.

Klaster DAX dapat mendukung hingga 11 simpul per klaster (simpul primer ditambah maksimum 10 replika baca).

Replika baca bertanggung jawab untuk hal berikut:

- Memenuhi permintaan aplikasi untuk data cache.
- Menghapus data dari cache sesuai dengan kebijakan pengosongan klaster.

Namun, tidak seperti simpul primer, replika baca tidak menulis ke DynamoDB.

Replika baca melayani dua tujuan tambahan:

- Skalabilitas. Jika ada sejumlah besar aplikasi klien yang perlu mengakses DAX secara bersamaan, Anda dapat menambahkan lebih banyak replika untuk penskalaan pembacaan. DAX menyebarkan beban secara merata di semua simpul di klaster. (Cara lain untuk meningkatkan throughput adalah dengan menggunakan jenis simpul cache yang lebih besar).
- Ketersediaan tinggi. Jika simpul primer mengalami kegagalan, DAX otomatis melakukan failover ke replika baca dan menetapkannya sebagai primer baru. Jika simpul replika gagal, simpul lain dalam klaster DAX masih dapat melayani permintaan hingga simpul yang mengalami kegagalan dapat dipulihkan. Untuk toleransi kesalahan maksimum, Anda harus melakukan deployment replika baca di Zona Ketersediaan terpisah. Konfigurasi ini memastikan bahwa klaster DAX Anda dapat terus berfungsi, meskipun seluruh Zona Ketersediaan menjadi tidak tersedia.

Important

Untuk penggunaan produksi, kami sangat menyarankan menggunakan DAX dengan setidaknya tiga simpul dan setiap simpul ditempatkan di Zona Ketersediaan yang berbeda. Tiga simpul diperlukan agar klaster DAX menjadi toleran terhadap kesalahan.
Satu klaster DAX dapat di-deploy dengan satu atau dua simpul untuk beban kerja pengembangan atau pengujian. Cluster satu simpul dan dua simpul tidak toleran terhadap kesalahan, dan kami tidak merekomendasikan penggunaan kurang dari tiga node untuk penggunaan produksi. Jika cluster satu node atau dua-node mengalami kesalahan perangkat lunak atau perangkat keras, cluster dapat menjadi tidak tersedia atau kehilangan data cache.

Important

Cluster DAX mendukung maksimal 500 tabel DynamoDB. Jika Anda melampaui 500 tabel, klaster Anda mungkin mengalami penurunan ketersediaan dan kinerja.

Wilayah dan zona ketersediaan

Cluster DAX di AWS Region hanya dapat berinteraksi dengan tabel DynamoDB yang berada di Region yang sama. Untuk alasan ini, pastikan bahwa Anda meluncurkan klaster DAX di Wilayah yang benar. Jika memiliki tabel DynamoDB di Wilayah lain, Anda juga harus meluncurkan klaster DAX di Wilayah tersebut.

Setiap Wilayah dirancang agar terisolasi sepenuhnya dari Wilayah lainnya. Setiap Wilayah terdiri dari beberapa Zona Ketersediaan. Dengan meluncurkan simpul Anda di berbagai Zona Ketersediaan, Anda dapat mencapai kemungkinan toleransi kesalahan terbesar.

> **Important**
>
> Jangan menempatkan semua simpul klaster Anda di satu Zona Ketersediaan. Dalam konfigurasi ini, klaster DAX Anda menjadi tidak tersedia jika ada kegagalan Zona Ketersediaan.
> Untuk penggunaan produksi, kami sangat menyarankan menggunakan DAX dengan setidaknya tiga simpul dan setiap simpul ditempatkan di Zona Ketersediaan yang berbeda. Tiga simpul diperlukan agar klaster DAX menjadi toleran terhadap kesalahan.
> Satu klaster DAX dapat di-deploy dengan satu atau dua simpul untuk beban kerja pengembangan atau pengujian. Klaster dengan satu dan dua simpul tidak toleran terhadap kesalahan dan sebaiknya jangan menggunakan kurang dari tiga simpul untuk produksi. Jika klaster dengan satu atau dua simpul mengalami kesalahan perangkat lunak atau perangkat keras, klaster dapat menjadi tidak tersedia atau kehilangan data cache.

Grup parameter

Grup parameter digunakan untuk mengelola pengaturan runtime klaster DAX. DAX memiliki beberapa parameter yang dapat Anda gunakan untuk mengoptimalkan performa (seperti menentukan kebijakan TTL untuk data cache). Grup parameter adalah set parameter bernama yang dapat Anda terapkan ke klaster. Dengan demikian, Anda dapat memastikan bahwa semua simpul dalam klaster tersebut dikonfigurasi dengan cara yang persis sama.

Grup keamanan

Klaster DAX berjalan di lingkungan Amazon Virtual Private Cloud (Amazon VPC). Lingkungan ini adalah jaringan virtual yang didedikasikan untuk AWS akun Anda dan terisolasi dari yang lain VPCs.

Grup keamanan bertindak sebagai firewall virtual untuk VPC Anda guna mengontrol lalu lintas masuk dan keluar.

Ketika meluncurkan sebuah klaster di VPC, Anda menambahkan aturan masuk ke grup keamanan Anda untuk mengizinkan lalu lintas jaringan masuk. Aturan masuk menentukan protokol (TCP) dan nomor port (8111) untuk klaster Anda. Setelah Anda menambahkan aturan ini ke grup keamanan, aplikasi yang berjalan dalam VPC Anda dapat mengakses klaster DAX.

ARN klaster

Setiap klaster DAX diberi Amazon Resource Name (ARN). Format ARN adalah sebagai berikut.

```
arn:aws:dax:region:accountID:cache/clusterName
```

Anda menggunakan ARN klaster dalam kebijakan IAM untuk menentukan izin operasi API DAX. Untuk informasi selengkapnya, lihat Kontrol akses DAX.

Titik akhir klaster

Setiap klaster DAX menyediakan klaster titik akhir untuk digunakan oleh aplikasi Anda. Dengan mengakses klaster menggunakan titik akhir, aplikasi Anda tidak perlu tahu nama host dan nomor port masing-masing simpul dalam klaster. Aplikasi Anda secara otomatis "mengetahui" semua simpul di klaster, bahkan jika Anda menambahkan atau menghapus replika baca.

Berikut adalah contoh titik akhir klaster di wilayah us-east-1 yang tidak dikonfigurasi untuk menggunakan enkripsi saat bergerak.

```
dax://my-cluster.l6fzcv.dax-clusters.us-east-1.amazonaws.com
```

Berikut adalah contoh titik akhir klaster di wilayah yang sama yang dikonfigurasi untuk menggunakan enkripsi saat bergerak.

```
daxs://my-encrypted-cluster.l6fzcv.dax-clusters.us-east-1.amazonaws.com
```

Titik akhir simpul

Masing-masing simpul individu dalam klaster DAX memiliki nama host dan nomor port sendiri. Berikut adalah contoh titik akhir simpul.

```
myDAXcluster-a.2cmrwl.clustercfg.dax.use1.cache.amazonaws.com:8111
```

Aplikasi Anda dapat mengakses simpul langsung menggunakan titik akhirnya. Namun, sebaiknya Anda memperlakukan klaster DAX sebagai satu unit dan mengaksesnya menggunakan titik akhir klaster. Titik akhir klaster mengisolasi aplikasi Anda agar tidak perlu mengelola daftar simpul dan memastikan daftar selalu terbaru ketika Anda menambahkan atau menghapus simpul dari klaster.

Grup subnet

Akses ke node cluster DAX dibatasi untuk aplikasi yang berjalan di EC2 instans Amazon dalam lingkungan VPC Amazon. Anda dapat menggunakan grup subnet untuk memberikan akses klaster dari EC2 instans Amazon yang berjalan pada subnet tertentu. Grup subnet adalah kumpulan subnet (biasanya privat) yang dapat ditetapkan untuk klaster Anda yang berjalan di lingkungan Amazon VPC.

Ketika membuat klaster DAX, Anda harus menentukan grup subnet. DAX menggunakan grup subnet tersebut untuk memilih subnet dan alamat IP dalam subnet tersebut yang akan dikaitkan dengan simpul Anda.

Peristiwa

DAX mencatat peristiwa penting dalam klaster Anda, seperti kegagalan untuk menambahkan simpul, keberhasilan dalam menambahkan simpul, atau perubahan pada grup keamanan. Dengan memantau peristiwa penting, Anda dapat mengetahui status klaster terbaru dan dapat mengambil tindakan korektif sesuai peristiwa tersebut. Anda dapat mengakses peristiwa ini menggunakan AWS Management Console atau `DescribeEvents` tindakan di API manajemen DAX.

Anda juga dapat meminta agar pemberitahuan dikirim ke topik Amazon Simple Notification Service (Amazon SNS). Dengan demikian, Anda akan segera mengetahui jika ada peristiwa yang terjadi di klaster DAX.

Periode pemeliharaan

Setiap cluster memiliki jendela pemeliharaan mingguan untuk menerapkan perubahan sistem. Karena perubahan diterapkan secara berurutan, node yang ada diganti dan node baru dengan perubahan yang diterapkan ditambahkan ke cluster. Selama periode ini, aplikasi Anda mungkin mengamati kesalahan sementara atau throttle. Oleh karena itu, kami menyarankan Anda menjadwalkan jendela pemeliharaan selama waktu penggunaan terendah Anda dan menyesuaikan jadwal ini secara berkala sesuai kebutuhan. Anda dapat menentukan rentang waktu dengan durasi hingga 24 jam. Selama waktu tersebut, aktivitas pemeliharaan yang Anda minta akan dilakukan.

Jika Anda tidak menentukan jendela pemeliharaan yang diinginkan saat membuat atau memodifikasi klaster cache, DAX menetapkan jendela pemeliharaan 60 menit pada hari kerja acak. Jendela pemeliharaan 60 menit ini dipilih secara acak dari blok waktu 8 jam untuk masing-masing. Wilayah AWS Tabel berikut mencantumkan blok waktu untuk tiap Wilayah dari periode waktu default yang ditetapkan.

Kode Wilayah	Nama Wilayah	Periode pemeliharaan
ap-northeast-1	Wilayah Asia Pacific (Tokyo)	13.00–21.00 UTC
ap-southeast-1	Wilayah Asia Pacific (Singapore)	14.00–22.00 UTC
ap-southeast-2	Wilayah Asia Pacific (Sydney)	12.00–20.00 UTC
ap-south-1	Wilayah Asia Pacific (Mumbai)	17.30–01.30 UTC
cn-northwest-1	Wilayah China (Ningxia)	23.00–07.00 UTC
cn-north-1	Wilayah Tiongkok (Beijing)	14.00–22.00 UTC
eu-central-1	Wilayah Eropa (Frankfurt)	23.00–07.00 UTC
eu-north-1	Wilayah Eropa (Stockholm)	01.00–09.00 UTC
eu-south-2	Wilayah Eropa (Spanyol)	21.00–05.00 UTC
eu-west-1	Wilayah Eropa (Irlandia)	22.00–06.00 UTC
eu-west-2	Wilayah Eropa (London)	23.00–07.00 UTC
eu-west-3	Wilayah Eropa (Paris)	23.00–07.00 UTC
sa-east-1	Wilayah South America (São Paulo)	01.00–09.00 UTC
us-east-1	Wilayah US East (N. Virginia)	03.00–11.00 UTC
us-east-2	Wilayah US East (Ohio)	23.00–07.00 UTC

Kode Wilayah	Nama Wilayah	Periode pemeliharaan
us-west-1	Wilayah Barat AS (N. Californi a)	06.00–14.00 UTC
us-west-2	Wilayah US West (Oregon)	06.00–14:00 UTC

Membuat klaster DAX

Bagian ini memandu Anda saat pertama kali menyiapkan dan menggunakan Amazon DynamoDB Accelerator (DAX) di lingkungan default Amazon Virtual Private Cloud (Amazon VPC). Anda dapat membuat cluster DAX pertama Anda menggunakan AWS Command Line Interface (AWS CLI) atau. AWS Management Console

Setelah membuat cluster DAX, Anda dapat mengaksesnya dari EC2 instans Amazon yang berjalan di VPC yang sama. Kemudian, Anda dapat menggunakan klaster DAX dengan program aplikasi. Untuk informasi selengkapnya, lihat Melakukan pengembangan dengan klien DynamoDB Accelerator (DAX).

Topik

- Membuat peran layanan IAM untuk DAX agar dapat mengakses DynamoDB
- Membuat cluster DAX menggunakan AWS CLI
- Membuat klaster DAX menggunakan AWS Management Console

Membuat peran layanan IAM untuk DAX agar dapat mengakses DynamoDB

Agar klaster DAX dapat mengakses tabel DynamoDB atas nama Anda, Anda harus membuat peran layanan. Peran layanan adalah peran AWS Identity and Access Management (IAM) yang mengizinkan AWS layanan untuk bertindak atas nama Anda. Peran layanan memungkinkan DAX mengakses tabel DynamoDB Anda, seolah-olah Anda sendiri yang mengakses tabel tersebut. Anda harus membuat peran layanan sebelum dapat membuat klaster DAX.

Jika Anda menggunakan konsol, alur kerja pembuatan klaster memeriksa keberadaan peran layanan DAX yang sudah ada sebelumnya. Jika tidak ditemukan, konsol membuat peran layanan baru untuk Anda. Untuk informasi selengkapnya, lihat the section called “Langkah 2: Buat klaster DAX”.

Jika Anda menggunakan AWS CLI, Anda harus menentukan peran layanan DAX yang telah Anda buat sebelumnya. Jika tidak, Anda harus membuat peran layanan baru sebelumnya. Untuk informasi selengkapnya, lihat Langkah 1: Buat peran layanan IAM untuk DAX untuk mengakses DynamoDB menggunakan AWS CLI.

Izin yang diperlukan untuk membuat peran layanan

Kebijakan `AdministratorAccess` yang dikelola AWS menyediakan semua izin yang diperlukan untuk membuat klaster DAX dan peran layanan. Jika pengguna Anda melampirkan `AdministratorAccess`, tidak diperlukan tindakan lebih lanjut.

Jika tidak, izin berikut harus ditambahkan untuk kebijakan IAM Anda sehingga pengguna IAM Anda dapat membuat peran layanan:

- `iam:CreateRole`
- `iam:CreatePolicy`
- `iam:AttachRolePolicy`
- `iam:PassRole`

Lampirkan izin tersebut ke pengguna yang mencoba melakukan tindakan.

> **Note**
>
> `iam:PassRole`Izin `iam:CreateRole`iam:CreatePolicy,`iam:AttachRolePolicy`,, dan tidak disertakan dalam kebijakan AWS terkelola untuk DynamoDB. Hal ini disengaja karena izin tersebut memberikan kemungkinan eskalasi hak istimewa: Artinya, pengguna dapat menggunakan izin untuk membuat kebijakan administrator baru dan kemudian melampirkan kebijakan tersebut untuk peran yang ada. Untuk alasan ini, Anda (administrator klaster DAX) harus secara eksplisit menambahkan izin ini ke kebijakan.

Pemecahan Masalah

Jika tidak ada izin `iam:CreateRole`, `iam:CreatePolicy`, dan `iam:AttachPolicy` di kebijakan pengguna Anda, akan muncul pesan kesalahan. Tabel berikut mencantumkan pesan tersebut dan menjelaskan cara memperbaiki masalah.

Jika Anda melihat pesan kesalahan ini...	Lakukan hal berikut:
`User: arn:aws:iam:: accountID :user/userName is not authorized to perform: iam:CreateRole on resource: arn:aws:iam:: accountID :role/service-role/ roleName`	Tambahkan `iam:CreateRole` ke kebijakan pengguna Anda.
`User: arn:aws:iam:: accountID :user/userName is not authorized to perform: iam:CreatePolicy on resource: policy policyName`	Tambahkan `iam:CreatePolicy` ke kebijakan pengguna Anda.
`User: arn:aws:iam:: accountID :user/userName is not authorized to perform: iam:AttachRolePolicy on resource: role daxServiceRole`	Tambahkan `iam:AttachRolePolicy` ke kebijakan pengguna Anda.

Untuk informasi selengkapnya tentang kebijakan IAM yang diperlukan untuk administrasi klaster DAX, lihat Kontrol akses DAX.

Membuat cluster DAX menggunakan AWS CLI

Bagian ini menjelaskan cara membuat klaster Amazon DynamoDB Accelerator (DAX) menggunakan AWS Command Line Interface (AWS CLI). Jika belum, Anda harus menginstal dan mengonfigurasi AWS CLI. Untuk melakukannya, lihat petunjuk berikut di AWS Command Line Interface Panduan Pengguna:

- Instalasi AWS CLI
- Mengonfigurasi AWS CLI

> **Important**
>
> Untuk mengelola kluster DAX menggunakan AWS CLI, instal atau tingkatkan ke versi 1.11.110 atau lebih tinggi.

Semua AWS CLI contoh menggunakan `us-west-2` Wilayah dan akun fiktif. IDs

Topik

- Langkah 1: Buat peran layanan IAM untuk DAX untuk mengakses DynamoDB menggunakan AWS CLI
- Langkah 2: Buat grup subnet
- Langkah 3: Buat cluster DAX menggunakan AWS CLI
- Langkah 4: Konfigurasikan aturan masuk grup keamanan menggunakan AWS CLI

Langkah 1: Buat peran layanan IAM untuk DAX untuk mengakses DynamoDB menggunakan AWS CLI

Sebelum dapat membuat klaster Amazon DynamoDB Accelerator (DAX), Anda harus membuat peran layanan untuk itu. Peran layanan adalah peran AWS Identity and Access Management (IAM) yang mengizinkan AWS layanan untuk bertindak atas nama Anda. Peran layanan memungkinkan DAX mengakses tabel DynamoDB Anda seolah-olah Anda sendiri yang mengakses tabel tersebut.

Pada langkah ini, Anda membuat kebijakan IAM lalu melampirkan kebijakan tersebut ke peran IAM. Hal ini memungkinkan Anda menetapkan peran untuk klaster DAX agar dapat melakukan operasi DynamoDB atas nama Anda.

Cara membuat peran layanan IAM untuk DAX

1. Buat file bernama `service-trust-relationship.json` dengan isi berikut ini.

```
{
    "Version": "2012-10-17",
    "Statement": [
       {
            "Effect": "Allow",
            "Principal": {
                "Service": "dax.amazonaws.com"
            },
            "Action": "sts:AssumeRole"
        }
    ]
}
```

2. Buat peran layanan.

```
aws iam create-role \
    --role-name DAXServiceRoleForDynamoDBAccess \
    --assume-role-policy-document file://service-trust-relationship.json
```

3. Buat file bernama `service-role-policy.json` dengan isi berikut ini.

```
{
    "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:accountID:*"
            ]
        }
    ]
}
```

   Ganti *`accountID`* dengan ID AWS akun Anda. Untuk menemukan ID AWS akun Anda, di sudut kanan atas konsol, pilih ID login Anda. ID AWS akun Anda muncul di menu tarik-turun.

   Dalam Amazon Resource Name (ARN) dalam contoh, *`accountID`* harus berupa angka 12 digit. Jangan gunakan tanda hubung atau tanda baca lainnya.

4. Buat kebijakan IAM untuk peran layanan.

```
aws iam create-policy \
    --policy-name DAXServicePolicyForDynamoDBAccess \
    --policy-document file://service-role-policy.json
```

Dalam output, perhatikan ARN untuk kebijakan yang Anda buat, seperti dalam contoh berikut.

```
arn:aws:iam::123456789012:policy/DAXServicePolicyForDynamoDBAccess
```

5. Lampirkan kebijakan pada peran layanan. Ganti *arn* dalam kode berikut dengan peran ARN yang sebenarnya dari langkah sebelumnya.

```
aws iam attach-role-policy \
    --role-name DAXServiceRoleForDynamoDBAccess \
    --policy-arn arn
```

Berikutnya, tentukan grup subnet untuk VPC default Anda. Grup subnet adalah kumpulan satu atau beberapa subnet di VPC Anda. Lihat Langkah 2: Buat grup subnet.

Langkah 2: Buat grup subnet

Ikuti prosedur ini untuk membuat grup subnet untuk klaster Amazon DynamoDB Accelerator (DAX) menggunakan (). AWS Command Line Interface AWS CLI

> **Note**
>
> Jika sudah membuat grup subnet untuk VPC default, Anda dapat melewati langkah ini.

DAX dirancang untuk berjalan di lingkungan Amazon Virtual Private Cloud (Amazon VPC). Jika Anda membuat akun AWS setelah 4 Desember 2013, Anda sudah memiliki VPC default di setiap Wilayah AWS . Untuk informasi selengkapnya, lihat VPC default dan subnet default di Panduan Pengguna Amazon VPC.

> **Note**
>
> VPC yang memiliki cluster DAX ini dapat berisi sumber daya lain dan bahkan titik akhir VPC untuk layanan lain kecuali titik akhir VPC untuk ElastiCache dan dapat mengakibatkan kesalahan untuk operasi cluster DAX.

Cara membuat grup subnet

1. Untuk menentukan pengidentifikasi VPC default Anda, masukkan perintah berikut.

```
aws ec2 describe-vpcs
```

Di output, perhatikan pengidentifikasi untuk VPC default Anda, seperti dalam contoh berikut.

```
vpc-12345678
```

2. Tentukan subnet IDs yang terkait dengan VPC default Anda. Ganti *vpcID* dengan ID VPC Anda yang sebenarnya — misalnya,. `vpc-12345678`

```
aws ec2 describe-subnets \
    --filters "Name=vpc-id,Values=vpcID" \
    --query "Subnets[*].SubnetId"
```

Di output, catat pengidentifikasi subnet, misalnya, `subnet-11111111`.

3. Buat grup subnet. Pastikan bahwa Anda menentukan setidaknya satu ID subnet di parameter `--subnet-ids`.

```
aws dax create-subnet-group \
    --subnet-group-name my-subnet-group \
    --subnet-ids subnet-11111111 subnet-22222222 subnet-33333333 subnet-44444444
```

Untuk membuat klaster, lihat Langkah 3: Buat cluster DAX menggunakan AWS CLI.

Langkah 3: Buat cluster DAX menggunakan AWS CLI

Ikuti prosedur ini untuk menggunakan AWS Command Line Interface (AWS CLI) untuk membuat klaster Amazon DynamoDB Accelerator (DAX) di Amazon VPC default Anda.

Cara membuat klaster DAX

1. Dapatkan Amazon Resource Name (ARN) untuk peran layanan Anda.

```
aws iam get-role \
    --role-name DAXServiceRoleForDynamoDBAccess \
    --query "Role.Arn" --output text
```

Di output, catat peran layanan ARN, seperti dalam contoh berikut.

```
arn:aws:iam::123456789012:role/DAXServiceRoleForDynamoDBAccess
```

2. Buat klaster DAX. Ganti *roleARN* dengan ARN dari langkah sebelumnya.

```
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
```

Semua simpul dalam klaster berjenis `dax.r4.large` (`--node-type`). Ada tiga simpul (`--replication-factor`), satu simpul primer dan dua replika.

> **Note**
>
> Karena `sudo` dan `grep` adalah kata kunci cadangan, Anda tidak dapat membuat klaster DAX dengan kata-kata ini dalam nama klaster. Misalnya, `sudo` dan `sudocluster` adalah nama klaster yang tidak valid.

Untuk melihat status klaster, masukkan perintah berikut.

```
aws dax describe-clusters
```

Status ditampilkan dalam output, misalnya, `"Status": "creating"`.

> **Note**
>
> Pembuatan klaster memerlukan waktu beberapa menit. Ketika klaster siap, statusnya berubah menjadi `available`. Sementara itu, lanjutkan ke Langkah 4: Konfigurasikan aturan masuk grup keamanan menggunakan AWS CLI dan ikuti petunjuk di sana.

Langkah 4: Konfigurasikan aturan masuk grup keamanan menggunakan AWS CLI

Simpul di klaster Amazon DynamoDB Accelerator (DAX) Anda menggunakan grup keamanan default untuk Amazon VPC Anda. Untuk grup keamanan default, Anda harus mengotorisasi lalu lintas masuk

pada port TCP 8111 untuk klaster tidak terenkripsi atau port 9111 untuk klaster terenkripsi. Hal ini memungkinkan EC2 instans Amazon di VPC Amazon Anda untuk mengakses kluster DAX Anda.

> **Note**
>
> Jika meluncurkan klaster DAX dengan grup keamanan yang berbeda (selain `default`), Anda harus melakukan prosedur ini untuk grup tersebut sebagai gantinya.

Cara mengonfigurasi aturan masuk grup keamanan

1. Untuk menentukan pengidentifikasi grup keamanan default, masukkan perintah berikut. Ganti *`vpcID`* dengan ID VPC Anda yang sebenarnya (dari Langkah 2: Buat grup subnet).

   ```
   aws ec2 describe-security-groups \
       --filters Name=vpc-id,Values=vpcID Name=group-name,Values=default \
       --query "SecurityGroups[*].{GroupName:GroupName,GroupId:GroupId}"
   ```

   Di output, catat pengidentifikasi grup keamanan, misalnya `sg-01234567`.

2. Kemudian masukkan perintah berikut. Ganti *`sgID`* dengan pengidentifikasi grup keamanan yang sebenarnya. Gunakan port `8111` untuk klaster yang tidak terenkripsi dan `9111` untuk klaster terenkripsi.

   ```
   aws ec2 authorize-security-group-ingress \
       --group-id sgID --protocol tcp --port 8111
   ```

Membuat klaster DAX menggunakan AWS Management Console

Bagian ini menjelaskan cara membuat klaster Amazon DynamoDB Accelerator (DAX) menggunakan AWS Management Console.

Topik

- Langkah 1: Buat grup subnet menggunakan AWS Management Console
- Langkah 2: Buat cluster DAX menggunakan AWS Management Console
- Langkah 3: Konfigurasikan aturan masuk grup keamanan menggunakan AWS Management Console

Langkah 1: Buat grup subnet menggunakan AWS Management Console

Ikuti prosedur ini untuk membuat grup subnet bagi klaster Amazon DynamoDB Accelerator (DAX) Anda menggunakan AWS Management Console.

> **Note**
>
> Jika sudah membuat grup subnet untuk VPC default, Anda dapat melewati langkah ini.

DAX dirancang untuk berjalan di lingkungan Amazon Virtual Private Cloud (Amazon VPC). Jika Anda membuat akun AWS setelah 4 Desember 2013, Anda sudah memiliki VPC default di setiap Wilayah AWS . Untuk informasi selengkapnya, lihat VPC default dan subnet default di Panduan Pengguna Amazon VPC.

Sebagai bagian dari proses pembuatan klaster DAX, Anda harus menentukan grup subnet. Grup subnet adalah kumpulan satu atau beberapa subnet di VPC Anda. Ketika Anda membuat klaster DAX, simpul di-deploy ke subnet dalam grup subnet.

> **Note**
>
> VPC yang memiliki cluster DAX ini dapat berisi sumber daya lain dan bahkan titik akhir VPC untuk layanan lain kecuali titik akhir VPC untuk ElastiCache dan dapat mengakibatkan kesalahan untuk operasi cluster DAX.

Cara membuat grup subnet

1. Buka konsol DynamoDB di. https://console.aws.amazon.com/dynamodb/
2. Pada panel navigasi, di bawah DAX, pilih Grup subnet.
3. Pilih Buat grup subnet.
4. Pada jendela Buat grup subnet, lakukan hal berikut:
    a. Nama—Masukkan nama singkat untuk grup subnet.
    b. Deskripsi—Masukkan deskripsi untuk grup subnet.
    c. ID VPC—Pilih pengidentifikasi untuk lingkungan Amazon VPC Anda.
    d. Subnet—Pilih satu atau beberapa subnet dari daftar.

> **Note**
>
> Subnet didistribusikan di beberapa Zona Ketersediaan. Jika Anda berencana untuk membuat cluster DAX multi-node (node utama dan satu atau lebih replika baca), sebaiknya Anda memilih beberapa subnet. IDs Kemudian DAX dapat men-deploy simpul klaster ke beberapa Zona Ketersediaan. Jika Zona Ketersediaan menjadi tidak tersedia, DAX otomatis melakukan failover ke Zona Ketersediaan yang masih bertahan. Klaster DAX Anda akan terus berfungsi tanpa gangguan.

Jika pengaturan sudah sesuai keinginan Anda, pilih Buat grup subnet.

Untuk membuat klaster, lihat Langkah 2: Buat cluster DAX menggunakan AWS Management Console.

Langkah 2: Buat cluster DAX menggunakan AWS Management Console

Ikuti prosedur ini untuk membuat klaster Amazon DynamoDB Accelerator (DAX) di Amazon VPC default Anda.

Cara membuat klaster DAX

1. Buka konsol DynamoDB di. https://console.aws.amazon.com/dynamodb/
2. Pada panel navigasi, di bawah DAX, pilih Klaster.
3. Pilih Buat klaster.
4. Pada jendela Buat klaster, lakukan hal berikut:
   a. Nama klaster—Masukkan nama singkat untuk klaster DAX Anda.

      > **Note**
      >
      > Karena `sudo` dan `grep` adalah kata kunci cadangan, Anda tidak dapat membuat klaster DAX dengan kata-kata ini dalam nama klaster. Misalnya, `sudo` dan `sudocluster` adalah nama klaster yang tidak valid.

   b. Deskripsi klaster—Masukkan deskripsi untuk klaster.
   c. Jenis simpul—Pilih jenis simpul untuk semua simpul di klaster.

d. Ukuran klaster—Pilih jumlah simpul di klaster. Sebuah klaster terdiri dari satu simpul primer dan hingga sembilan replika baca.

> **Note**
>
> Jika Anda ingin membuat klaster simpul tunggal, pilih 1. Klaster Anda akan terdiri dari satu simpul primer.
> Jika Anda ingin membuat klaster multi-simpul, pilih angka antara 3 (satu primer dan dua replika baca) dan 10 (satu primer dan sembilan replika baca).
>
> > **Important**
> >
> > Untuk penggunaan produksi, sebaiknya gunakan DAX dengan setidaknya tiga simpul dan tempatkan setiap simpul di Zona Ketersediaan yang berbeda. Tiga simpul diperlukan agar klaster DAX menjadi toleran terhadap kesalahan.
> > Satu klaster DAX dapat di-deploy dengan satu atau dua simpul untuk beban kerja pengembangan atau pengujian. Klaster dengan satu dan dua simpul tidak toleran terhadap kesalahan dan sebaiknya jangan menggunakan kurang dari tiga simpul untuk produksi. Jika klaster dengan satu atau dua simpul mengalami kesalahan perangkat lunak atau perangkat keras, klaster dapat menjadi tidak tersedia atau kehilangan data cache.

e. Pilih Berikutnya.

f. Grup subnet—Pilih Pilih yang ada dan pilih grup subnet yang Anda buat di Langkah 1: Buat grup subnet menggunakan AWS Management Console.

g. Kontrol akses—Pilih grup keamanan default.

h. Zona Ketersediaan (AZ)—Pilih Otomatis.

i. Pilih berikutnya.

j. Peran layanan IAM untuk akses DynamoDB—Pilih Buat baru dan masukkan informasi berikut:

- Nama peran IAM—Masukkan nama untuk peran IAM, misalnya `DAXServiceRole`. Konsol membuat peran IAM baru dan klaster DAX Anda mengambil peran ini pada saat runtime.
- Pilih kotak di samping Buat kebijakan.

- Kebijakan peran IAM—Pilih Baca/Tulis. Hal ini memungkinkan klaster DAX melakukan operasi pembacaan dan penulisan di DynamoDB.
- Nama kebijakan IAM baru—Bidang ini akan terisi saat Anda memasukkan nama peran IAM. Anda juga dapat memasukkan nama untuk kebijakan IAM, misalnya, `DAXServicePolicy`. Konsol membuat kebijakan IAM baru dan melampirkan kebijakan ke peran IAM.
- Akses ke tabel DynamoDB—Pilih Semua tabel.

k. Enkripsi—Pilih Aktifkan enkripsi saat diam dan Aktifkan enkripsi saat bergerak Untuk informasi selengkapnya, lihat Enkripsi DAX saat diam dan Enkripsi bergerak DAX.

Peran layanan terpisah untuk DAX untuk mengakses Amazon juga EC2 diperlukan. DAX otomatis membuat peran layanan ini untuk Anda. Untuk informasi selengkapnya, lihat Menggunakan peran terkait layanan untuk DAX.

5. Jika pengaturan sudah sesuai keinginan Anda, pilih Berikutnya.
6. Grup parameter—Pilih Pilih yang ada.
7. Periode pemeliharaan—Pilih Tidak ada preferensi jika Anda tidak memiliki preferensi kapan peningkatan perangkat lunak diterapkan, atau pilih Tentukan jendela waktu dan berikan opsi Hari Kerja, Waktu (UTC) dan Mulai dalam (jam) untuk menjadwalkan periode pemeliharaan Anda.
8. Tanda—Pilih Tambahkan tanda baru untuk memasukkan pasangan kunci/nilai dengan tujuan penandaan.
9. Pilih Berikutnya.

Pada layar Tinjau dan buat, Anda dapat meninjau semua pengaturan. Jika Anda sudah siap untuk membuat klaster, pilih Buat klaster.

Pada layar Klaster, klaster DAX Anda akan terdaftar dengan status Membuat.

> **Note**
>
> Pembuatan klaster memerlukan waktu beberapa menit. Ketika klaster siap, statusnya berubah menjadi Tersedia.
> Sementara itu, lanjutkan ke Langkah 3: Konfigurasikan aturan masuk grup keamanan menggunakan AWS Management Console dan ikuti petunjuk di sana.

Langkah 3: Konfigurasikan aturan masuk grup keamanan menggunakan AWS Management Console

Klaster Amazon DynamoDB Accelerator (DAX) berkomunikasi melalui port TCP 8111 (untuk klaster tidak terenkripsi) atau 9111 (untuk klaster terenkripsi), sehingga Anda harus mengotorisasi lalu lintas masuk pada port tersebut. Hal ini memungkinkan EC2 instans Amazon di VPC Amazon Anda untuk mengakses kluster DAX Anda.

> **Note**
>
> Jika meluncurkan klaster DAX dengan grup keamanan yang berbeda (selain `default`), Anda harus melakukan prosedur ini untuk grup tersebut sebagai gantinya.

Cara mengonfigurasi aturan masuk grup keamanan

1. Buka EC2 konsol Amazon di https://console.aws.amazon.com/ec2/.
2. Pada panel navigasi, pilih Grup Keamanan.
3. Pilih grup keamanan default. Pada menu Tindakan, pilih Edit aturan masuk.
4. Pilih Tambahkan Aturan dan masukkan informasi berikut:
   - Rentang Port—Masukkan 8111 (jika klaster tidak dienkripsi) atau 9111 (jika klaster dienkripsi).
   - Sumber—Biarkan dalam nilai Kustom, lalu pilih bidang pencarian di sebelah kanan. Menu drop-down akan ditampilkan. Pilih pengidentifikasi grup keamanan default.
5. Pilih Simpan aturan untuk menyimpan perubahan Anda.
6. Untuk memperbarui nama di konsol, buka properti Nama dan pilih opsi Edit yang ditampilkan.

Model konsistensi DAX dan DynamoDB

Amazon DynamoDB Accelerator (DAX) adalah layanan caching write-through yang dirancang untuk memudahkan proses menambahkan cache ke tabel DynamoDB. Karena DAX beroperasi secara terpisah dari DynamoDB, Anda harus memahami model konsistensi DAX dan DynamoDB untuk memastikan bahwa aplikasi Anda berfungsi seperti yang diharapkan.

Dalam banyak kasus penggunaan, cara aplikasi Anda menggunakan DAX memengaruhi konsistensi data dalam klaster DAX serta konsistensi data antara DAX dan DynamoDB.

Topik

- Konsistensi di antara simpul klaster DAX
- Perilaku cache item DAX
- Perilaku cache kueri DAX
- Bacaan sangat konsisten dan transaksional
- Caching negatif
- Strategi untuk penulisan

Konsistensi di antara simpul klaster DAX

Guna mencapai ketersediaan tinggi untuk aplikasi Anda, sebaiknya sediakan setidaknya tiga simpul untuk klaster DAX Anda. Kemudian tempatkan simpul tersebut di beberapa Zona Ketersediaan dalam Wilayah.

Ketika klaster DAX Anda berjalan, klaster mereplikasi data di antara semua simpul dalam klaster (dengan asumsi bahwa Anda menyediakan lebih dari satu simpul). Pertimbangkan aplikasi yang berhasil melakukan `UpdateItem` menggunakan DAX. Tindakan ini menyebabkan cache item di simpul primer dimodifikasi dengan nilai baru. Nilai tersebut kemudian direplikasi ke semua simpul lain dalam klaster. Replikasi ini akhirnya konsisten dan biasanya membutuhkan waktu kurang dari satu detik untuk diselesaikan.

Dalam skenario ini, dua klien dapat membaca kunci yang sama dari klaster DAX yang sama tetapi menerima nilai yang berbeda, tergantung simpul yang diakses setiap klien. Semua simpul akan konsisten ketika pembaruan telah sepenuhnya direplikasi di seluruh simpul dalam klaster. (Perilaku ini mirip dengan sifat akhir konsisten DynamoDB.)

Jika Anda sedang membuat aplikasi yang menggunakan DAX, aplikasi tersebut harus dirancang agar dapat menoleransi data akhir konsisten.

Perilaku cache item DAX

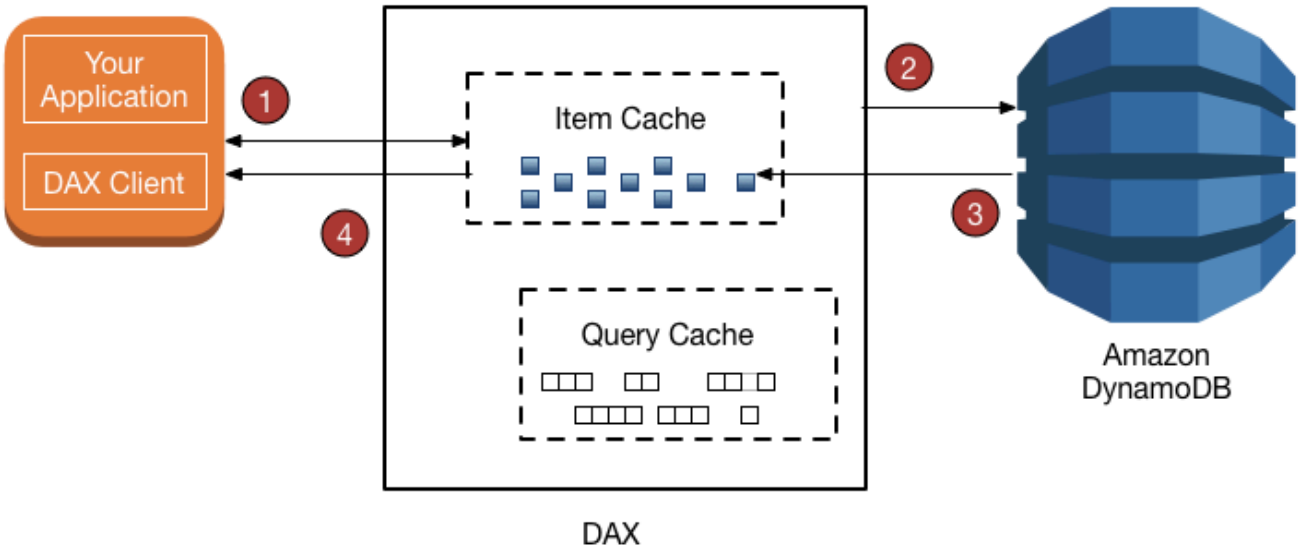
Setiap klaster DAX memiliki dua cache yang berbeda, yaitu cache item dan cache kueri. Untuk informasi selengkapnya, lihat DAX: Cara kerjanya.

Bagian ini membahas implikasi konsistensi dari pembacaan dari dan penulisan ke cache item DAX.

Konsistensi pembacaan

Dengan DynamoDB, operasi `GetItem` melakukan bacaan akhir konsisten secara default. Anggaplah Anda menggunakan `UpdateItem` dengan klien DynamoDB. Jika kemudian Anda mencoba membaca item yang sama segera setelahnya, Anda mungkin melihat data yang muncul sebelum pembaruan. Hal ini disebabkan penundaan penyebaran di semua lokasi penyimpanan DynamoDB. Konsistensi biasanya tercapai dalam hitungan detik. Jadi jika mencoba kembali pembacaan, Anda mungkin akan melihat item yang diperbarui.

Saat Anda menggunakan `GetItem` dengan klien DAX, operasi (dalam kasus ini, bacaan akhir konsisten) berlanjut seperti yang ditunjukkan berikut ini.



1. Klien DAX mengeluarkan permintaan `GetItem`. DAX mencoba membaca item yang diminta dari cache item. Jika item ada dalam cache (cache hit), DAX mengembalikannya ke aplikasi.
2. Jika item tidak tersedia (cache miss), DAX melakukan operasi `GetItem` akhir konsisten terhadap DynamoDB.
3. DynamoDB mengembalikan item yang diminta dan DAX menyimpannya dalam cache item.
4. DAX mengembalikan item ke aplikasi.
5. (Tidak ditampilkan) Jika klaster DAX berisi lebih dari satu simpul, item direplikasi ke semua simpul lain dalam klaster.

Item tetap dalam cache item DAX, tunduk pada pengaturan Time to Live (TTL) dan algoritma paling lama tidak digunakan (LRU) untuk cache. Untuk informasi selengkapnya, lihat DAX: Cara kerjanya.

Namun, selama periode ini, DAX tidak membaca ulang item dari DynamoDB. Jika orang lain memperbarui item menggunakan klien DynamoDB dan sepenuhnya melewati DAX, permintaan `GetItem` menggunakan klien DAX memberikan hasil yang berbeda dari permintaan `GetItem` menggunakan klien DynamoDB. Dalam skenario ini, DAX dan DynamoDB memegang nilai yang tidak konsisten untuk kunci yang sama hingga TTL untuk item DAX kedaluwarsa.

Jika aplikasi memodifikasi data dalam tabel DynamoDB dasar dan melewati DAX, aplikasi perlu mengantisipasi dan menoleransi inkonsistensi data yang mungkin terjadi.

> **Note**
>
> Selain `GetItem`, klien DAX juga mendukung permintaan `BatchGetItem`. `BatchGetItem` pada dasarnya adalah pembungkus satu atau beberapa permintaan `GetItem`, sehingga DAX memperlakukan masing-masing sebagai operasi `GetItem` individu.

Konsistensi penulisan

DAX adalah cache write-through, yang memudahkan proses mempertahankan konsistensi cache item DAX dengan tabel DynamoDB dasar.

Klien DAX mendukung operasi API penulisan yang sama seperti DynamoDB (`PutItem`, `UpdateItem`, `DeleteItem`, `BatchWriteItem`, dan `TransactWriteItems`). Ketika Anda menggunakan operasi ini dengan klien DAX, item yang ada di DAX dan DynamoDB diubah. DAX memperbarui item dalam cache item, terlepas dari nilai TTL untuk item ini.

Misalnya, anggaplah Anda membuat permintaan `GetItem` dari klien DAX untuk membaca item dari tabel `ProductCatalog`. (Kunci partisinya adalah `Id` dan tidak ada kunci urutan.) Anda mengambil item dengan `Id` `101`. Nilai `QuantityOnHand` untuk item itu adalah `42`. DAX menyimpan item dalam cache item dengan TTL tertentu. Untuk contoh ini, TTL diasumsikan 10 menit. Lalu, 3 menit kemudian, aplikasi lain menggunakan klien DAX untuk memperbarui item yang sama sehingga nilai `QuantityOnHand` sekarang adalah `41`. Dengan asumsi bahwa item tidak diperbarui lagi, pembacaan berikutnya untuk item yang sama dalam 10 menit ke depan akan menampilkan nilai cache untuk `QuantityOnHand` (`41`).

Cara DAX memproses penulisan

DAX ditujukan untuk aplikasi yang memerlukan pembacaan performa tinggi. Sebagai cache write-through, DAX meneruskan penulisan Anda ke DynamoDB secara bersamaan, kemudian mereplikasi pembaruan yang dihasilkan untuk cache item Anda di semua simpul dalam klaster secara otomatis dan asinkron. Anda tidak perlu mengelola logika pembatalan cache karena DAX menanganinya untuk Anda.

DAX mendukung operasi penulisan berikut: `PutItem`, `UpdateItem`, `DeleteItem`, `BatchWriteItem`, dan `TransactWriteItems`.

Ketika Anda mengirim permintaan `PutItem`, `UpdateItem`, `DeleteItem`, atau `BatchWriteItem` ke DAX, hal berikut akan terjadi:

- DAX mengirimkan permintaan ke DynamoDB.
- DynamoDB membalas DAX, mengonfirmasi bahwa penulisan berhasil.
- DAX menulis item ke cache item.
- DAX menampilkan status berhasil ke pemohon.

Ketika Anda mengirim permintaan `TransactWriteItems` ke DAX, hal berikut akan terjadi:

- DAX mengirimkan permintaan ke DynamoDB.
- DynamoDB membalas DAX, mengonfirmasi bahwa transaksi selesai.
- DAX menampilkan status berhasil ke pemohon.
- Di latar belakang, DAX membuat permintaan `TransactGetItems` untuk setiap item dalam permintaan `TransactWriteItems` guna menyimpan item dalam cache item. `TransactGetItems` digunakan untuk memastikan isolasi yang dapat diserialisasi.

Jika penulisan ke DynamoDB gagal dengan alasan apa pun, termasuk throttling, item tidak di-cache di DAX. Pengecualian untuk kegagalan ditampilkan ke pemohon. Hal ini memastikan bahwa data tidak ditulis ke cache DAX kecuali sudah berhasil ditulis ke DynamoDB.

> **Note**
>
> Setiap penulisan ke DAX akan mengubah status cache item. Namun, menulis ke cache item tidak memengaruhi cache kueri. (Cache item dan cache kueri DAX digunakan untuk tujuan yang berbeda dan beroperasi secara terpisah dari satu sama lain).

Perilaku cache kueri DAX

DAX melakukan cache untuk hasil dari permintaan `Query` dan `Scan` dalam cache kueri. Namun, hasil tersebut tidak memengaruhi cache item sama sekali. Saat aplikasi Anda mengajukan permintaan `Query` atau `Scan` dengan DAX, set hasil disimpan di cache kueri, bukan di cache item. Anda tidak dapat melakukan "pemanasan" cache item dengan melakukan operasi `Scan` karena cache item dan cache kueri adalah entitas terpisah.

Konsistensi query-update-query

Pembaruan untuk cache item atau tabel DynamoDB dasar tidak membatalkan atau mengubah hasil yang disimpan dalam cache kueri.

Sebagai gambaran, pertimbangkan skenario berikut. Aplikasi bekerja dengan tabel `DocumentRevisions`, yang memiliki `DocId` sebagai kunci partisi dan `RevisionNumber` sebagai kunci urutan.

1. Klien mengeluarkan `Query` untuk `DocId 101`, untuk semua item dengan `RevisionNumber` lebih dari atau sama dengan 5. DAX menyimpan set hasil dalam cache kueri dan mengembalikan set hasil untuk pengguna.
2. Klien mengeluarkan permintaan `PutItem` untuk `DocId 101` dengan nilai `RevisionNumber 20`.
3. Klien mengeluarkan `Query` yang sama seperti yang dijelaskan pada langkah 1 (`DocId 101` dan `RevisionNumber >= 5`).

Dalam skenario ini, set hasil yang di-cache untuk `Query` yang dikeluarkan di langkah 3 identik dengan set hasil yang di-cache di langkah 1. Alasannya adalah karena DAX tidak membatalkan set hasil `Query` atau `Scan` berdasarkan pembaruan untuk masing-masing item. Operasi `PutItem` dari langkah 2 hanya tercermin dalam cache kueri DAX ketika TTL untuk `Query` kedaluwarsa.

Aplikasi Anda harus mempertimbangkan nilai TTL untuk cache kueri dan berapa lama aplikasi Anda dapat menoleransi hasil yang tidak konsisten antara cache kueri dan cache item.

Bacaan sangat konsisten dan transaksional

Untuk melakukan permintaan `GetItem`, `BatchGetItem`, `Query`, atau `Scan` yang sangat konsisten, Anda perlu mengatur parameter `ConsistentRead` ke benar. DAX meneruskan permintaan bacaan sangat konsisten ke DynamoDB. Ketika menerima respons dari DynamoDB, DAX mengembalikan hasil ke klien, tetapi tidak melakukan cache hasil. DAX tidak dapat menyediakan bacaan sangat konsisten sendiri karena tidak digabungkan dengan erat ke DynamoDB. Untuk alasan ini, setiap pembacaan berikutnya dari DAX harus merupakan bacaan akhir konsisten. Setiap bacaan sangat konsisten berikutnya juga harus diteruskan ke DynamoDB.

DAX menangani permintaan `TransactGetItems` dengan cara yang sama seperti bacaan sangat konsisten. DAX meneruskan semua permintaan `TransactGetItems` ke DynamoDB. Ketika menerima respons dari DynamoDB, DAX mengembalikan hasil ke klien, tetapi tidak melakukan cache hasil.

Caching negatif

DAX mendukung entri cache negatif baik di cache item dan di cache kueri. Entri cache negatif terjadi ketika DAX tidak dapat menemukan item yang diminta dalam tabel DynamoDB dasar. DAX melakukan cache hasil kosong dan mengembalikan hasil ke pengguna, bukan menghasilkan pesan kesalahan.

Misalnya, anggaplah aplikasi mengirimkan permintaan `GetItem` ke klaster DAX dan tidak ada item yang cocok dalam cache item DAX. Hal ini menyebabkan DAX membaca item yang sesuai dari tabel DynamoDB dasar. Jika item tidak ada di DynamoDB, DAX menyimpan item kosong dalam cache item dan mengembalikan item kosong ke aplikasi. Sekarang, anggaplah bahwa aplikasi mengirimkan permintaan `GetItem` lain untuk item yang sama. DAX menemukan item kosong dalam cache item dan langsung mengembalikannya ke aplikasi. DAX tidak melihat DynamoDB sama sekali.

Entri cache negatif tetap ada di cache item DAX hingga TTL item kedaluwarsa, LRU diinvokasi, atau item diubah menggunakan `PutItem`, `UpdateItem`, atau `DeleteItem`.

Cache kueri DAX menangani hasil cache negatif dengan cara yang sama. Jika aplikasi melakukan `Query` atau `Scan` dan cache kueri DAX tidak berisi hasil cache, DAX mengirimkan permintaan ke DynamoDB. Jika tidak ada item yang cocok dalam set hasil, DAX menyimpan set hasil kosong di cache kueri dan mengembalikan set hasil kosong ke aplikasi. Permintaan `Query` atau `Scan` berikutnya menghasilkan set hasil (kosong) yang sama, hingga TTL untuk hasil tersebut kedaluwarsa.

Strategi untuk penulisan

Perilaku write-through DAX sesuai untuk banyak pola aplikasi. Namun, ada beberapa pola aplikasi yang mungkin tidak cocok dengan model write-through.

Untuk aplikasi yang sensitif terhadap latensi, menulis melalui DAX menimbulkan lompatan jaringan tambahan. Jadi menulis ke DAX akan sedikit lebih lambat dibandingkan menulis langsung ke DynamoDB. Jika aplikasi Anda sensitif terhadap latensi tulis, Anda dapat mengurangi latensi dengan menulis langsung ke DynamoDB. Untuk informasi selengkapnya, lihat Write-around.

Untuk aplikasi dengan operasi tulis intensif (seperti yang melakukan pemuatan data massal), Anda mungkin tidak ingin menulis semua data melalui DAX karena hanya sebagian kecil dari data yang akan dibaca oleh aplikasi. Ketika Anda menulis sejumlah besar data melalui DAX, DAX harus menginvokasi algoritma LRU agar menyediakan ruang di cache untuk item baru yang akan dibaca. Hal ini mengurangi efektivitas DAX sebagai cache baca.

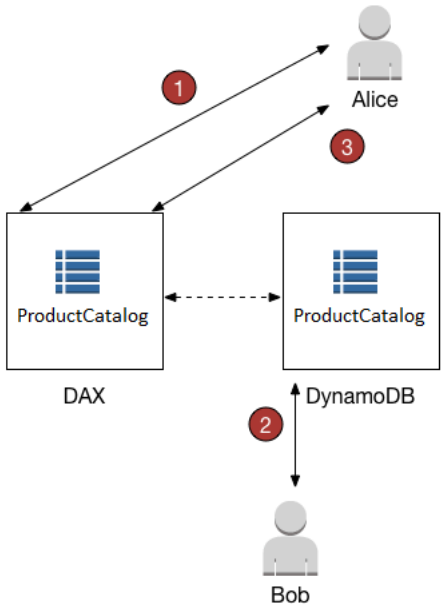
Ketika Anda menulis item ke DAX, status cache item diubah untuk mengakomodasi item baru. (Misalnya, DAX mungkin perlu menghapus data lama dari cache item untuk menyediakan ruang bagi item baru.) Item baru tetap dalam cache item, tunduk pada algoritma LRU cache dan pengaturan TTL untuk cache. Selama item tetap ada dalam cache item, DAX tidak membaca ulang item dari DynamoDB.

Write-through

Cache item DAX menerapkan kebijakan write-through. Untuk informasi selengkapnya, lihat Cara DAX memproses penulisan.

Ketika Anda menulis item, DAX memastikan bahwa item cache disinkronkan dengan item yang ada di DynamoDB. Hal ini sangat membantu untuk aplikasi yang perlu membaca ulang item segera setelah menulisnya. Namun, jika aplikasi lain menulis langsung ke tabel DynamoDB, item dalam cache item DAX tidak lagi sinkron dengan DynamoDB.

Sebagai gambaran, pertimbangkan dua pengguna (Alice dan Bob), yang bekerja dengan tabel `ProductCatalog`. Alice mengakses tabel menggunakan DAX, tetapi Bob melewati DAX dan mengakses tabel secara langsung di DynamoDB.



1. Alice memperbarui item di tabel `ProductCatalog`. DAX meneruskan permintaan ke DynamoDB dan pembaruan berhasil. Kemudian, DAX menulis item ke cache item dan mengembalikan respons berhasil ke Alice. Sejak saat itu sampai item tersebut akhirnya dihapus dari cache, setiap pengguna yang membaca item dari DAX akan melihat item dengan pembaruan Alice.
2. Beberapa waktu kemudian, Bob memperbarui item `ProductCatalog` yang sama seperti yang ditulis Alice. Namun, Bob memperbarui item secara langsung di DynamoDB. DAX tidak secara otomatis memperbarui item cache sebagai respons pembaruan melalui DynamoDB. Oleh karena itu, pengguna DAX tidak melihat pembaruan Bob.
3. Alice membaca item dari DAX lagi. Item ada di cache item, jadi DAX mengembalikannya ke Alice tanpa mengakses tabel DynamoDB.

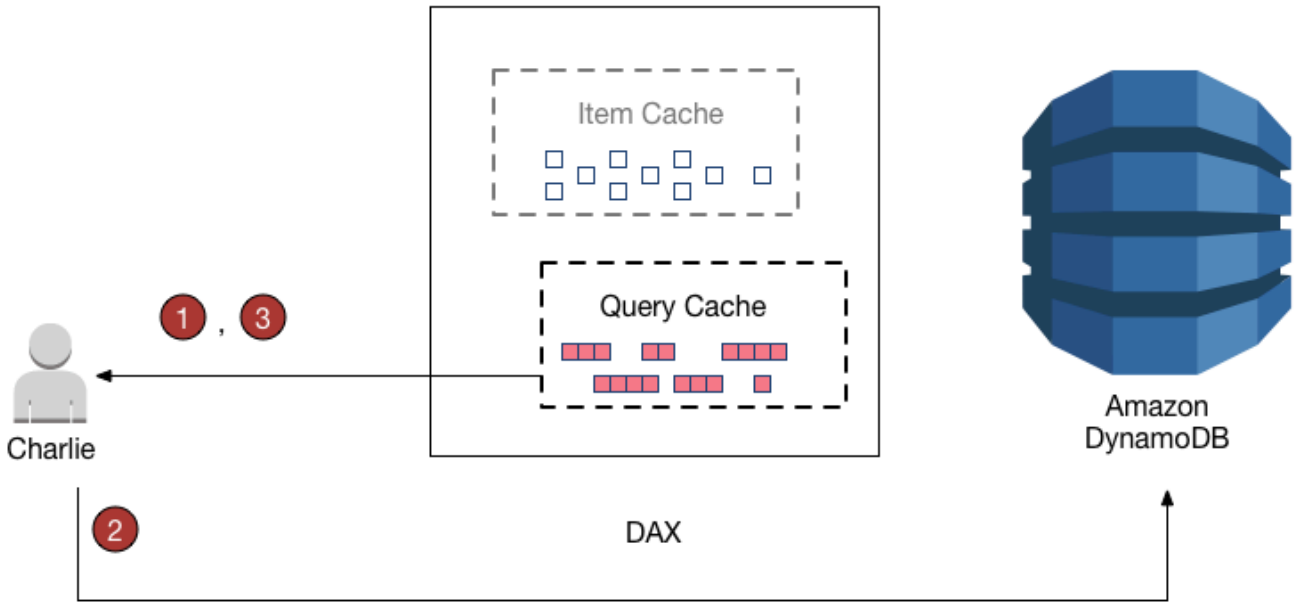
Dalam skenario ini, Alice dan Bob melihat representasi yang berbeda dari item `ProductCatalog` yang sama. Hal ini terjadi hingga DAX mengeluarkan item dari cache item, atau hingga pengguna lain memperbarui item yang sama lagi menggunakan DAX.

Write-around

Jika aplikasi Anda perlu menulis data dalam jumlah besar (seperti beban data massal), mungkin wajar untuk melewati DAX dan menulis data langsung ke DynamoDB. Strategi seperti write-around mengurangi latensi penulisan. Namun, cache item tidak akan sinkron dengan data di DynamoDB.

Jika Anda memutuskan untuk menggunakan strategi write-around, ingat bahwa DAX mengisi cache item setiap kali aplikasi menggunakan klien DAX untuk membaca data. Hal ini dapat menguntungkan dalam beberapa kasus karena memastikan bahwa hanya data yang paling sering dibaca yang di-cache (kebalikan dari data yang paling sering ditulis).

Sebagai contoh, pertimbangkan pengguna (Charlie) yang ingin bekerja dengan tabel yang berbeda, tabel `GameScores`, menggunakan DAX. Kunci partisi untuk `GameScores` adalah `UserId`, sehingga semua skor Charlie akan memiliki `UserId` yang sama.



1. Charlie ingin mengambil semua skornya, jadi dia mengirimkan `Query` ke DAX. Dengan asumsi bahwa permintaan ini belum dikeluarkan sebelumnya, DAX meneruskan kueri ke DynamoDB untuk diproses. Hasil akan disimpan dalam cache kueri DAX, dan kemudian dikembalikan ke Charlie. Set hasil tetap tersedia dalam cache kueri hingga dihapus.
2. Sekarang anggaplah bahwa Charlie memainkan game Meteor Blasters dan mencapai skor tinggi. Charlie mengirim permintaan `UpdateItem` ke DynamoDB, memodifikasi item dalam tabel `GameScores`.
3. Terakhir, Charlie memutuskan untuk menjalankan kembali `Query` sebelumnya untuk mengambil semua data dari `GameScores`. Charlie tidak melihat skor tinggi untuk Meteor Blasters dalam hasil. Hal ini karena hasil kueri berasal dari cache kueri, bukan cache item. Kedua cache terpisah dari satu sama lain, sehingga perubahan dalam satu cache tidak memengaruhi cache lainnya.

DAX tidak memperbarui set hasil dalam cache kueri dengan data terbaru dari DynamoDB. Setiap set hasil dalam cache kueri adalah hasil terbaru saat operasi `Query` atau `Scan` dilakukan. Jadi, hasil `Query` Charlie tidak mencerminkan operasi `PutItem`. Hal ini terjadi hingga DAX menghapus set hasil dari cache kueri.

Melakukan pengembangan dengan klien DynamoDB Accelerator (DAX)

Untuk menggunakan DAX dari aplikasi, Anda menggunakan klien DAX untuk bahasa pemrograman. Klien DAX dirancang agar tidak banyak mengganggu aplikasi Amazon DynamoDB Anda yang sudah ada hanya dengan beberapa modifikasi kode sederhana.

> **Note**
>
> Klien DAX untuk berbagai bahasa pemrograman tersedia di situs berikut:
>
> - http://dax-sdk.s3 - website-us-west -2.amazonaws.com

Bagian ini menunjukkan cara meluncurkan EC2 instans Amazon di VPC Amazon default Anda, menyambung ke instance, dan menjalankan contoh aplikasi. Bagian ini juga menyediakan informasi tentang cara mengubah aplikasi yang ada sehingga dapat menggunakan klaster DAX.

Topik

- Tutorial: Menjalankan aplikasi sampel menggunakan DynamoDB Accelerator (DAX)
- Memodifikasi aplikasi yang ada untuk menggunakan DAX

Tutorial: Menjalankan aplikasi sampel menggunakan DynamoDB Accelerator (DAX)

Tutorial ini menunjukkan cara meluncurkan EC2 instans Amazon di virtual private cloud (VPC) default Anda, sambungkan ke instans, dan jalankan contoh aplikasi yang menggunakan Amazon DynamoDB Accelerator (DAX).

> **Note**
>
> Untuk menyelesaikan tutorial ini, Anda harus menjalankan klaster DAX di VPC default. Jika Anda belum membuat klaster DAX, lihat petunjuk di Membuat klaster DAX.

Topik

- Langkah 1: Luncurkan EC2 instans Amazon
- Langkah 2: Buat pengguna dan kebijakan
- Langkah 3: Konfigurasikan EC2 instans Amazon
- Langkah 4: Jalankan aplikasi sampel

Langkah 1: Luncurkan EC2 instans Amazon

Jika klaster Amazon DynamoDB Accelerator (DAX) tersedia, Anda dapat meluncurkan instans Amazon di EC2 Amazon Virtual Private Cloud (Amazon VPC) default. Kemudian Anda dapat menginstal dan menjalankan perangkat lunak klien DAX pada instans tersebut.

Untuk meluncurkan sebuah EC2 instance

1. Masuk ke AWS Management Console dan buka EC2 konsol Amazon di https://console.aws.amazon.com/ec2/.
2. Pilih Luncurkan Instans dan lakukan hal berikut:

   Langkah 1: Pilih Amazon Machine Image (AMI)

   1. Dalam daftar AMIs, temukan Amazon Linux AMI, dan pilih Pilih.

   Langkah 2: Pilih Jenis Instans

   1. Dalam daftar jenis instans, pilih t2.micro.
   2. Pilih Berikutnya: Konfigurasikan Detail Instans.

   Langkah 3: Konfigurasikan Detail Instans

   1. Untuk Jaringan, pilih VPC default Anda.

2. Pilih Berikutnya: Tambahkan Penyimpanan.

Langkah 4: Tambahkan Penyimpanan

1. Lewati langkah ini dengan memilih Berikutnya: Tambahkan Tanda.

Langkah 5: Tambahkan Tanda

1. Lewati langkah ini dengan memilih Berikutnya: Konfigurasikan Grup Keamanan.

Langkah 6: Konfigurasikan Grup Keamanan

1. Pilih Pilih grup keamanan yang ada.
2. Dalam daftar grup keamanan, pilih default. Ini adalah grup keamanan default untuk VPC Anda.
3. Pilih Berikutnya: Tinjau dan Luncurkan.

Langkah 7: Tinjau Peluncuran Instans

1. Pilih Luncurkan.

3. Di jendela Pilih pasangan kunci yang ada atau buat pasangan kunci baru, lakukan salah satu hal berikut:

   - Jika Anda tidak memiliki EC2 key pair Amazon, pilih Create a new key pair dan ikuti petunjuknya. Anda diminta untuk mengunduh file kunci privat (file `.pem`). Anda memerlukan file ini nanti saat Anda masuk ke EC2 instans Amazon Anda.
   - Jika Anda sudah memiliki EC2 key pair Amazon yang sudah ada, buka Select a key pair dan pilih key pair Anda dari daftar. Anda harus sudah memiliki file kunci pribadi (`.pem`file) yang tersedia untuk masuk ke EC2 instans Amazon Anda.

4. Setelah mengonfigurasi pasangan kunci, pilih Luncurkan Instans.
5. Di panel navigasi konsol, pilih EC2 Dasbor, lalu pilih instance yang Anda luncurkan. Pada panel bawah, di tab Deskripsi, cari DNS Publik untuk instans Anda, misalnya: `ec2-11-22-33-44.us-west-2.compute.amazonaws.com`. Catat nama DNS publik ini karena Anda membutuhkannya untuk Langkah 3: Konfigurasikan EC2 instans Amazon.

> **Note**
>
> Dibutuhkan beberapa menit agar EC2 instans Amazon Anda tersedia. Sementara itu, lanjutkan ke Langkah 2: Buat pengguna dan kebijakan dan ikuti petunjuk di sana.

Langkah 2: Buat pengguna dan kebijakan

Pada langkah ini, Anda membuat pengguna dengan kebijakan yang memberikan akses ke klaster Amazon DynamoDB Accelerator (DAX) dan menggunakan DynamoDB. AWS Identity and Access Management Kemudian, Anda dapat menjalankan aplikasi yang berinteraksi dengan klaster DAX Anda.

Mendaftar untuk Akun AWS

Jika Anda tidak memiliki Akun AWS, selesaikan langkah-langkah berikut untuk membuatnya.

Untuk mendaftar untuk Akun AWS

1. Buka https://portal.aws.amazon.com/billing/pendaftaran.
2. Ikuti petunjuk online.

   Bagian dari prosedur pendaftaran melibatkan menerima panggilan telepon atau pesan teks dan memasukkan kode verifikasi pada keypad telepon.

   Saat Anda mendaftar untuk sebuah Akun AWS, sebuah Pengguna root akun AWSdibuat. Pengguna root memiliki akses ke semua Layanan AWS dan sumber daya di akun. Sebagai praktik keamanan terbaik, tetapkan akses administratif ke pengguna, dan gunakan hanya pengguna root untuk melakukan tugas yang memerlukan akses pengguna root.

AWS mengirimkan email konfirmasi setelah proses pendaftaran selesai. Kapan saja, Anda dapat melihat aktivitas akun Anda saat ini dan mengelola akun Anda dengan masuk https://aws.amazon.comke/ dan memilih Akun Saya.

Buat pengguna dengan akses administratif

Setelah Anda mendaftar Akun AWS, amankan Pengguna root akun AWS, aktifkan AWS IAM Identity Center, dan buat pengguna administratif sehingga Anda tidak menggunakan pengguna root untuk tugas sehari-hari.

Amankan Anda Pengguna root akun AWS

1. Masuk ke AWS Management Consolesebagai pemilik akun dengan memilih pengguna Root dan memasukkan alamat Akun AWS email Anda. Di laman berikutnya, masukkan kata sandi.

   Untuk bantuan masuk dengan menggunakan pengguna root, lihat Masuk sebagai pengguna root di AWS Sign-In Panduan Pengguna.
2. Mengaktifkan autentikasi multi-faktor (MFA) untuk pengguna root Anda.

   Untuk petunjuk, lihat Mengaktifkan perangkat MFA virtual untuk pengguna Akun AWS root (konsol) Anda di Panduan Pengguna IAM.

Buat pengguna dengan akses administratif

1. Aktifkan Pusat Identitas IAM.

   Untuk mendapatkan petunjuk, silakan lihat Mengaktifkan AWS IAM Identity Center di Panduan Pengguna AWS IAM Identity Center .
2. Di Pusat Identitas IAM, berikan akses administratif ke pengguna.

   Untuk tutorial tentang menggunakan Direktori Pusat Identitas IAM sebagai sumber identitas Anda, lihat Mengkonfigurasi akses pengguna dengan default Direktori Pusat Identitas IAM di Panduan AWS IAM Identity Center Pengguna.

Masuk sebagai pengguna dengan akses administratif

- Untuk masuk dengan pengguna Pusat Identitas IAM, gunakan URL masuk yang dikirim ke alamat email saat Anda membuat pengguna Pusat Identitas IAM.

  Untuk bantuan masuk menggunakan pengguna Pusat Identitas IAM, lihat Masuk ke portal AWS akses di Panduan AWS Sign-In Pengguna.

Tetapkan akses ke pengguna tambahan

1. Di Pusat Identitas IAM, buat set izin yang mengikuti praktik terbaik menerapkan izin hak istimewa paling sedikit.

   Untuk petunjuknya, lihat Membuat set izin di Panduan AWS IAM Identity Center Pengguna.

2. Tetapkan pengguna ke grup, lalu tetapkan akses masuk tunggal ke grup.

   Untuk petunjuk, lihat Menambahkan grup di Panduan AWS IAM Identity Center Pengguna.

Untuk memberikan akses dan menambahkan izin bagi pengguna, grup, atau peran Anda:

- Pengguna dan grup di AWS IAM Identity Center:

  Buat rangkaian izin. Ikuti instruksi di Buat rangkaian izin di Panduan Pengguna AWS IAM Identity Center .
- Pengguna yang dikelola di IAM melalui penyedia identitas:

  Buat peran untuk federasi identitas. Ikuti instruksi dalam Buat peran untuk penyedia identitas pihak ketiga (federasi) dalam Panduan Pengguna IAM.
- Pengguna IAM:
  - Buat peran yang dapat diambil pengguna Anda. Ikuti instruksi dalam Buat peran untuk pengguna IAM dalam Panduan Pengguna IAM.
  - (Tidak disarankan) Lampirkan kebijakan langsung ke pengguna atau tambahkan pengguna ke grup pengguna. Ikuti petunjuk di Menambahkan izin ke pengguna (konsol) dalam Panduan Pengguna IAM.

Cara menggunakan editor kebijakan JSON untuk membuat kebijakan

1. Masuk ke AWS Management Console dan buka konsol IAM di https://console.aws.amazon.com/iam/.
2. Pada panel navigasi di sebelah kiri, pilih Kebijakan.

   Jika ini pertama kalinya Anda memilih Kebijakan, akan muncul halaman Selamat Datang di Kebijakan Terkelola. Pilih Memulai.
3. Di bagian atas halaman, pilih Buat kebijakan.
4. Di bagian Editor kebijakan, pilih opsi JSON.
5. Masukkan atau tempel dokumen kebijakan JSON. Untuk detail bahasa kebijakan IAM, lihat Referensi kebijakan JSON IAM.
6. Selesaikan peringatan keamanan, kesalahan, atau peringatan umum yang dihasilkan selama validasi kebijakan, lalu pilih Berikutnya.

> **Note**
>
> Anda dapat beralih antara opsi editor Visual dan JSON kapan saja. Namun, jika Anda melakukan perubahan atau memilih Berikutnya di editor Visual, IAM dapat merestrukturisasi kebijakan Anda untuk mengoptimalkannya bagi editor visual. Untuk informasi selengkapnya, lihat Restrukturisasi kebijakan dalam Panduan Pengguna IAM.

7. (Opsional) Saat membuat atau mengedit kebijakan AWS Management Console, Anda dapat membuat templat kebijakan JSON atau YAMB yang dapat Anda gunakan dalam AWS CloudFormation templat.

   Untuk melakukan ini, di editor Kebijakan pilih Tindakan, lalu pilih Buat CloudFormation templat. Untuk mempelajari selengkapnya AWS CloudFormation, lihat referensi jenis AWS Identity and Access Management sumber daya di Panduan AWS CloudFormation Pengguna.
8. Setelah selesai menambahkan izin ke kebijakan, pilih Berikutnya.
9. Pada halaman Tinjau dan buat, masukkan Nama kebijakan dan Deskripsi (opsional) untuk kebijakan yang Anda buat. Tinjau Izin yang ditentukan dalam kebijakan ini untuk melihat izin yang diberikan oleh kebijakan Anda.
10. (Opsional) Tambahkan metadata ke kebijakan dengan melampirkan tanda sebagai pasangan nilai kunci. Untuk informasi selengkapnya tentang penggunaan tag di IAM, lihat Tag untuk AWS Identity and Access Management sumber daya di Panduan Pengguna IAM.
11. Pilih Buat kebijakan untuk menyimpan kebijakan baru Anda.

Dokumen kebijakan – Salin dan tempel dokumen berikut untuk membuat kebijakan JSON.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Action": [
                "dax:*"
            ],
            "Effect": "Allow",
            "Resource": [
                "*"
            ]
        },
```

```
        {
            "Action": [
                "dynamodb:*"
            ],
            "Effect": "Allow",
            "Resource": [
                "*"
            ]
        }
    ]
}
```

Langkah 3: Konfigurasikan EC2 instans Amazon

Ketika EC2 instans Amazon Anda tersedia, Anda dapat masuk ke instans dan menyiapkannya untuk digunakan.

> **Note**
>
> Langkah-langkah berikut mengasumsikan bahwa Anda terhubung ke EC2 instans Amazon Anda dari komputer yang menjalankan Linux. Untuk cara lain untuk terhubung, lihat Connect to Your Linux Instance di Panduan EC2 Pengguna Amazon.

Untuk mengkonfigurasi EC2 instance

1. Buka EC2 konsol Amazon di https://console.aws.amazon.com/ec2/.
2. Gunakan `ssh` perintah untuk masuk ke EC2 instance Amazon Anda, seperti yang ditunjukkan pada contoh berikut.

   ```
   ssh -i my-keypair.pem ec2-user@public-dns-name
   ```

   Anda perlu menentukan file kunci privat (file `.pem`) dan nama DNS publik instans Anda. (Lihat Langkah 1: Luncurkan EC2 instans Amazon).

   ID login adalah `ec2-user`. Tidak diperlukan kata sandi.
3. Setelah Anda masuk ke EC2 instans Anda, konfigurasikan AWS kredensional Anda seperti yang ditunjukkan berikut. Masukkan ID kunci AWS akses dan kunci rahasia (fromLangkah 2: Buat

pengguna dan kebijakan), dan atur nama Region default ke Region Anda saat ini. (Pada contoh berikut, nama Wilayah default adalah `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]:
```

Setelah meluncurkan dan mengonfigurasi EC2 instans Amazon, Anda dapat menguji fungsionalitas DAX menggunakan salah satu contoh aplikasi yang tersedia. Untuk informasi selengkapnya, lihat Langkah 4: Jalankan aplikasi sampel.

Langkah 4: Jalankan aplikasi sampel

Untuk membantu menguji fungsionalitas Amazon DynamoDB Accelerator (DAX), Anda dapat menjalankan salah satu contoh aplikasi yang tersedia di instans Amazon Anda. EC2

Topik

- Node.js dan DAX
- DAX SDK untuk Go
- Java dan DAX
- .NET dan DAX
- Python dan DAX

Node.js dan DAX

Konfigurasi klien default untuk Node.js

Saat mengonfigurasi klien DAX JavaScript SDK, Anda dapat menyesuaikan berbagai parameter untuk mengoptimalkan kinerja, penanganan koneksi, dan ketahanan kesalahan. Tabel berikut menguraikan pengaturan konfigurasi default yang mengontrol cara klien Anda berinteraksi dengan klaster DAX, termasuk nilai batas waktu, mekanisme coba lagi, manajemen kredensi, dan opsi pemantauan kesehatan. Untuk informasi selengkapnya, lihat DBClient Operasi Dynamo.

Default klien DAX JS SDK

Parameter	Jenis	Deskripsi
`region` opsional	`string`	Wilayah AWS Untuk digunakan untuk klien DAX (contoh - 'us-east-1'). Ini adalah parameter yang diperlukan jika tidak disediaka n melalui variabel lingkungan.
`endpoint` wajib	`string`	Titik akhir Cluster yang terhubung dengan SDK. Contoh: Tidak terenkripsi — .region.a mazonaws.com dax-cluster-name Terenkripsi - daxs: //my-clus ter.l6fzcv. dax-clusters.us-ea st-1.amazonaws.com
`requestTimeout` default 6000 ms	`number`	Ini menentukan waktu maksimum klien akan menunggu respons dari DAX.
`writeRetries` default 1	`number`	Jumlah percobaan ulang untuk mencoba permintaan tulis yang gagal.
`readRetries` default 1	`number`	Jumlah percobaan ulang untuk mencoba permintaan baca yang gagal.
`maxRetries` default 1	`number`	Jumlah maksimum percobaan ulang untuk mencoba permintaan yang gagal.

Parameter	Jenis	Deskripsi
		Jika readRetries/writeRetries disetel, maka konfigurasi yang disetel di ReadRetries dan WriteRetries diprioritaskan daripada maxRetries.
`connectTimeout` default 10000 ms	`number`	Batas waktu (dalam milidetik) untuk membuat koneksi ke salah satu node cluster.
`maxConcurrentConnections` standar 100	`number`	Membatasi jumlah total koneksi bersamaan yang dapat dibuat oleh instance klien per node dalam cluster DAX.

Parameter	Jenis	Deskripsi
`maxRetryDelay` default 7000 ms	`number`	Ketika server DAX menunjukkan pemulihan diperlukan dengan menyetel `waitForRecoveryBeforeRetrying` flag ke true, klien akan berhenti sebelum mencoba lagi. Selama periode pemulihan ini, `maxRetryDelay` parameter menentukan waktu tunggu maksimum antara percobaan ulang. Konfigurasi khusus pemulihan ini hanya berlaku ketika server DAX dalam mode pemulihan. Untuk semua skenario lainnya, perilaku coba lagi mengikuti salah satu dari dua pola: baik penundaan eksponensial berdasarkan jumlah coba lagi (diatur oleh`writeRetries` ,, atau `maxRetries` parameter)`readRetries` , atau coba lagi langsung tergantung pada jenis pengecualian.

Parameter	Jenis	Deskripsi
`credentials` opsional	`AwsCredentialIdentity` | `AwsCredentialIdentityProvider`	AWS Kredensyal yang digunakan untuk mengautentikasi permintaan. Ini dapat diberikan sebagai AwsCredentialIdentity atau AwsCredentialIdentityProvider. Jika tidak disediakan, AWS SDK akan secara otomatis menggunakan rantai penyedia kredensi default. Contoh: `{ accessKeyId: 'AKIA... ' , secretAccessKey: ' ... ', SessionToken:'... '} `* @default Menggunakan rantai penyedia kredensi AWS default.
`healthCheckInterval` default 5000 ms	`number`	Interval (dalam milidetik) antara pemeriksaan kesehatan cluster. Interval yang lebih rendah akan lebih sering diperiksa.
`healthCheckTimeout` default 1000 ms	`number`	Batas waktu (dalam milidetik) untuk pemeriksaan kesehatan selesai.

Parameter	Jenis	Deskripsi
`skipHostnameVerification` default palsu	`boolean`	Lewati verifikasi nama host koneksi TLS. Ini tidak berdampak pada cluster yang tidak terenkripsi. Standarnya adalah melakukan verifikasi nama host, menyetel ini ke True akan melewati verifikasi. Pastikan Anda memahami implikasi dari mematikannya, yang merupakan ketidakmampuan untuk mengotentikasi cluster yang Anda sambungkan.
`unhealthyConsecutiveErrorCount` default 5	`number`	Menetapkan jumlah kesalahan berturut-turut yang diperlukan untuk memberi sinyal node tidak sehat dalam interval pemeriksaan kesehatan.
`clusterUpdateInterval` default 4000 ms	`number`	Mengembalikan interval antara polling anggota cluster untuk perubahan keanggotaan.
`clusterUpdateThreshold` standar 125	`number`	Mengembalikan ambang batas di bawah mana cluster tidak akan disurvei untuk perubahan keanggotaan.
`credentailProvider` opsional | default null	`AwsCredentialIdentityProvider`	Penyedia yang Ditetapkan Pengguna untuk AWS kredensil yang digunakan untuk mengautentikasi permintaan ke DAX.

Konfigurasi pagination untuk DaxDocument

Nama	Tipe	Detail
`client`	DaxDocument	Contoh DaxDocument tipe.
`pageSize`	number	Menentukan jumlah item per halaman.
`startingToken` Opsional	any	LastEvaluatedKey dari tanggapan sebelumnya dapat digunakan untuk permintaan berikutnya.

Untuk penggunaan pagination, lihatthe section called “TryDax.js”.

Migrasi ke DAX Node.js SDK V3

Panduan migrasi ini akan membantu Anda mentransisikan aplikasi DAX Node.js yang ada. SDK baru memerlukan Node.js 18 atau lebih tinggi dan memperkenalkan beberapa perubahan penting dalam cara Anda menyusun kode DynamoDB Accelerator Anda. Panduan ini akan memandu Anda melalui perbedaan utama, termasuk perubahan sintaks, metode impor baru, dan pola pemrograman asinkron yang diperbarui.

V2 Node.js penggunaan 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);
    }
});
```

Penggunaan V3 Node.js DAX

Untuk Menggunakan DAX Node.js V3 Node versi 18 atau lebih tinggi adalah versi yang disukai. Untuk pindah ke Node 18, gunakan yang berikut ini:

```
// 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)
}
```

DAX SDK untuk Node.js v3.x kompatibel dengan AWS SDK untuk Node.js v3.x. DAX SDK untuk Node.js v3.x mendukung penggunaan klien agregat. Harap dicatat bahwa DAX tidak mendukung pembuatan klien telanjang. Untuk detail selengkapnya tentang fitur yang tidak didukung, lihatthe section called “Fitur tidak paritas dengan AWS SDK V3”.

Ikuti langkah-langkah ini untuk menjalankan contoh aplikasi Node.js di EC2 instans Amazon Anda.

Cara menjalankan sampel Node.js untuk DAX

1. Siapkan Node.js di EC2 instans Amazon Anda, sebagai berikut:

   a. Instal manajer versi simpul (nvm).

   ```
   curl -o- https://raw.githubusercontent.com/nvm-sh/nvm/v0.40.1/install.sh | bash
   ```

   b. Gunakan nvm untuk menginstal Node.js.

   ```
   nvm install 18
   ```

   c. Gunakan nvm untuk menggunakan Node 18

   ```
   nvm use 18
   ```

   d. Pastikan bahwa Node.js sudah diinstal dan berjalan dengan benar.

   ```
   node -e "console.log('Running Node.js ' + process.version)"
   ```

   Tindakan ini akan menampilkan pesan berikut.

```
Running Node.js v18.x.x
```

2. Instal klien DaxDocument Node.js menggunakan node package manager (npm).

```
npm install @amazon-dax-sdk/lib-dax
```

TryDaxkode sampel

Untuk mengevaluasi manfaat kinerja DynamoDB Accelerator (DAX), ikuti langkah-langkah berikut untuk menjalankan pengujian sampel yang membandingkan waktu operasi baca antara DynamoDB standar dan cluster DAX.

1. Setelah Anda menyiapkan ruang kerja dan menginstal dependensi `lib-dax` sebagai dependensi, salin the section called “TryDax.js” ke proyek Anda.
2. Jalankan program terhadap cluster DAX Anda. Untuk menentukan titik akhir klaster DAX Anda, pilih salah satu dari berikut ini:

   - Menggunakan konsol DynamoDB — Pilih klaster DAX Anda. Titik akhir klaster ditampilkan pada konsol, seperti dalam contoh berikut.

     ```
     dax://my-cluster.l6fzcv.dax-clusters.us-east-1.amazonaws.com
     ```

   - Menggunakan AWS CLI — Masukkan perintah berikut.

     ```
     aws dax describe-clusters --query "Clusters[*].ClusterDiscoveryEndpoint"
     ```

     Titik akhir klaster ditampilkan pada output, seperti dalam contoh berikut.

     ```
     {
         "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. Sekarang jalankan program dengan menentukan titik akhir cluster sebagai parameter baris perintah.

```
node TryDax.js dax://my-cluster.l6fzcv.dax-clusters.us-east-1.amazonaws.com
```

Anda akan melihat output yang serupa dengan yang berikut:

```
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.
```

Catat informasi waktu. Jumlah mikrodetik yang diperlukan untuk`GetItem`,`Query`, `Scan` tes.

4. Dalam hal ini, Anda menjalankan program terhadap cluster DAX. Sekarang, Anda akan menjalankan program lagi, kali ini melawan DynamoDB.
5. Sekarang jalankan program lagi, tapi kali ini, tanpa URL endpoint cluster sebagai parameter baris perintah.

```
node TryDax.js
```

Lihat output dan perhatikan informasi waktu. Waktu yang telah berlalu untuk`GetItem`,`Query`, dan `Scan` harus jauh lebih rendah dengan DAX dibandingkan dengan DynamoDB.

Fitur tidak paritas dengan AWS SDK V3

- Klien telanjang - Dax Node.js V3 tidak mendukung klien telanjang.

```
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 tidak mendukung penggunaan fungsi Middleware.

```
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 untuk Go

Ikuti prosedur ini untuk menjalankan aplikasi sampel Amazon DynamoDB Accelerator (DAX) SDK for Go di instans Amazon Anda. EC2

Cara menjalankan sampel SDK untuk Go untuk DAX

1. Siapkan SDK for Go di instans EC2 Amazon Anda:

   a. Instal bahasa pemrograman Go (`Golang`).

   ```
   sudo yum install -y golang
   ```

   b. Pastikan bahwa Golang sudah diinstal dan berjalan dengan benar.

   ```
   go version
   ```

   Pesan seperti ini akan muncul.

   ```
   go version go1.23.4 linux/amd64
   ```

2. Instal aplikasi Golang sampel.

   ```
   go get github.com/aws-samples/sample-aws-dax-go-v2
   ```

3. Jalankan program Golang berikut. Program pertama membuat tabel DynamoDB bernama `TryDaxGoTable`. Program kedua menulis data ke tabel.

```
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. Jalankan program Golang berikut.

```
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
```

   Perhatikan informasi waktu, yaitu jumlah milidetik yang diperlukan untuk pengujian `GetItem`, `Query`, dan `Scan`.

5. Pada langkah sebelumnya, Anda menjalankan program terhadap titik akhir DynamoDB. Sekarang, jalankan program lagi, tetapi kali ini operasi `GetItem`, `Query`, dan `Scan` diproses oleh klaster DAX Anda.

   Untuk menentukan titik akhir klaster DAX Anda, pilih salah satu dari berikut ini:

   - Menggunakan konsol DynamoDB — Pilih klaster DAX Anda. Titik akhir klaster ditampilkan pada konsol, seperti dalam contoh berikut.

     ```
     dax://my-cluster.l6fzcv.dax-clusters.us-east-1.amazonaws.com
     ```

   - Menggunakan AWS CLI — Masukkan perintah berikut.

     ```
     aws dax describe-clusters --query "Clusters[*].ClusterDiscoveryEndpoint"
     ```

     Titik akhir klaster ditampilkan pada output, seperti dalam contoh berikut.

```
{
    "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"
}
```

Sekarang jalankan program lagi, tetapi kali ini tentukan titik akhir klaster sebagai parameter baris perintah.

```
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
```

Periksa sisa output dan perhatikan informasi waktu. Waktu berlalu untuk `GetItem`, `Query`, dan `Scan` harus jauh lebih rendah dengan DAX dibandingkan dengan DynamoDB.

6. Jalankan program Golang berikut untuk menghapus `TryDaxGoTable`.

```
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
```

Fitur tidak paritas dengan V2 AWS SDK untuk Go

Middleware Stack — DAX Go V2 tidak mendukung penggunaan Middleware Stacks melalui. APIoptions Untuk informasi selengkapnya, lihat Menyesuaikan Permintaan Klien AWS SDK untuk Go v2 dengan Middleware.

Contoh:

```
// 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
```

```
}
```

Output:

```
failed to execute command: custom middleware through APIOptions is not supported in DAX
 client
exit status 1
```

Konfigurasi klien default untuk Go

Panduan ini akan memandu Anda melalui opsi konfigurasi yang memungkinkan Anda untuk menyempurnakan kinerja klien DAX Anda, manajemen koneksi, dan perilaku logging. Dengan memahami pengaturan default dan cara menyesuaikannya, Anda dapat mengoptimalkan interaksi aplikasi Go Anda dengan DAX.

Dalam bagian ini

- Default Klien DAX Go SDK
- Pembuatan klien

Default Klien DAX Go SDK

Parameter	Jenis	Deskripsi
`Region` wajib	`string`	Wilayah AWS Untuk digunakan untuk klien DAX (contoh- 'us-east-1'). Ini adalah parameter yang diperlukan jika tidak disediaka n melalui lingkungan.
`HostPorts` wajib	`[] string`	Daftar titik akhir klaster DAX yang terhubung dengan SDK. Misalnya: Tidak Terenkripsi - dax: //my-cluster.l6fzcv. dax-clusters.us-east-1.amazonaws.com

Parameter	Jenis	Deskripsi
		Terenkripsi - daxs: //my-cluster.l6fzcv. dax-clusters.us-east-1.amazonaws.com
`MaxPendingConnectionsPerHost` default 10	`number`	Jumlah upaya koneksi bersamaan. (Koneksi dapat dalam proses dibuat secara bersamaan.)
`ClusterUpdateThreshold` default 125 * Time.millisecond	`time.Duration`	Waktu minimum yang harus berlalu antara penyegaran cluster.
`ClusterUpdateInterval` default 4 * waktu.detik	`time.Duration`	Interval di mana klien akan secara otomatis menyegarkan informasi cluster DAX.
`IdleConnectionsReapDelay` default 30 * waktu.Detik	`time.Duration`	Interval di mana klien akan menutup koneksi idle di klien DAX.
`ClientHealthCheckInterval` default 5 * waktu.detik	`time.Duration`	Interval di mana klien akan melakukan pemeriksaan kesehatan pada endpoint cluster DAX.
`Credentials` default	`aws.CredentialsProvider`	AWS Kredensi yang digunakan oleh klien DAX untuk mengautentikasi permintaan ke layanan DAX. Lihat Penyedia Kredensyal dan Kredensyal.

Parameter	Jenis	Deskripsi
`DialContext` default	`func`	Fungsi kustom yang digunakan oleh klien DAX untuk membuat koneksi ke cluster DAX.
`SkipHostnameVerification` default palsu	bool	Lewati verifikasi nama host koneksi TLS. Pengaturan ini hanya memengaruhi kluster terenkripsi. Ketika disetel ke True, itu menonaktifkan verifikasi nama host. Menonaktifkan verifikasi berarti Anda tidak dapat mengautentikasi identitas klaster yang Anda sambungkan, yang menimbulkan risiko keamanan. Secara default, verifikasi nama host diaktifkan.
`RouteManagerEnabled` default palsu	`bool`	Bendera ini digunakan untuk menghapus rute yang menghadapi kesalahan jaringan.
`RequestTimeout` default 60 * waktu.detik	`time.Duration`	Ini menentukan waktu maksimum klien akan menunggu respons dari DAX. Prioritas: Batas waktu konteks (jika disetel) > `RequestTimmeout` (jika disetel) > Default 60s`RequestTimeout` .

Parameter	Jenis	Deskripsi
`WriteRetries` default 2	`number`	Jumlah percobaan ulang untuk mencoba permintaan tulis yang gagal.
`ReadRetries` default 2	`number`	Jumlah percobaan ulang untuk mencoba permintaan baca yang gagal.
`RetryDelay` standar 0	`time.Duration`	Penundaan untuk kesalahan non-throttled (dalam hitungan detik) untuk percobaan ulang ketika permintaan gagal.
`Logger` opsional	`logging.Logger`	Logger adalah antarmuka untuk entri logging pada klasifikasi tertentu.
`LogLevel` utilitas default. LogOff	`number`	Loglevel ini didefinisikan hanya untuk DAX. Itu dapat diimpor menggunakan github.com/aws/aws-dax-go-v2/tree/main/dax/utils. `const ( LogOff LogLevelType = 0 LogDebug LogLevelType = 1 LogDebugWithRequestRetries LogLevelType = 2 )`

> **Note**
>
> Untuk`time.Duration`, unit defaultnya adalah nanodetik. Jika kita tidak menentukan unit apa pun untuk parameter apa pun maka itu akan menganggapnya sebagai detik nano: `daxCfg.ClusterUpdateInterval = 10` berarti 10 detik nano. (`daxCfg.ClusterUpdateInterval = 10 * time.Millisecond`berarti 10 milidetik).

Pembuatan klien

Untuk membuat klien DAX:

- Buat konfigurasi DAX, lalu buat klien DAX menggunakan konfigurasi DAX. Dengan menggunakan ini, Anda dapat menimpa konfigurasi DAX jika diperlukan.

```
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)
```

Migrasi ke DAX Go SDK V2

Panduan migrasi ini akan membantu Anda mentransisikan aplikasi DAX Go yang ada.

Penggunaan 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)
}
```

Penggunaan 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)
}
```

Untuk detail penggunaan API selengkapnya, lihat AWS contoh.

Java dan DAX

DAX SDK untuk Java 2.x kompatibel dengan AWS SDK untuk Java 2.x. Ini dibangun di atas Java 8+ dan termasuk dukungan untuk non-blocking I/O. Untuk informasi tentang menggunakan DAX dengan SDK for AWS Java 1.x, lihat. Menggunakan DAX dengan AWS SDK for Java 1.x

Menggunakan klien sebagai dependensi Maven

Ikuti langkah-langkah berikut untuk menggunakan klien untuk DAX SDK untuk Java dalam aplikasi Anda sebagai dependensi.

1. Unduh dan instal Apache Maven. Untuk informasi selengkapnya, lihat Mengunduh Apache Maven dan Menginstal Apache Maven.
2. Tambahkan dependensi Maven klien ke file Project Object Model (POM) aplikasi Anda. Dalam contoh ini, ganti *x.x.x* dengan nomor versi sebenarnya dari klien.

```
<!--Dependency:-->
<dependencies>
    <dependency>
        <groupId>software.amazon.dax</groupId>
        <artifactId>amazon-dax-client</artifactId>
        <version>x.x.x</version>
    </dependency>
</dependencies>
```

TryDax kode sampel

Setelah menyiapkan ruang kerja dan menambahkan DAX SDK sebagai dependensi, salin TryDax.jawa ke proyek Anda.

Jalankan kode menggunakan perintah ini.

```
java -cp classpath TryDax
```

Anda akan melihat output seperti yang berikut ini.

```
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.
```

Perhatikan informasi waktu, yaitu jumlah milidetik yang diperlukan untuk pengujian `GetItem` dan `Query`. Dalam hal ini, Anda menjalankan program terhadap titik akhir DynamoDB. Sekarang Anda akan menjalankan program lagi, kali ini terhadap klaster DAX.

Untuk menentukan titik akhir klaster DAX Anda, pilih salah satu dari berikut ini:

- Pada konsol DynamoDB, pilih klaster DAX Anda. Titik akhir klaster ditampilkan pada konsol, seperti dalam contoh berikut.

  ```
  dax://my-cluster.l6fzcv.dax-clusters.us-east-1.amazonaws.com
  ```

- Menggunakan AWS CLI, masukkan perintah berikut:

  ```
  aws dax describe-clusters --query "Clusters[*].ClusterDiscoveryEndpoint"
  ```

  Alamat, port, dan URL titik akhir klaster ditampilkan pada output, seperti dalam contoh berikut.

  ```
  {
      "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"
  }
  ```

Sekarang jalankan program lagi, tetapi kali ini tentukan URL titik akhir klaster sebagai parameter baris perintah.

```
java -cp classpath TryDax dax://my-cluster.l6fzcv.dax-clusters.us-east-1.amazonaws.com
```

Lihat output dan perhatikan informasi waktu. Waktu berlalu untuk `GetItem` dan `Query` harus jauh lebih rendah dengan DAX dibandingkan dengan DynamoDB.

Metrik SDK

Dengan DAX SDK for Java 2.x, Anda dapat mengumpulkan metrik tentang klien layanan di aplikasi Anda dan menganalisis output di Amazon. CloudWatch Lihat Mengaktifkan metrik SDK untuk informasi selengkapnya.

> **Note**
>
> DAX SDK untuk Java hanya mengumpulkan metrik `ApiCallSuccessful` dan `ApiCallDuration`.

TryDax.jawa

```
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 dan DAX

Ikuti langkah-langkah ini untuk menjalankan sampel.NET di EC2 instans Amazon Anda.

> **Note**
>
> Tutorial ini menggunakan .NET 6 SDK, tetapi juga akan berfungsi dengan .NET Core SDK. Anda akan diberi tahu cara menjalankan program di Amazon VPC default untuk mengakses klaster Amazon DynamoDB Accelerator (DAX) Anda. Jika mau, Anda dapat menggunakan AWS Toolkit for Visual Studio untuk menulis aplikasi.NET dan menyebarkannya ke VPC Anda.
>
> Untuk informasi selengkapnya, lihat Membuat dan Melakukan Deployment Aplikasi Elastic Beanstalk di .NET Menggunakan AWS Toolkit for Visual Studio di Panduan Developer AWS Elastic Beanstalk .

Cara menjalankan sampel .NET untuk DAX

1. Buka Halaman Microsoft Downloads dan unduh .NET 6 (atau .NET Core) SDK terbaru untuk Linux. File yang diunduh adalah `dotnet-sdk-N.N.N-linux-x64.tar.gz`.
2. Ekstrak file SDK.

```
mkdir dotnet
tar zxvf dotnet-sdk-N.N.N-linux-x64.tar.gz -C dotnet
```

Ganti *N.N.N* dengan nomor versi .NET SDK sebenarnya (misalnya:6.0.100).

3. Verifikasi instalasi.

```
alias dotnet=$HOME/dotnet/dotnet
dotnet --version
```

Tindakan ini akan mencetak nomor versi .NET SDK.

> **Note**
>
> Sebagai ganti nomor versi, Anda mungkin menerima kesalahan berikut:
> kesalahan: libunwind.so.8: tidak dapat membuka file objek yang dibagikan: File atau direktori tersebut tidak ada
> Untuk mengatasi kesalahan ini, instal paket `libunwind`.
>
> ```
> sudo yum install -y libunwind
> ```
>
> Setelah melakukan ini, Anda akan dapat menjalankan perintah `dotnet --version` tanpa kesalahan.

4. Buat proyek .NET baru.

```
dotnet new console -o myApp
```

Ini membutuhkan beberapa menit untuk melakukan one-time-only pengaturan. Setelah selesai, jalankan sampel proyek.

```
dotnet run --project myApp
```

Anda akan menerima pesan berikut: `Hello World!`

5. File `myApp/myApp.csproj` berisi metadata tentang proyek Anda. Untuk menggunakan klien DAX di aplikasi Anda, modifikasi file sehingga terlihat seperti berikut ini.

```
<Project Sdk="Microsoft.NET.Sdk">
    <PropertyGroup>
        <OutputType>Exe</OutputType>
        <TargetFramework>net6.0</TargetFramework>
```

```
    </PropertyGroup>
    <ItemGroup>
        <PackageReference Include="AWSSDK.DAX.Client" Version="*" />
    </ItemGroup>
</Project>
```

6. Unduh sampel kode sumber program (file `.zip`).

```
wget http://docs.aws.amazon.com/amazondynamodb/latest/developerguide/samples/
TryDax.zip
```

Setelah unduhan selesai, ekstrak file sumber.

```
unzip TryDax.zip
```

7. Sekarang jalankan program sampel satu per satu. Untuk setiap program, salin isinya ke `myApp/Program.cs`, kemudian jalankan proyek `MyApp`.

Jalankan program .NET berikut. Program pertama membuat tabel DynamoDB bernama `TryDaxTable`. Program kedua menulis data ke tabel.

```
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. Berikutnya, jalankan beberapa program untuk melakukan operasi `GetItem`, `Query`, dan `Scan` di klaster DAX Anda. Untuk menentukan titik akhir klaster DAX Anda, pilih salah satu dari berikut ini:

- Menggunakan konsol DynamoDB — Pilih klaster DAX Anda. Titik akhir klaster ditampilkan pada konsol, seperti dalam contoh berikut.

```
dax://my-cluster.l6fzcv.dax-clusters.us-east-1.amazonaws.com
```

- Menggunakan AWS CLI — Masukkan perintah berikut.

```
aws dax describe-clusters --query "Clusters[*].ClusterDiscoveryEndpoint"
```

Titik akhir klaster ditampilkan pada output, seperti dalam contoh berikut.

```
{
    "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"
}
```

Sekarang jalankan program berikut untuk menentukan titik akhir klaster Anda sebagai parameter baris perintah. (Ganti titik akhir sampel dengan titik akhir klaster DAX Anda yang sebenarnya).

```
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
```

Perhatikan informasi waktu, yaitu jumlah milidetik yang diperlukan untuk pengujian `GetItem`, `Query`, dan `Scan`.

9. Jalankan program .NET berikut untuk menghapus `TryDaxTable`.

```
cp TryDax/dotNet/06-DeleteTable.cs myApp/Program.cs
dotnet run --project myApp
```

Untuk informasi selengkapnya tentang program tersebut, lihat bagian berikut:

- 01- CreateTable .cs
- 02-Write-Data.cs
- 03- GetItem -Test.cs
- 04-Query-Test.cs
- 05-Scan-Test.cs
- 06- DeleteTable .cs

01- CreateTable .cs

Program `01-CreateTable.cs` membuat tabel (`TryDaxTable`). Program .NET yang tersisa di bagian ini bergantung pada tabel ini.

```
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

Program `02-Write-Data.cs` menulis data pengujian ke `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- GetItem -Test.cs

Program `03-GetItem-Test.cs` melakukan operasi `GetItem` di `TryDaxTable`.

```
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

Program `04-Query-Test.cs` melakukan operasi `Query` di `TryDaxTable`.

```
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

Program `05-Scan-Test.cs` melakukan operasi `Scan` di `TryDaxTable`.

```
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();
        }
    }
}
```

06- DeleteTable .cs

Program `06-DeleteTable.cs` menghapus `TryDaxTable`. Jalankan program ini setelah Anda selesai melakukan pengujian.

```
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 dan DAX

Ikuti prosedur ini untuk menjalankan contoh aplikasi Python pada instance Amazon EC2 Anda.

Cara menjalankan sampel Python untuk DAX

1. Instal klien Python DAX menggunakan utilitas `pip`.

   ```
   pip install amazon-dax-client
   ```

2. Unduh sampel kode sumber program (file `.zip`).

   ```
   wget http://docs.aws.amazon.com/amazondynamodb/latest/developerguide/samples/
   TryDax.zip
   ```

   Setelah unduhan selesai, ekstrak file sumber.

   ```
   unzip TryDax.zip
   ```

3. Jalankan program Python berikut. Program pertama membuat tabel Amazon DynamoDB bernama `TryDaxTable`. Program kedua menulis data ke tabel.

```
python 01-create-table.py
python 02-write-data.py
```

4. Jalankan program Python berikut.

```
python 03-getitem-test.py
python 04-query-test.py
python 05-scan-test.py
```

   Perhatikan informasi waktu, yaitu jumlah milidetik yang diperlukan untuk pengujian `GetItem`, `Query`, dan `Scan`.

5. Pada langkah sebelumnya, Anda menjalankan program terhadap titik akhir DynamoDB. Sekarang jalankan program lagi, tetapi kali ini, operasi `GetItem`, `Query` dan `Scan` diproses oleh klaster DAX Anda.

   Untuk menentukan titik akhir klaster DAX Anda, pilih salah satu dari berikut ini:

   - Menggunakan konsol DynamoDB — Pilih klaster DAX Anda. Titik akhir klaster ditampilkan pada konsol, seperti dalam contoh berikut.

     ```
     dax://my-cluster.l6fzcv.dax-clusters.us-east-1.amazonaws.com
     ```

   - Menggunakan AWS CLI — Masukkan perintah berikut.

     ```
     aws dax describe-clusters --query "Clusters[*].ClusterDiscoveryEndpoint"
     ```

     Titik akhir klaster ditampilkan pada output, seperti dalam contoh ini.

     ```
     {
         "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"
     }
     ```

   Jalankan program lagi, tetapi kali ini tentukan titik akhir klaster sebagai parameter baris perintah.

```
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
```

Periksa sisa output dan perhatikan informasi waktu. Waktu berlalu untuk `GetItem`, `Query`, dan `Scan` harus jauh lebih rendah dengan DAX dibandingkan dengan DynamoDB.

6. Jalankan program Python berikut untuk menghapus `TryDaxTable`.

```
python 06-delete-table.py
```

Untuk informasi selengkapnya tentang program tersebut, lihat bagian berikut:

- 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

Program `01-create-table.py` membuat tabel (`TryDaxTable`). Program Python yang tersisa di bagian ini bergantung pada tabel ini.

```
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

Program `02-write-data.py` menulis data pengujian ke `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

Program `03-getitem-test.py` melakukan operasi `GetItem` di `TryDaxTable`. Contoh ini diberikan untuk Wilayah eu-west-1.

```
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

Program `04-query-test.py` melakukan operasi `Query` di `TryDaxTable`.

```
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

Program `05-scan-test.py` melakukan operasi `Scan` di `TryDaxTable`.

```
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

Program `06-delete-table.py` menghapus `TryDaxTable`. Jalankan program ini setelah Anda selesai menguji fungsionalitas Amazon DynamoDB Accelerator (DAX).

```
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!")
```

Memodifikasi aplikasi yang ada untuk menggunakan DAX

Jika sudah memiliki aplikasi Java yang menggunakan Amazon DynamoDB, Anda dapat memodifikasinya agar dapat mengakses klaster DynamoDB Accelerator (DAX). Anda tidak perlu menulis ulang seluruh aplikasi karena klien DAX Java mirip dengan klien tingkat rendah DynamoDB yang disertakan dalam SDK for Java 2.x. AWS Lihat Bekerja dengan item di DynamoDB untuk detailnya.

> **Note**
>
> Contoh ini menggunakan AWS SDK for Java 2.x. Untuk SDK untuk Java versi 1.x lama, lihat Memodifikasi SDK untuk Java 1.x yang ada agar menggunakan DAX.

Untuk mengubah program Anda, ganti klien DynamoDB dengan klien DAX.

```
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();
```

Anda juga dapat menggunakan pustaka tingkat tinggi yang merupakan bagian dari AWS SDK for Java 2.x, menggantikan klien DynamoDB dengan klien DAX.

```
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();
```

Untuk informasi selengkapnya, lihat Memetakan item dalam tabel DynamoDB.

Mengelola klaster DAX

Bagian ini membahas beberapa tugas manajemen umum untuk klaster Amazon DynamoDB Accelerator (DAX).

Topik

- Izin IAM untuk mengelola klaster DAX
- Menskalakan klaster DAX
- Menyesuaikan pengaturan klaster DAX
- Mengonfigurasi pengaturan TTL
- Dukungan penandaan untuk DAX
- AWS CloudTrail integrasi
- Menghapus klaster DAX

Izin IAM untuk mengelola klaster DAX

Saat Anda mengelola klaster DAX menggunakan AWS Management Console atau AWS Command Line Interface (AWS CLI), kami sangat menyarankan agar Anda mempersempit cakupan tindakan yang dapat dilakukan pengguna. Dengan demikian, Anda membantu mengurangi risiko sekaligus mengikuti prinsip hak akses paling rendah.

Diskusi berikut berfokus pada kontrol akses untuk manajemen APIs DAX. Untuk informasi selengkapnya, lihat Amazon DynamoDB Accelerator di Referensi API Amazon DynamoDB.

> **Note**
>
> Untuk informasi lebih rinci tentang mengelola izin AWS Identity and Access Management (IAM), lihat berikut ini:
>
> - IAM dan pembuatan klaster DAX: Membuat klaster DAX.
> - Operasi bidang data IAM dan DAX: Kontrol akses DAX.

Untuk manajemen DAX APIs, Anda tidak dapat melakukan cakupan tindakan API ke sumber daya tertentu. Elemen `Resource` harus diatur ke `"*"`. Hal ini berbeda dari operasi API bidang data DAX,

seperti `GetItem`, `Query`, dan `Scan`. Operasi bidang data terekspos melalui klien DAX dan operasi tersebut dapat dicakup untuk sumber daya tertentu.

Untuk menggambarkannya, pertimbangkan dokumen kebijakan IAM berikut.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Action": [
                "dax:*"
            ],
            "Effect": "Allow",
            "Resource": [
                "arn:aws:dax:us-west-2:123456789012:cache/DAXCluster01"
            ]
        }
    ]
}
```

Anggaplah maksud dari kebijakan ini adalah untuk mengizinkan panggilan API manajemen DAX untuk klaster `DAXCluster01` dan hanya klaster tersebut.

Sekarang anggaplah bahwa pengguna mengeluarkan AWS CLI perintah berikut.

```
aws dax describe-clusters
```

Perintah ini gagal dengan pengecualian yang Tidak Diizinkan karena panggilan API `DescribeClusters` dasar tidak dapat dicakup ke klaster tertentu. Meskipun kebijakan valid secara sintaksis, perintah gagal karena elemen `Resource` harus diatur ke `"*"`. Namun, jika pengguna menjalankan program yang mengirimkan panggilan bidang data DAX (seperti `GetItem` atau `Query`) ke `DAXCluster01`, panggilan tersebut akan berhasil. Ini karena bidang data DAX APIs dapat dicakup ke sumber daya tertentu (dalam hal ini,). `DAXCluster01`

Jika Anda ingin menulis satu kebijakan IAM komprehensif untuk mencakup manajemen DAX APIs dan bidang data DAX APIs, kami sarankan Anda menyertakan dua pernyataan berbeda dalam dokumen kebijakan. Salah satu pernyataan ini harus membahas bidang data DAX APIs, sedangkan pernyataan lainnya membahas manajemen DAX. APIs

Contoh kebijakan berikut menunjukkan pendekatan ini. Perhatikan bagaimana `DAXDataAPIs` dicakup ke sumber daya `DAXCluster01`, tetapi sumber daya untuk `DAXManagementAPIs` harus

"*". Tindakan yang ditampilkan dalam setiap pernyataan hanya sebagai ilustrasi. Anda dapat menyesuaikan semuanya sesuai kebutuhan untuk aplikasi Anda.

```
{
    "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": [
                "*"
            ]
        }
    ]
}
```

Menskalakan klaster DAX

Ada dua opsi yang tersedia untuk menskalakan klaster DAX. Pilihan pertama adalah penskalaan horizontal, dengan menambahkan replika baca ke klaster. Pilihan kedua adalah penskalaan vertikal, dengan memilih jenis simpul yang berbeda. Untuk memperoleh saran tentang cara memilih ukuran klaster dan jenis simpul yang sesuai untuk aplikasi Anda, lihat Panduan pengukuran klaster DAX.

Penskalaan horizontal

Dengan penskalaan horizontal, Anda dapat meningkatkan throughput untuk operasi baca dengan menambahkan lebih banyak replika baca ke klaster. Satu klaster DAX mendukung hingga 10 replika baca dan Anda dapat menambahkan atau menghapus replika saat klaster sedang berjalan.

Ketika Anda menambahkan node baru, Anda harus menyinkronkan data cache dari node peer. Oleh karena itu, waktu penambahan bervariasi berdasarkan ukuran cache dan beban kerja aplikasi Anda. Sebagai praktik terbaik, kami menyarankan Anda melakukan pra-skala cluster Anda untuk memenuhi puncak lalu lintas yang diharapkan. Untuk informasi tentang pedoman ukuran kanan dan rekomendasi pemantauan, lihat. Panduan pengukuran klaster DAX

AWS CLI Contoh berikut menunjukkan cara menambah atau mengurangi jumlah node. Argumen `--new-replication-factor` menentukan jumlah total simpul di klaster. Salah satu simpul adalah simpul primer, dan simpul lainnya adalah replika baca.

```
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**
>
> Status klaster berubah menjadi `modifying` ketika Anda mengubah faktor replikasi. Status berubah menjadi `available` ketika modifikasi selesai.

Penskalaan vertikal

Jika Anda memiliki set data kerja yang besar, mungkin akan lebih bermanfaat jika aplikasi Anda menggunakan jenis simpul yang lebih besar. Simpul yang lebih besar memungkinkan klaster menyimpan lebih banyak data dalam memori, mengurangi cache miss, dan meningkatkan performa penerapan aplikasi secara keseluruhan. (Semua simpul dalam klaster DAX harus dari jenis yang sama).

Jika klaster DAX Anda memiliki tingkat operasi tulis atau cache miss tinggi, penggunaan jenis simpul yang lebih besar mungkin juga akan bermanfaat bagi aplikasi Anda. Operasi tulis dan cache miss menggunakan sumber daya pada simpul primer klaster. Oleh karena itu, menggunakan jenis simpul yang lebih besar mungkin akan meningkatkan performa simpul primer dan memungkinkan throughput yang lebih tinggi untuk jenis operasi ini.

Anda tidak dapat mengubah jenis simpul di klaster DAX yang sedang berjalan. Sebagai gantinya, Anda harus membuat klaster baru dengan jenis simpul yang diinginkan. Untuk daftar jenis simpul yang didukung, lihat Simpul.

Anda dapat membuat cluster DAX baru menggunakan AWS Management Console, AWS CloudFormation AWS CLI, atau AWS SDK. (Untuk AWS CLI, gunakan `--node-type` parameter untuk menentukan jenis node.)

Menyesuaikan pengaturan klaster DAX

Ketika Anda membuat klaster DAX, pengaturan default berikut digunakan:

- Pengosongan cache otomatis diaktifkan dengan Time to Live (TTL) selama 5 menit
- Tidak ada preferensi untuk Zona Ketersediaan
- Tidak ada preferensi untuk periode pemeliharaan
- Notifikasi dinonaktifkan

Untuk klaster baru, Anda dapat menyesuaikan pengaturan pada saat pembuatan. Untuk melakukannya di AWS Management Console, hapus Gunakan pengaturan default untuk mengubah pengaturan berikut:

- Jaringan dan Keamanan —Memungkinkan Anda menjalankan node cluster DAX individual di Availability Zone yang berbeda dalam Region saat ini AWS . Jika Anda memilih Tidak Ada Preferensi, simpul akan didistribusikan di antara Zona Ketersediaan secara otomatis.
- Grup Parameter—set parameter bernama yang diterapkan untuk setiap simpul di klaster. Anda dapat menggunakan grup parameter untuk menentukan perilaku TTL cache. Anda dapat mengubah nilai dari setiap parameter yang diberikan dalam grup parameter (kecuali grup parameter default `default.dax.1.0`) kapan pun.
- Periode Pemeliharaan—Adalah periode waktu mingguan untuk menerapkan peningkatan dan patch perangkat lunak ke simpul dalam klaster. Anda dapat memilih hari mulai, waktu mulai, dan durasi periode pemeliharaan. Jika Anda memilih Tidak Ada Preferensi, jendela pemeliharaan dipilih secara acak dari blok waktu 8 jam per Wilayah. Untuk informasi selengkapnya, lihat Periode pemeliharaan.

> **Note**
>
> Grup Parameter dan Periode Pemeliharaan juga dapat diubah kapan pun pada klaster yang berjalan.

Ketika proses pemeliharaan berlangsung, DAX dapat memberi tahu Anda menggunakan Amazon Simple Notification Service (Amazon SNS). Untuk mengonfigurasi notifikasi, pilih opsi dari pemilih Topik untuk notifikasi SNS. Anda dapat membuat topik Amazon SNS baru atau menggunakan topik yang sudah ada.

Untuk informasi selengkapnya tentang cara mengatur dan berlangganan topik SNS, lihat Mulai menggunakan Amazon SNS dalam Panduan Developer Amazon Simple Notification Service.

Mengonfigurasi pengaturan TTL

DAX mengelola dua cache untuk data yang dibaca dari DynamoDB:

- Cache item—Untuk item yang diambil menggunakan `GetItem` atau `BatchGetItem`.
- Cache kueri—Untuk set hasil yang diambil menggunakan `Query` atau `Scan`.

Untuk informasi selengkapnya, lihat Cache item dan Cache kueri.

TTL default untuk masing-masing cache ini adalah 5 menit. Jika ingin menggunakan pengaturan TTL yang berbeda, Anda dapat meluncurkan klaster DAX menggunakan grup parameter kustom. Untuk melakukannya di konsol, pilih DAX | Grup parameter di panel navigasi.

Anda juga dapat melakukan tugas ini menggunakan AWS CLI. Contoh berikut menunjukkan cara meluncurkan klaster DAX baru menggunakan grup parameter kustom. Dalam contoh ini, TTL cache item diatur ke 10 menit dan TTL cache kueri diatur ke 3 menit.

1. Buat grup parameter baru.

```
aws dax create-parameter-group \
    --parameter-group-name custom-ttl
```

2. Atur TTL cache item ke 10 menit (600000 milidetik).

```
aws dax update-parameter-group \
    --parameter-group-name custom-ttl \
    --parameter-name-values "ParameterName=record-ttl-millis,ParameterValue=600000"
```

3. Atur TTL cache kueri ke 3 menit (180000 milidetik).

```
aws dax update-parameter-group \
    --parameter-group-name custom-ttl \
    --parameter-name-values "ParameterName=query-ttl-millis,ParameterValue=180000"
```

4. Verifikasi bahwa parameter telah diatur dengan benar.

```
aws dax describe-parameters --parameter-group-name custom-ttl \
    --query "Parameters[*].[ParameterName,Description,ParameterValue]"
```

Sekarang Anda dapat meluncurkan klaster DAX baru dengan grup parameter ini.

```
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**
>
> Anda tidak dapat mengubah grup parameter yang sedang digunakan oleh instans DAX berjalan.

Dukungan penandaan untuk DAX

Banyak AWS layanan, termasuk DynamoDB, mendukung penandaan — kemampuan untuk memberi label sumber daya dengan nama yang ditentukan pengguna. Anda dapat menetapkan tag ke kluster DAX, memungkinkan Anda mengidentifikasi dengan cepat semua AWS sumber daya yang memiliki tag yang sama, atau untuk mengkategorikan AWS tagihan berdasarkan tag yang Anda tetapkan.

Untuk informasi selengkapnya, lihat Menambahkan tag dan label ke sumber daya di DynamoDB.

Menggunakan AWS Management Console

Cara mengelola tanda klaster DAX

1. Buka konsol DynamoDB di. https://console.aws.amazon.com/dynamodb/
2. Pada panel navigasi, di bawah DAX, pilih Klaster.
3. Pilih klaster yang ingin Anda gunakan.
4. Pilih tab Tanda. Anda dapat menambahkan, mencantumkan, mengedit, atau menghapus tanda di sini.

   Jika pengaturan sudah sesuai keinginan Anda, pilih Terapkan Perubahan.

Menggunakan AWS CLI

Saat Anda menggunakan tag klaster AWS CLI untuk mengelola DAX, Anda harus terlebih dahulu menentukan Nama Sumber Daya Amazon (ARN) untuk klaster. Contoh berikut menunjukkan cara menentukan ARN untuk klaster bernama `MyDAXCluster`.

```
aws dax describe-clusters \
    --cluster-name MyDAXCluster \
    --query "Clusters[*].ClusterArn"
```

Dalam output, ARN akan tampak mirip dengan ini: `arn:aws:dax:us-west-2:123456789012:cache/MyDAXCluster`

Contoh berikut menunjukkan cara menandai klaster.

```
aws dax tag-resource  \
    --resource-name arn:aws:dax:us-west-2:123456789012:cache/MyDAXCluster \
    --tags="Key=ClusterUsage,Value=prod"
```

Daftar semua tanda untuk klaster.

```
aws dax list-tags \
    --resource-name arn:aws:dax:us-west-2:123456789012:cache/MyDAXCluster
```

Untuk menghapus tanda, tentukan kuncinya.

```
aws dax untag-resource  \
    --resource-name arn:aws:dax:us-west-2:123456789012:cache/MyDAXCluster \
    --tag-keys ClusterUsage
```

AWS CloudTrail integrasi

DAX terintegrasi dengan AWS CloudTrail, memungkinkan Anda untuk mengaudit aktivitas cluster DAX. Anda dapat menggunakan CloudTrail log untuk melihat semua perubahan yang telah dibuat di tingkat cluster. Anda juga dapat melihat perubahan untuk komponen klaster seperti simpul, grup subnet, dan grup parameter. Untuk informasi selengkapnya, lihat Pencatatan log operasi DynamoDB menggunakan AWS CloudTrail.

Menghapus klaster DAX

Jika tidak lagi menggunakan klaster DAX, Anda harus menghapusnya untuk menghindari tagihan biaya atas sumber daya yang tidak digunakan.

Anda dapat menghapus klaster DAX menggunakan konsol atau AWS CLI. Berikut adalah contohnya.

```
aws dax delete-cluster --cluster-name mydaxcluster
```

Memantau DynamoDB Accelerator

Pemantauan merupakan bagian penting dalam menjaga keandalan, ketersediaan, dan kinerja Amazon DynamoDB Accelerator (DAX) dan solusi Anda. AWS Anda harus mengumpulkan data pemantauan dari semua bagian AWS solusi Anda sehingga Anda dapat lebih mudah men-debug kegagalan multi-titik, jika terjadi.

Namun, sebelum mulai memantau DAX, Anda harus membuat rencana pemantauan yang mencakup jawaban atas pertanyaan berikut:

- Apa saja sasaran pemantauan Anda?
- Sumber daya apa yang akan Anda pantau?
- Seberapa sering Anda akan memantau sumber daya ini?
- Alat pemantauan apa yang akan Anda gunakan?
- Siapa yang akan melakukan tugas pemantauan?
- Siapa yang harus diberi tahu saat terjadi kesalahan?

Topik

- Alat pemantauan untuk DynamoDB Accelerator
- Pemantauan CloudWatch dengan Amazon
- Mencatat operasi DAX menggunakan AWS CloudTrail

Alat pemantauan untuk DynamoDB Accelerator

AWS menyediakan alat yang dapat Anda gunakan untuk memantau Amazon DynamoDB Accelerator (DAX). Anda dapat mengonfigurasi beberapa alat tersebut agar melakukan pemantauan untuk Anda, dan beberapa alat lainnya memerlukan intervensi manual. Sebaiknya Anda mengotomatisasi tugas pemantauan sebanyak mungkin.

Topik

- Alat pemantauan otomatis
- Alat pemantauan manual

Alat pemantauan otomatis

Anda dapat menggunakan alat pemantauan otomatis berikut untuk memantau DAX dan melapor saat terjadi masalah:

- CloudWatch Alarm Amazon — Tonton satu metrik selama periode waktu yang Anda tentukan, dan lakukan satu atau beberapa tindakan berdasarkan nilai metrik relatif terhadap ambang batas tertentu selama beberapa periode waktu. Tindakannya adalah pemberitahuan yang dikirim ke topik Amazon Simple Notification Service (Amazon SNS) atau kebijakan Amazon Auto EC2 Scaling. CloudWatch alarm tidak memanggil tindakan hanya karena mereka berada dalam keadaan tertentu; negara harus telah berubah dan dipertahankan untuk sejumlah periode tertentu. Untuk informasi selengkapnya, lihat Memantau metrik di DynamoDB dengan Amazon CloudWatch.
- Amazon CloudWatch Logs — Pantau, simpan, dan akses file log Anda dari AWS CloudTrail atau sumber lain. Untuk informasi selengkapnya, lihat Memantau File Log di Panduan CloudWatch Pengguna Amazon.
- CloudWatch Acara Amazon — Cocokkan acara dan arahkan ke satu atau beberapa fungsi atau aliran target untuk membuat perubahan, menangkap informasi status, dan mengambil tindakan korektif. Untuk informasi selengkapnya, lihat Apa itu CloudWatch Acara Amazon di Panduan CloudWatch Pengguna Amazon.
- AWS CloudTrail Pemantauan Log - Bagikan file log antar akun, pantau file CloudTrail log secara real time dengan mengirimkannya ke CloudWatch Log, menulis aplikasi pemrosesan log di Java, dan validasi bahwa file log Anda tidak berubah setelah pengiriman oleh CloudTrail. Untuk informasi selengkapnya, lihat Bekerja dengan File CloudTrail Log di Panduan AWS CloudTrail Pengguna.

Alat pemantauan manual

Bagian penting lainnya dari pemantauan DAX melibatkan pemantauan secara manual item yang tidak CloudWatch tercakup oleh alarm. DAX, CloudWatch, Trusted Advisor, dan AWS Management Console dasbor lainnya memberikan at-a-glance pandangan tentang keadaan lingkungan Anda AWS . Sebaiknya Anda juga memeriksa file log pada DAX.

- Dasbor DAX menunjukkan hal berikut:
  - Kondisi layanan
- CloudWatch Halaman beranda menunjukkan yang berikut:
  - Alarm dan status saat ini
  - Grafik alarm dan sumber daya

- Status kesehatan layanan

Selain itu, Anda dapat menggunakan CloudWatch untuk melakukan hal berikut:

- Membuat dasbor yang disesuaikan untuk memantau layanan yang penting bagi Anda.
- Data metrik grafik untuk memecahkan masalah dan mengungkap tren.
- Cari dan telusuri semua metrik AWS sumber daya Anda.
- Buat dan sunting alarm untuk menerima pemberitahuan tentang masalah.

Pemantauan CloudWatch dengan Amazon

Anda dapat memantau DynamoDB Accelerator (DAX) menggunakan CloudWatch Amazon, yang mengumpulkan dan memproses data mentah dari DAX menjadi metrik hampir real-time yang dapat dibaca. Statistik tersebut dicatat untuk jangka waktu dua minggu. Artinya, Anda dapat mengakses informasi historis untuk mendapatkan perspektif yang lebih baik tentang performa aplikasi web atau layanan Anda. Secara default, data metrik DAX dikirim secara CloudWatch otomatis. Untuk informasi selengkapnya, lihat Apa itu Amazon CloudWatch? di Panduan CloudWatch Pengguna Amazon.

Topik

- Bagaimana cara menggunakan metrik DAX?
- Melihat metrik dan dimensi DAX
- Membuat CloudWatch alarm untuk memantau DAX
- Pemantauan produksi

Bagaimana cara menggunakan metrik DAX?

Metrik yang dilaporkan oleh DAX memberikan informasi yang dapat Anda analisis dengan berbagai cara. Daftar berikut menunjukkan beberapa penggunaan umum untuk metrik. Daftar ini adalah saran untuk memulai, bukan daftar komprehensif.

Bagaimana saya dapat melakukannya?	Metrik Terkait
Menentukan apakah terjadi kesalahan sistem	Pantau `FaultRequestCount` untuk menentukan apakah setiap permintaan menghasilkan kode HTTP 500 (kesalahan server). Ini dapat menunjukkan kesalahan layanan internal

Bagaimana saya dapat melakukannya?	Metrik Terkait
	DAX atau HTTP 500 dalam SystemErrors metrik tabel yang mendasarinya.
Menentukan apakah terjadi kesalahan pengguna	Pantau `ErrorRequestCount` untuk menentukan apakah setiap permintaan menghasilkan kode HTTP 400 (kesalahan klien). Jika melihat jumlah kesalahan bertambah, Anda mungkin ingin menyelidiki dan memastikan bahwa Anda mengirim permintaan klien yang benar.
Menentukan apakah terjadi cache miss	Pantau `ItemCacheMisses` untuk menentukan berapa kali item tidak ditemukan dalam cache, serta `QueryCacheMisses` dan `ScanCacheMisses` untuk menentukan berapa kali hasil kueri atau pemindaian tidak ditemukan dalam cache.
Memantau laju hit cache	Gunakan CloudWatch Metric Math untuk menentukan metrik hit rate cache menggunakan ekspresi matematika. Misalnya, untuk cache item, Anda dapat menggunakan ekspresi m1/SUM([m1, m2])*100, dengan m1 adalah metrik `ItemCacheHits` dan m2 adalah metrik `ItemCacheMisses` untuk klaster Anda. Untuk cache kueri dan pemindaian, Anda dapat mengikuti pola yang sama menggunakan metrik cache kueri dan pemindaian.

Melihat metrik dan dimensi DAX

Saat Anda berinteraksi dengan Amazon DynamoDB, ia mengirimkan metrik dan dimensi ke Amazon. CloudWatch Anda dapat menggunakan prosedur berikut untuk melihat metrik DynamoDB Accelerator (DAX).

Cara melihat metrik (konsol)

Metrik dikelompokkan terlebih dahulu berdasarkan namespace layanan, lalu berdasarkan berbagai kombinasi dimensi dalam setiap namespace.

1. Buka CloudWatch konsol di https://console.aws.amazon.com/cloudwatch/.

2. Di panel navigasi, pilih Metrik.
3. PILIH namespace DAX.

Cara melihat metrik (AWS CLI)

- Pada prompt perintah, gunakan perintah berikut.

```
aws cloudwatch list-metrics --namespace "AWS/DAX"
```

Metrik dan dimensi DAX

Bagian berikut berisi metrik dan dimensi yang dikirimkan DAX. CloudWatch

Metrik DAX

Metrik berikut tersedia dari DAX. DAX mengirimkan metrik CloudWatch hanya ketika mereka memiliki nilai bukan nol.

> **Note**
>
> CloudWatch menggabungkan metrik DAX berikut pada interval satu menit:
>
> - `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

Tidak semua statistik, seperti `Average` atau `Sum`, berlaku untuk setiap metrik. Namun, semua nilai ini tersedia melalui konsol DAX, atau dengan menggunakan CloudWatch konsol, AWS CLI, atau AWS SDKs untuk semua metrik. Dalam tabel berikut, setiap metrik memiliki daftar statistik valid yang berlaku untuk metrik tersebut.

Metrik	Deskripsi
`CPUUtilization`	Persentase pemanfaatan CPU simpul atau klaster.

Metrik	Deskripsi
	Unit: `Percent` Statistik Valid: • `Minimum` • `Maximum` • `Average`
`CacheMemoryUtilization`	Persentase memori cache tersedia yang digunakan oleh cache item dan cache kueri pada simpul atau klaster. Data cache mulai dihapus sebelum pemanfaatan memori mencapai 100% (lihat metrik `EvictedSize`). Jika `CacheMemoryUtilization` mencapai 100% pada setiap simpul, permintaan tulis akan dibatasi dan Anda harus mempertimbangkan untuk beralih ke klaster dengan jenis simpul yang lebih besar. Unit: `Percent` Statistik Valid: • `Minimum` • `Maximum` • `Average`
`NetworkBytesIn`	Jumlah byte yang diterima di semua antarmuka jaringan oleh simpul atau klaster. Unit: `Bytes` Statistik Valid: • `Minimum` • `Maximum` • `Average`

Metrik	Deskripsi
NetworkBytesOut	Jumlah byte yang dikirim di semua antarmuka jaringan oleh simpul atau klaster. Metrik ini mengidentifikasi volume lalu lintas yang keluar dari segi jumlah byte pada simpul atau klaster tunggal. Unit: Bytes Statistik Valid: • Minimum • Maximum • Average
BaselineNetworkBytesInUtilization	Persentase bandwidth jaringan dasar yang dikonsumsi pada waktu tertentu untuk lalu lintas masuk. Sebagai referensi, 50% berarti setengah dari bandwidth jaringan yang tersedia untuk lalu lintas masuk sedang digunakan. Unit: Percent Statistik Valid: • Minimum • Maximum • Average • SampleCount • Sum

Metrik	Deskripsi
`BaselineNetworkBytesOutUtilization`	Persentase bandwidth jaringan dasar yang dikonsumsi pada waktu tertentu untuk lalu lintas keluar. Sebagai referensi, 50% berarti setengah dari bandwidth jaringan yang tersedia untuk lalu lintas keluar sedang digunakan. Unit: `Percent` Statistik Valid: • `Minimum` • `Maximum` • `Average` • `SampleCount` • `Sum`
`NetworkPacketsIn`	Jumlah paket yang diterima di semua antarmuka jaringan oleh simpul atau klaster. Unit: `Count` Statistik Valid: • `Minimum` • `Maximum` • `Average`

Metrik	Deskripsi
NetworkPacketsOut	Jumlah paket yang dikirim di semua antarmuka jaringan oleh simpul atau klaster. Metrik ini mengidentifikasi volume lalu lintas yang keluar dari segi jumlah paket pada simpul atau klaster tunggal. Unit: Count Statistik Valid: • Minimum • Maximum • Average
GetItemRequestCount	Jumlah permintaan GetItem yang ditangani oleh simpul atau klaster. Unit: Count Statistik Valid: • Minimum • Maximum • Average • SampleCount • Sum

Metrik	Deskripsi
BatchGetItemRequestCount	Jumlah permintaan BatchGetItem yang ditangani oleh simpul atau klaster. Unit: Count Statistik Valid: • Minimum • Maximum • Average • SampleCount • Sum
BatchWriteItemRequestCount	Jumlah permintaan BatchWriteItem yang ditangani oleh simpul atau klaster. Unit: Count Statistik Valid: • Minimum • Maximum • Average • SampleCount • Sum

Metrik	Deskripsi
DeleteItemRequestCount	Jumlah permintaan DeleteItem yang ditangani oleh simpul atau klaster. Unit: Count Statistik Valid: • Minimum • Maximum • Average • SampleCount • Sum
PutItemRequestCount	Jumlah permintaan PutItem yang ditangani oleh simpul atau klaster. Unit: Count Statistik Valid: • Minimum • Maximum • Average • SampleCount • Sum

Metrik	Deskripsi
UpdateItemRequestCount	Jumlah permintaan UpdateItem yang ditangani oleh simpul atau klaster. Unit: Count Statistik Valid: • Minimum • Maximum • Average • SampleCount • Sum
TransactWriteItemsCount	Jumlah permintaan TransactWriteItems yang ditangani oleh simpul atau klaster. Unit: Count Statistik Valid: • Minimum • Maximum • Average • SampleCount • Sum

Metrik	Deskripsi
`TransactGetItemsCount`	Jumlah permintaan `TransactGetItems` yang ditangani oleh simpul atau klaster. Unit: `Count` Statistik Valid: • `Minimum` • `Maximum` • `Average` • `SampleCount` • `Sum`
`ItemCacheHits`	Berapa kali item dikembalikan dari cache oleh simpul atau klaster. Unit: `Count` Statistik Valid: • `Minimum` • `Maximum` • `Average` • `SampleCount` • `Sum`

Metrik	Deskripsi
`ItemCacheMisses`	Berapa kali item tidak ada dalam cache simpul atau klaster, dan harus diambil dari DynamoDB. Unit: `Count` Statistik Valid: • `Minimum` • `Maximum` • `Average` • `SampleCount` • `Sum`
`QueryCacheHits`	Berapa kali hasil kueri dikembalikan dari cache simpul atau klaster. Unit: `Count` Statistik Valid: • `Minimum` • `Maximum` • `Average` • `SampleCount` • `Sum`

Metrik	Deskripsi
QueryCacheMisses	Berapa kali hasil kueri tidak ada di cache simpul atau klaster, dan harus diambil dari DynamoDB. Unit: Count Statistik Valid: • Minimum • Maximum • Average • SampleCount • Sum
ScanCacheHits	Berapa kali hasil pemindaian dikembalikan dari cache simpul atau klaster. Unit: Count Statistik Valid: • Minimum • Maximum • Average • SampleCount • Sum

Metrik	Deskripsi
`ScanCacheMisses`	Berapa kali hasil pemindaian tidak ada di cache simpul atau klaster, dan harus diambil dari DynamoDB. Unit: `Count` Statistik Valid: • `Minimum` • `Maximum` • `Average` • `SampleCount` • `Sum`
`TotalRequestCount`	Jumlah total permintaan yang ditangani oleh simpul atau klaster. Unit: `Count` Statistik Valid: • `Minimum` • `Maximum` • `Average` • `SampleCount` • `Sum`

Metrik	Deskripsi
`ErrorRequestCount`	Jumlah total permintaan yang mengakibatkan kesalahan pengguna dilaporkan oleh simpul atau klaster. Permintaan yang dibatasi oleh simpul atau klaster disertakan. Unit: `Count` Statistik Valid: • `Minimum` • `Maximum` • `Average` • `SampleCount` • `Sum`
`ThrottledRequestCount`	Jumlah total permintaan yang dibatasi oleh simpul atau klaster. Permintaan yang dibatasi oleh DynamoDB tidak disertakan, dan dapat dipantau menggunakan Metrik DynamoDB. Unit: `Count` Statistik Valid: • `Minimum` • `Maximum` • `Average` • `SampleCount` • `Sum`

Metrik	Deskripsi
`FaultRequestCount`	Jumlah total permintaan yang mengakibatkan kesalahan internal dilaporkan oleh simpul atau klaster. Unit: `Count` Statistik Valid: • `Minimum` • `Maximum` • `Average` • `SampleCount` • `Sum`
`FailedRequestCount`	Jumlah total permintaan yang mengakibatkan kesalahan dilaporkan oleh simpul atau klaster. Unit: `Count` Statistik Valid: • `Minimum` • `Maximum` • `Average` • `SampleCount` • `Sum`

Metrik	Deskripsi
QueryRequestCount	Jumlah permintaan kueri yang ditangani oleh simpul atau klaster. Unit: Count Statistik Valid: • Minimum • Maximum • Average • SampleCount • Sum
ScanRequestCount	Jumlah permintaan pemindaian yang ditangani oleh simpul atau klaster. Unit: Count Statistik Valid: • Minimum • Maximum • Average • SampleCount • Sum

Metrik	Deskripsi
`ClientConnections`	Jumlah koneksi simultan yang dibuat oleh klien ke simpul atau klaster. Unit: `Count` Statistik Valid: • `Minimum` • `Maximum` • `Average` • `SampleCount` • `Sum`
`EstimatedDbSize`	Perkiraan jumlah data cache dalam cache item dan cache kueri oleh simpul atau klaster. Unit: `Bytes` Statistik Valid: • `Minimum` • `Maximum` • `Average`

Metrik	Deskripsi
`EvictedSize`	Jumlah data yang dihapus oleh simpul atau klaster untuk menyediakan ruang bagi data yang baru diminta. Jika laju miss naik dan Anda juga melihat metrik ini bertambah, hal ini mungkin berarti bahwa set pekerjaan Anda telah meningkat . Anda harus mempertimbangkan untuk beralih ke klaster dengan jenis simpul yang lebih besar. Unit: `Bytes` Statistik Valid: • `Minimum` • `Maximum` • `Average` • `Sum`

Metrik	Deskripsi
`CPUCreditUsage`	Jumlah kredit CPU yang digunakan oleh simpul untuk pemanfaatan CPU. Satu kredit CPU sama dengan satu vCPU yang berjalan pada pemanfaatan 100% selama satu menit atau kombinasi yang setara dari CPUs v, pemanfaatan, dan waktu (misalnya, satu vCPU berjalan pada pemanfaatan 50% selama dua menit atau dua CPUs v berjalan pada pemanfaatan 25% selama dua menit). Metrik kredit CPU tersedia hanya dalam frekuensi lima menit. Jika Anda menentukan periode lebih dari lima menit, gunakan statistik `Sum`, bukan `Average`. Unit: `Credits (vCPU-minutes)` Statistik Valid: • `Minimum` • `Maximum` • `Average` • `SampleCount` • `Sum`

Metrik	Deskripsi
`CPUCreditBalance`	Jumlah kredit CPU yang diperoleh yang diakumulasi oleh simpul sejak diluncurkan atau dimulai. Kredit diakumulasi dalam saldo kredit setelah diperoleh, dan dihapus dari saldo kredit saat dibelanjakan. Saldo kredit memiliki batasan maksimum, ditentukan oleh ukuran simpul DAX. Setelah batas tercapai, setiap kredit baru yang diperoleh akan dibuang. Kredit dalam `CPUCreditBalance` tersedia untuk digunakan oleh simpul hingga melonjak melebihi pemanfaatan CPU acuan. Unit: `Credits (vCPU-minutes)` Statistik Valid: • `Minimum` • `Maximum` • `Average` • `SampleCount` • `Sum`

Metrik	Deskripsi
CPUSurplusCreditBalance	Jumlah kredit surplus yang telah digunakan oleh simpul DAX ketika nilai CPUCreditBalance nol. Nilai CPUSurplusCreditBalance dibayarkan oleh kredit CPU yang diperoleh. Jika jumlah kredit surplus melebihi jumlah kredit maksimum yang dapat diperoleh simpul dalam jangka waktu 24 jam, kredit surplus yang digunakan di atas jumlah maksimum akan dikenakan biaya tambahan. Unit: Credits (vCPU-minutes) Statistik Valid: • Minimum • Maximum • Average • SampleCount • Sum

Metrik	Deskripsi
`CPUSurplusCreditsCharged`	Jumlah kredit surplus yang dibelanjakan yang tidak dibayarkan oleh kredit CPU yang diperoleh, dan dengan demikian menimbulkan biaya tambahan. Kredit surplus dikenai biaya saat kredit surplus yang digunakan melebihi jumlah kredit maksimum yang dapat diperoleh simpul dalam periode 24 jam. Kredit surplus yang digunakan di atas jumlah maksimum akan dikenakan biaya pada akhir jam atau ketika simpul dihentikan. Unit: `Credits (vCPU-minutes)` Statistik Valid: • `Minimum` • `Maximum` • `Average` • `SampleCount` • `Sum`

> **Note**
>
> Metrik `CPUCreditUsage`, `CPUCreditBalance`, `CPUSurplusCreditBalance`, dan `CPUSurplusCreditsCharged` hanya tersedia untuk simpul T3.

Dimensi untuk Metrik DAX

Metrik untuk DAX memenuhi syarat berdasarkan nilai untuk akun, ID klaster, atau ID klaster dan kombinasi ID simpul. Anda dapat menggunakan CloudWatch konsol untuk mengambil data DAX sepanjang salah satu dimensi dalam tabel berikut.

Dimensi	Deskripsi
`Account`	Menyediakan statistik agregat di semua simpul dalam akun.
`ClusterId`	Membatasi data ke klaster.
`ClusterId, NodeId`	Membatasi data ke simpul dalam klaster.

Membuat CloudWatch alarm untuk memantau DAX

Anda dapat membuat CloudWatch alarm Amazon yang mengirimkan pesan Amazon Simple Notification Service (Amazon SNS) saat alarm berubah status. Alarm mengawasi metrik tunggal selama periode waktu yang Anda tentukan. Alarm tersebut melakukan satu atau beberapa tindakan berdasarkan nilai metrik yang relatif terhadap ambang batas tertentu selama beberapa periode waktu. Tindakan tersebut adalah notifikasi yang dikirim ke topik Amazon SNS atau kebijakan Penskalaan Otomatis. Alarm memanggil tindakan untuk perubahan status berkelanjutan saja. CloudWatch alarm tidak memanggil tindakan hanya karena mereka berada dalam keadaan tertentu. Status harus diubah dan dipertahankan selama jangka waktu tertentu.

Bagaimana cara agar saya mendapatkan pemberitahuan mengenai cache miss kueri?

1. Buat topik Amazon SNS, `arn:aws:sns:us-west-2:522194210714:QueryMissAlarm`.

   Untuk informasi selengkapnya, lihat Mengatur Layanan Pemberitahuan Sederhana Amazon di Panduan CloudWatch Pengguna Amazon.
2. Buat 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. Uji 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**
>
> Anda dapat menambah atau mengurangi ambang batas sesuai dengan yang memungkinkan untuk aplikasi Anda. Anda juga dapat menggunakan CloudWatch Metric Math untuk menentukan metrik tingkat kehilangan cache dan mengatur alarm di atas metrik tersebut.

Bagaimana saya bisa diberi tahu jika permintaan menyebabkan kesalahan internal di cluster?

1. Buat topik Amazon SNS, `arn:aws:sns:us-west-2:123456789012:notify-on-system-errors`.

   Untuk informasi selengkapnya, lihat Mengatur Layanan Pemberitahuan Sederhana Amazon di Panduan CloudWatch Pengguna Amazon.

2. Buat 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. Uji 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
```

Pemantauan produksi

Anda harus menetapkan acuan untuk performa DAX normal di lingkungan Anda dengan mengukur performa di berbagai waktu dan dalam kondisi beban yang berbeda. Saat memantau DAX, Anda harus mempertimbangkan untuk menyimpan data pemantauan historis. Data yang disimpan ini memberi Anda acuan untuk membandingkan data performa saat ini, mengidentifikasi pola performa normal dan anomali performa, serta merancang metode untuk mengatasi masalah.

Untuk menetapkan acuan, setidaknya Anda harus memantau item-item berikut baik saat pengujian beban dan saat produksi.

- Pemanfaatan CPU dan permintaan yang dibatasi, sehingga Anda dapat menentukan apakah mungkin perlu menggunakan jenis simpul yang lebih besar di klaster Anda. Pemanfaatan CPU cluster Anda tersedia melalui `CPUUtilization` CloudWatch metrik. Statistik rata-rata pada metrik ini memberikan tampilan pemanfaatan CPU rata-rata di semua node di cluster Anda. Untuk keputusan penskalaan klaster, kami menyarankan Anda menggunakan stat maksimum yang merupakan pemanfaatan maksimum di semua node.

> **Note**
>
> AWS telah meningkatkan granularitas `CPUUtilization` metrik. Anda mungkin mengamati perubahan metrik mulai dari 2024-05-17 hingga 2024-06-22.

- Latensi operasi (seperti yang diukur pada sisi klien) harus tetap konsisten dalam persyaratan latensi aplikasi Anda.
- Tingkat kesalahan harus tetap rendah, seperti yang terlihat dari`ErrorRequestCount`,`FaultRequestCount`, dan `FailedRequestCount` CloudWatch metrik.
- Konsumsi byte jaringan, sehingga Anda dapat menentukan apakah Anda harus menggunakan lebih banyak node atau tipe node yang lebih besar di cluster Anda. Untuk memantau konsumsi, Anda dapat mengatur peringatan `BaselineNetworkBytesInUtilization` dan `BaselineNetworkBytesOutUtilization` metrik yang tersedia CloudWatch, yang menunjukkan persentase konsumsi bandwidth jaringan yang tersedia untuk jenis instans Anda, masing-masing untuk lalu lintas masuk dan keluar.
- Pemanfaatan memori cache dan ukuran yang dikosongkan, sehingga Anda dapat menentukan apakah jenis simpul klaster memiliki cukup memori untuk menahan set pekerjaan, dan jika tidak, alihkan ke jenis simpul yang lebih besar.

> **Note**
>
> Jika ada sejumlah besar cache miss dan penulisan cache, pemanfaatan memori cache dapat meningkat hingga 100% dan dapat menyebabkan waktu henti ketersediaan.

- Koneksi klien, sehingga Anda dapat memantau setiap lonjakan yang tidak bisa dijelaskan dalam koneksi ke klaster.

Mencatat operasi DAX menggunakan AWS CloudTrail

Amazon DynamoDB Accelerator (DAX) terintegrasi AWS CloudTrail dengan, layanan yang menyediakan catatan tindakan yang diambil oleh pengguna, peran, atau layanan di DAX. AWS

Untuk mempelajari lebih lanjut tentang DAX dan CloudTrail, lihat bagian DynamoDB Accelerator (DAX) di. Pencatatan log operasi DynamoDB menggunakan AWS CloudTrail

Instans DAX T3/T2 yang dapat melonjak

DAX memungkinkan Anda memilih antara instans kinerja tetap (seperti R4, R5, dan R7) dan instans kinerja burstable (seperti T2 dan T3). Instans performa yang dapat melonjak memberikan tingkat acuan performa CPU dengan kemampuan untuk melonjak melebihi acuan saat diperlukan.

Performa acuan dan kemampuan melonjak diatur oleh kredit CPU. Instans performa yang dapat melonjak mengakumulasikan kredit CPU secara terus-menerus pada tingkat yang ditentukan oleh ukuran instans, ketika beban kerja di bawah ambang batas acuan. Kredit ini kemudian dapat digunakan ketika beban kerja meningkat. Kredit CPU memberikan performa inti CPU penuh selama satu menit.

Banyak beban kerja tidak memerlukan CPU tingkat tinggi secara konsisten, tetapi mendapat manfaat yang signifikan dari memiliki akses penuh ke sangat cepat CPUs ketika mereka membutuhkannya. Instans performa yang dapat melonjak dirancang khusus untuk kasus penggunaan ini. Jika membutuhkan performa CPU yang tinggi secara konsisten untuk basis data Anda, sebaiknya gunakan instans performa tetap.

Rangkaian instans T2 DAX

Instans T2 DAX adalah instans performa tujuan umum yang dapat melonjak yang memberikan tingkat acuan performa CPU dengan kemampuan untuk melonjak melebihi acuan. Instans T2 adalah pilihan bagus untuk beban kerja pengujian dan pengembangan yang membutuhkan prediktabilitas harga. Instans DAX T2 dikonfigurasi untuk mode standar, yang berarti jika instans kehabisan kredit yang diakumulasikan, pemanfaatan CPU akan diturunkan secara bertahap ke tingkat acuan. Untuk informasi selengkapnya tentang mode standar, lihat Mode standar untuk instans performa burstable di EC2 Panduan Pengguna Amazon.

Rangkaian instans T3 DAX

Instans T3 DAX adalah jenis instans tujuan umum yang dapat melonjak yang menyediakan tingkat acuan performa CPU dengan kemampuan untuk melonjakkan penggunaan CPU kapan saja selama yang diperlukan. Instans T3 menawarkan keseimbangan komputasi, memori, dan sumber daya jaringan serta ideal untuk beban kerja dengan penggunaan CPU sedang yang mengalami lonjakan sementara saat digunakan. Instans T3 DAX dikonfigurasi untuk mode tidak terbatas, yang berarti dapat melonjak melampaui acuan dalam jangka waktu 24 jam dengan biaya tambahan. Untuk informasi selengkapnya tentang mode tak terbatas, lihat Mode tak terbatas untuk instans performa burstable di EC2 Panduan Pengguna Amazon.

Instans T3 DAX dapat mempertahankan performa CPU yang tinggi selama beban kerja memerlukannya. Untuk sebagian besar beban kerja tujuan umum, instans T3 akan memberikan performa yang memadai tanpa biaya tambahan. Harga instans T3 per jam otomatis mencakup semua lonjakan sementara saat digunakan ketika pemanfaatan CPU rata-rata dari instans T3 berada pada atau kurang dari acuan dalam jangka waktu 24 jam.

Misalnya, instans `dax.t3.small` menerima kredit terus-menerus dengan laju 24 kredit CPU per jam. Kemampuan ini memberikan performa acuan setara dengan 20% dari inti CPU (20% × 60 menit = 12 menit). Jika instans tidak menggunakan kredit yang diterima, kredit tersebut akan disimpan dalam saldo kredit CPU hingga maksimum 576 kredit CPU. Saat instans `t3.small` perlu melonjak 20% di atas inti, instans akan menarik saldo kredit CPU untuk menangani lonjakan ini secara otomatis.

Saat instans T2 DAX dibatasi ke tingkat performa acuan setelah saldo kredit CPU habis digunakan, instans T3 DAX dapat melonjak melebihi acuan meskipun saldo kredit CPU nol. Untuk sebagian besar beban kerja dengan pemanfaatan CPU rata-rata berada pada atau di bawah performa acuan, harga per jam dasar untuk `t3.small` mencakup semua lonjakan CPU. Jika instans berjalan dengan pemanfaatan CPU rata-rata 25% (5% di atas acuan) selama periode 24 jam setelah saldo kredit CPU habis digunakan, akan dikenakan tambahan 11,52 sen (9,6 sen/vCPU-jam × 1 vCPU × 5% × 24 jam). Lihat Harga Amazon DynamoDB untuk mendapatkan detail harga.

Kontrol akses DAX

DynamoDB Accelerator (DAX) dirancang untuk bekerja sama dengan DynamoDB guna menambahkan lapisan caching ke aplikasi Anda dengan lancar. Namun, DAX dan DynamoDB memiliki mekanisme kontrol akses terpisah. Kedua layanan menggunakan AWS Identity and Access Management (IAM) untuk menerapkan kebijakan keamanan masing-masing, tetapi model keamanan untuk DAX dan DynamoDB berbeda.

Kami sangat menyarankan agar Anda memahami kedua model keamanan, sehingga dapat menerapkan langkah-langkah keamanan yang tepat untuk aplikasi Anda yang menggunakan DAX.

Bagian ini menjelaskan mekanisme kontrol akses yang disediakan oleh DAX dan memberikan contoh kebijakan IAM yang dapat disesuaikan dengan kebutuhan Anda.

Dengan DynamoDB, Anda dapat membuat kebijakan IAM yang membatasi tindakan yang dapat dilakukan pengguna pada sumber daya DynamoDB individu. Misalnya, Anda dapat membuat peran pengguna yang hanya mengizinkan pengguna melakukan tindakan hanya baca di tabel DynamoDB

tertentu. (Untuk informasi selengkapnya, lihat Manajemen Identitas dan Akses untuk Amazon DynamoDB.) Sebagai perbandingan, model keamanan DAX berfokus pada keamanan klaster dan kemampuan klaster untuk melakukan tindakan API DynamoDB atas nama Anda.

> **Warning**
>
> Jika saat ini Anda menggunakan kebijakan dan peran IAM untuk membatasi akses ke data tabel DynamoDB, penggunaan DAX dapat merusak kebijakan tersebut. Misalnya, pengguna dapat memiliki akses ke tabel DynamoDB melalui DAX tetapi tidak memiliki akses eksplisit ke tabel yang sama yang mengakses DynamoDB secara langsung. Untuk informasi selengkapnya, lihat Manajemen Identitas dan Akses untuk Amazon DynamoDB.
> DAX tidak memberlakukan pemisahan tingkat pengguna pada data di DynamoDB. Sebaliknya, pengguna mewarisi izin kebijakan IAM klaster DAX ketika mengakses klaster tersebut. Dengan demikian, ketika mengakses tabel DynamoDB melalui DAX, kontrol akses yang berlaku adalah izin di kebijakan IAM klaster DAX. Tidak ada izin lain yang diakui.
> Jika memerlukan isolasi, sebaiknya Anda membuat klaster DAX tambahan dan lingkup kebijakan IAM untuk setiap klaster. Misalnya, Anda dapat membuat beberapa klaster DAX dan mengizinkan setiap klaster mengakses hanya satu tabel.

Peran layanan IAM untuk DAX

Ketika membuat sebuah klaster DAX, Anda harus mengasosiasikan instans dengan peran IAM. Hal ini dikenal sebagai peran layanan untuk klaster.

Misalkan Anda ingin membuat cluster DAX baru bernama DAXCluster01. Anda dapat membuat peran layanan bernama DAXServicePeran, dan mengaitkan peran dengan DAXCluster01. Kebijakan untuk DAXServicePeran akan menentukan tindakan DynamoDB DAXClusteryang dapat dilakukan 01, atas nama pengguna yang berinteraksi dengan 01. DAXCluster

Saat membuat peran layanan, Anda harus menentukan hubungan kepercayaan antara DAXServicePeran dan layanan DAX itu sendiri. Hubungan kepercayaan menentukan entitas yang dapat mengambil peran dan memanfaatkan izinnya. Berikut ini adalah contoh dokumen hubungan kepercayaan untuk DAXServicePeran:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
```

```
            "Effect": "Allow",
            "Principal": {
                "Service": "dax.amazonaws.com"
            },
            "Action": "sts:AssumeRole"
        }
    ]
}
```

Hubungan kepercayaan ini memungkinkan klaster DAX untuk mengambil DAXServicePeran dan melakukan panggilan API DynamoDB atas nama Anda.

Tindakan API DynamoDB yang diizinkan dijelaskan dalam dokumen kebijakan IAM, yang Anda lampirkan ke Peran. DAXService Berikut adalah contoh dokumen kebijakan.

```
{
    "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"
            ]
        }
    ]
}
```

Kebijakan ini mengizinkan DAX melakukan tindakan API DynamoDB yang diperlukan di tabel DynamoDB. Tindakan `dynamodb:DescribeTable` diperlukan agar DAX dapat mempertahankan

metadata tentang tabel, dan tindakan lain adalah tindakan pembacaan dan penulisan yang dilakukan pada item dalam tabel. Tabel bernama `Books` berada di Wilayah us-west-2 dan dimiliki oleh ID akun AWS `123456789012`.

> **Note**
>
> DAX mendukung mekanisme untuk mencegah masalah deputi yang membingungkan selama akses Cross-service. Untuk informasi selengkapnya, lihat Masalah confused deputy di Panduan Pengguna IAM.

Kebijakan IAM untuk mengizinkan akses klaster DAX

Setelah membuat klaster DAX, Anda perlu memberikan izin untuk pengguna sehingga pengguna dapat mengakses klaster DAX.

Misalnya, Anda ingin memberikan akses ke DAXCluster01 ke pengguna bernama Alice. Pertama-tama Anda akan membuat kebijakan IAM (AliceAccessPolicy) yang mendefinisikan cluster DAX dan tindakan API DAX yang dapat diakses oleh penerima. Kemudian, Anda akan memberikan akses dengan melampirkan kebijakan ini ke pengguna Alice.

Dokumen kebijakan berikut memberikan penerima akses penuh pada DAXCluster01.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Action": [
                "dax:*"
            ],
            "Effect": "Allow",
            "Resource": [
                "arn:aws:dax:us-west-2:123456789012:cache/DAXCluster01"
            ]
        }
    ]
}
```

Dokumen kebijakan mengizinkan akses ke klaster DAX, tetapi tidak memberikan izin DynamoDB apa pun. (Izin DynamoDB diberikan oleh peran layanan DAX).

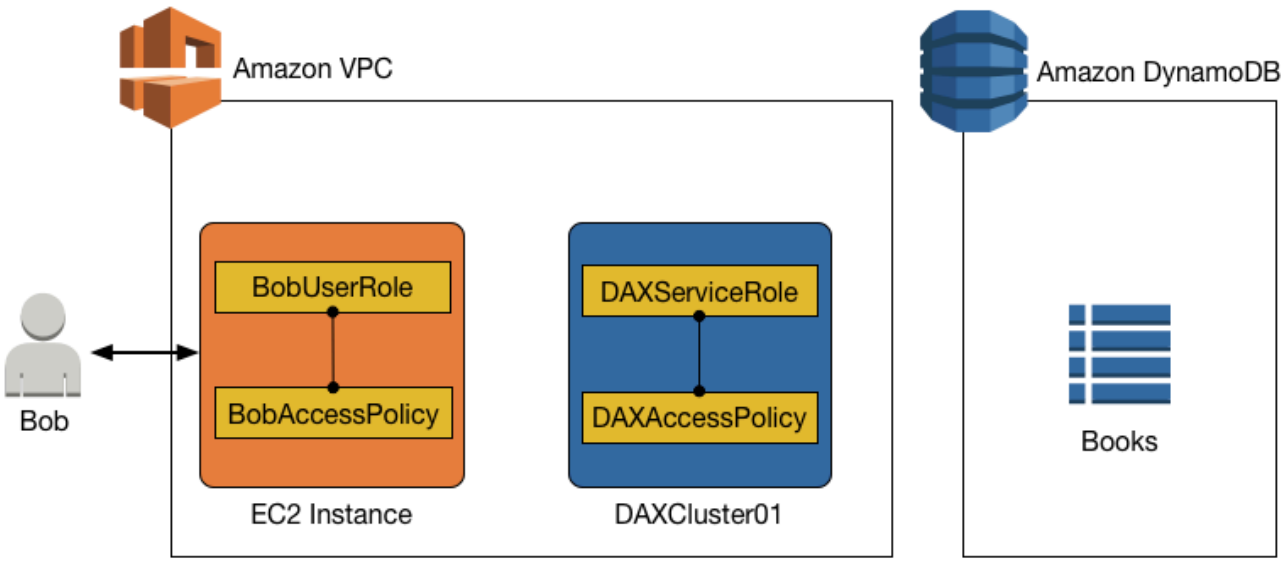
Untuk pengguna Alice, Anda harus terlebih dahulu membuat `AliceAccessPolicy` dengan dokumen kebijakan yang ditampilkan sebelumnya. Kemudian, Anda akan melampirkan kebijakan tersebut pada Alice.

> **Note**
>
> Selain melampirkan kebijakan kepada pengguna, Anda dapat melampirkannya ke peran IAM. Dengan cara itu, semua pengguna yang mengambil peran tersebut akan memiliki izin yang Anda tetapkan dalam kebijakan.

Kebijakan pengguna dan peran layanan DAX menentukan sumber daya DynamoDB dan tindakan API yang dapat diakses oleh penerima melalui DAX.

Studi Kasus: Mengakses DynamoDB dan DAX

Skenario berikut dapat membantu Anda memahami lebih jauh tentang kebijakan IAM yang digunakan dengan DAX. (Skenario ini disebut di seluruh bagian ini). Diagram berikut menunjukkan gambaran umum tingkat tinggi skenario.



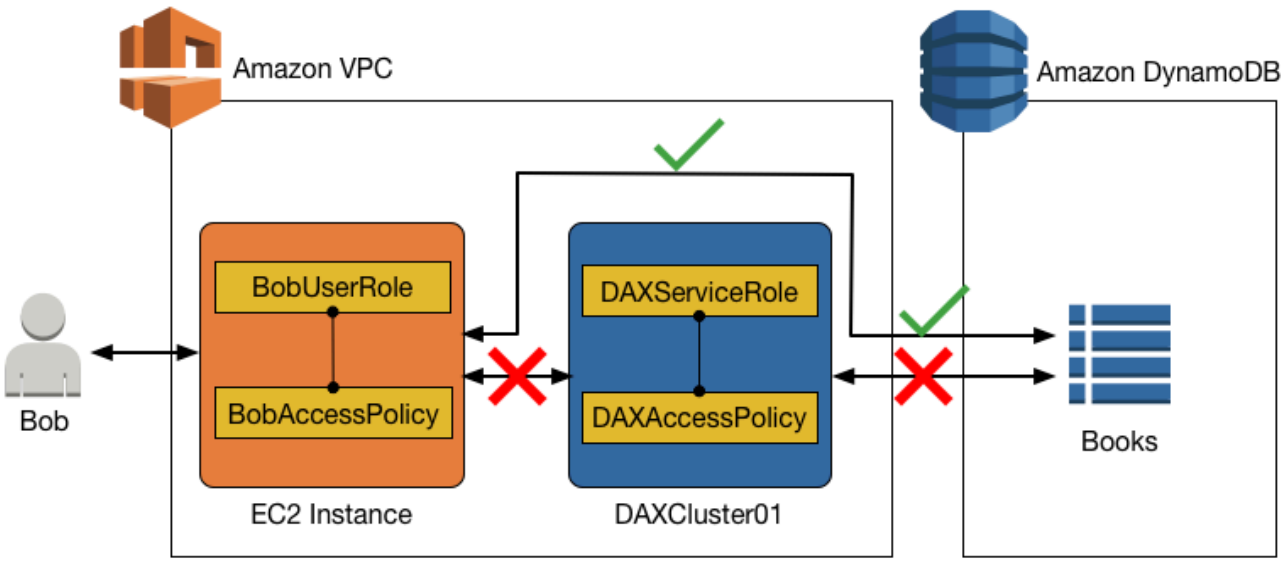
Dalam skenario ini, terdapat entitas berikut:

- Pengguna (Bob).

- Peran IAM (`BobUserRole`). Bob mengambil peran ini saat runtime.
- Kebijakan IAM (`BobAccessPolicy`). Kebijakan ini dilampirkan pada `BobUserRole`. `BobAccessPolicy` menentukan sumber daya DynamoDB dan DAX yang dapat diakses `BobUserRole`.
- Klaster DAX (`DAXCluster01`).
- Peran layanan IAM (`DAXServiceRole`). Peran ini mengizinkan `DAXCluster01` untuk mengakses DynamoDB.
- Kebijakan IAM (`DAXAccessPolicy`). Kebijakan ini dilampirkan pada`DAXServiceRole`. `DAXAccessPolicy`mendefinisikan APIs DynamoDB dan sumber daya `DAXCluster01` yang diizinkan untuk mengakses.
- Tabel DynamoDB (`Books`).

Kombinasi pernyataan kebijakan di `BobAccessPolicy` dan `DAXAccessPolicy` menentukan apa yang dapat dilakukan Bob dengan tabel `Books`. Misalnya, Bob mungkin dapat mengakses `Books` secara langsung (menggunakan titik akhir DynamoDB), secara tidak langsung (menggunakan klaster DAX), atau keduanya. Bob mungkin juga dapat membaca data dari `Books`, menulis data ke `Books`, atau keduanya.

Akses ke DynamoDB, tetapi tidak ada akses ke DAX



Mengizinkan akses langsung ke tabel DynamoDB sekaligus mencegah akses tidak langsung menggunakan klaster DAX dapat dilakukan. Untuk akses langsung ke DynamoDB, izin untuk `BobUserRole` ditentukan oleh `BobAccessPolicy` (yang terlampir pada peran).

Akses hanya baca ke DynamoDB (khusus)

Bob dapat mengakses DynamoDB dengan `BobUserRole`. Kebijakan IAM yang dilampirkan pada role (`BobAccessPolicy`) ini menentukan tabel DynamoDB `BobUserRole` yang dapat mengakses, dan APIs apa yang dapat dipanggil. `BobUserRole`

Pertimbangkan dokumen kebijakan berikut untuk `BobAccessPolicy`.

```
{
    "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"
        }
    ]
}
```

Ketika dilampirkan ke `BobAccessPolicy`, dokumen ini mengizinkan `BobUserRole` untuk mengakses titik akhir DynamoDB dan melakukan operasi hanya baca pada tabel `Books`.

DAX tidak muncul dalam kebijakan ini, sehingga akses melalui DAX ditolak.

Akses baca/tulis ke DynamoDB (khusus)

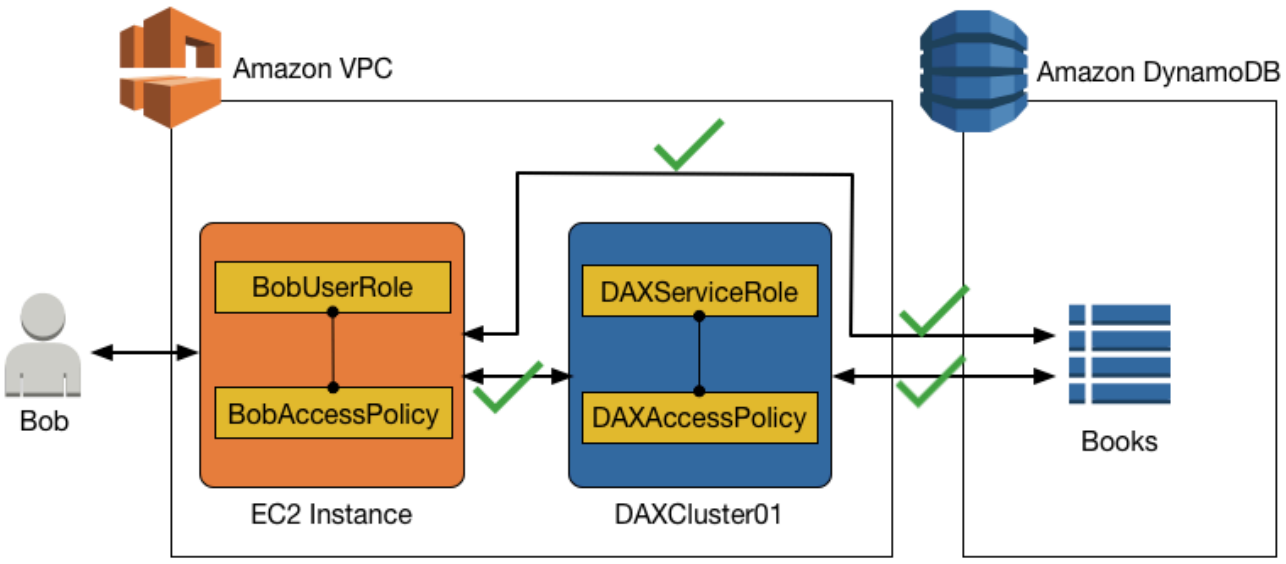
Jika `BobUserRole` memerlukan akses baca/tulis ke DynamoDB, kebijakan berikut akan berfungsi.

```
{
    "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"
        }
    ]
}
```

Sekali lagi, DAX tidak muncul dalam kebijakan ini, sehingga akses melalui DAX ditolak.

Akses ke DynamoDB dan DAX



Untuk mengizinkan akses ke klaster DAX, Anda harus menyertakan tindakan khusus DAX dalam kebijakan IAM.

Tindakan khusus DAX berikut sesuai dengan tindakan serupa yang memiliki nama serupa di API DynamoDB:

- `dax:GetItem`
- `dax:BatchGetItem`
- `dax:Query`
- `dax:Scan`
- `dax:PutItem`
- `dax:UpdateItem`
- `dax:DeleteItem`
- `dax:BatchWriteItem`
- `dax:ConditionCheckItem`

Hal yang sama berlaku untuk kunci syarat `dax:EnclosingOperation`.

Akses hanya baca ke DynamoDB dan akses hanya baca ke DAX

Anggaplah Bob membutuhkan akses hanya baca ke tabel `Books`, dari DynamoDB dan dari DAX. Kebijakan berikut (dilampirkan pada `BobUserRole`) memberikan akses ini.

```
{
    "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"
        }
    ]
}
```

Kebijakan ini memiliki pernyataan untuk DAX access (`DAXAccessStmt`) dan pernyataan lain untuk Dynamo DBaccess (`DynamoDBAccessStmt`). Pernyataan tersebut mengizinkan Bob mengirim permintaan `GetItem`, `BatchGetItem`, `Query`, dan `Scan` ke `DAXCluster01`.

Namun, peran layanan untuk `DAXCluster01` juga akan membutuhkan akses hanya baca ke tabel `Books` di DynamoDB. Kebijakan IAM berikut, yang terlampir pada `DAXServiceRole`, akan memenuhi persyaratan ini.

```
{
    "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"
        }
    ]
}
```

Akses baca/tulis ke DynamoDB dan hanya baca dengan DAX

Untuk peran pengguna tertentu, Anda dapat memberikan akses baca/tulis ke tabel DynamoDB sekaligus mengizinkan akses hanya baca melalui DAX.

Untuk Bob, kebijakan IAM untuk `BobUserRole` perlu mengizinkan DynamoDB membaca dan menulis tindakan pada tabel `Books`, sekaligus mendukung tindakan hanya baca melalui `DAXCluster01`.

Dokumen kebijakan contoh untuk `BobUserRole` berikut memberikan akses ini.

```
{
    "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"
        }
    ]
}
```

Selain itu, `DAXServiceRole` akan memerlukan kebijakan IAM yang mengizinkan `DAXCluster01` melakukan tindakan hanya baca pada tabel `Books`.

```
{
    "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/writeakses ke DAX

Sekarang anggaplah bahwa Bob memerlukan akses baca/tulis ke tabel `Books`, langsung dari DynamoDB atau secara tidak langsung dari `DAXCluster01`. Dokumen kebijakan yang dilampirkan pada `BobAccessPolicy` berikut memberikan akses ini.

```
{
    "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": [
```

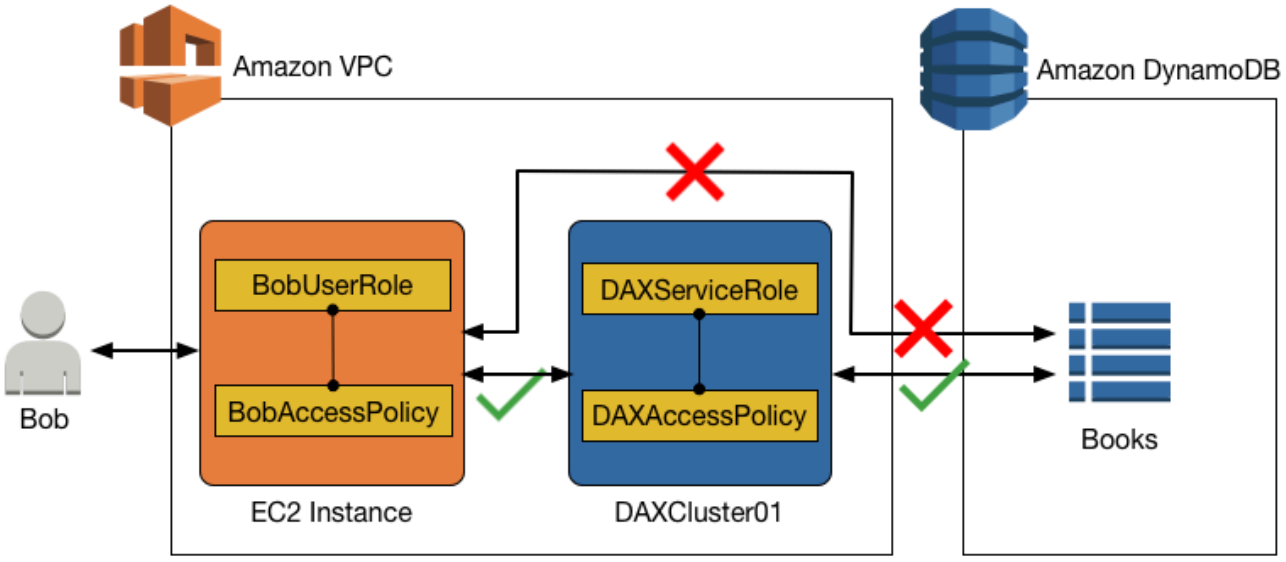
```
                "dynamodb:GetItem",
                "dynamodb:BatchGetItem",
                "dynamodb:Query",
                "dynamodb:Scan",
                "dynamodb:PutItem",
                "dynamodb:UpdateItem",
                "dynamodb:DeleteItem",
                "dynamodb:BatchWriteItem",
                "dynamodb:DescribeTable",
                "dynamodb:ConditionCheckItem"
            ],
            "Resource": "arn:aws:dynamodb:us-west-2:123456789012:table/Books"
        }
    ]
}
```

Selain itu, `DAXServiceRole` akan memerlukan kebijakan IAM yang mengizinkan `DAXCluster01` melakukan tindakan baca/tulis pada tabel `Books`.

```
{
    "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"
        }
    ]
}
```

Akses ke DynamoDB melalui DAX, tetapi tidak ada akses langsung ke DynamoDB

Dalam skenario ini, Bob dapat mengakses tabel `Books` melalui DAX, tetapi tidak memiliki akses langsung ke tabel `Books` di DynamoDB. Dengan demikian, ketika mendapatkan akses ke DAX, Bob juga mendapatkan akses ke tabel DynamoDB yang mungkin tidak dapat diakses. Ketika Anda mengonfigurasi kebijakan IAM untuk peran layanan DAX, ingat bahwa setiap pengguna yang diberikan akses ke klaster DAX melalui kebijakan akses pengguna mendapatkan akses ke tabel yang ditentukan dalam kebijakan tersebut. Dalam kasus ini, `BobAccessPolicy` mendapat akses ke tabel yang ditentukan di `DAXAccessPolicy`.



Jika saat ini Anda menggunakan kebijakan dan peran IAM untuk membatasi akses ke tabel dan data DynamoDB, penggunaan DAX dapat merusak kebijakan tersebut. Dalam kebijakan berikut, Bob memiliki akses ke tabel DynamoDB melalui DAX, tetapi tidak memiliki akses langsung eksplisit ke tabel yang sama di DynamoDB.

Dokumen kebijakan berikut (`BobAccessPolicy`), yang dilampirkan pada `BobUserRole`, memberikan akses ini.

```
{
    "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"
        }
    ]
}
```

Dalam kebijakan akses ini, tidak ada izin untuk mengakses DynamoDB secara langsung.

Bersama dengan `BobAccessPolicy`, `DAXAccessPolicy` berikut memberi `BobUserRole` akses ke tabel DynamoDB `Books` meskipun `BobUserRole` tidak dapat langsung mengakses tabel `Books`.

```
{
    "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"
```

```
        }
    ]
}
```

Seperti yang ditunjukkan contoh ini, ketika Anda mengonfigurasi kontrol akses untuk kebijakan akses pengguna dan kebijakan akses klaster DAX, Anda harus sepenuhnya memahami end-to-end akses untuk memastikan bahwa prinsip hak istimewa paling sedikit dipatuhi. Pastikan juga bahwa pemberian akses ke klaster DAX kepada pengguna tidak merusak kebijakan kontrol akses yang telah ada sebelumnya.

Enkripsi DAX saat diam

Enkripsi Amazon DynamoDB Accelerator (DAX) saat diam menyediakan lapisan perlindungan data tambahan dengan membantu mengamankan data Anda dari akses tidak sah ke penyimpanan yang mendasari. Kebijakan organisasi, peraturan industri atau pemerintah, dan persyaratan kepatuhan mungkin memerlukan penggunaan enkripsi saat diam untuk melindungi data Anda. Anda dapat menggunakan enkripsi untuk meningkatkan keamanan data aplikasi Anda yang di-deploy di cloud.

Dengan enkripsi saat istirahat, data yang disimpan oleh DAX pada disk dienkripsi menggunakan Advanced Encryption Standard 256-bit, yang juga dikenal sebagai enkripsi AES-256. DAX menulis data ke disk sebagai bagian dari penyebaran perubahan dari simpul primer ke replika baca.

Enkripsi DAX saat istirahat secara otomatis terintegrasi dengan AWS Key Management Service (AWS KMS) untuk mengelola kunci default layanan tunggal yang digunakan untuk mengenkripsi cluster Anda. Jika kunci default layanan tidak ada saat Anda membuat kluster DAX terenkripsi, AWS KMS secara otomatis akan membuat kunci AWS terkelola baru untuk Anda. Kunci ini digunakan dengan cluster terenkripsi yang dibuat di masa depan. AWS KMS menggabungkan perangkat keras dan perangkat lunak yang aman dan sangat tersedia untuk menyediakan sistem manajemen kunci yang diskalakan untuk cloud.

Setelah data Anda dienkripsi, DAX menangani dekripsi data Anda secara transparan tanpa terlalu banyak memengaruhi performa. Anda tidak perlu mengubah aplikasi untuk menggunakan enkripsi.

> **Note**
>
> DAX tidak memanggil AWS KMS untuk setiap operasi DAX tunggal. DAX hanya menggunakan kunci pada peluncuran klaster. Meskipun akses dicabut, DAX masih dapat

mengakses data hingga klaster dimatikan. AWS KMS Kunci yang ditentukan pelanggan tidak didukung.

Enkripsi DAX saat diam tersedia untuk jenis simpul klaster berikut.

Rangkaian	Jenis simpul
Dioptimalkan memori (R4, R5, dan R7)	dax.r4.large dax.r4.xlarge dax.r4.2xlarge dax.r4.4xlarge 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

Rangkaian	Jenis simpul
	dax.r7i.4xlarge dax.r7i.8xlarge dax.r7i.12xlarge dax.r7i.16xlarge dax.r7i.24xlarge
Tujuan umum (T2)	dax.t2.small dax.t2.medium
Tujuan umum (T3)	dax.t3.small dax.t3.medium

> **Important**
>
> Enkripsi DAX saat diam tidak didukung untuk jenis simpul `dax.r3.*`.

Anda tidak dapat mengaktifkan atau menonaktifkan enkripsi saat diam setelah klaster dibuat. Anda harus membuat ulang klaster untuk mengaktifkan enkripsi saat diam jika belum diaktifkan saat dibuat.

Enkripsi DAX saat istirahat ditawarkan tanpa biaya tambahan (biaya penggunaan kunci AWS KMS enkripsi berlaku). Untuk informasi selengkapnya tentang harga, lihat Harga Amazon DynamoDB.

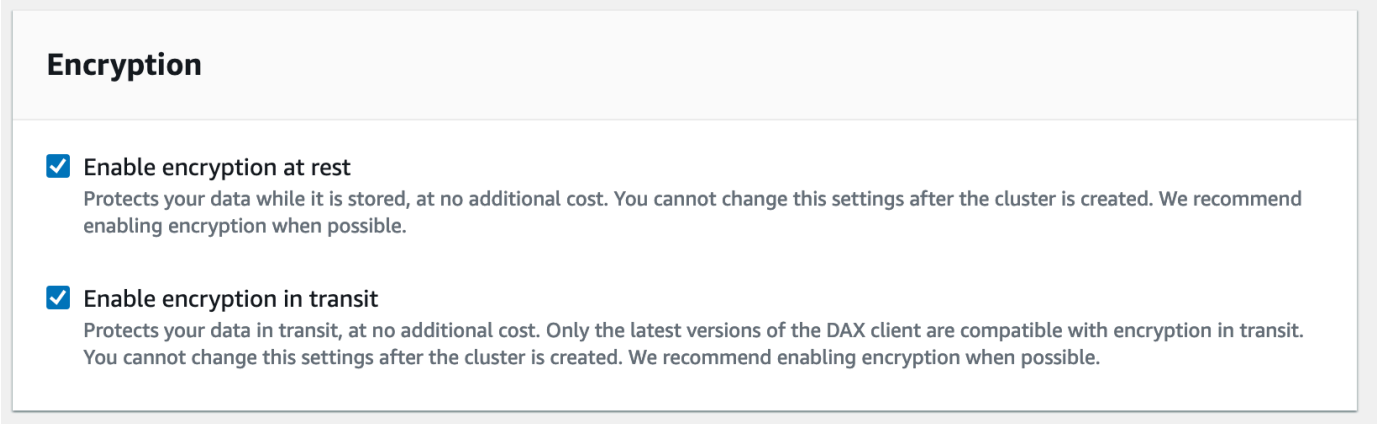
Mengaktifkan enkripsi saat diam menggunakan AWS Management Console

Ikuti langkah berikut untuk mengaktifkan enkripsi DAX saat diam pada tabel menggunakan konsol.

Cara mengaktifkan enkripsi DAX saat diam

1. Masuk ke AWS Management Console dan buka konsol DynamoDB di. https://console.aws.amazon.com/dynamodb/
2. Di panel navigasi di sisi kiri konsol, di bawah DAX, pilih Klaster.

3. Pilih Buat klaster.
4. Untuk Nama klaster, masukkan nama singkat untuk klaster Anda. Pilih jenis simpul untuk semua simpul di klaster, dan untuk ukuran klaster, gunakan simpul **3**.
5. Di Enkripsi, pastikan bahwa Aktifkan enkripsi dipilih.



6. Setelah memilih peran IAM, grup subnet, grup keamanan, dan pengaturan klaster, pilih Luncurkan klaster.

Untuk mengonfirmasi bahwa klaster telah dienkripsi, periksa detail klaster di bagian panel Klaster. Enkripsi harus DIAKTIFKAN.

Enkripsi bergerak DAX

Amazon DynamoDB Accelerator (DAX) mendukung enkripsi data bergerak antara aplikasi dan klaster DAX Anda, sehingga memungkinkan Anda menggunakan DAX dalam aplikasi dengan persyaratan enkripsi yang ketat.

Terlepas dari apakah Anda memilih enkripsi saat bergerak atau tidak, lalu lintas antara aplikasi dan klaster DAX Anda tetap ada di Amazon VPC. Lalu lintas ini diarahkan ke Antarmuka Jaringan Elastis dengan pribadi IPs di VPC Anda yang dilampirkan ke node cluster Anda. Dengan VPC Anda sebagai batas kepercayaan, Anda memiliki kontrol yang signifikan atas keamanan data Anda melalui penggunaan alat standar seperti grup keamanan, segmentasi subnet dengan Jaringan, ACLs dan penelusuran aliran VPC. Enkripsi DAX saat bergerak menambah tingkat acuan kerahasiaan, memastikan bahwa semua permintaan dan tanggapan antara aplikasi dan klaster dienkripsi oleh keamanan lapisan pengangkutan (TLS) dan koneksi ke klaster dapat dikonfirmasi oleh verifikasi sertifikat klaster x509. Data yang ditulis ke disk oleh DAX juga dapat dienkripsi jika Anda memilih enkripsi saat diam ketika membuat klaster DAX.

Menggunakan enkripsi saat bergerak dengan DAX cukup mudah. Cukup pilih opsi ini saat membuat klaster baru dan gunakan versi terbaru dari salah satu Klien DAX di aplikasi Anda. Klaster yang menggunakan enkripsi saat bergerak tidak mendukung lalu lintas yang tidak terenkripsi, sehingga tidak ada peluang kesalahan dalam mengonfigurasi aplikasi Anda dan melewati enkripsi. Klien DAX akan menggunakan sertifikat x509 klaster untuk mengautentikasi identitas klaster ketika menetapkan koneksi guna memastikan bahwa permintaan DAX Anda mengarah ke tujuan yang diinginkan. Semua metode pembuatan cluster DAX mendukung enkripsi dalam perjalanan:,, semua AWS Management Console AWS CLI SDKs, dan. AWS CloudFormation

Enkripsi saat bergerak tidak dapat diaktifkan pada klaster DAX yang sudah ada. Untuk menggunakan enkripsi saat bergerak dalam aplikasi DAX yang sudah ada, buat klaster baru dengan enkripsi saat bergerak diaktifkan, alihkan lalu lintas aplikasi Anda ke klaster tersebut, lalu hapus klaster lama.

Menggunakan peran IAM tertaut layanan untuk DAX

Amazon DynamoDB Accelerator (DAX) AWS Identity and Access Management menggunakan peran terkait layanan (IAM). Peran tertaut layanan adalah jenis peran IAM unik yang tertaut langsung ke DAX. Peran terkait layanan telah ditentukan sebelumnya oleh DAX dan mencakup semua izin yang diperlukan layanan untuk memanggil layanan lain AWS atas nama Anda.

Peran tertaut layanan memudahkan penyiapan DAX karena Anda tidak perlu menambahkan izin yang diperlukan secara manual. DAX menentukan izin peran tertaut layanan, dan kecuali jika ditentukan lain, hanya DAX yang dapat mengasumsikan perannya. Izin yang ditentukan mencakup kebijakan kepercayaan dan kebijakan izin. Kebijakan izin tersebut tidak dapat dilampirkan ke entitas IAM lainnya.

Anda dapat menghapus peran tersebut setelah terlebih dahulu menghapus sumber daya yang terkait. Hal ini melindungi sumber daya Anda karena Anda tidak dapat menghapus izin untuk mengakses sumber daya secara tidak sengaja.

Untuk informasi tentang layanan lain yang mendukung peran tertaut layanan, lihat Layanan AWS yang Berfungsi dengan IAM di Panduan Pengguna IAM. Cari layanan yang memiliki Ya dalam kolom Peran tertaut layanan. Pilih Ya untuk melihat dokumentasi peran terkait layanan untuk layanan tersebut.

Topik

- Izin peran tertaut layanan untuk DAX

- Membuat peran tertaut layanan untuk DAX
- Mengedit peran tertaut layanan untuk DAX
- Menghapus peran tertaut layanan untuk DAX

Izin peran tertaut layanan untuk DAX

DAX menggunakan peran tertaut layanan bernama `AWSServiceRoleForDAX`. Peran ini memungkinkan DAX memanggil layanan atas nama klaster DAX Anda.

> **Important**
>
> Peran tertaut layanan `AWSServiceRoleForDAX` membuat Anda lebih mudah untuk menyiapkan dan mempertahankan klaster DAX. Namun, Anda masih harus memberi setiap klaster akses ke DynamoDB sebelum dapat menggunakannya. Untuk informasi selengkapnya, lihat Kontrol akses DAX.

Peran tertaut layanan `AWSServiceRoleForDAX` memercayai layanan berikut untuk mengambil peran tersebut:

- `dax.amazonaws.com`

Kebijakan izin peran memungkinkan DAX menyelesaikan tindakan berikut pada sumber daya yang ditentukan:

- Tindakan pada `ec2`:
  - `AuthorizeSecurityGroupIngress`
  - `CreateNetworkInterface`
  - `CreateSecurityGroup`
  - `DeleteNetworkInterface`
  - `DeleteSecurityGroup`
  - `DescribeAvailabilityZones`
  - `DescribeNetworkInterfaces`
  - `DescribeSecurityGroups`
  - `DescribeSubnets`

- `DescribeVpcs`
- `ModifyNetworkInterfaceAttribute`
- `RevokeSecurityGroupIngress`

Anda harus mengonfigurasikan izin untuk mengizinkan entitas IAM (seperti pengguna, grup, atau peran) untuk membuat, menyunting, atau menghapus peran terhubung dengan layanan. Untuk informasi selengkapnya, silakan lihat Izin Peran Tertaut Layanan di Panduan Pengguna IAM.

Untuk mengizinkan entitas IAM membuat peran terkait layanan AWSService RoleFor DAX

Tambahkan pernyataan kebijakan berikut ini ke izin untuk entitas IAM tersebut.

```
{
    "Effect": "Allow",
    "Action": [
        "iam:CreateServiceLinkedRole"
    ],
    "Resource": "*",
    "Condition": {"StringLike": {"iam:AWSServiceName": "dax.amazonaws.com"}}
}
```

Membuat peran tertaut layanan untuk DAX

Anda tidak perlu membuat peran terkait layanan secara manual. Saat Anda membuat klaster, DAX membuat peran tertaut layanan untuk Anda.

> **Important**
>
> Jika Anda menggunakan layanan DAX sebelum 28 Februari 2018, DAX membuat peran `AWSServiceRoleForDAX` di akun Anda saat mulai mendukung peran tertaut layanan. Untuk informasi selengkapnya, lihat Peran Baru Muncul di AWS Akun Saya di Panduan Pengguna IAM.

Jika Anda menghapus peran tertaut layanan ini dan ingin membuatnya lagi, Anda dapat mengulangi proses yang sama untuk membuat kembali peran tersebut di akun Anda. Saat Anda membuat instans atau klaster, DAX membuat peran tertaut layanan lagi untuk Anda.

Mengedit peran tertaut layanan untuk DAX

DAX tidak mengizinkan Anda mengedit peran tertaut layanan `AWSServiceRoleForDAX`. Setelah membuat peran terkait layanan, Anda tidak dapat mengubah nama peran karena berbagai entitas mungkin mereferensikan peran tersebut. Namun, Anda dapat mengedit penjelasan peran menggunakan IAM. Untuk informasi selengkapnya, lihat Mengedit Peran Tertaut Layanan dalam Panduan Pengguna IAM.

Menghapus peran tertaut layanan untuk DAX

Jika tidak perlu lagi menggunakan fitur atau layanan yang memerlukan peran tertaut layanan, sebaiknya Anda menghapus peran tersebut. Dengan begitu, Anda tidak memiliki entitas yang tidak digunakan yang tidak dipantau atau dikelola secara aktif. Namun, Anda harus menghapus semua klaster DAX sebelum dapat menghapus peran tertaut layanan.

Membersihkan peran terkait layanan

Sebelum dapat menggunakan IAM untuk menghapus peran tertaut layanan, Anda harus mengonfirmasi terlebih dahulu bahwa peran tersebut tidak memiliki sesi aktif dan menghapus sumber daya yang digunakan oleh peran tersebut.

Untuk memeriksa apakah peran terkait layanan memiliki sesi aktif di konsol IAM

1. Masuk ke AWS Management Console dan buka konsol IAM di https://console.aws.amazon.com/iam/.
2. Di panel navigasi konsol IAM, pilih Peran. Kemudian pilih nama (bukan kotak centang) peran AWSServiceRoleForDAX.
3. Di halaman Ringkasan untuk peran yang dipilih, pilih tab Penasihat Akses.
4. Di tab Penasihat Akses, tinjau aktivitas terbaru untuk peran tertaut layanan tersebut.

> **Note**
>
> Jika Anda tidak yakin apakah DAX menggunakan peran `AWSServiceRoleForDAX` tersebut, coba hapus peran tersebut. Jika layanan menggunakan peran tersebut, penghapusan akan gagal dan Anda dapat melihat Wilayah tempat peran tersebut digunakan. Jika peran tersebut digunakan, Anda harus menghapus klaster DAX sebelum dapat menghapus peran tersebut. Anda tidak dapat mencabut sesi untuk peran tertaut layanan.

Jika ingin menghapus peran `AWSServiceRoleForDAX`, Anda harus terlebih dahulu menghapus semua klaster DAX Anda.

Menghapus semua klaster DAX

Gunakan salah satu dari prosedur berikut untuk menghapus setiap klaster DAX Anda.

Cara menghapus klaster DAX (konsol)

1. Buka konsol DynamoDB di. https://console.aws.amazon.com/dynamodb/
2. Pada panel navigasi, di bawah DAX, pilih Klaster.
3. Pilih Tindakan, lalu pilih Hapus.
4. Di kotak Hapus konfirmasi klaster, pilih Hapus.

Untuk menghapus klaster DAX (AWS CLI)

Lihat delete-cluster di Referensi Perintah AWS CLI .

Untuk menghapus klaster DAX (API)

Lihat DeleteClusterdi Referensi API Amazon DynamoDB.

Menghapus peran tertaut layanan

Untuk menghapus peran tertaut layanan secara manual menggunakan IAM

Gunakan konsol IAM, CLI IAM, atau API CLI untuk menghapus peran tertaut layanan `AWSServiceRoleForDAX`. Untuk informasi selengkapnya, lihat Menghapus Peran Terkait Layanan di Panduan Pengguna IAM.

Mengakses DAX di seluruh akun AWS

Bayangkan Anda memiliki klaster DynamoDB Accelerator (DAX) yang berjalan di AWS satu akun (akun A), dan kluster DAX harus dapat diakses dari instans Amazon Elastic Compute Cloud (EC2Amazon) di akun lain (akun B). AWS Dalam tutorial ini, Anda menyelesaikan ini dengan meluncurkan EC2 instance di akun B dengan peran IAM dari akun B. Anda kemudian menggunakan kredenal keamanan sementara dari EC2 instance untuk mengambil peran IAM dari akun A. Akhirnya, Anda menggunakan kredenal keamanan sementara dari asumsi peran IAM di akun A untuk melakukan panggilan aplikasi melalui koneksi peering VPC Amazon ke cluster DAX di akun A. Untuk melakukan tugas-tugas ini, Anda akan membutuhkan akses administratif di kedua akun. AWS

> **Important**
>
> Klaster DAX tidak dapat mengakses tabel DynamoDB dari akun yang berbeda.

Topik

- Menyiapkan IAM
- Menyiapkan VPC
- Mengubah klien DAX untuk mengizinkan akses lintas akun

Menyiapkan IAM

1. Buat file teks bernama `AssumeDaxRoleTrust.json` dengan konten berikut, yang EC2 memungkinkan Amazon bekerja atas nama Anda.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "Service": "ec2.amazonaws.com"
            },
            "Action": "sts:AssumeRole"
        }
    ]
}
```

2. Di akun B, buat peran yang EC2 dapat digunakan Amazon saat meluncurkan instance.

```
aws iam create-role \
    --role-name AssumeDaxRole \
    --assume-role-policy-document file://AssumeDaxRoleTrust.json
```

3. Buat file teks bernama `AssumeDaxRolePolicy.json` dengan konten berikut, yang memungkinkan kode berjalan pada EC2 instance di akun B untuk mengambil peran IAM di akun A. Ganti *`accountA`* dengan ID aktual akun A.

```
{
```

```
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "sts:AssumeRole",
            "Resource": "arn:aws:iam::accountA:role/DaxCrossAccountRole"
        }
    ]
}
```

4. Tambahkan kebijakan tersebut ke peran yang baru Anda buat.

```
aws iam put-role-policy \
    --role-name AssumeDaxRole \
    --policy-name AssumeDaxRolePolicy \
    --policy-document file://AssumeDaxRolePolicy.json
```

5. Buat profil instans untuk mengizinkan instans menggunakan peran.

```
aws iam create-instance-profile \
    --instance-profile-name AssumeDaxInstanceProfile
```

6. Kaitkan peran dengan profil instans.

```
aws iam add-role-to-instance-profile \
    --instance-profile-name AssumeDaxInstanceProfile \
    --role-name AssumeDaxRole
```

7. Buat file teks bernama `DaxCrossAccountRoleTrust.json` berisi konten berikut, yang mengizinkan akun B mengambil peran akun A. Ganti *`accountB`* dengan ID aktual akun B.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "AWS": "arn:aws:iam::accountB:role/AssumeDaxRole"
            },
            "Action": "sts:AssumeRole"
        }
    ]
```

```
}
```

8. Di akun A, buat peran yang dapat diambil oleh akun B.

```
aws iam create-role \
    --role-name DaxCrossAccountRole \
    --assume-role-policy-document file://DaxCrossAccountRoleTrust.json
```

9. Buat file teks bernama `DaxCrossAccountPolicy.json` yang mengizinkan akses ke klaster DAX. Ganti *`dax-cluster-arn`* dengan Nama Sumber Daya Amazon (ARN) yang benar dari cluster DAX Anda.

```
{
    "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": "dax-cluster-arn"
        }
    ]
}
```

10. Di akun A, tambahkan kebijakan ke peran tersebut.

```
aws iam put-role-policy \
    --role-name DaxCrossAccountRole \
    --policy-name DaxCrossAccountPolicy \
    --policy-document file://DaxCrossAccountPolicy.json
```

Menyiapkan VPC

1. Temukan grup subnet klaster DAX akun A. Ganti *cluster-name* dengan nama cluster DAX yang harus diakses akun B.

```
aws dax describe-clusters \
    --cluster-name cluster-name
    --query 'Clusters[0].SubnetGroup'
```

2. Dengan menggunakan itu*subnet-group*, temukan VPC cluster.

```
aws dax describe-subnet-groups \
    --subnet-group-name subnet-group \
    --query 'SubnetGroups[0].VpcId'
```

3. Dengan menggunakan itu*vpc-id*, temukan CIDR VPC.

```
aws ec2 describe-vpcs \
    --vpc vpc-id \
    --query 'Vpcs[0].CidrBlock'
```

4. Dari akun B, buat VPC menggunakan CIDR yang berbeda dan tidak tumpang tindih dari yang ditemukan di langkah sebelumnya. Kemudian, buat setidaknya satu subnet. Anda dapat menggunakan wizard pembuatan VPC di AWS Management Console atau. AWS CLI
5. Dari akun B, minta koneksi peering ke VPC akun A seperti yang dijelaskan di Membuat dan menerima koneksi peering VPC. Dari akun A, terima koneksi.
6. Dari akun B, temukan tabel perutean VPC baru. Ganti *vpc-id* dengan ID VPC yang Anda buat di akun B.

```
aws ec2 describe-route-tables \
    --filters 'Name=vpc-id,Values=vpc-id' \
    --query 'RouteTables[0].RouteTableId'
```

7. Tambahkan rute untuk mengirim lalu lintas yang dibuat untuk CIDR akun A ke koneksi peering VPC. Ingatlah untuk mengganti masing-masing *user input placeholder* dengan nilai yang benar untuk akun Anda.

```
aws ec2 create-route \
    --route-table-id accountB-route-table-id \
    --destination-cidr accountA-vpc-cidr \
```

```
    --vpc-peering-connection-id peering-connection-id
```

8. Dari akun A, temukan tabel rute cluster DAX menggunakan tabel yang *vpc-id* Anda temukan sebelumnya.

```
aws ec2 describe-route-tables \
    --filters 'Name=vpc-id, Values=accountA-vpc-id' \
    --query 'RouteTables[0].RouteTableId'
```

9. Dari akun A, tambahkan rute untuk mengirim lalu lintas yang dibuat untuk CIDR akun B ke koneksi peering VPC. Ganti masing-masing *user input placeholder* dengan nilai yang benar untuk akun Anda.

```
aws ec2 create-route \
    --route-table-id accountA-route-table-id \
    --destination-cidr accountB-vpc-cidr \
    --vpc-peering-connection-id peering-connection-id
```

10. Dari akun B, luncurkan EC2 instance di VPC yang Anda buat sebelumnya. Berikan `AssumeDaxInstanceProfile`. Anda dapat menggunakan wizard peluncuran di AWS Management Console atau AWS CLI. Perhatikan grup keamanan instans.

11. Dari akun A, temukan grup keamanan yang digunakan oleh klaster DAX. Ingatlah untuk mengganti *cluster-name* dengan nama cluster DAX Anda.

```
aws dax describe-clusters \
    --cluster-name cluster-name \
    --query 'Clusters[0].SecurityGroups[0].SecurityGroupIdentifier'
```

12. Perbarui grup keamanan klaster DAX untuk mengizinkan lalu lintas masuk dari grup keamanan EC2 instans yang Anda buat di akun B. Ingatlah untuk mengganti *user input placeholders* dengan nilai yang benar untuk akun Anda.

```
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
```

Pada titik ini, aplikasi pada EC2 instans akun B dapat menggunakan profil instance untuk mengambil `arn:aws:iam::accountA-id:role/DaxCrossAccountRole` peran dan menggunakan cluster DAX.

Mengubah klien DAX untuk mengizinkan akses lintas akun

> **Note**
>
> AWS Security Token Service (AWS STS) kredensial adalah kredensial sementara. Beberapa klien menangani pembaruan secara otomatis, sementara klien lain memerlukan logika tambahan untuk memperbarui kredensialnya. Sebaiknya Anda mengikuti panduan dokumentasi yang sesuai.

Java

Bagian ini membantu Anda mengubah kode klien DAX yang ada untuk mengizinkan akses DAX lintas akun. Jika belum memiliki kode klien DAX, Anda dapat menemukan contoh kode yang berfungsi di tutorial Java dan DAX.

1. Tambahkan impor berikut.

```
import com.amazonaws.auth.STSAssumeRoleSessionCredentialsProvider;
import com.amazonaws.services.securitytoken.AWSSecurityTokenService;
import
 com.amazonaws.services.securitytoken.AWSSecurityTokenServiceClientBuilder;
```

2. Dapatkan penyedia kredensial dari AWS STS dan buat objek klien DAX. Ingatlah untuk mengganti masing-masing *user input placeholder* dengan nilai yang benar untuk akun Anda.

```
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

Bagian ini membantu Anda mengubah kode klien DAX yang ada untuk mengizinkan akses DAX lintas akun. Jika belum memiliki kode klien DAX, Anda dapat menemukan contoh kode yang berfungsi di tutorial .NET dan DAX.

1. Tambahkan AWSSDK. SecurityToken NuGet paket untuk solusinya.

```
<PackageReference Include="AWSSDK.SecurityToken" Version="latest version" />
```

2. Gunakan paket `SecurityToken` dan `SecurityToken.Model`.

```
using Amazon.SecurityToken;
using Amazon.SecurityToken.Model;
```

3. Dapatkan kredensial sementara dari `AmazonSimpleTokenService` dan buat objek `ClusterDaxClient`. Ingatlah untuk mengganti masing-masing *`user input placeholder`* dengan nilai yang benar untuk akun Anda.

```
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

Bagian ini membantu Anda mengubah kode klien DAX yang ada untuk mengizinkan akses DAX lintas akun. Jika Anda belum memiliki kode klien DAX, Anda dapat menemukan contoh kode yang berfungsi. GitHub

1. Impor paket AWS STS dan sesi.

```
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. Dapatkan kredensial sementara dari `AmazonSimpleTokenService` dan buat objek klien DAX. Ingatlah untuk mengganti masing-masing *`user input placeholder`* dengan nilai yang benar untuk akun Anda.

```
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

Bagian ini membantu Anda mengubah kode klien DAX yang ada untuk mengizinkan akses DAX lintas akun. Jika belum memiliki kode klien DAX, Anda dapat menemukan contoh kode yang berfungsi di tutorial Python dan DAX.

1. Impor `boto3`.

```
import boto3
```

2. Dapatkan kredensial sementara dari `sts` dan buat objek `AmazonDaxClient`. Ingatlah untuk mengganti masing-masing *`user input placeholder`* dengan nilai yang benar untuk akun Anda.

```
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

Bagian ini membantu Anda mengubah kode klien DAX yang ada untuk mengizinkan akses DAX lintas akun. Jika belum memiliki kode klien DAX, Anda dapat menemukan contoh kode yang berfungsi di tutorial Node.js dan DAX. Ingatlah untuk mengganti masing-masing *`user input placeholder`* dengan nilai yang benar untuk akun Anda.

```
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);
});
```

Panduan pengukuran klaster DAX

Panduan ini memberikan saran untuk memilih ukuran klaster dan jenis simpul Amazon DynamoDB Accelerator (DAX) yang sesuai untuk aplikasi Anda. Petunjuk ini memandu Anda melakukan langkah-langkah untuk memperkirakan lalu lintas DAX aplikasi, memilih konfigurasi klaster, dan mengujinya.

Jika Anda memiliki klaster DAX yang sudah ada dan ingin mengevaluasi apakah klaster memiliki jumlah dan ukuran simpul yang sesuai, lihat Menskalakan klaster DAX.

Topik

- Gambaran Umum
- Memperkirakan lalu lintas
- Pengujian beban

Gambaran Umum

Penting menskalakan klaster DAX untuk beban kerja secara tepat, baik ketika membuat klaster baru atau mempertahankan klaster yang sudah ada. Seiring berjalannya waktu dan perubahan beban kerja aplikasi, Anda harus meninjau kembali keputusan penskalaan secara berkala untuk memastikan bahwa semuanya masih tepat.

Biasanya, proses mengikuti langkah-langkah berikut:

1. Memperkirakan lalu lintas. Dalam langkah ini, Anda membuat prediksi tentang volume lalu lintas yang akan dikirimkan oleh aplikasi ke DAX, sifat lalu lintas (operasi baca vs. tulis), dan laju hit cache yang diharapkan.
2. Pengujian beban. Pada langkah ini, Anda membuat klaster dan mengirim lalu lintas yang mencerminkan estimasi Anda dari langkah sebelumnya ke klaster tersebut. Ulangi langkah ini hingga Anda menemukan konfigurasi klaster yang sesuai.
3. Pemantauan produksi. Saat aplikasi menggunakan DAX dalam produksi, Anda harus memantau klaster untuk terus memvalidasi bahwa klaster masih melakukan penskalaan dengan benar sebagai perubahan beban kerja Anda dari waktu ke waktu.

Memperkirakan lalu lintas

Ada tiga faktor utama yang menjadi ciri beban kerja DAX umum:

- Laju hit cache
- Baca unit kapasitas (RCUs) per detik
- Tulis unit kapasitas (WCUs) per detik

Memperkirakan laju hit cache

Jika Anda sudah memiliki cluster DAX, Anda dapat menggunakan CloudWatch metrik `ItemCacheHits` dan `ItemCacheMisses` Amazon untuk menentukan tingkat hit cache. laju hit cache sama dengan `ItemCacheHits` / (`ItemCacheHits` + `ItemCacheMisses`). Jika beban kerja Anda mencakup `Query` atau `Scan`, Anda juga harus melihat metrik `QueryCacheHits`, `QueryCacheMisses`, `ScanCacheHits`, dan `ScanCacheMisses`. Laju hit cache bervariasi antara satu aplikasi dengan aplikasi yang lain dan sangat dipengaruhi oleh pengaturan Time to Live (TTL) klaster. Laju hit umum untuk aplikasi yang menggunakan DAX adalah 85-95 persen.

Memperkirakan unit kapasitas baca dan tulis

Jika Anda sudah memiliki tabel DynamoDB untuk aplikasi Anda, lihat `ConsumedReadCapacityUnits` dan metrik. `ConsumedWriteCapacityUnits` CloudWatch Gunakan statistik `Sum` dan bagi dengan jumlah detik dalam periode tersebut.

Jika Anda juga sudah memiliki klaster DAX, ingat bahwa metrik `ConsumedReadCapacityUnits` DynamoDB hanya memperhitungkan cache miss. Jadi, untuk mengetahui unit kapasitas baca per detik yang ditangani oleh klaster DAX Anda, bagi angka dengan laju cache miss Anda (yaitu, 1 - laju hit cache).

Jika Anda belum memiliki tabel DynamoDB, lihat dokumentasi tentang unit kapasitas baca dan tulis untuk memperkirakan lalu lintas Anda berdasarkan perkiraan tingkat permintaan aplikasi Anda, item yang diakses per permintaan, dan ukuran item.

Ketika membuat estimasi lalu lintas, rencanakan pertumbuhan di waktu mendatang dan untuk puncak yang diharapkan dan tidak terduga untuk memastikan bahwa klaster Anda memiliki ruang kosong yang cukup untuk meningkatkan lalu lintas.

Pengujian beban

Langkah berikutnya setelah memperkirakan lalu lintas adalah menguji konfigurasi klaster dengan beban.

1. Untuk pengujian beban awal, sebaiknya Anda memulai dengan jenis simpul `dax.r4.large`, biaya performa tetap paling murah, jenis simpul dengan memori yang dioptimalkan.
2. Klaster yang toleran terhadap kesalahan memerlukan setidaknya tiga simpul yang tersebar di tiga Zona Ketersediaan. Dalam kasus ini, jika Zona Ketersediaan menjadi tidak tersedia, jumlah Zona Ketersediaan efektif berkurang sepertiga. Untuk pengujian beban awal, sebaiknya Anda mulai

dengan klaster dua simpul, yang menyimulasikan kegagalan satu Zona Ketersediaan di klaster tiga simpul.

3. Dorong lalu lintas berkelanjutan (seperti yang diperkirakan di langkah sebelumnya) ke klaster pengujian selama pengujian beban.
4. Pantau performa klaster selama pengujian beban.

Idealnya, profil lalu lintas yang Anda dorong selama pengujian beban harus semirip mungkin dengan lalu lintas aplikasi Anda yang sebenarnya. Hal ini termasuk distribusi operasi (misalnya, 70 persen `GetItem`, 25 persen `Query`, dan 5 persen `PutItem`), tingkat permintaan untuk setiap operasi, jumlah item yang diakses per permintaan, dan distribusi ukuran item. Untuk mencapai laju hit cache yang mirip dengan laju hit cache yang diharapkan aplikasi Anda, perhatikan distribusi kunci di lalu lintas pengujian Anda.

> **Note**
>
> Hati-hati saat memuat pengujian jenis simpul T2 (`dax.t2.small` dan `dax.t2.medium`). Jenis simpul T2 menyediakan performa CPU yang dapat melonjak yang bervariasi dari waktu ke waktu tergantung saldo kredit CPU simpul. Klaster DAX yang berjalan pada simpul T2 mungkin tampak beroperasi secara normal, tetapi jika ada simpul yang mengalami lonjakan di atas performa acuan instansnya, simpul akan menggunakan saldo kredit CPU yang diakumulasikan. Ketika saldo kredit hampir habis, performa akan menurun secara bertahap ke tingkat performa acuan.

Pantau klaster DAX Anda selama pengujian beban untuk menentukan apakah jenis simpul yang Anda gunakan untuk pengujian beban adalah jenis simpul yang tepat untuk Anda. Selain itu, selama pengujian beban, Anda harus memantau tingkat permintaan dan laju hit cache untuk memastikan bahwa infrastruktur pengujian Anda benar-benar mendorong jumlah lalu lintas yang diinginkan.

Anda harus memperhatikan konsumsi byte jaringan dari jenis instans klaster yang dipilih. Melebihi bandwidth dasar yang tersedia untuk EC2 instans Amazon menunjukkan bahwa klaster Anda mungkin tidak menopang beban kerja aplikasi Anda, dan perlu diskalakan.

Jika pengujian beban menunjukkan bahwa konfigurasi klaster yang dipilih tidak dapat mempertahankan beban kerja aplikasi, Anda harus beralih ke jenis simpul yang lebih besar, terutama jika melihat pemanfaatan CPU yang tinggi pada simpul primer di klaster, tingkat pengosongan tinggi, atau pemanfaatan memori cache yang tinggi. Jika laju hit konsisten tinggi dan rasio lalu lintas baca

terhadap lalu lintas tulis tinggi, Anda dapat mempertimbangkan untuk menambahkan lebih banyak simpul ke klaster Anda. Lihat Menskalakan klaster DAX untuk mendapatkan panduan tambahan tentang waktu yang tepat untuk menggunakan jenis simpul yang lebih besar (penskalaan vertikal) atau menambahkan lebih banyak simpul (penskalaan horizontal).

Anda harus mengulangi pengujian beban setelah membuat perubahan pada konfigurasi klaster.

Pemodelan data untuk tabel DynamoDB

Sebelum kita menyelami pemodelan data, penting untuk memahami beberapa hal-hal dasar DynamoDB. DynamoDB adalah basis data NoSQL kunci-nilai yang memungkinkan skema fleksibel. Kumpulan atribut data selain atribut kunci untuk setiap item dapat berupa seragam atau diskrit. Skema kunci DynamoDB adalah dalam bentuk kunci primer sederhana di mana kunci partisi secara unik mengidentifikasi item, atau dalam bentuk kunci primer komposit di mana kombinasi kunci partisi dan kunci urutan secara unik mendefinisikan item. Kunci partisi di-hash untuk menentukan lokasi fisik data dan mengambilnya. Oleh karena itu, penting untuk memilih atribut dengan kardinalitas tinggi dan dapat diskalakan secara horizontal sebagai kunci partisi untuk memastikan distribusi data yang merata. Atribut kunci sortir adalah opsional dalam skema kunci dan memiliki kunci pengurutan memungkinkan pemodelan one-to-many hubungan dan membuat koleksi item di DynamoDB. Kunci urutan juga disebut sebagai kunci rentang, yang digunakan untuk mengurutkan item dalam koleksi item dan juga memungkinkan operasi berbasis rentang yang fleksibel.

Untuk detail selengkapnya dan praktik terbaik skema kunci DynamoDB, Anda dapat merujuk ke hal berikut:

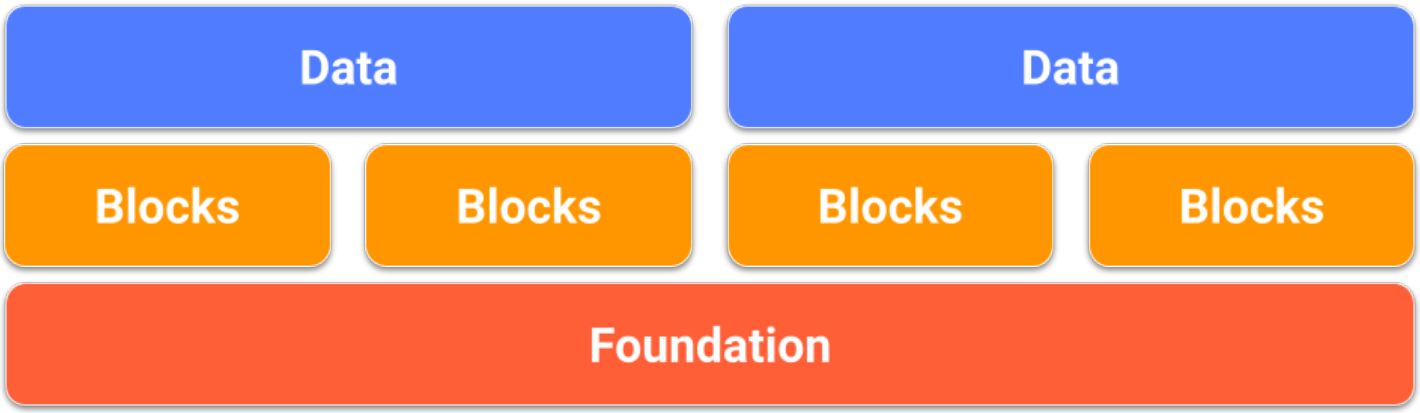
- the section called “Partisi dan distribusi data di DynamoDB”
- the section called “Desain kunci partisi”
- the section called “Desain kunci urutan”
- Memilih kunci partisi DynamoDB yang tepat

Indeks sekunder kerap diperlukan untuk mendukung pola kueri tambahan di DynamoDB. Indeks sekunder adalah tabel bayangan tempat data yang sama diatur melalui skema kunci yang berbeda dibandingkan dengan tabel dasar. Indeks sekunder lokal (LSI) berbagi kunci partisi yang sama dengan tabel dasar dan memungkinkan untuk memiliki kunci urutan alternatif yang memungkinkan untuk berbagi kapasitas tabel dasar. Indeks sekunder global (GSI) dapat memiliki kunci partisi yang berbeda serta atribut kunci urutan yang berbeda dari tabel dasar, yang berarti manajemen throughput untuk GSI tidak tergantung pada tabel dasar.

Untuk detail selengkapnya tentang indeks sekunder dan praktik terbaik, Anda dapat merujuk ke hal berikut:

- the section called “Bekerja dengan indeks”
- the section called “Indeks sekunder”

Sekarang mari kita lihat pemodelan data lebih dekat. Proses merancang skema yang fleksibel dan sangat dioptimalkan pada DynamoDB, atau basis data NoSQL apa pun dalam hal ini, dapat menjadi keterampilan yang menantang untuk dipelajari. Sasaran dari modul ini adalah untuk membantu Anda mengembangkan diagram alur mental untuk merancang skema yang akan membawa Anda dari kasus penggunaan ke produksi. Kita akan mulai dengan pengantar pilihan dasar dari desain apa pun, desain tabel tunggal versus desain multitabel. Kemudian, kita akan meninjau banyak pola desain (blok bangunan) yang dapat digunakan untuk mencapai berbagai hasil organisasi atau performa untuk aplikasi Anda. Terakhir, kita menyertakan berbagai paket desain skema lengkap untuk berbagai kasus penggunaan dan industri.



Topik

- Koleksi item - cara memodelkan one-to-many hubungan di DynamoDB
- Fondasi Pemodelan Data di DynamoDB
- Blok bangunan pemodelan data di DynamoDB
- Paket desain skema pemodelan data di DynamoDB
- Praktik terbaik untuk memodelkan data relasional di DynamoDB

Koleksi item - cara memodelkan one-to-many hubungan di DynamoDB

Di DynamoDB, koleksi item adalah sekelompok item yang berbagi nilai kunci partisi yang sama, yang berarti item tersebut terkait. Koleksi item adalah mekanisme utama untuk memodelkan one-to-many hubungan di DynamoDB. Koleksi item hanya bisa ada pada tabel atau indeks yang dikonfigurasi untuk menggunakan kunci primer komposit.

> **Note**
>
> Koleksi item bisa ada di tabel dasar atau indeks sekunder. Untuk informasi selengkapnya secara spesifik tentang bagaimana kumpulan item berinteraksi dengan indeks, lihat Kumpulan item dalam Indeks Sekunder Lokal.

Perhatikan tabel berikut yang menunjukkan tiga pengguna berbeda dan inventaris dalam game mereka:

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

Untuk beberapa item di setiap koleksi, kunci urutan adalah gabungan informasi yang digunakan untuk mengelompokkan data, misalnya `inventory::armor`, `inventory::weapon`, atau `info`. Setiap koleksi item dapat memiliki kombinasi yang berbeda dari atribut ini sebagai kunci urutan. Pengguna `account1234` memiliki item `inventory::weapons`, sementara pengguna `account1387` tidak memilikinya (karena mereka belum menemukan apa pun). Pengguna `account1138` hanya menggunakan dua item untuk kunci urutan mereka (karena mereka belum memiliki inventaris) sedangkan pengguna lain menggunakan tiga item.

DynamoDB memungkinkan Anda secara selektif mengambil item dari koleksi item ini untuk melakukan hal berikut:

- Mengambil semua item dari pengguna tertentu

- Mengambil hanya satu item dari pengguna tertentu
- Mengambil semua item dari jenis tertentu milik pengguna tertentu

Mempercepat kueri dengan mengatur data Anda dengan koleksi item

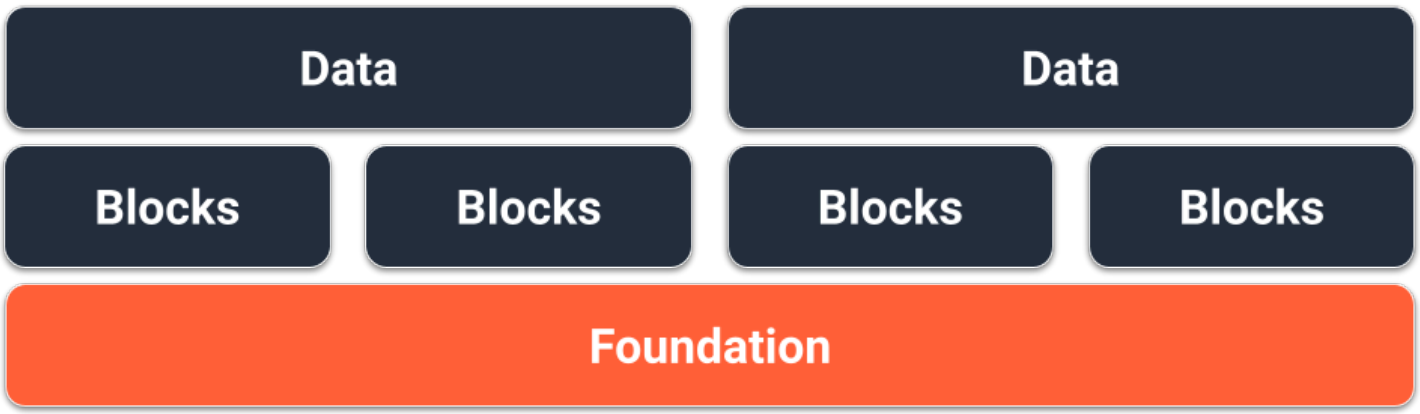
Dalam contoh ini, masing-masing item dalam tiga kumpulan item ini mewakili pemain dan model data yang telah kita pilih, berdasarkan pola akses game dan pemain. Data apa yang dibutuhkan game? Kapan game membutuhkannya? Seberapa sering game membutuhkannya? Berapa biaya untuk melakukannya dengan cara ini? Keputusan pemodelan data ini dibuat berdasarkan jawaban atas pertanyaan-pertanyaan ini.

Dalam game ini, ada halaman berbeda yang disajikan kepada pemain untuk inventaris senjata dan halaman lain untuk baju besi. Saat pemain membuka inventarisnya, senjata ditampilkan terlebih dahulu karena kami ingin halaman tersebut dimuat dengan sangat cepat, sedangkan halaman inventaris berikutnya dapat dimuat setelahnya. Karena masing-masing jenis item ini bisa berukuran cukup besar seiring pemain memperoleh lebih banyak item dalam game, kami memutuskan bahwa setiap halaman inventaris akan menjadi itemnya sendiri dalam koleksi item pemain di basis data.

Bagian berikut membahas lebih lanjut tentang bagaimana Anda dapat berinteraksi dengan koleksi item melalui operasi `Query`.

Fondasi Pemodelan Data di DynamoDB

Bagian ini mencakup lapisan fondasi dengan memeriksa dua jenis desain tabel: tabel tunggal dan multitabel.



Fondasi desain tabel tunggal

Salah satu pilihan untuk fondasi skema DynamoDB adalah desain tabel tunggal. Desain tabel tunggal adalah pola yang memungkinkan Anda menyimpan beberapa jenis (entitas) data dalam tabel DynamoDB tunggal. Hal ini bertujuan untuk mengoptimalkan pola akses data, meningkatkan performa, dan mengurangi biaya dengan menghilangkan kebutuhan untuk memelihara beberapa tabel dan hubungan yang kompleks di antara tabel-tabel tersebut. Hal ini dimungkinkan karena DynamoDB menyimpan item dengan kunci partisi yang sama (dikenal sebagai koleksi item) pada partisi yang sama satu sama lain. Dalam desain ini, berbagai jenis data disimpan sebagai item dalam tabel yang sama, dan setiap item diidentifikasi dengan kunci urutan yang unik.

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

Keuntungan

- Lokalitas data untuk mendukung kueri untuk beberapa jenis entitas dalam satu panggilan basis data
- Mengurangi biaya keuangan dan latensi pembacaan secara keseluruhan:
  - Satu kueri untuk dua item dengan total kurang dari 4KB adalah 0,5 RCU yang pada akhirnya konsisten

  - Dua kueri untuk dua item dengan total kurang dari 4KB adalah 1 RCU yang pada akhirnya konsisten (masing-masing 0,5 RCU)
  - Waktu untuk mengembalikan dua panggilan database terpisah akan rata-rata lebih tinggi dari satu panggilan
- Mengurangi jumlah tabel untuk dikelola:
  - Izin tidak perlu dipertahankan di beberapa kebijakan atau peran IAM
  - Manajemen kapasitas untuk tabel dirata-ratakan di seluruh entitas, biasanya menghasilkan pola konsumsi yang lebih dapat diprediksi
  - Pemantauan membutuhkan alarm yang lebih sedikit
  - Kunci Enkripsi yang Dikelola Pelanggan hanya perlu dirotasi pada satu tabel
- Memperlancar lalu lintas ke tabel:
  - Dengan menggabungkan beberapa pola penggunaan ke tabel yang sama, penggunaan secara keseluruhan cenderung lebih lancar (seperti performa indeks saham yang cenderung lebih lancar dibandingkan dengan saham individual) yang bekerja lebih baik untuk mencapai penggunaan yang lebih tinggi dengan tabel mode yang telah disediakan

Kekurangan

- Kurva pembelajaran bisa curam karena desain paradoks dibandingkan dengan basis data relasional
- Persyaratan data harus konsisten di semua jenis entitas
  - Pencadangan semuanya atau tidak sama sekali jadi jika beberapa data tidak penting untuk misi, pertimbangkan untuk menyimpannya di tabel terpisah
  - Enkripsi tabel dibagikan di semua item. Untuk aplikasi multi-penghuni dengan persyaratan enkripsi penghuni individual, enkripsi di sisi klien akan diperlukan
  - Tabel dengan campuran data historis dan data operasional tidak akan mendapatkan banyak manfaat dengan mengaktifkan Kelas Penyimpanan Akses Jarang. Untuk informasi selengkapnya, silakan lihat Kelas tabel DynamoDB
- Semua data yang diubah akan disebarkan ke DynamoDB Streams meskipun hanya sebagian dari entitas yang perlu diproses.
  - Berkat filter peristiwa Lambda, hal ini tidak akan memengaruhi tagihan Anda saat menggunakan Lambda, tetapi akan menjadi biaya tambahan saat menggunakan Kinesis Consumer Library
- Saat menggunakan GraphQL, desain tabel tunggal akan lebih sulit diterapkan

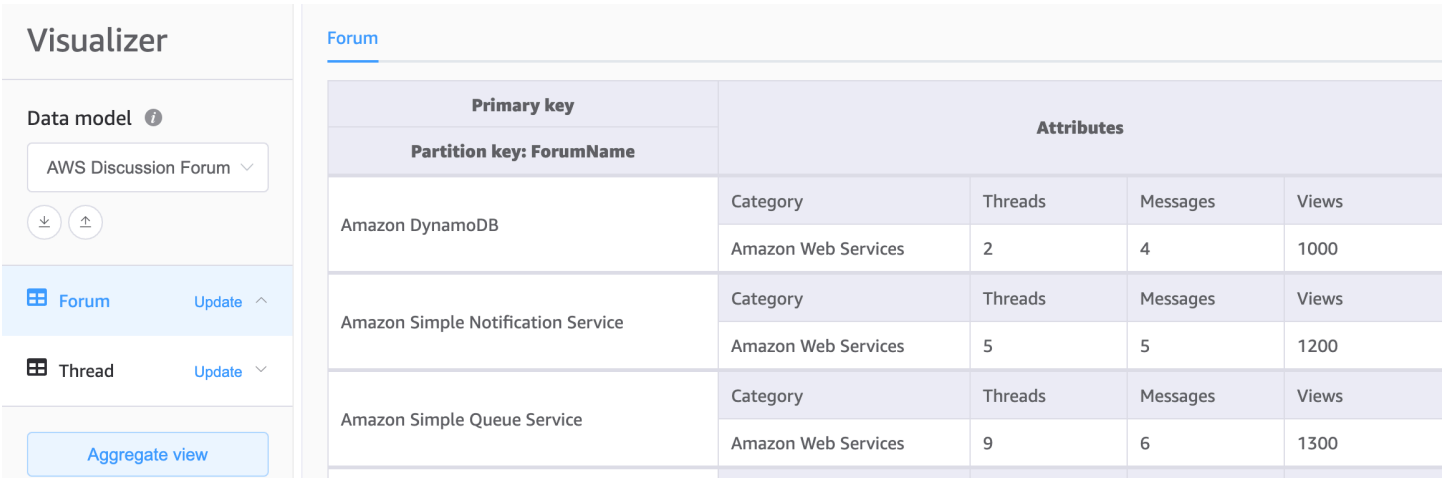
- Saat menggunakan klien SDK tingkat tinggi seperti `DynamoDBMapper` Java atau Enhanced Client, akan lebih sulit untuk memproses hasil karena item dalam respons yang sama dapat dikaitkan dengan kelas yang berbeda

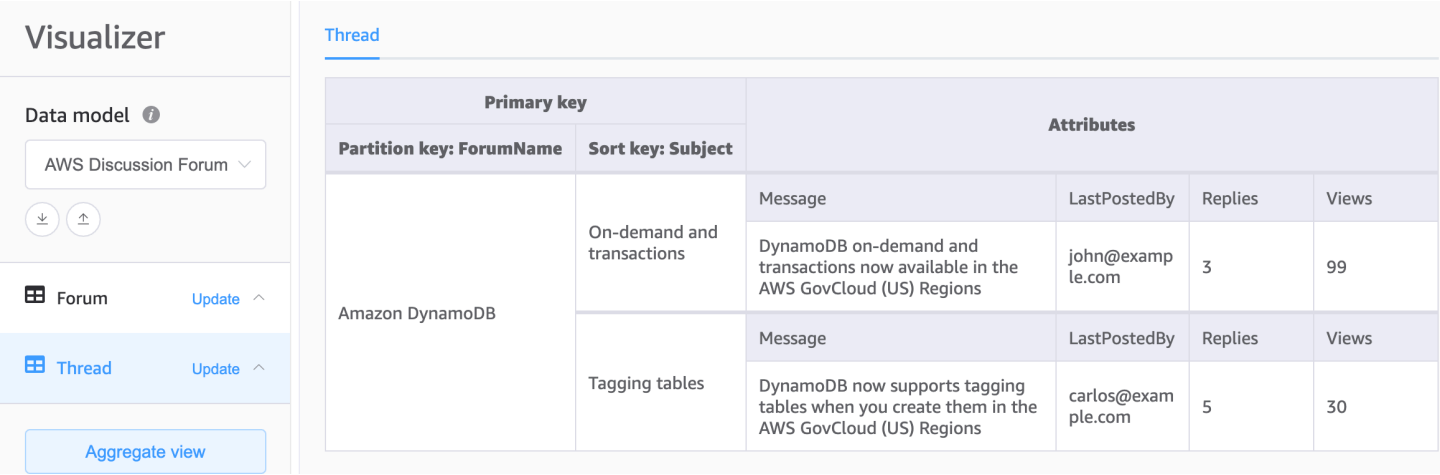
Kapan harus digunakan

Desain tabel tunggal adalah pola desain yang direkomendasikan untuk DynamoDB kecuali kasus penggunaan Anda akan sangat dipengaruhi oleh salah satu kekurangan di atas. Bagi sebagian besar pelanggan, manfaat jangka panjang lebih besar daripada tantangan jangka pendeknya, yaitu merancang tabel mereka dengan cara ini.

Fondasi desain multitabel

Pilihan kedua untuk fondasi skema DynamoDB kita adalah desain multitabel. Desain multitabel adalah pola yang lebih mirip dengan desain basis data tradisional tempat Anda menyimpan satu jenis (entitas) data dalam setiap tabel DynamoDB. Data dalam setiap tabel tetap akan diatur oleh kunci partisi, sehingga performa dalam satu jenis entitas akan dioptimalkan untuk skalabilitas dan performa, tetapi kueri di beberapa tabel harus dilakukan secara independen.





Keuntungan

- Perancangannya lebih sederhana bagi mereka yang tidak terbiasa bekerja dengan desain tabel tunggal
- Implementasi resolver GraphQL yang lebih mudah karena setiap resolver dipetakan ke satu entitas (tabel)
- Memungkinkan persyaratan data unik di berbagai jenis entitas:
  - Cadangan dapat dibuat untuk tabel individu yang sangat penting untuk misi
  - Enkripsi tabel dapat dikelola untuk setiap tabel. Untuk aplikasi multi-penghuni dengan persyaratan enkripsi penghuni individual, tabel penghuni terpisah memungkinkan setiap pelanggan memiliki kunci enkripsinya sendiri
  - Kelas Penyimpanan Akses Jarang dapat diaktifkan hanya pada tabel dengan data historis untuk merealisasikan manfaat penghematan biaya penuh. Untuk informasi selengkapnya, silakan lihat Kelas tabel DynamoDB
- Setiap tabel akan memiliki aliran data perubahannya sendiri yang memungkinkan fungsi Lambda khusus dirancang untuk setiap jenis item daripada prosesor monolitik tunggal

Kekurangan

- Untuk pola akses yang memerlukan data di beberapa tabel, beberapa pembacaan dari DynamoDB akan diperlukan dan data mungkin perlu diproses/digabungkan pada kode klien.
- Operasi dan pemantauan beberapa tabel membutuhkan lebih banyak CloudWatch alarm dan setiap tabel harus diskalakan secara independen

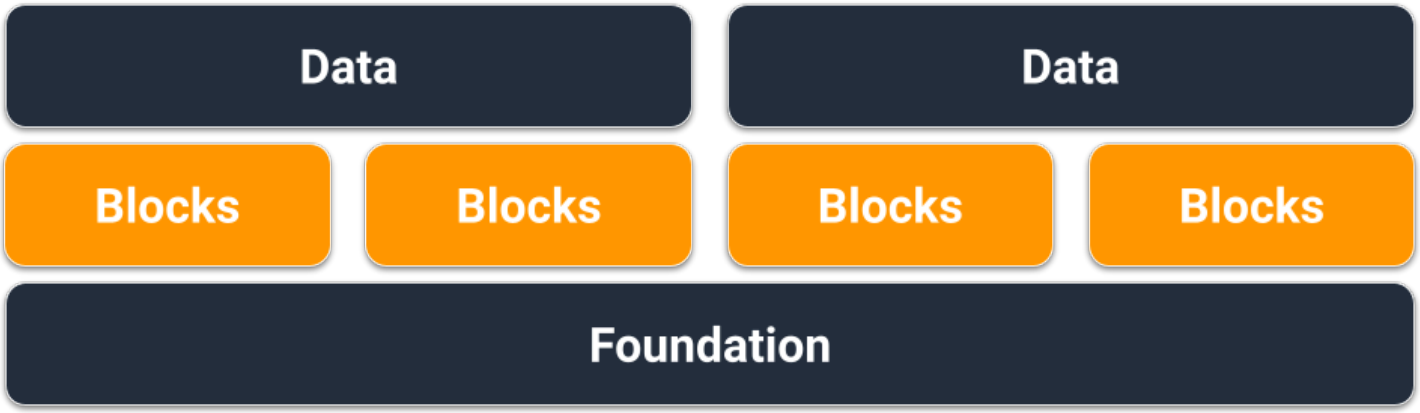
- Setiap izin tabel perlu dikelola secara terpisah. Penambahan tabel nantinya akan memerlukan perubahan pada kebijakan atau peran IAM yang diperlukan

Kapan harus digunakan

Jika pola akses aplikasi Anda tidak memiliki kebutuhan untuk melakukan kueri untuk beberapa entitas atau tabel bersama-sama, maka desain multitabel adalah pendekatan yang baik dan memadai.

Blok bangunan pemodelan data di DynamoDB

Bagian ini membahas lapisan blok bangunan yang dapat Anda gunakan dalam pola deasin untuk aplikasi Anda.



Topik

- Blok bangunan kunci urutan komposit
- Blok bangunan multi-penghunian
- Blok bangunan indeks jarang
- Blok bangunan waktu untuk tayang
- Blok bangunan pengarsipan waktu untuk tayang
- Blok bangunan partisi vertikal
- Blok bangunan sharding penulisan

Blok bangunan kunci urutan komposit

Bicara tentang NoSQL, orang-orang mungkin juga menganggapnya sebagai non-relasional. Pada akhirnya, tidak ada hal yang menghalangi relasi untuk ditempatkan ke dalam skema DynamoDB,

meski tampilannya berbeda dari basis data relasional dan kunci asingnya. Salah satu pola paling kritis yang dapat kita gunakan untuk mengembangkan hierarki logis data kita di DynamoDB adalah kunci urutan komposit. Gaya yang paling umum untuk merancangnya adalah dengan memisahkan setiap lapisan hierarki (layer induk > lapisan turunan > lapisan cucu) dengan tagar. Misalnya, `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

Sementara kunci partisi di DynamoDB selalu membutuhkan nilai yang tepat untuk kueri data, kita dapat menerapkan kondisi parsial ke kunci urutan dari kiri ke kanan, mirip dengan melintasi pohon biner.

Dalam contoh di atas, ada toko e-Commerce dengan Keranjang Belanja yang perlu dipertahankan di seluruh sesi pengguna. Setiap kali pengguna masuk, mereka mungkin ingin melihat seluruh Keranjang Belanja, termasuk item yang disimpan untuk nanti. Namun ketika mereka memasuki halaman checkout, hanya item di keranjang aktif yang harus dimuat untuk dibeli. Karena kedua `KeyConditions` ini secara eksplisit meminta kunci urutan CART, data daftar keinginan tambahan diabaikan begitu saja oleh DynamoDB pada waktu baca. Meskipun item yang disimpan dan aktif adalah bagian dari keranjang yang sama, kita perlu memperlakukannya secara berbeda di berbagai bagian aplikasi, jadi menerapkan `KeyCondition` ke prefiks kunci urutan adalah cara yang paling optimal untuk hanya mengambil data yang diperlukan untuk setiap bagian aplikasi.

Fitur utama dari blok bangunan ini

- Item terkait disimpan secara lokal satu sama lain untuk akses data yang efektif
- Menggunakan `KeyCondition` ekspresi, himpunan bagian dari hierarki dapat diambil secara selektif yang berarti tidak ada yang sia-sia RCUs

- Bagian yang berbeda dari aplikasi dapat menyimpan item-nya di bawah prefiks tertentu, yang mencegah penimpaan item atau penulisan yang bertentangan

Blok bangunan multi-penghunian

Banyak pelanggan menggunakan DynamoDB sebagai host data untuk aplikasi multi-penghuni mereka. Untuk skenario ini, kami ingin merancang skema dengan cara yang menyimpan semua data dari penghuni tunggal di partisi logisnya sendiri dari tabel. Hal ini memanfaatkan konsep Koleksi Item, yang merupakan istilah untuk semua item dalam tabel DynamoDB dengan kunci partisi yang sama. Untuk informasi selengkapnya tentang cara pendekatan DynamoDB terhadap multi-penghuni, lihat Multi-penghuni di 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]

Untuk contoh ini, kami menjalankan situs hosting foto dengan potensi ribuan pengguna. Setiap pengguna hanya akan mengunggah foto ke profil mereka sendiri pada awalnya, tetapi secara default kami tidak akan mengizinkan pengguna untuk melihat foto pengguna lain. Tingkat isolasi tambahan idealnya akan ditambahkan ke otorisasi panggilan setiap pengguna ke API Anda untuk memastikan mereka hanya meminta data dari partisi mereka sendiri, tetapi pada tingkat skema, kunci partisi unik sudah cukup.

Fitur utama dari blok bangunan ini

- Jumlah data yang dibaca oleh satu pengguna atau penghuni hanya bisa sebanyak jumlah total item di partisi mereka

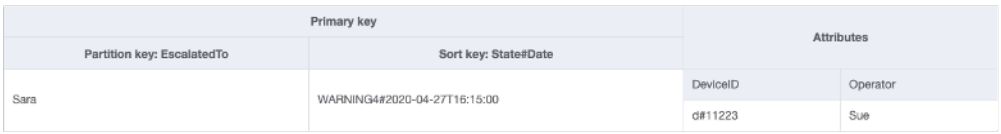
- Penghapusan data penghuni karena penutupan akun atau permintaan kepatuhan dapat dilakukan dengan taktis dan murah. Cukup jalankan kueri di mana kunci partisi sama dengan ID penyewanya, lalu jalankan operasi `DeleteItem` untuk setiap kunci primer yang dikembalikan

> **Note**
>
> Dirancang dengan mempertimbangkan multi-tenancy, Anda dapat menggunakan penyedia kunci enkripsi yang berbeda di satu tabel untuk mengisolasi data dengan aman. AWS SDK Enkripsi Basis Data untuk Amazon DynamoDB memungkinkan Anda menyertakan enkripsi di sisi klien dalam beban kerja DynamoDB Anda. Anda dapat melakukan enkripsi tingkat atribut, memungkinkan Anda mengenkripsi nilai atribut tertentu sebelum menyimpannya di tabel DynamoDB Anda dan mencari atribut terenkripsi tanpa mendekripsi seluruh basis data sebelumnya.

Blok bangunan indeks jarang

Terkadang pola akses memerlukan pencarian item yang cocok dengan item langka atau item yang menerima status (yang memerlukan respons yang dieskalasikan). Daripada melakukan kueri secara reguler di seluruh set data untuk item-item ini, kita dapat memanfaatkan fakta bahwa indeks sekunder global (GSI) jarang dimuat dengan data. Ini berarti bahwa hanya item dalam tabel dasar yang memiliki atribut yang ditentukan dalam indeks yang akan direplikasi ke indeks.

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

Dalam contoh ini, kita melihat kasus penggunaan IOT di mana setiap perangkat di lapangan melaporkan kembali status secara reguler. Untuk sebagian besar laporan, kita berharap perangkat melaporkan semuanya baik-baik saja, tetapi kadang-kadang kesalahan bisa saja muncul, dan hal itu harus dieskalasikan ke teknisi perbaikan. Untuk laporan dengan eskalasi, atribut `EscalatedTo` ditambahkan item, tetapi malah tidak muncul. GSI dalam contoh ini dipartisi oleh `EscalatedTo`, dan karena GSI membawa kunci dari tabel dasar, kita masih dapat melihat DeviceID yang melaporkan kesalahan dan pada pukul berapa.

Meskipun pembacaan lebih murah daripada penulisan di DynamoDB, indeks jarang adalah alat yang sangat efektif untuk kasus penggunaan di mana instans jenis item tertentu jarang terjadi tetapi pembacaan untuk menemukannya merupakan hal biasa.

Fitur utama dari blok bangunan ini

- Biaya tulis dan penyimpanan untuk GSI jarang hanya berlaku untuk item yang cocok dengan pola kunci, sehingga biaya GSI bisa jauh lebih rendah daripada yang lain yang memiliki semua item GSIs yang direplikasi ke mereka
- Kunci urutan komposit masih dapat digunakan untuk menyaring item yang cocok dengan kueri yang diinginkan, misalnya, stempel waktu dapat digunakan untuk kunci urutan agar hanya melihat kesalahan yang dilaporkan dalam X menit terakhir (`SK > 5 minutes ago, ScanIndexForward: False`)

Blok bangunan waktu untuk tayang

Sebagian besar data memiliki beberapa durasi waktu yang dapat dianggap layak disimpan dalam penyimpanan data primer. Untuk memfasilitasi penuaan data dari DynamoDB, ada fitur yang disebut waktu untuk tayang (TTL). Fitur TTL memungkinkan Anda untuk menentukan atribut tertentu pada tingkat tabel yang memerlukan pemantauan untuk item dengan stempel waktu epoch (itu di masa lalu). Hal ini memungkinkan Anda untuk menghapus catatan kedaluwarsa dari tabel secara gratis.

> **Note**
>
> Jika Anda menggunakan Global Tables versi 2019.11.21 (Saat ini) dari tabel global dan Anda juga menggunakan fitur Time to Live, DynamoDB mereplikasi penghapusan TTL ke

semua tabel replika. Penghapusan TTL awal tidak mengonsumsi kapasitas tulis di wilayah tempat TTL kedaluwarsa. Namun, penghapusan TTL yang direplikasi untuk tabel replika mengonsumsi unit kapasitas tulis yang direplikasi ketika menggunakan kapasitas di setiap Wilayah replika dan biaya akan berlaku.

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

Dalam contoh ini, kami memiliki aplikasi yang dirancang untuk memungkinkan pengguna membuat pesan berumur pendek. Ketika pesan dibuat di DynamoDB, atribut TTL diatur ke tanggal tujuh hari di masa depan oleh kode aplikasi. Dalam kira-kira tujuh hari, DynamoDB akan melihat bahwa stempel jangka waktu item ini sudah ada di masa lalu kemudian menghapusnya.

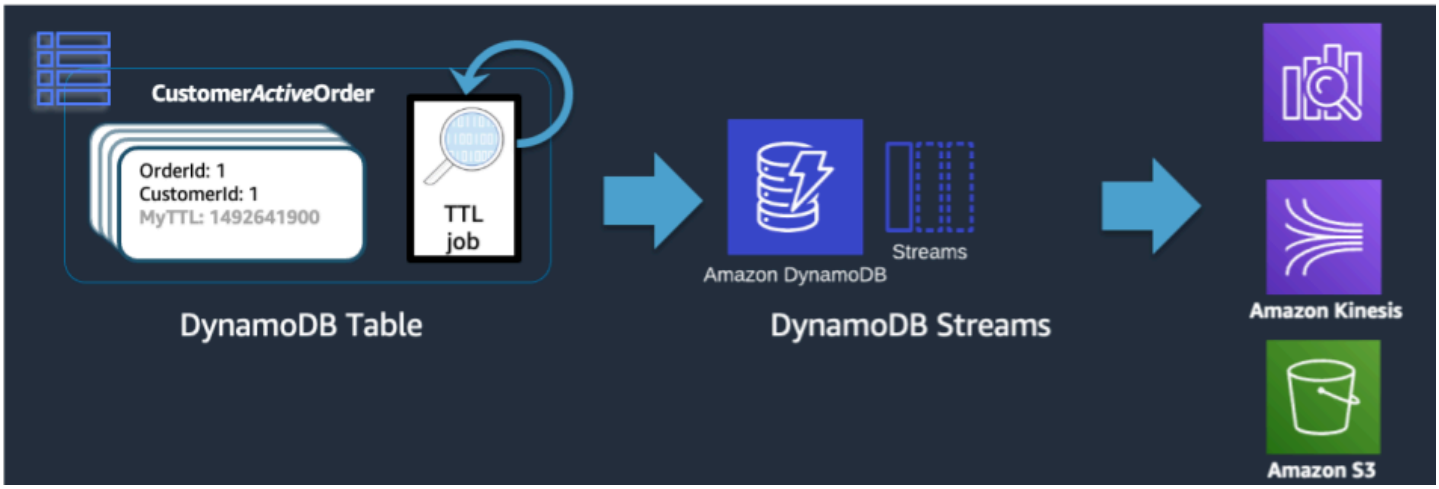
Karena penghapusan yang dilakukan oleh TTL gratis, sangat disarankan untuk menggunakan fitur ini guna menghapus data historis dari tabel. Ini akan mengurangi tagihan penyimpanan keseluruhan setiap bulan dan kemungkinan akan mengurangi biaya pembacaan pengguna karena data yang akan diambil oleh kueri mereka menjadi lebih sedikit. Meskipun TTL diaktifkan pada tingkat tabel, Anda bebas menentukan item atau entitas mana yang akan membuat atribut TTL dan seberapa jauh stempel jangka waktu diatur ke depannya.

Fitur utama dari blok bangunan ini

- Penghapusan TTL dijalankan di belakang layar tanpa berdampak pada performa tabel Anda
- TTL adalah proses asinkron yang berjalan kira-kira setiap enam jam, tetapi penghapusan catatan kedaluwarsa dapat memakan waktu lebih dari 48 jam
  - Jangan mengandalkan penghapusan TTL untuk kasus penggunaan seperti catatan kunci atau manajemen status jika data basi harus dibersihkan dalam waktu kurang dari 48 jam
- Anda dapat memberi atribut TTL dengan nama atribut yang valid, tetapi nilainya harus berupa jenis angka

Blok bangunan pengarsipan waktu untuk tayang

Meski TTL adalah alat yang efektif untuk menghapus data lama dari DynamoDB, banyak kasus penggunaan mewajibkan arsip data disimpan untuk jangka waktu yang lebih lama dibandingkan penyimpanan data primer. Dalam instans, kita dapat memanfaatkan penghapusan catatan berwaktu TTL untuk mendorong catatan kedaluwarsa ke dalam penyimpanan data jangka panjang.



Ketika penghapusan TTL dilakukan oleh DynamoDB, TTL masih didorong ke Aliran DynamoDB sebagai peristiwa `Delete`. Ketika penghapusan dilakukan oleh TTL DynamoDB, ada atribut pada catatan aliran `principal:dynamodb`. Menggunakan pelanggan Lambda ke Aliran DynamoDB, kita dapat menerapkan filter peristiwa hanya untuk atribut pengguna utama DynamoDB dan mengetahui bahwa catatan apa pun yang cocok dengan filter tersebut akan didorong ke penyimpanan arsip seperti S3 Glacier.

Fitur utama dari blok bangunan ini

- Setelah pembacaan latensi rendah DynamoDB tidak lagi diperlukan untuk item historis, memigrasikannya ke layanan penyimpanan yang lebih dingin, seperti S3 Glacier, dapat mengurangi biaya penyimpanan secara signifikan sekaligus memenuhi kebutuhan kepatuhan data dari kasus penggunaan Anda
- Jika data disimpan di Amazon S3, alat analitik hemat biaya seperti Amazon Athena atau Redshift Spectrum dapat digunakan untuk melakukan analisis historis data

Blok bangunan partisi vertikal

Pengguna yang sudah familier dengan basis data model dokumen juga akan familier dengan gagasan menyimpan semua data terkait dalam satu dokumen JSON. Sementara DynamoDB

mendukung tipe data JSON, itu tidak mendukung `KeyConditions` eksekusi pada JSON bersarang. Karena itulah `KeyConditions` yang menentukan berapa banyak data yang dibaca dari disk dan secara efektif berapa banyak kueri RCUs yang dikonsumsi, ini dapat mengakibatkan inefisiensi pada skala. Untuk mengoptimalkan penulisan dan pembacaan DynamoDB dengan lebih baik, kami sarankan untuk memecah entitas individual dokumen menjadi item DynamoDB individual, yang juga disebut sebagai partisi vertikal.

```
{
        ”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

Partisi vertikal, seperti yang ditunjukkan di atas, adalah contoh kunci desain tabel tunggal dalam kenyataannya, tetapi juga dapat diimplementasikan di beberapa tabel jika diinginkan. Karena tagihan DynamoDB menulis dalam kelipatan 1 KB, Anda idealnya harus mempartisi dokumen dengan cara yang menghasilkan item di bawah 1 KB.

Fitur utama dari blok bangunan ini

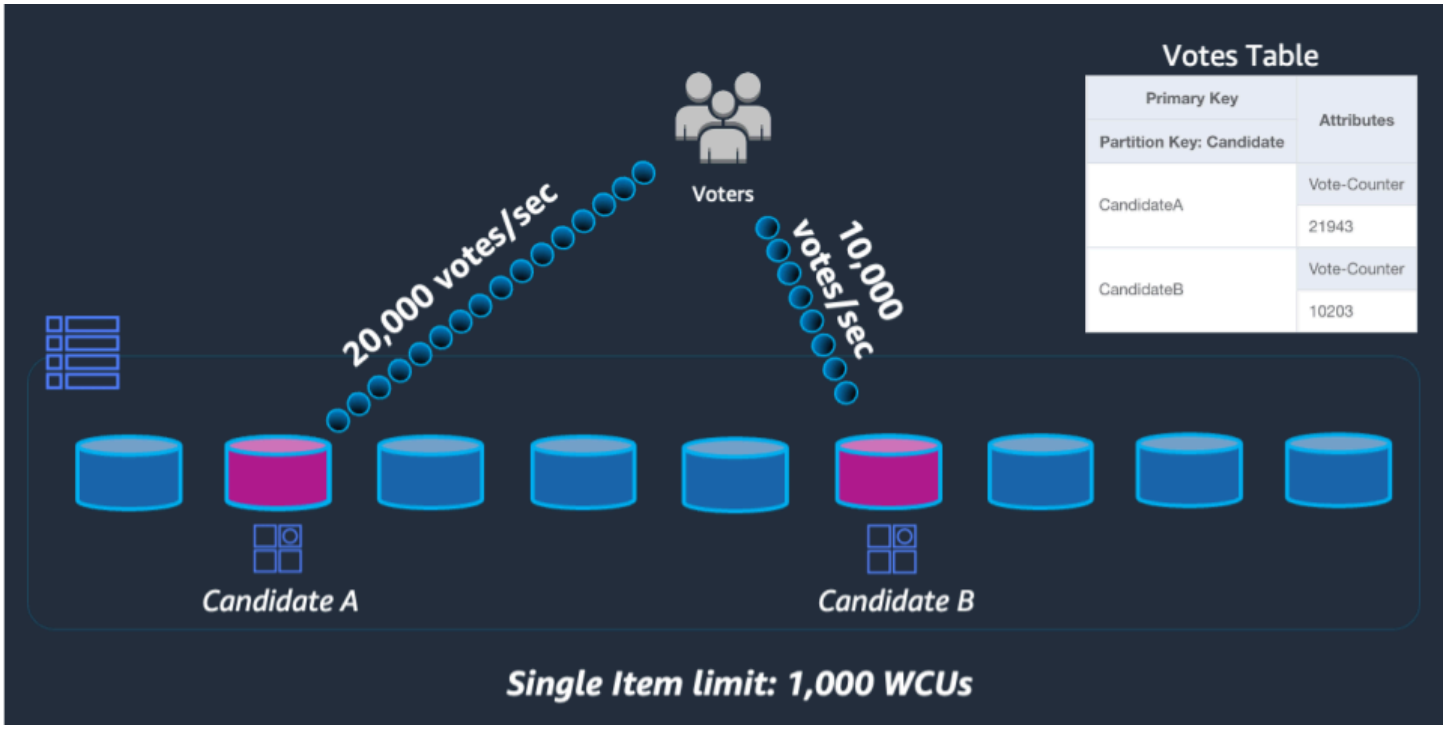
- Hierarki hubungan data dipertahankan melalui prefiks kunci urutan agar struktur dokumen tunggal dapat dibangun kembali sisi klien jika diperlukan
- Komponen tunggal dari struktur data dapat diperbarui secara independen, sehingga item kecil dapat diperbarui meski hanya 1 WCU
- Dengan menggunakan kunci urutan `BeginsWith`, aplikasi dapat mengambil data serupa dalam satu kueri, yang menggabungkan biaya baca untuk mengurangi total biaya/latensi
- Dokumen besar bisa saja lebih besar dari batas ukuran item individu 400 KB di DynamoDB dan partisi vertikal membantu mengatasi batas ini

Blok bangunan sharding penulisan

Salah satu dari sedikit batasan keras yang dimiliki DynamoDB adalah pembatasan jumlah throughput yang dapat dipertahankan oleh satu partisi fisik per detik (tidak harus satu kunci partisi). Batasan ini untuk saat ini:

- 1000 WCU (atau 1000 <= 1 KB item ditulis per detik) dan 3000 RCU (atau 3000 <= 4 KB dibaca per detik) sangat konsisten atau
- 6000 <=4 KB dibaca per detik pada akhirnya konsisten

Jika permintaan terhadap tabel melebihi salah satu dari batas ini, kesalahan dikirim kembali ke SDK klien `ThroughputExceededException`, yang umumnya disebut sebagai throttling. Kasus penggunaan yang memerlukan operasi baca di luar batas itu sebagian besar akan dilayani paling baik dengan menempatkan cache baca di depan DynamoDB, tetapi operasi penulisan memerlukan desain tingkat skema yang dikenal sebagai sharding penulisan.



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

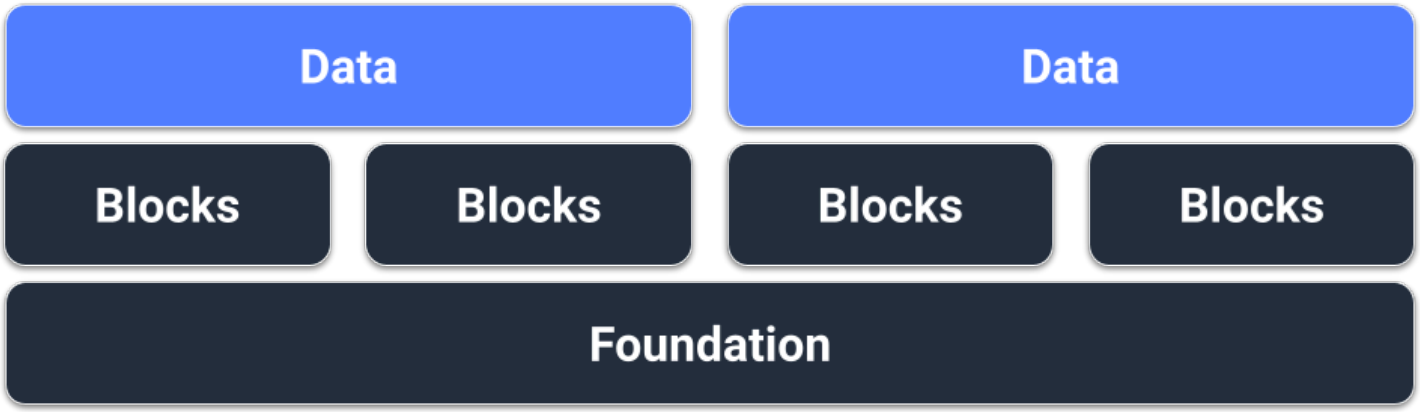
Untuk mengatasi masalah ini, kita akan menambahkan bilangan bulat acak ke akhir kunci partisi untuk setiap kontestan dalam kode `UpdateItem` aplikasi. Rentang generator bilangan bulat acak harus memiliki pencocokan batas atas atau melebihi jumlah penulisan per detik yang diharapkan untuk kontestan tertentu dibagi 1000. Untuk mendukung 20.000 suara per detik, hal ini akan terlihat seperti rand(0,19). Setelah data disimpan di bawah partisi logis yang terpisah, data harus digabungkan bersama kembali pada waktu baca. Karena total suara tidak perlu real time, fungsi Lambda yang dijadwalkan untuk membaca semua partisi suara setiap X menit dapat melakukan agregasi sesekali untuk setiap kontestan dan menuliskannya kembali ke catatan total suara tunggal untuk pembacaan langsung.

Fitur utama dari blok bangunan ini

- Untuk kasus penggunaan dengan throughput tulis yang sangat tinggi untuk kunci partisi tertentu yang tidak dapat dihindari, operasi tulis dapat tersebar secara artifisial di beberapa partisi DynamoDB
- GSIs dengan kunci partisi kardinalitas rendah juga harus menggunakan pola ini karena pelambatan pada GSI akan menerapkan tekanan balik untuk menulis operasi pada tabel dasar

Paket desain skema pemodelan data di DynamoDB

Pelajari tentang paket desain skema pemodelan data untuk DynamoDB, termasuk kasus penggunaan, pola akses, dan desain skema akhir untuk jejaring sosial, profil game, manajemen keluhan, pembayaran berulang, status perangkat, dan toko online.



Prasyarat

Sebelum mencoba merancang skema untuk DynamoDB, kita harus terlebih dahulu mengumpulkan beberapa data prasyarat tentang kasus penggunaan yang perlu didukung oleh skema. Tidak seperti basis data relasional, DynamoDB dipecah secara default, artinya data akan berada di beberapa server di belakang layar sehingga desain lokalitas data menjadi hal yang penting. Kita harus menyusun daftar berikut untuk setiap desain skema:

- Daftar entitas (Diagram ER)
- Estimasi volume dan throughput untuk setiap entitas
- Pola akses yang perlu didukung (kueri dan penulisan)
- Persyaratan retensi data

Topik

- Desain skema jejaring sosial di DynamoDB
- Desain skema profil game di DynamoDB
- Desain skema sistem manajemen pengaduan di DynamoDB
- Desain skema pembayaran berulang di DynamoDB
- Memantau pembaruan status perangkat di DynamoDB
- Menggunakan DynamoDB sebagai penyimpanan data untuk toko online

Desain skema jejaring sosial di DynamoDB

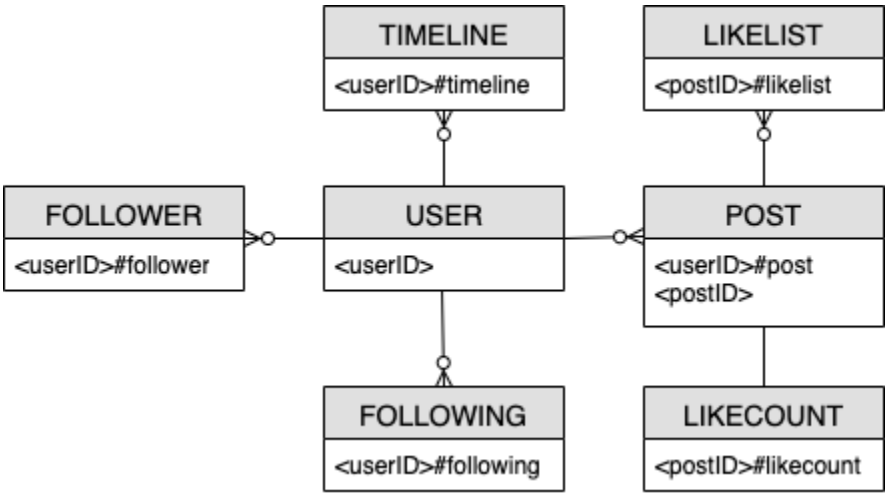
Kasus penggunaan bisnis jejaring sosial

Kasus penggunaan ini membahas penggunaan DynamoDB sebagai jejaring sosial. Jejaring sosial adalah layanan online yang memungkinkan berbagai pengguna berinteraksi satu sama lain. Jejaring sosial yang akan kita desain akan memungkinkan pengguna melihat garis waktu yang terdiri dari postingan mereka, pengikut mereka, orang yang mereka ikuti, dan postingan yang ditulis oleh orang yang mereka ikuti. Pola akses untuk desain skema ini adalah:

- Mendapatkan informasi pengguna untuk userID tertentu
- Mendapatkan daftar pengikut untuk userID tertentu
- Mendapatkan daftar orang yang diikuti untuk userID tertentu
- Mendapatkan daftar postingan untuk userID tertentu
- Mendapatkan daftar pengguna yang menyukai postingan untuk postID tertentu
- Mendapatkan jumlah suka untuk postID tertentu
- Mendapatkan lini masa untuk userID tertentu

Diagram hubungan entitas jejaring sosial

Ini adalah diagram hubungan entitas (ERD) yang akan kita gunakan untuk desain skema jejaring sosial.



Pola akses jejaring sosial

Ini adalah pola akses yang akan kita pertimbangkan untuk desain skema jaringan sosial.

- `getUserInfoByUserID`
- `getFollowerListByUserID`
- `getFollowingListByUserID`
- `getPostListByUserID`
- `getUserLikesByPostID`
- `getLikeCountByPostID`
- `getTimelineByUserID`

Evolusi desain skema jejaring sosial

DynamoDB adalah basis data NoSQL, sehingga Anda tidak dapat melakukan penggabungan, yaitu operasi yang menggabungkan data dari beberapa basis data. Pelanggan yang tidak terbiasa dengan DynamoDB mungkin menerapkan filosofi desain sistem manajemen basis data relasional (RDBMS) (seperti membuat tabel untuk setiap entitas) ke DynamoDB ketika mereka tidak perlu melakukannya. Tujuan desain tabel tunggal DynamoDB adalah untuk menulis data dalam bentuk pra-penggabungan sesuai pola akses aplikasi, lalu segera menggunakan data tanpa komputasi tambahan. Untuk informasi selengkapnya, lihat Single-table vs. multi-table design in DynamoDB.

Sekarang, mari melanjutkan ke cara mengembangkan desain skema untuk menangani semua pola akses.

Langkah 1: Tangani pola akses 1 (**`getUserInfoByUserID`**)

Untuk mendapatkan informasi pengguna tertentu, kita perlu `Query` tabel dasar dengan kondisi kunci `PK=<userID>`. Operasi kueri memungkinkan Anda melakukan paginasi hasil, yang berguna ketika pengguna memiliki banyak pengikut. Untuk informasi selengkapnya tentang Kueri, lihat Menanyakan tabel di DynamoDB.

Dalam contoh, kita melacak dua jenis data untuk pengguna: yaitu "count" dan "info". "Count" pengguna mencerminkan berapa banyak pengikut yang dimiliki, berapa banyak pengguna yang diikuti, dan berapa banyak postingan yang telah dibuat. "Info" pengguna mencerminkan informasi pribadi seperti nama mereka.

Kita melihat dua jenis data yang diwakili oleh dua item di bawah ini. Item dengan "count" dalam kunci urutan (SK) lebih mungkin berubah daripada item dengan "info". DynamoDB menganggap ukuran item seperti yang muncul sebelum dan setelah pembaruan dan throughput yang disediakan akan mencerminkan ukuran item yang lebih besar. Meskipun Anda hanya memperbarui subset atribut item,

`UpdateItem` masih akan menggunakan throughput yang disediakan dalam jumlah penuh (lebih besar dari ukuran item sebelum dan sesudah). Anda bisa mendapatkan item melalui satu operasi `Query` dan menggunakan `UpdateItem` untuk menambah atau mengurangi dari atribut numerik yang ada.

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://....	...

Langkah 2: Tangani pola akses 2 (**`getFollowerListByUserID`**)

Untuk mendapatkan daftar pengguna yang mengikuti pengguna tertentu, kita perlu melakukan `Query` tabel dasar dengan kondisi kunci `PK=<userID>#follower`.

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				

Langkah 3: Tangani pola akses 3 (**`getFollowingListByUserID`**)

Untuk mendapatkan daftar pengguna yang diikuti pengguna tertentu, kita perlu melakukan `Query` tabel dasar dengan kondisi kunci `PK=<userID>#following`. Anda kemudian dapat menggunakan `TransactWriteItems`operasi untuk mengelompokkan beberapa permintaan bersama-sama dan melakukan hal berikut:

- Tambahkan Pengguna A ke daftar pengikut Pengguna B, lalu tambahkan jumlah pengikut Pengguna B satu per satu.
- Tambahkan Pengguna B ke daftar pengikut Pengguna A, lalu tambahkan jumlah pengikut Pengguna A satu per satu.

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				

Langkah 4: Tangani pola akses 4 (**`getPostListByUserID`**)

Untuk mendapatkan daftar postingan yang dibuat pengguna tertentu, kita perlu melakukan `Query` tabel dasar dengan kondisi kunci `PK=<userID>#post`. Satu hal penting yang perlu diperhatikan di sini adalah bahwa posting pengguna IDs harus inkremental: nilai PostID kedua harus lebih besar dari nilai PostID pertama (karena pengguna ingin melihat posting mereka dengan cara yang diurutkan). Anda dapat melakukan ini dengan membuat posting IDs berdasarkan nilai waktu seperti Universally Unique Lexicographically Sortable Identifier (ULID).

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	

Langkah 5: Tangani pola akses 5 (**`getUserLikesByPostID`**)

Untuk mendapatkan daftar pengguna yang menyukai postingan pengguna tertentu, kita perlu melakukan `Query` tabel dasar dengan kondisi kunci `PK=<postID>#likelist`. Pendekatan ini sama dengan pola yang kita gunakan untuk mengambil daftar pengikut dan pengguna yang diikuti dalam pola akses 2 (`getFollowerListByUserID`) dan pola akses 3 (`getFollowingListByUserID`).

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				

Langkah 6: Tangani pola akses 6 (**`getLikeCountByPostID`**)

Untuk mendapatkan jumlah suka dalam posting tertentu, kita harus melakukan `GetItem` operasi pada tabel dasar dengan kondisi kunci `PK=<postID>#likecount`. Pola akses ini dapat menyebabkan masalah throttling setiap kali pengguna dengan banyak pengikut (seperti selebritas) membuat postingan karena throttling terjadi ketika throughput partisi melebihi 1000 WCU per detik. Masalah ini tidak dihasilkan oleh DynamoDB, tetapi hanya muncul di DynamoDB karena berada di akhir tumpukan perangkat lunak.

Anda harus mengevaluasi apakah penting bahwa semua pengguna melihat jumlah suka secara bersamaan atau hal tersebut dapat dilakukan secara bertahap dari waktu ke waktu. Secara umum, jumlah suka postingan tidak harus langsung 100% akurat. Anda dapat menerapkan strategi ini dengan menempatkan antrean antara aplikasi Anda dan DynamoDB agar pembaruan terjadi secara berkala.

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			

Langkah 7: Tangani pola akses 7 (**`getTimelineByUserID`**)

Untuk mendapatkan lini masa pengguna tertentu, kita harus melakukan `Query` operasi pada tabel dasar dengan kondisi kunci `PK=<userID>#timeline`. Mari pertimbangkan skenario ketika pengikut pengguna perlu melihat postingan secara sinkron. Setiap kali pengguna menulis postingan, daftar pengikut mereka dibaca dan userID serta postID mereka perlahan dimasukkan ke dalam kunci lini masa semua pengikutnya. Kemudian, ketika aplikasi Anda dimulai, Anda dapat membaca kunci lini masa dengan operasi `Query` dan mengisi layar lini masa dengan kombinasi userID dan postID menggunakan operasi `BatchGetItem` untuk semua item baru. Anda tidak dapat membaca lini masa dengan panggilan API, tetapi ini adalah solusi yang lebih hemat biaya jika postingan sering diedit.

Lini masa adalah tempat yang menampilkan postingan terbaru, jadi kita perlu cara untuk membersihkan yang lama. Dibandingkan menggunakan WCU untuk menghapusnya, Anda dapat menggunakan fitur TTL DynamoDB untuk melakukannya secara gratis.

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			

Semua pola akses dan bagaimana desain skema mengatasinya dirangkum dalam tabel di bawah ini:

Pola akses	Basis table/ GSI/LSI	Operasi	Nilai kunci partisi	Nilai kunci urutan	Kondisi/filter lainnya
getUserInfoByUserID	Tabel dasar	Kueri	PK=<userID>		
getFollowerListByUserID	Tabel dasar	Kueri	PK=<userID>#follower		

Pola akses	Basis table/ GSI/LSI	Operasi	Nilai kunci partisi	Nilai kunci urutan	Kondisi/filter lainnya
getFollowingListByUserID	Tabel dasar	Kueri	PK=<userID>#following		
getPostListByUserID	Tabel dasar	Kueri	PK=<userID>#post		
getUserLikesByPostID	Tabel dasar	Kueri	PK=<postID>#likelist		
getLikeCountByPostID	Tabel dasar	GetItem	PK=<postID>#likecount		
getTimelineByuserID	Tabel dasar	Kueri	PK=<userID>#timeline		

Skema akhir jejaring sosial

Berikut adalah desain skema akhir. Untuk mengunduh desain skema ini sebagai file JSON, lihat Contoh DynamoDB di. GitHub

Tabel dasar:

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			

Menggunakan NoSQL Workbench dengan desain skema ini

Anda dapat mengimpor skema akhir ini ke NoSQL Workbench, sebuah alat visual yang menyediakan fitur pemodelan data, visualisasi data, dan pengembangan kueri untuk DynamoDB, guna mengeksplorasi dan mengedit proyek baru Anda lebih lanjut. Ikuti langkah-langkah berikut untuk memulai:

1. Unduh NoSQL Workbench. Untuk informasi selengkapnya, lihat the section called “Unduh”.
2. Unduh file skema JSON yang tercantum di atas, yang sudah dalam format model NoSQL Workbench.

3. Impor file skema JSON ke NoSQL Workbench. Untuk informasi selengkapnya, lihat the section called “Mengimpor model yang ada”.
4. Setelah mengimpor model data ke NoSQL Workbench, Anda dapat mengeditnya. Untuk informasi selengkapnya, lihat the section called “Mengedit model yang ada”.
5. Untuk memvisualisasikan model data Anda, menambahkan data sampel, atau mengimpor data sampel dari file CSV, gunakan fitur Data Visualizer di NoSQL Workbench.

Desain skema profil game di DynamoDB

Kasus penggunaan bisnis profil game

Kasus penggunaan ini berbicara tentang penggunaan DynamoDB untuk menyimpan profil pemain untuk sistem game. Pengguna (dalam hal ini, pemain) perlu membuat profil sebelum mereka dapat berinteraksi dengan banyak game modern, terutama yang online. Profil game biasanya mencakup hal berikut:

- Informasi dasar seperti nama pengguna
- Data game seperti item dan peralatan
- Catatan game seperti tugas dan aktivitas
- Informasi sosial seperti daftar teman

Untuk memenuhi persyaratan akses kueri data ketat untuk aplikasi ini, kunci primer (kunci partisi dan kunci urutan) akan menggunakan nama generik (PK dan SK) agar dapat dibebani dengan berbagai jenis nilai seperti yang akan kita lihat di bawah ini.

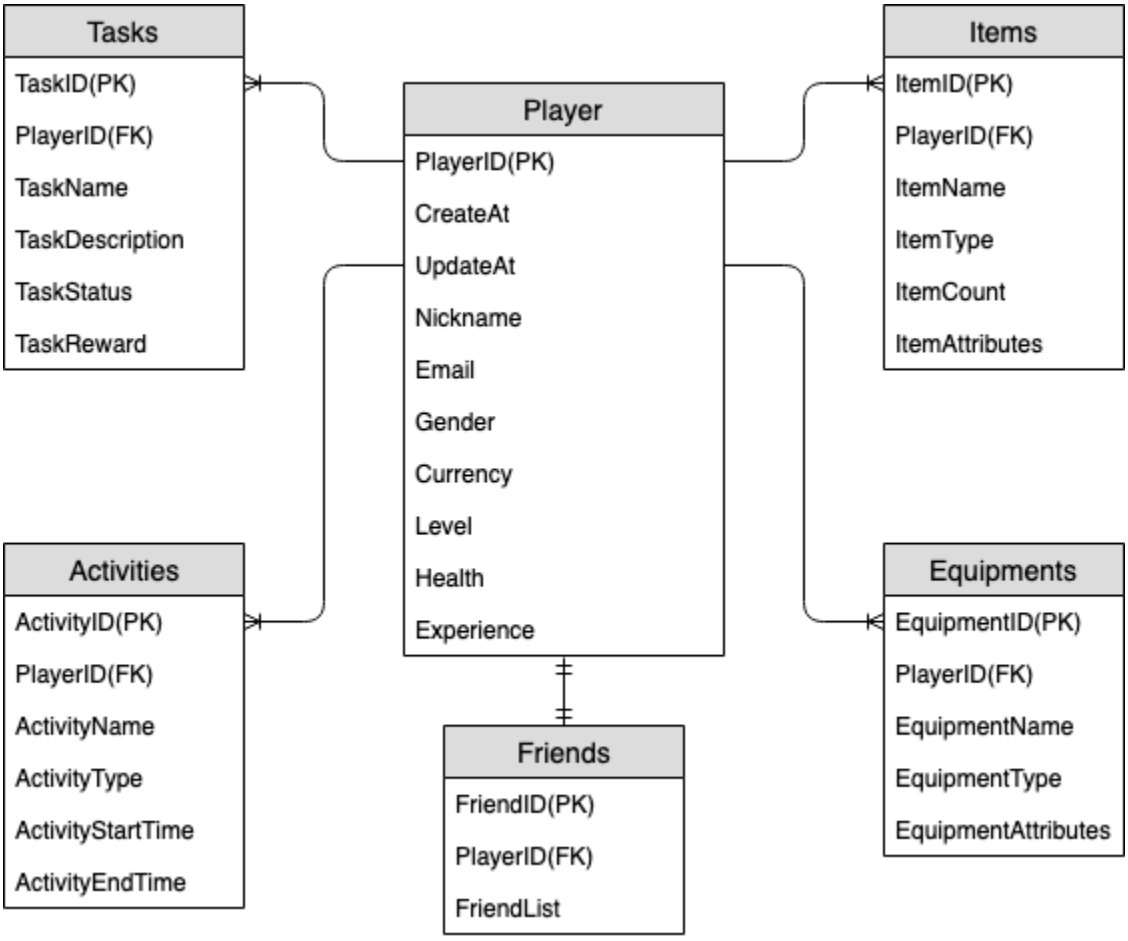
Pola akses untuk desain skema ini adalah:

- Mendapatkan daftar teman pengguna
- Mendapatkan semua informasi pemain
- Mendapatkan daftar item pengguna
- Mendapatkan item tertentu dari daftar item pengguna
- Memperbarui karakter pengguna
- Memperbarui jumlah item untuk pengguna

Ukuran profil game akan bervariasi di game yang berbeda. Mengompresi nilai atribut yang besar dapat membuatnya sesuai dengan batas item di DynamoDB dan mengurangi biaya. Strategi manajemen throughput akan bergantung pada berbagai faktor, seperti: jumlah pemain, jumlah game yang dimainkan per detik, dan beban kerja musiman. Biasanya untuk game yang baru diluncurkan, jumlah pemain dan tingkat popularitasnya tidak diketahui, jadi kita akan mulai dengan mode throughput sesuai permintaan.

Diagram hubungan entitas profil game

Ini adalah diagram hubungan entitas (ERD) yang akan kita gunakan untuk desain skema sistem profil game.



Pola akses profil game

Ini adalah pola akses yang akan kita pertimbangkan untuk desain skema jaringan sosial.

- `getPlayerFriends`
- `getPlayerAllProfile`

- `getPlayerAllItems`
- `getPlayerSpecificItem`
- `updateCharacterAttributes`
- `updateItemCount`

Evolusi desain skema profil game

Dari ERD di atas, kita dapat melihat bahwa ini adalah tipe one-to-many hubungan pemodelan data. Dalam DynamoDB one-to-many, model data dapat diatur ke dalam koleksi item, yang berbeda dari database relasional tradisional di mana beberapa tabel dibuat dan ditautkan melalui kunci asing. Koleksi item adalah sekelompok item yang berbagi nilai kunci partisi yang sama, tetapi memiliki nilai kunci urutan yang berbeda. Dalam koleksi item, setiap item memiliki nilai kunci urutan unik yang membedakannya dari item lainnya. Dengan pertimbangan ini, mari kita gunakan pola berikut untuk nilai `HASH` dan `RANGE` untuk setiap jenis entitas.

Untuk memulai, kita menggunakan nama generik seperti `PK` dan `SK` untuk menyimpan berbagai jenis entitas dalam tabel yang sama agar model bisa tetap digunakan hingga ke depannya. Untuk keterbacaan yang lebih baik, kita dapat menyertakan prefiks untuk menunjukkan jenis data atau menyertakan atribut arbitrer yang disebut `Entity_type` atau `Type`. Dalam contoh saat ini, kita menggunakan string yang dimulai dengan `player` untuk menyimpan `player_ID` sebagai `PK`; gunakan `entity name#` sebagai prefiks `SK`, dan tambahkan atribut `Type` untuk menunjukkan jenis entitas bagian data ini. Hal ini memungkinkan kami untuk mendukung penyimpanan lebih banyak jenis entitas ke depannya, dan menggunakan teknologi canggih seperti GSI Overloading dan Sparse GSI untuk memenuhi lebih banyak pola akses.

Mari kita mulai menerapkan pola akses. Pola akses seperti menambahkan pemain dan menambahkan peralatan dapat diwujudkan melalui operasi `PutItem`, jadi kita dapat mengabaikannya. Dalam dokumen ini, kita akan fokus pada pola akses tipikal yang tercantum di atas.

Langkah 1: Atasi pola akses 1 (**`getPlayerFriends`**)

Kita mengatasi pola akses 1 (`getPlayerFriends`) dengan langkah ini. Dalam desain kita saat ini, pertemanan bersifat sederhana dan jumlah teman dalam game ini kecil. Agar simpel, kita menggunakan jenis data daftar untuk menyimpan daftar teman (pemodelan 1:1). Dalam desain ini, kita menggunakan `GetItem` untuk memenuhi pola akses ini. Dalam operasi `GetItem`, kita secara eksplisit memberikan kunci partisi dan nilai kunci urutan untuk mendapatkan item tertentu.

Namun, jika sebuah game memiliki banyak teman, dan hubungan di antara mereka rumit (seperti pertemanan menjadi dua arah dengan komponen undangan dan penerimaan), perlu menggunakan many-to-many hubungan untuk menyimpan setiap teman secara individual, untuk skala ke ukuran daftar teman yang tidak terbatas. Dan jika perubahan pertemanan melibatkan operasi pada beberapa item pada saat yang sama, transaksi DynamoDB dapat digunakan untuk mengelompokkan beberapa tindakan bersama-sama dan mengirimkannya sebagai satu atau all-or-nothing `TransactWriteItems`operasi. `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"}}}]

Langkah 2: Tangani pola akses 2 (**`getPlayerAllProfile`**), 3 (**`getPlayerAllItems`**), dan 4 (**`getPlayerSpecificItem`**)

Kita menangani pola akses 2 (`getPlayerAllProfile`), 3 (`getPlayerAllItems`), dan 4 (`getPlayerSpecificItem`) menggunakan langkah ini. Kesamaan ketiga pola akses ini adalah kueri rentang, yang menggunakan operasi `Query`. Bergantung pada ruang lingkup kueri, Kondisi Kunci dan Ekspresi Filter digunakan, yang umum digunakan dalam pengembangan praktis.

Dalam operasi Kueri, kita memberikan nilai tunggal untuk Kunci Partisi dan mendapatkan semua item dengan nilai Kunci Partisi tersebut. Pola akses 2 (`getPlayerAllProfile`) diimplementasikan dengan cara ini. Secara opsional, kita dapat menambahkan ekspresi kondisi kunci urutan — sebuah string yang menentukan item yang akan dibaca dari tabel. Pola akses 3 (`getPlayerAllItems`) diimplementasikan dengan menambahkan kondisi kunci dari kunci urutan begins_with `ITEMS#`. Selanjutnya, untuk menyederhanakan pengembangan sisi aplikasi, kita dapat menggunakan ekspresi filter untuk mengimplementasikan pola akses 4 (`getPlayerSpecificItem`).

Berikut adalah contoh kode pseudo menggunakan ekspresi filter yang memfilter item dari kategori `Weapon`:

```
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"}}}
	ITEMS#003	Type	ItemName	ItemType	ItemCount	ItemAttributes
		Item	Sword of the Dragon	Weapon	1	{"M":{"ATK":{"N":"100"},"DEF":{"N":"50"}}}

> **Note**
>
> Filter ekspresi diterapkan setelah Kueri selesai, namun sebelum hasilnya dikembalikan ke klien. Oleh karena itu, Kueri menggunakan jumlah kapasitas baca yang sama, terlepas dari ada atau tidak adanya ekspresi filter.

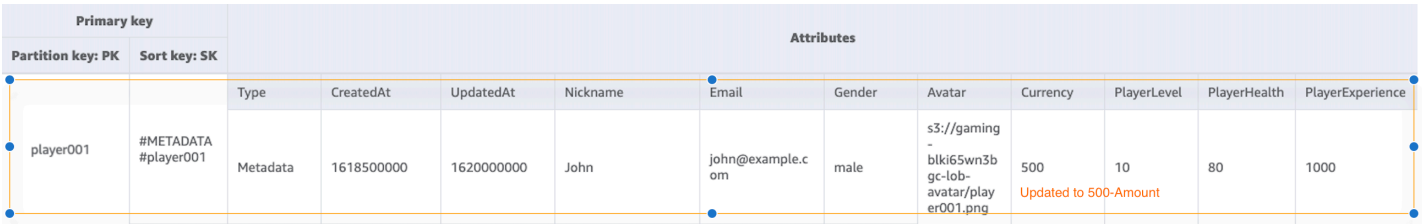
Jika pola aksesnya adalah melakukan kueri pada set data besar dan menyaring sejumlah besar data untuk menyimpan sebagian kecil data saja, pendekatan yang tepat adalah merancang Kunci Partisi dan Kunci Urutan DynamoDB secara lebih efektif. Misalnya, dalam contoh di atas untuk mendapatkan `ItemType` tertentu, jika ada banyak item untuk setiap pemain dan melakukan kueri untuk `ItemType` tertentu adalah pola akses yang tipikal, membawa `ItemType` ke `SK` sebagai kunci komposit merupakan hal yang lebih efisien. Model data akan terlihat seperti ini: `ITEMS#ItemType#ItemId`.

Langkah 3: Atasi pola akses 5 (**`updateCharacterAttributes`**) dan 6 (**`updateItemCount`**)

Kita menangani pola akses 5 (`updateCharacterAttributes`), dan 6 (`updateItemCount`) menggunakan langkah ini. Ketika pemain perlu memodifikasi karakter, seperti mengurangi mata uang, atau memodifikasi jumlah senjata tertentu dalam item mereka, gunakan `UpdateItem` untuk menerapkan pola akses ini. Untuk memperbarui mata uang pemain tetapi juga memastikannya tidak pernah berada di bawah jumlah minimum, kita dapat menambahkan the section called “Contoh CLI” untuk mengurangi saldo hanya jika saldo lebih besar dari atau sama dengan jumlah minimum. Berikut adalah contoh kode pseudo:

```
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

Saat melakukan pengembangan dengan DynamoDB dan menggunakan Penghitung Atom untuk mengurangi inventaris, kita dapat memastikan idempotensi menggunakan penguncian optimis. Berikut adalah contoh kode pseudo untuk Penghitung Atom:

```
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"}}}

Selain itu, dalam skenario di mana pemain membeli item dengan mata uang, seluruh proses perlu mengurangi mata uang dan menambahkan item pada saat yang sama. Kita dapat menggunakan Transaksi DynamoDB untuk mengelompokkan beberapa tindakan bersama-sama dan mengirimkannya sebagai satu atau all-or-nothing `TransactWriteItems` operasi. `TransactGetItems` `TransactWriteItems`adalah operasi penulisan sinkron dan idempoten yang mengelompokkan hingga 100 tindakan penulisan dalam satu operasi. all-or-nothing Tindakan tersebut diselesaikan secara atomik, sehingga semuanya berhasil atau tidak satu pun yang berhasil. Transaksi membantu menghilangkan risiko duplikasi atau hilangnya mata uang. Untuk informasi selengkapnya tentang transaksi, lihat Contoh DynamoDB transactions.

Semua pola akses dan bagaimana desain skema mengatasinya dirangkum dalam tabel di bawah ini:

Pola akses	Basis table/ GSI/LSI	Operasi	Nilai kunci partisi	Nilai kunci urutan	Kondisi/filter lainnya
getPlayer Friends	Tabel dasar	GetItem	PK=PlayerID	SK=“FRIENDS#playerID”	

Pola akses	Basis table/ GSI/LSI	Operasi	Nilai kunci partisi	Nilai kunci urutan	Kondisi/filter lainnya
getPlayer AllProfil	Tabel dasar	Kueri	PK=PlayerID		
getPlayer AllBarang	Tabel dasar	Kueri	PK=PlayerID	SK begins_with “ITEMS#”	
getPlayer SpecificBarang	Tabel dasar	Kueri	PK=PlayerID	SK begins_with “ITEMS#”	filterExpression: "ItemType=:itemType”: {“:itemType” expressionAttributeValues: “Senjata”}
updateCharacterAttributes	Tabel dasar	UpdateItem	PK=PlayerID	SK=“#METADATA#playerID”	UpdateExpression: “SET mata uang = mata uang -:jumlah” : “mata Condition Expression uang>=:minAmount”

Pola akses	Basis table/ GSI/LSI	Operasi	Nilai kunci partisi	Nilai kunci urutan	Kondisi/filter lainnya
updateIte mCount	Tabel dasar	UpdateItem	PK=PlayerID	SK =“ITEMS#I temID”	update-ex pression: “SET ItemCount = ItemCount -:incr”: '{” :incr” expressio n-attribu te-values: {"N” :"1"}}'

Skema akhir profil game

Berikut adalah desain skema akhir. Untuk mengunduh desain skema ini sebagai file JSON, lihat Contoh DynamoDB di. GitHub

Tabel dasar:

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"}}}						

Menggunakan NoSQL Workbench dengan desain skema ini

Anda dapat mengimpor skema akhir ini ke NoSQL Workbench, sebuah alat visual yang menyediakan fitur pemodelan data, visualisasi data, dan pengembangan kueri untuk DynamoDB, guna mengeksplorasi dan mengedit proyek baru Anda lebih lanjut. Ikuti langkah-langkah berikut untuk memulai:

1. Unduh NoSQL Workbench. Untuk informasi selengkapnya, lihat the section called “Unduh”.
2. Unduh file skema JSON yang tercantum di atas, yang sudah dalam format model NoSQL Workbench.
3. Impor file skema JSON ke NoSQL Workbench. Untuk informasi selengkapnya, lihat the section called “Mengimpor model yang ada”.
4. Setelah mengimpor model data ke NoSQL Workbench, Anda dapat mengeditnya. Untuk informasi selengkapnya, lihat the section called “Mengedit model yang ada”.
5. Untuk memvisualisasikan model data Anda, menambahkan data sampel, atau mengimpor data sampel dari file CSV, gunakan fitur Data Visualizer di NoSQL Workbench.

Desain skema sistem manajemen pengaduan di DynamoDB

Kasus penggunaan bisnis sistem manajemen pengaduan

DynamoDB adalah basis data yang cocok untuk kasus penggunaan sistem manajemen pengaduan (atau pusat kontak) karena sebagian besar pola akses yang terkait dengannya adalah pencarian transaksional berbasis kunci-nilai. Pola akses tipikal dalam skenario ini adalah:

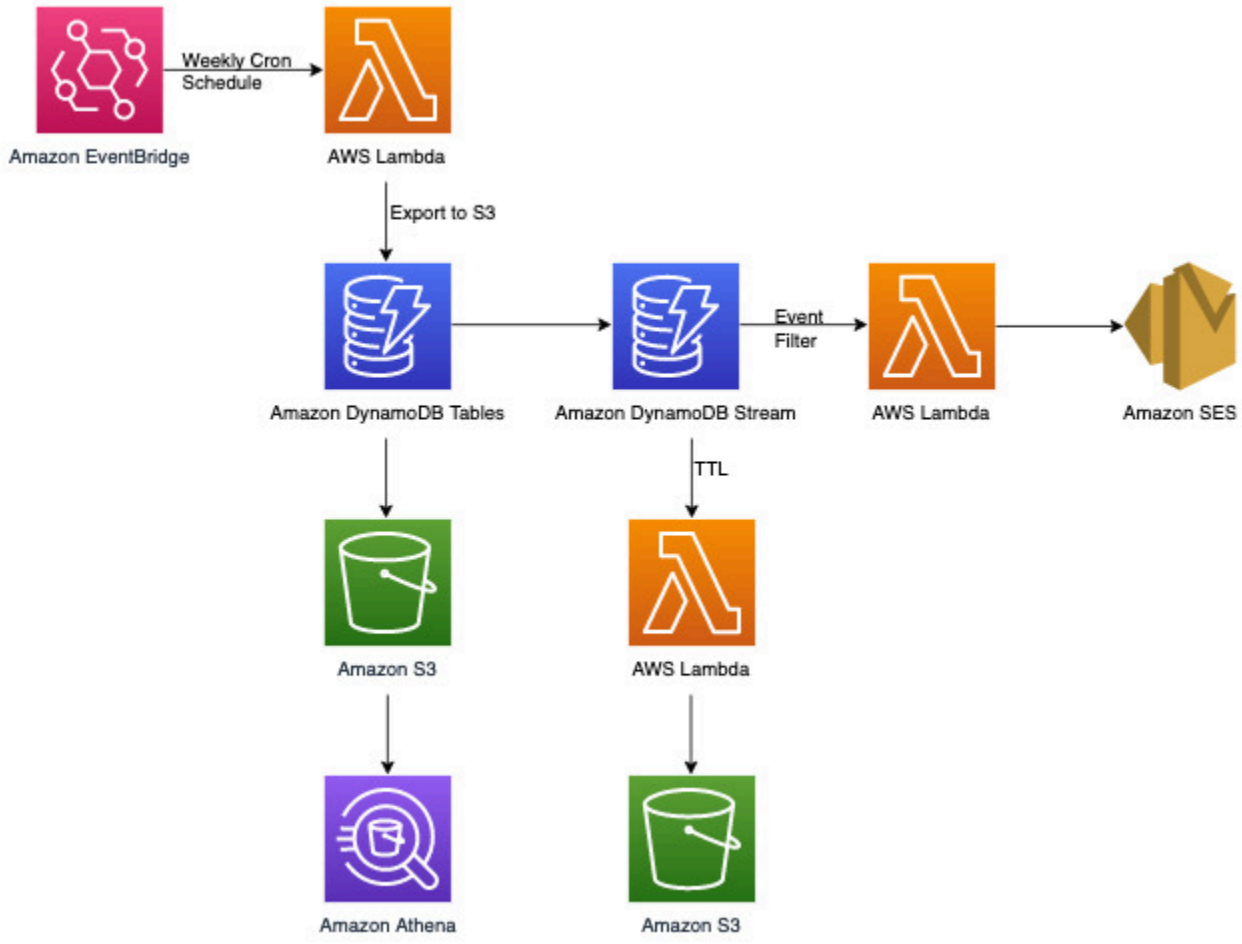
- Membuat dan memperbarui pengaduan
- Mengeskalasi pengaduan
- Membuat dan membaca komentar tentang pengaduan
- Mendapatkan semua pengaduan dari pelanggan
- Mendapatkan semua komentar oleh agen dan mendapatkan semua eskalasi

Beberapa komentar mungkin memiliki lampiran yang menjelaskan pengaduan atau solusi. Meskipun ini semua adalah pola akses kunci-nilai, mungkin ada persyaratan tambahan seperti mengirimkan pemberitahuan ketika komentar baru ditambahkan ke pengaduan atau menjalankan kueri analitik untuk menemukan distribusi pengaduan berdasarkan tingkat keparahan (atau kinerja agen) per

minggu. Persyaratan tambahan yang terkait dengan manajemen siklus hidup atau kepatuhan adalah mengarsipkan data pengaduan yang sudah tercatat selama tiga tahun.

Diagram arsitektur sistem manajemen pengaduan

Diagram berikut menunjukkan diagram arsitektur sistem manajemen keluhan. Diagram ini menunjukkan Layanan AWS integrasi berbeda yang digunakan sistem manajemen keluhan.



Terlepas dari pola akses transaksional kunci-nilai yang akan kita tangani di bagian pemodelan data DynamoDB nanti, ada tiga persyaratan non-transaksional. Diagram arsitektur di atas dapat dipecah menjadi tiga alur kerja berikut:

1. Kirim pemberitahuan saat komentar baru ditambahkan ke pengaduan
2. Jalankan kueri analitik pada data mingguan
3. Arsipkan data yang berusia lebih dari tiga tahun

Mari kita lihat setiap alur kerja tersebut lebih mendalam.

Kirim pemberitahuan saat komentar baru ditambahkan ke pengaduan

Kita dapat menggunakan alur kerja di bawah ini untuk mencapai persyaratan ini:

DynamoDB Streams adalah mekanisme pengambilan data perubahan untuk merekam semua aktivitas tulis pada tabel DynamoDB Anda. Anda dapat mengonfigurasi fungsi Lambda untuk memicu beberapa atau semua perubahan ini. Filter peristiwa dapat dikonfigurasi pada pemicu Lambda untuk memfilter peristiwa yang tidak relevan dengan kasus penggunaan. Dalam instans ini, kita dapat menggunakan filter untuk memicu Lambda hanya ketika komentar baru ditambahkan dan mengirimkan pemberitahuan ke ID email yang relevan yang dapat diambil dari AWS Secrets Manager atau penyimpanan kredensial lainnya.

Jalankan kueri analitik pada data mingguan

DynamoDB cocok untuk beban kerja yang utamanya difokuskan pada pemrosesan transaksional online (OLTP). Untuk 10-20% pola akses lainnya dengan persyaratan analitik, data dapat diekspor ke S3 dengan fitur Ekspor ke Amazon S3 yang dikelola tanpa berdampak pada lalu lintas langsung di tabel DynamoDB. Lihat alur kerja di bawah ini:

Amazon EventBridge dapat digunakan untuk memicu AWS Lambda sesuai jadwal - ini memungkinkan Anda mengonfigurasi ekspresi cron agar pemanggilan Lambda berlangsung secara berkala. Lambda dapat menginvokasi panggilan API `ExportToS3` dan menyimpan data DynamoDB di S3. Data S3 ini kemudian dapat diakses oleh mesin SQL seperti Amazon Athena untuk menjalankan kueri analitik pada data DynamoDB tanpa memengaruhi beban kerja transaksional langsung pada tabel. Contoh kueri Athena untuk menemukan jumlah pengaduan per tingkat keparahan akan terlihat seperti ini:

```
SELECT Item.severity.S as "Severity", COUNT(Item) as "Count"
FROM "complaint_management"."data"
WHERE NOT Item.severity.S = ''
GROUP BY Item.severity.S ;
```

Hal ini memunculkan hasil kueri Athena berikut:

Results (3)

Search rows

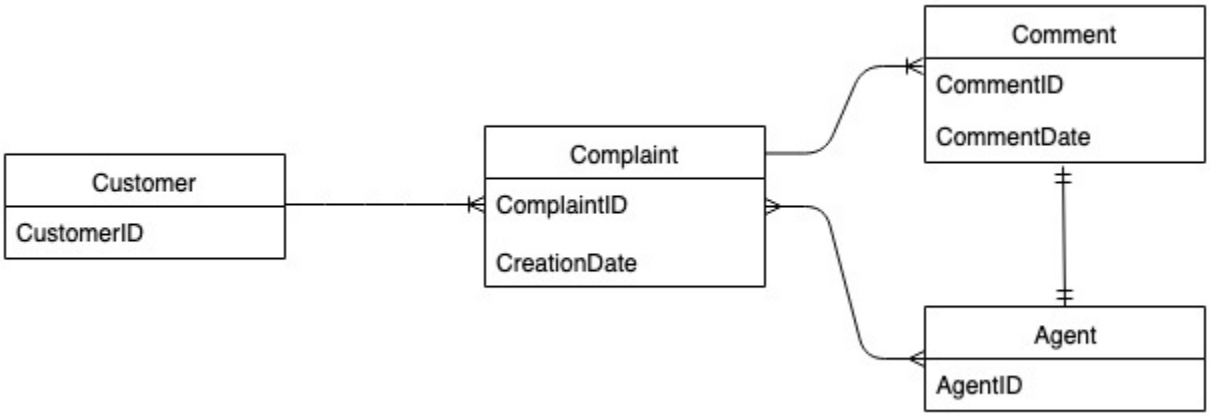
#	Severity	Count
1	P3	1
2	P2	2
3	P1	1

Arsipkan data yang berusia lebih dari tiga tahun

Anda dapat memanfaatkan fitur Waktu untuk Beroperasi (TTL) DynamoDB guna menghapus data usang dari tabel DynamoDB Anda tanpa biaya tambahan (kecuali dalam kasus replika tabel global untuk versi 2019.11.21 (Terbaru), di mana penghapusan TTL yang direplikasi ke Wilayah lain akan mengonsumsi kapasitas tulis). Data ini muncul dan dapat dikonsumsi dari DynamoDB Streams untuk diarsipkan ke Amazon S3. Alur kerja untuk persyaratan ini adalah sebagai berikut:

Diagram hubungan entitas sistem manajemen pengaduan

Ini adalah diagram hubungan entitas (ERD) yang akan kita gunakan untuk desain skema sistem manajemen pengaduan.



Pola akses sistem manajemen pengaduan

Ini adalah pola akses yang akan kita pertimbangkan untuk desain skema manajemen pengaduan.

1. createComplaint
2. updateComplaint
3. updateSeveritybyComplaintID
4. getComplaintByKeluhanD
5. addCommentByKeluhanD
6. getAllCommentsByComplaintID
7. getLatestCommentByComplaintID
8. dapatkan IDAnd keluhan AComplaintby pelangganID
9. getAllComplaintsByCustomerID
10. escalateComplaintByKeluhanD
11. getAllEscalatedKeluhan
12. getEscalatedComplaintsByAgentID (pesanan dari yang terbaru ke terlama)
13. getCommentsByAgenTid (antara dua tanggal)

Evolusi desain skema sistem manajemen pengaduan

Karena ini adalah sistem manajemen pengaduan, sebagian besar pola akses berkisar pada pengaduan sebagai entitas utama. `ComplaintID` yang sangat penting akan memastikan distribusi data yang merata di partisi-partisi yang mendasarinya, dan juga merupakan kriteria pencarian yang paling umum untuk pola akses yang teridentifikasi. Oleh karena itu, `ComplaintID` adalah kandidat kunci partisi yang baik dalam set data ini.

Langkah 1: Tangani pola akses 1 (**`createComplaint`**), 2 (**`updateComplaint`**), 3 (**`updateSeveritybyComplaintID`**), dan 4 (**`getComplaintByComplaintID`**)

Kita dapat menggunakan kunci urutan umum yang disebut "metadata" (atau "AA") untuk menyimpan informasi khusus pengaduan seperti`CustomerID`, `State`, `Severity`, dan `CreationDate`. Kita menggunakan operasi tunggal dengan `PK=ComplaintID` dan `SK=“metadata”` untuk melakukan hal berikut:

1. `PutItem` untuk membuat pengaduan baru
2. `UpdateItem` untuk memperbarui tingkat keparahan atau bidang lain dalam metadata pengaduan
3. `GetItem` untuk mengambil metadata untuk pengaduan

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>

Langkah 2: Atasi pola akses 5 (**`addCommentByComplaintID`**)

Pola akses ini membutuhkan model one-to-many hubungan antara keluhan dan komentar atas keluhan. Kita akan menggunakan teknik partisi vertikal di sini untuk menggunakan kunci urutan dan membuat koleksi item dengan berbagai jenis data. Jika kita melihat pola akses 6 (`getAllCommentsByComplaintID`) dan 7 (`getLatestCommentByComplaintID`), kita tahu bahwa komentar perlu diurutkan berdasarkan waktu. Beberapa komentar dapat masuk pada saat yang sama, sehingga kita dapat menggunakan teknik kunci urutan komposit untuk menambahkan waktu dan `CommentID` di atribut kunci urutan.

Opsi lain untuk menangani kemungkinan benturan komentar seperti itu adalah dengan meningkatkan perincian stempel waktu atau menambahkan angka tambahan sebagai sufiks alih-alih menggunakan

`Comment_ID`. Dalam hal ini, kita akan mengawali nilai kunci urutan untuk item yang sesuai dengan komentar dengan “comm#” untuk mengaktifkan operasi berbasis kisaran.

Kita juga perlu memastikan bahwa `currentState` dalam metadata pengaduan mencerminkan keadaan saat komentar baru ditambahkan. Menambahkan komentar mungkin menunjukkan bahwa pengaduan telah ditugaskan ke agen atau telah diselesaikan dan lain sebagainya. Untuk menggabungkan penambahan komentar dan pembaruan status saat ini dalam metadata keluhan, all-or-nothing dengan cara tertentu, kami akan menggunakan API. TransactWriteItems Status tabel kini terlihat seperti ini:

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>

Kita tambahkan beberapa data lagi dalam tabel dan juga tambahkan `ComplaintID` sebagai bidang terpisah dari PK kita untuk pemeriksaan model di masa depan jika kita membutuhkan indeks tambahan di `ComplaintID`. Perhatikan juga bahwa beberapa komentar mungkin memiliki lampiran yang akan kami simpan di Amazon Simple Storage Service dan hanya mempertahankan referensi mereka atau URLs di DynamoDB. Hal ini adalah praktik terbaik untuk menjaga basis data transaksional seramping mungkin untuk mengoptimalkan biaya dan performa. Data sekarang terlihat seperti ini:

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>

Langkah 3: Atasi pola akses 6 (**`getAllCommentsByComplaintID`**) dan 7 (**`getLatestCommentByComplaintID`**)

Untuk mendapatkan semua komentar untuk pengaduan, kita dapat menggunakan operasi `query` dengan kondisi `begins_with` pada kunci urutan. Daripada menghabiskan kapasitas baca tambahan untuk membaca entri metadata dan kemudian menyaring hasil yang relevan, memiliki kondisi kunci urutan seperti ini membantu kita untuk hanya membaca apa yang kita butuhkan. Misalnya, operasi kueri dengan `PK=Complaint123` dan SK begins_with `comm#` akan mengembalikan yang berikut saat melewatkan entri metadata:

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>

Karena kita memerlukan komentar terbaru untuk pengaduan dalam pola 7 (`getLatestCommentByComplaintID`), mari kita gunakan dua parameter kueri tambahan:

1. `ScanIndexForward` harus diatur ke False untuk mendapatkan hasil yang diurutkan dalam urutan menurun
2. `Limit` harus diatur ke 1 untuk mendapatkan (hanya satu) komentar terbaru

Mirip dengan pola akses 6 (`getAllCommentsByComplaintID`), kita melewatkan entri metadata menggunakan `begins_with comm#` sebagai kondisi kunci urutan. Sekarang, Anda dapat melakukan pola akses 7 pada desain ini menggunakan operasi kueri dengan `PK=Complaint123` dan `SK=begins_with comm#`, `ScanIndexForward=False`, serta `Limit` 1. Item yang ditargetkan berikut akan dikembalikan sebagai hasilnya:

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>

Mari tambahkan lebih banyak data dummy ke tabel.

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>

Langkah 4: Atasi pola akses 8 (**`getAComplaintbyCustomerIDAndComplaintID`**) dan 9 (**`getAllComplaintsByCustomerID`**)

Pola akses 8 (`getAComplaintbyCustomerIDAndComplaintID`) dan 9 (`getAllComplaintsByCustomerID`) memperkenalkan kriteria pencarian baru:`CustomerID`. Mengambilnya dari tabel yang ada memerlukan `Scan` yang mahal untuk membaca semua data dan kemudian memfilter item yang relevan untuk `CustomerID` yang bersangkutan. Kita dapat membuat

pencarian ini lebih efisien dengan membuat indeks sekunder global (GSI) dengan `CustomerID` sebagai kunci partisi. Mengingat one-to-many hubungan antara pelanggan dan keluhan serta pola akses 9 (`getAllComplaintsByCustomerID`), `ComplaintID` akan menjadi kandidat yang tepat untuk kunci pengurutan.

Data di GSI akan terlihat seperti ini:

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>

Contoh kueri pada GSI ini untuk pola akses 8 (`getAComplaintbyCustomerIDAndComplaintID`) adalah: `customer_id=custXYZ`, `sort key=Complaint1321`. Hasilnya akan menjadi:

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>

Untuk mendapatkan semua pengaduan bagi pelanggan untuk pola akses 9 (`getAllComplaintsByCustomerID`), kueri pada GSI adalah: `customer_id=custXYZ` sebagai kondisi kunci partisi. Hasilnya akan menjadi:

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>

Langkah 5: Atasi pola akses 10 (**`escalateComplaintByComplaintID`**)

Akses ini memperkenalkan aspek eskalasi. Untuk mengeskalasi pengaduan, kita dapat menggunakan `UpdateItem` untuk menambahkan atribut seperti `escalated_to` dan `escalation_time` ke item metadata pengaduan yang ada. DynamoDB menyediakan desain skema fleksibel yang berarti satu set atribut non-kunci dapat dibuat seragam atau terpisah di berbagai item. Lihat contohnya di bawah ini:

`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

Langkah 6: Atasi pola akses 11 (**`getAllEscalatedComplaints`**) dan 12 (**`getEscalatedComplaintsByAgentID`**)

Hanya segelintir pengaduan yang diharapkan akan dieskalasikan dari seluruh set data. Oleh karena itu, membuat indeks pada atribut terkait eskalasi akan menghasilkan pencarian yang efisien serta penyimpanan GSI yang hemat biaya. Kita bisa melakukan hal ini dengan memanfaatkan teknik indeks jarang. GSI dengan kunci partisi sebagai `escalated_to` dan kunci urutan sebagai `escalation_time` akan terlihat seperti ini:

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>

Untuk mendapatkan semua pengaduan yang dieskalasikan untuk pola akses 11 (`getAllEscalatedComplaints`), kita cukup memindai GSI ini. Perhatikan bahwa pemindaian ini akan berperforma baik dan hemat biaya karena ukuran GSI. Untuk mendapatkan pengaduan yang dieskalasikan untuk agen tertentu (pola akses 12 (`getEscalatedComplaintsByAgentID`)), kunci partisi akan berupa `escalated_to=agentID` dan kita mengatur `ScanIndexForward` ke `False` untuk mengurutkan dari yang terbaru ke yang terlama.

Langkah 7: Atasi pola akses 13 (**`getCommentsByAgentID`**)

Untuk pola akses terakhir, kita perlu melakukan pencarian berdasarkan dimensi baru: `AgentID`. Kita juga membutuhkan pengurutan berbasis waktu untuk membaca komentar di antara dua tanggal, jadi kita membuat GSI dengan `agent_id` sebagai kunci partisi dan `comm_date` sebagai kunci urutan. Data dalam GSI ini akan terlihat seperti berikut:

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"]

Contoh kueri pada GSI ini adalah `partition key agentID=AgentA` dan `sort key=comm_date between (2023-04-30T12:30:00, 2023-05-01T09:00:00)`, yang hasilnya adalah:

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"]

Semua pola akses dan bagaimana desain skema mengatasinya dirangkum dalam tabel di bawah ini:

Pola akses	Basis table/ GSI/LSI	Operasi	Nilai kunci partisi	Nilai kunci urutan	Kondisi/filter lainnya
createComplaint	Tabel dasar	PutItem	PK=complaint_id	SK=metadata	
updateComplaint	Tabel dasar	UpdateItem	PK=complaint_id	SK=metadata	
updateSeveritybyComplaintID	Tabel dasar	UpdateItem	PK=complaint_id	SK=metadata	
getComplaintByKeluhanD	Tabel dasar	GetItem	PK=complaint_id	SK=metadata	
addCommentByKeluhanD	Tabel dasar	TransactWriteItems	PK=complaint_id	SK=metadata, SK=comm#c	

Pola akses	Basis table/ GSI/LSI	Operasi	Nilai kunci partisi	Nilai kunci urutan	Kondisi/filter lainnya
				omm_date# comm_id	
getAllCommentsByComplaintID	Tabel dasar	Kueri	PK=complaint_id	SK begins_with "comm#"	
getLatestCommentByComplaintID	Tabel dasar	Kueri	PK=complaint_id	SK begins_with "comm#"	scan_index_forward =False, Limit 1
dapatkanIDAndkeluhanAComplaintbypelangganID	Customer_complaint_GSI	Kueri	customer_id=customer_id	complaint_id = complaint_id	
getAllComplaintsByCustomerID	Customer_complaint_GSI	Kueri	customer_id=customer_id	N/A	
escalateComplaintByKeluhanD	Tabel dasar	UpdateItem	PK=complaint_id	SK=metadata	
getAllEscalatedKeluhan	Escalations_GSI	Scan	N/A	N/A	

Pola akses	Basis table/ GSI/LSI	Operasi	Nilai kunci partisi	Nilai kunci urutan	Kondisi/filter lainnya
getEscalatedComplaintsByAgentID (pesanan dari yang terbaru ke terlama)	Escalations_GSI	Kueri	escalated_to=agent_id	N/A	scan_index_forward=False
getCommentsByAgenTid (antara dua tanggal)	Agents_Comments_GSI	Kueri	agent_id=agent_id	SK antara (date1, date2)	

Skema akhir sistem manajemen pengaduan

Berikut adalah desain skema akhir. Untuk mengunduh desain skema ini sebagai file JSON, lihat Contoh DynamoDB di. GitHub

Tabel dasar

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"]

Menggunakan NoSQL Workbench dengan desain skema ini

Anda dapat mengimpor skema akhir ini ke NoSQL Workbench, sebuah alat visual yang menyediakan fitur pemodelan data, visualisasi data, dan pengembangan kueri untuk DynamoDB, guna mengeksplorasi dan mengedit proyek baru Anda lebih lanjut. Ikuti langkah-langkah berikut untuk memulai:

1. Unduh NoSQL Workbench. Untuk informasi selengkapnya, lihat the section called “Unduh”.
2. Unduh file skema JSON yang tercantum di atas, yang sudah dalam format model NoSQL Workbench.
3. Impor file skema JSON ke NoSQL Workbench. Untuk informasi selengkapnya, lihat the section called “Mengimpor model yang ada”.
4. Setelah mengimpor model data ke NoSQL Workbench, Anda dapat mengeditnya. Untuk informasi selengkapnya, lihat the section called “Mengedit model yang ada”.
5. Untuk memvisualisasikan model data Anda, menambahkan data sampel, atau mengimpor data sampel dari file CSV, gunakan fitur Data Visualizer di NoSQL Workbench.

Desain skema pembayaran berulang di DynamoDB

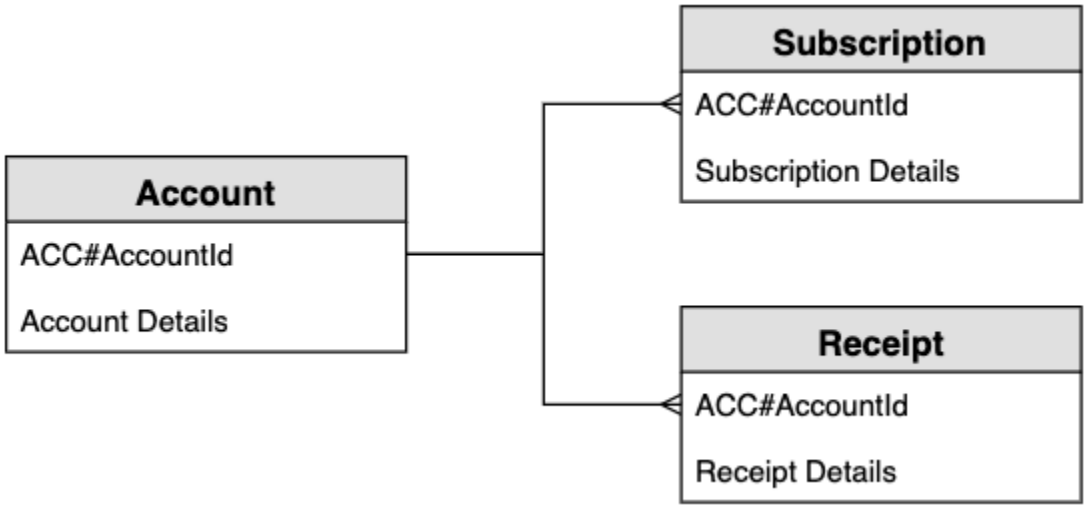
Kasus penggunaan bisnis pembayaran berulang

Kasus penggunaan ini membahas penggunaan DynamoDB untuk menerapkan sistem pembayaran berulang. Model data memiliki entitas berikut: akun, langganan, dan tanda terima. Spesifikasi untuk kasus penggunaan kita meliputi hal berikut:

- Setiap akun dapat memiliki beberapa langganan
- Langganan memiliki `NextPaymentDate` ketika pembayaran berikutnya perlu diproses dan `NextReminderDate` ketika pengingat email dikirim ke pelanggan
- Ada item untuk langganan yang disimpan dan diperbarui saat pembayaran diproses (ukuran item rata-rata sekitar 1 KB dan throughput-nya tergantung jumlah akun dan langganan)
- Pemroses pembayaran juga akan membuat tanda terima sebagai bagian dari proses yang disimpan dalam tabel dan diatur agar kedaluwarsa setelah jangka waktu tertentu dengan menggunakan atribut TTL

Diagram hubungan entitas pembayaran berulang

Ini adalah diagram hubungan entitas (ERD) yang akan kita gunakan untuk desain skema sistem pembayaran berulang.



Pola akses sistem pembayaran berulang

Ini adalah pola akses yang akan kita pertimbangkan untuk desain skema pembayaran berulang.

1. `createSubscription`
2. `createReceipt`
3. `updateSubscription`
4. `getDueRemindersByDate`
5. `getDuePaymentsByDate`
6. `getSubscriptionsByAccount`
7. `getReceiptsByAccount`

Desain skema pembayaran berulang

Nama generik PK dan SK digunakan untuk atribut kunci guna memungkinkan penyimpanan berbagai jenis entitas dalam tabel yang sama seperti entitas akun, langganan, dan tanda terima. Pertama, pengguna membuat langganan dan setuju untuk membayar sejumlah biaya pada hari yang sama setiap bulan sebagai harga atas suatu produk. Pengguna dapat memilih salah satu hari dalam sebulan untuk memproses pembayaran. Terdapat juga pengingat yang akan dikirim sebelum pembayaran diproses. Aplikasi bekerja dengan menjalankan dua tugas batch yang berjalan setiap hari: satu tugas batch mengirimkan pengingat yang jatuh tempo hari itu dan tugas batch lainnya memproses semua pembayaran yang jatuh tempo hari itu.

Langkah 1: Tangani pola akses 1 (**`createSubscription`**)

Pola akses 1 (`createSubscription`) awalnya digunakan untuk membuat langganan, dan detail yang mencakup SKU, `NextPaymentDate`, `NextReminderDate`, dan `PaymentDetails` diatur. Langkah ini menunjukkan status tabel hanya untuk satu akun dengan satu langganan. Mungkin ada beberapa langganan dalam koleksi item jadi ini adalah one-to-many hubungan.

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

Langkah 2: Tangani pola akses 2 (**`createReceipt`**) dan 3 (**`updateSubscription`**)

Pola akses 2 (`createReceipt`) digunakan untuk membuat item tanda terima. Setelah pembayaran diproses setiap bulan, pemroses pembayaran akan menulis tanda terima kembali ke tabel dasar. Di sana, bisa ada beberapa tanda terima dalam koleksi item jadi ini adalah one-to-many hubungan. Pemroses pembayaran juga akan memperbarui item langganan (Pola akses 3

(`updateSubscription`)) untuk memperbarui `NextReminderDate` atau `NextPaymentDate` bulan berikutnya.

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

Langkah 3: Tangani pola akses 4 (**`getDueRemindersByDate`**)

Aplikasi memproses pengingat untuk pembayaran dalam batch untuk hari ini. Oleh karena itu, aplikasi perlu mengakses langganan pada dimensi yang berbeda, yaitu tanggal, bukan akun. Kasus penggunaan ini bagus untuk indeks sekunder global (GSI). Pada langkah ini kita menambahkan indeks `GSI-1`, yang menggunakan `NextReminderDate` sebagai kunci partisi GSI. Kita tidak perlu mereplikasi semua item. GSI ini adalah indeks jarang dan item tanda terima tidak direplikasi. Kita juga tidak perlu memproyeksikan semua atribut—kita hanya perlu menyertakan subset atribut. Gambar di bawah ini menunjukkan skema `GSI-1` dan memberikan informasi yang diperlukan aplikasi untuk mengirim email pengingat.

Primary key		Attributes				
Partition key: NextReminderDate	**Sort key: LastReminderDate**					
2023-06-21	2023-05-21T14:15:39.247Z	SK	PK	SKU	Email	NextPaymentDate
		SUB#123#SKU#999	ACC#123	999	s@s.com	2023-06-28

Langkah 4: Tangani pola akses 5 (**`getDuePaymentsByDate`**)

Aplikasi memproses pembayaran dalam batch untuk hari ini dengan cara yang sama seperti pengingat. Kita menambahkan indeks `GSI-2` pada langkah ini, yang menggunakan `NextPaymentDate` sebagai kunci partisi GSI. Kita tidak perlu mereplikasi semua item. GSI ini adalah indeks jarang dan item tanda terima tidak direplikasi. Gambar di bawah ini menunjukkan skema `GSI-2`.

Primary key		Attributes						
Partition key: NextPaymentDate	**Sort key: LastPaymentDate**							
2023-06-28	2023-05-18T14:15:39.247Z	PK	SK	Email	PaymentDay	PaymentAmount	SKU	PaymentDetails
		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"}}

Langkah 5: Tangani pola akses 6 (**`getSubscriptionsByAccount`**) dan 7 (**`getReceiptsByAccount`**)

Aplikasi dapat mengambil semua langganan untuk akun dengan menggunakan kueri pada tabel dasar yang menargetkan pengidentifikasi akun (PK) dan menggunakan operator rentang untuk mendapatkan semua item dengan “SUB#” sebagai awalan SK. Aplikasi juga dapat menggunakan struktur kueri yang sama untuk mengambil semua tanda terima menggunakan operator rentang untuk mendapatkan semua item dengan “REC#” sebagai awalan SK. Hal ini memungkinkan kita memenuhi pola akses 6 (`getSubscriptionsByAccount`) dan 7 (`getReceiptsByAccount`). Aplikasi menggunakan pola akses tersebut sehingga pengguna dapat melihat langganan saat ini dan tanda terima lama mereka selama enam bulan terakhir. Tidak ada perubahan pada skema tabel dalam langkah ini dan kita dapat melihat di bawah ini bagaimana kita hanya menargetkan item berlangganan dalam pola akses 6 (`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		Email	PaymentDay	PaymentAmount	LastPaymentDate	NextPaymentDate	LastReminderDate	NextReminderDate	SKU	PaymentDetails	CreatedDate
	SUB#123#SKU#999	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

Semua pola akses dan bagaimana desain skema mengatasinya dirangkum dalam tabel di bawah ini:

Pola akses	Basis table/GSI/LSI	Operasi	Nilai kunci partisi	Nilai kunci urutan
createSubscription	Tabel dasar	PutItem	ACC#account_id	SUB#<SUBID>#SKU<SKUID>
createReceipt	Tabel dasar	PutItem	ACC#account_id	REC#< > #SKU RecieptDate <SKUID>
updateSubscription	Tabel dasar	UpdateItem	ACC#account_id	SUB#<SUBID>#SKU<SKUID>
getDueRemindersByDate	GSI-1	Kueri	<NextReminderDate>	

Pola akses	Basis table/GSI/ LSI	Operasi	Nilai kunci partisi	Nilai kunci urutan
getDuePay mentsByDate	GSI-2	Kueri	<NextPaym entDate>	
getSubscr iptionsByAkun	Tabel dasar	Kueri	ACC#account_id	SK begins_with “SUB#”
getReceip tsByAkun	Tabel dasar	Kueri	ACC#account_id	SK begins_with “REC#”

Skema akhir pembayaran berulang

Berikut adalah desain skema akhir. [Untuk mengunduh desain skema ini sebagai file JSON, lihat Contoh DynamoDB di.](#) GitHub

Tabel dasar

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"}}

Menggunakan NoSQL Workbench dengan desain skema ini

Anda dapat mengimpor skema akhir ini ke NoSQL Workbench, sebuah alat visual yang menyediakan fitur pemodelan data, visualisasi data, dan pengembangan kueri untuk DynamoDB, guna mengeksplorasi dan mengedit proyek baru Anda lebih lanjut. Ikuti langkah-langkah berikut untuk memulai:

1. Unduh NoSQL Workbench. Untuk informasi selengkapnya, lihat the section called “Unduh”.
2. Unduh file skema JSON yang tercantum di atas, yang sudah dalam format model NoSQL Workbench.
3. Impor file skema JSON ke NoSQL Workbench. Untuk informasi selengkapnya, lihat the section called “Mengimpor model yang ada”.
4. Setelah mengimpor model data ke NoSQL Workbench, Anda dapat mengeditnya. Untuk informasi selengkapnya, lihat the section called “Mengedit model yang ada”.
5. Untuk memvisualisasikan model data Anda, menambahkan data sampel, atau mengimpor data sampel dari file CSV, gunakan fitur Data Visualizer di NoSQL Workbench.

Memantau pembaruan status perangkat di DynamoDB

Kasus penggunaan ini membahas tentang penggunaan DynamoDB untuk memantau pembaruan status perangkat (atau perubahan status perangkat) di DynamoDB.

Kasus penggunaan

Dalam kasus penggunaan IoT (pabrik pintar misalnya), banyak perangkat yang perlu dipantau oleh operator dan perangkat tersebut secara berkala mengirim status atau log ke sistem pemantauan. Ketika ada masalah dengan sebuah perangkat, status perangkat tersebut berubah dari normal menjadi peringatan. Ada tingkat log atau status yang berbeda tergantung tingkat keparahan dan jenis perilaku abnormal pada perangkat. Sistem kemudian menugaskan operator untuk memeriksa perangkat dan mereka dapat mengeskalasikan masalah kepada supervisor jika diperlukan.

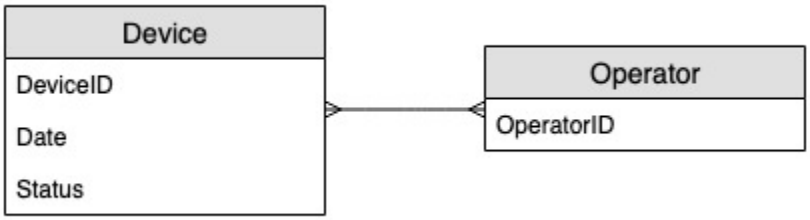
Beberapa pola akses tipikal untuk sistem ini meliputi:

- Membuat entri log untuk sebuah perangkat
- Mendapatkan semua log untuk status perangkat tertentu yang menampilkan log terbaru terlebih dahulu
- Mendapatkan semua log untuk operator tertentu di antara dua tanggal

- Mendapatkan semua log yang dieskalasi untuk supervisor tertentu
- Mendapatkan semua log yang dieskalasi dengan status perangkat tertentu untuk supervisor tertentu
- Mendapatkan semua log yang dieskalasi dengan status perangkat tertentu untuk supervisor tertentu pada tanggal tertentu

Diagram hubungan entitas

Ini adalah diagram hubungan entitas (ERD) yang akan kita gunakan untuk memantau pembaruan status perangkat.



Pola akses

Ini adalah pola akses yang akan kita pertimbangkan untuk memantau pembaruan status perangkat.

1. `createLogEntryForSpecificDevice`
2. `getLogsForSpecificDevice`
3. `getWarningLogsForSpecificDevice`
4. `getLogsForOperatorBetweenTwoDates`
5. `getEscalatedLogsForSupervisor`
6. `getEscalatedLogsWithSpecificStatusForSupervisor`
7. `getEscalatedLogsWithSpecificStatusForSupervisorForDate`

Evolusi desain skema

Langkah 1: Atasi pola akses 1 (**`createLogEntryForSpecificDevice`**) dan 2 (**`getLogsForSpecificDevice`**)

Unit penskalaan untuk sistem pelacakan perangkat adalah perangkat individual. Dalam sistem ini, `deviceID` mengidentifikasi perangkat secara unik. Ini menjadikan `deviceID` kandidat yang

baik untuk kunci partisi. Setiap perangkat mengirim informasi ke sistem pelacakan secara berkala (misalnya, setiap lima menit atau lebih). Pengurutan ini menjadikan tanggal sebagai kriteria pengurutan logis dan oleh karena itu, menjadi kunci urutan. Data sampel dalam kasus ini akan terlihat seperti ini:

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

Untuk mengambil entri log perangkat tertentu, kita dapat melakukan operasi kueri dengan kunci partisi `DeviceID="d#12345"`.

Langkah 2: Atasi pola akses 3 (**`getWarningLogsForSpecificDevice`**)

Karena `State` merupakan atribut non-kunci, mengatasi pola akses 3 dengan skema saat ini akan memerlukan ekspresi filter. Di DynamoDB, ekspresi filter diterapkan setelah data dibaca menggunakan ekspresi kondisi kunci. Misalnya, jika kita mengambil log peringatan untuk`d#12345`,

operasi kueri dengan kunci partisi `DeviceID="d#12345"` akan membaca empat item dari tabel di atas dan kemudian menyaring satu item tanpa status peringatan. Pendekatan ini tidak efisien dalam skala besar. Ekspresi filter dapat menjadi cara yang baik untuk mengecualikan item dalam kueri jika rasio item yang dikecualikan rendah atau kueri jarang dilakukan. Namun, untuk kasus dengan banyak item diambil dari tabel dan sebagian besar item disaring, kita dapat terus mengembangkan desain tabel agar berjalan lebih efisien.

Mari kita ubah cara menangani pola akses ini menggunakan kunci urutan komposit. Anda dapat mengimpor data sampel dari DeviceStateLog_3.json di mana kunci pengurutan diubah menjadi. `State#Date` Kunci urutan ini adalah komposisi atribut `State`, `#`, dan `Date`. Dalam contoh ini, `#` digunakan sebagai pembatas. Data sekarang terlihat seperti ini:

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

Untuk mengambil log peringatan untuk perangkat saja, kueri menjadi lebih tertarget dengan skema ini. Kondisi kunci untuk kueri menggunakan kunci partisi `DeviceID="d#12345"` dan kunci urutan `State#Date begins_with “WARNING”`. Kueri ini hanya akan membaca tiga item yang relevan dengan status peringatan.

Langkah 3: Atasi pola akses 4 (**`getLogsForOperatorBetweenTwoDates`**)

Anda dapat mengimpor DeviceStateLog_4.json D di mana `Operator` atribut ditambahkan ke `DeviceStateLog` tabel dengan contoh data.

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

Karena `Operator` bukan kunci partisi, tidak ada cara untuk melakukan pencarian nilai kunci langsung pada tabel ini berdasarkan `OperatorID`. Kita perlu membuat koleksi item baru dengan

indeks sekunder global aktif pada `OperatorID`. Pola akses memerlukan pencarian berdasarkan tanggal, sehingga Tanggal adalah atribut kunci urutan untuk indeks sekunder global (GSI). Tampilan GSI sekarang menjadi:

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

Untuk pola akses 4 (`getLogsForOperatorBetweenTwoDates`), Anda dapat mengueri GSI ini dengan kunci partisi `OperatorID=Liz` dan kunci urutan `Date` antara `2020-04-11T05:58:00` dan `2020-04-24T14:50:00`.

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

Langkah 4: Tangani pola akses 5 (**`getEscalatedLogsForSupervisor`**), 6 (**`getEscalatedLogsWithSpecificStatusForSupervisor`**), dan 7 (**`getEscalatedLogsWithSpecificStatusForSupervisorForDate`**)

Kami akan menggunakan indeks jarang untuk mengatasi pola akses ini.

Indeks sekunder global secara default bersifat jarang, jadi hanya item dalam tabel dasar yang berisi atribut kunci primer indeks yang benar-benar akan muncul di indeks. Ini adalah cara lain untuk mengecualikan item yang tidak relevan untuk pola akses yang dimodelkan.

Anda dapat mengimpor DeviceStateLog_6.json di mana `EscalatedTo` atribut ditambahkan ke `DeviceStateLog` tabel dengan contoh data. Seperti disebutkan sebelumnya, tidak semua log akan dieskalasikan ke supervisor.

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

Anda sekarang dapat membuat GSI baru di mana `EscalatedTo` adalah kunci partisi dan `State#Date` adalah kunci urutan. Perhatikan bahwa hanya item yang memiliki atribut `EscalatedTo` dan `State#Date` yang muncul dalam indeks.

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

Pola akses lainnya dirangkum sebagai berikut:

Semua pola akses dan bagaimana desain skema mengatasinya dirangkum dalam tabel di bawah ini:

Pola akses	Basis table/ GSI/LSI	Operasi	Nilai kunci partisi	Nilai kunci urutan	Kondisi/filter lainnya
createLogEntryForSpecificDevice	Tabel dasar	PutItem	DeviceID=deviceId	State#Date=state#date	
getLogsForSpecificDevice	Tabel dasar	Kueri	DeviceID=deviceId	State#Date begins_with "state1#"	ScanIndexForward = Salah
getWarningLogsForSpecificDevice	Tabel dasar	Kueri	DeviceID=deviceId	State#Date begins_with "WARNING"	
getLogsForOperatorBetweenTwoDates	GSI-1	Kueri	Operator=operatorName	Tanggal antara date1 dan date2	
getEscalatedLogsForSupervisor	GSI-2	Kueri	EscalatedTo=Namapengawas		
getEscalatedLogsWithSpecifi	GSI-2	Kueri	EscalatedTo=Namapengawas	State#Date begins_with "state1#"	

Pola akses	Basis table/ GSI/LSI	Operasi	Nilai kunci partisi	Nilai kunci urutan	Kondisi/filter lainnya
cStatusFo rSupervisor					
getEscala tedLogsWi thSpecifi cStatusFo rSupervis orForDate	GSI-2	Kueri	Escalated To=Nama pengawas	State#Dat e begins_wi th "state1#d ate1"	

Skema akhir

Berikut adalah desain skema akhir. Untuk mengunduh desain skema ini sebagai file JSON, lihat Contoh DynamoDB di. GitHub

Tabel dasar

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	DeviceID	Operator	Date	State
Sara	WARNING4#2020-04-27T16:15:00	d#11223	Sue	2020-04-27T16:15:00	WARNING4

Menggunakan NoSQL Workbench dengan desain skema ini

Anda dapat mengimpor skema akhir ini ke NoSQL Workbench, sebuah alat visual yang menyediakan fitur pemodelan data, visualisasi data, dan pengembangan kueri untuk DynamoDB, guna mengeksplorasi dan mengedit proyek baru Anda lebih lanjut. Ikuti langkah-langkah berikut untuk memulai:

1. Unduh NoSQL Workbench. Untuk informasi selengkapnya, lihat the section called “Unduh”.
2. Unduh file skema JSON yang tercantum di atas, yang sudah dalam format model NoSQL Workbench.
3. Impor file skema JSON ke NoSQL Workbench. Untuk informasi selengkapnya, lihat the section called “Mengimpor model yang ada”.
4. Setelah mengimpor model data ke NoSQL Workbench, Anda dapat mengeditnya. Untuk informasi selengkapnya, lihat the section called “Mengedit model yang ada”.
5. Untuk memvisualisasikan model data Anda, menambahkan data sampel, atau mengimpor data sampel dari file CSV, gunakan fitur Data Visualizer di NoSQL Workbench.

Menggunakan DynamoDB sebagai penyimpanan data untuk toko online

Kasus penggunaan ini membahas penggunaan DynamoDB sebagai penyimpanan data untuk toko online (atau e-store).

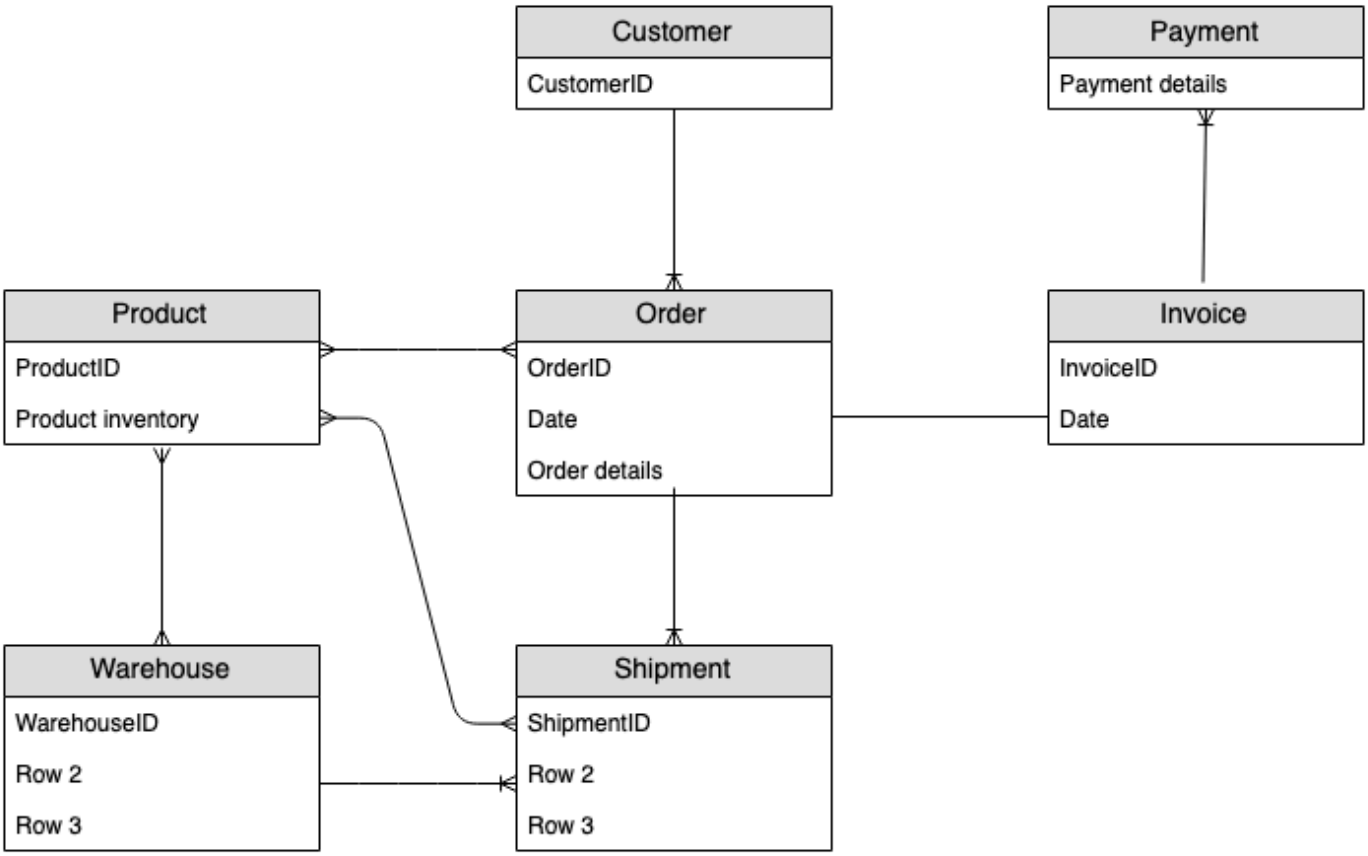
Kasus penggunaan

Toko online memungkinkan pengguna menelusuri berbagai produk dan pada akhirnya membelinya. Berdasarkan faktur yang dihasilkan, pelanggan dapat membayar menggunakan kode diskon atau kartu hadiah lalu membayar jumlah yang tersisa dengan kartu kredit. Produk yang dibeli akan diambil dari salah satu gudang dan akan dikirim ke alamat yang diberikan. Pola akses umum untuk toko online meliputi:

- Mendapatkan pelanggan untuk customerId tertentu
- Mendapatkan produk untuk productId tertentu
- Mendapatkan gudang untuk warehouseId tertentu
- Mendapatkan inventaris produk untuk semua gudang berdasarkan productId
- Mendapatkan pesanan untuk orderId tertentu
- Mendapatkan semua produk untuk orderId tertentu
- Mendapatkan faktur untuk orderId tertentu
- Mendapatkan semua pengiriman untuk orderId tertentu
- Mendapatkan semua pesanan untuk productId tertentu dan rentang tanggal tertentu
- Mendapatkan faktur untuk invoiceId tertentu
- Mendapatkan semua pembayaran untuk invoiceId tertentu
- Mendapatkan detail pengiriman untuk shipmentId tertentu
- Mendapatkan semua pengiriman untuk warehouseId tertentu
- Mendapatkan inventaris semua produk untuk warehouseId tertentu
- Mendapatkan semua faktur untuk customerId tertentu dan rentang tanggal tertentu
- Mendapatkan semua produk yang dipesan untuk customerId tertentu dan rentang tanggal tertentu

Diagram hubungan entitas

Ini adalah diagram hubungan entitas (ERD) yang akan kita gunakan untuk memodelkan DynamoDB sebagai penyimpanan data untuk toko online.



Pola akses

Ini adalah pola akses yang akan dipertimbangkan saat menggunakan DynamoDB sebagai penyimpanan data untuk toko online.

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

Evolusi desain skema

MenggunakanNoSQL Workbench untuk DynamoDB, import AnOnlineShop_1.json untuk membuat model data baru dipanggil AnOnlineShop dan tabel baru dipanggil. OnlineShop Perhatikan bahwa kita menggunakan nama generik PK dan SK untuk kunci partisi dan kunci urutan. Hal ini adalah praktik yang digunakan untuk menyimpan berbagai jenis entitas dalam tabel yang sama.

Langkah 1: Tangani pola akses 1 (**getCustomerByCustomerId**)

Impor AnOnlineShop_2.json untuk menangani pola akses 1 (). getCustomerByCustomerId Beberapa entitas tidak memiliki hubungan dengan entitas lain, jadi kita akan menggunakan nilai yang sama dari PK dan SK untuk entitas tersebut. Dalam contoh data, perhatikan bahwa kunci menggunakan prefiks c# untuk membedakan customerId dari entitas lain yang akan ditambahkan nanti. Praktik ini diulang untuk entitas lain juga.

Untuk menangani pola akses ini, operasi GetItem dapat digunakan dengan PK=customerId dan SK=customerId.

Langkah 2: Tangani pola akses 2 (**getProductByProductId**)

Impor AnOnlineShop_3.json ke alamat pola akses 2 (getProductByProductId) untuk entitas. product Entitas produk diawali oleh p# dan atribut kunci urutan yang sama digunakan untuk menyimpan customerID serta productID. Penamaan generik dan partisi vertikal memungkinkan kita membuat koleksi item untuk desain tabel tunggal yang efektif.

Untuk menangani pola akses ini, operasi GetItem dapat digunakan dengan PK=productId dan SK=productId.

Langkah 3: Tangani pola akses 3 (**getWarehouseByWarehouseId**)

Impor AnOnlineShop_4.json ke alamat pola akses 3 (getWarehouseByWarehouseId) untuk entitas. warehouse Saat ini entitas customer, product, dan warehouse ditambahkan ke tabel

yang sama. Entitas tersebut dibedakan menggunakan prefiks dan atribut `EntityType`. Atribut jenis (atau penamaan prefiks) meningkatkan keterbacaan model. Keterbacaan akan terpengaruh jika kita hanya menyimpan alfanumerik IDs untuk entitas yang berbeda dalam atribut yang sama. Akan sulit untuk membedakan satu entitas dari yang lain tanpa adanya pengidentifikasi ini.

Untuk menangani pola akses ini, operasi `GetItem` dapat digunakan dengan `PK=warehouseId` dan `SK=warehouseId`.

Tabel dasar:

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"}}	

Langkah 4: Tangani pola akses 4 (**`getProductInventoryByProductId`**)

Impor AnOnlineShop_5.json ke pola akses alamat 4 (). `getProductInventoryByProductId` `warehouseItem`entitas digunakan untuk melacak jumlah produk di setiap gudang. Item ini biasanya akan diperbarui ketika produk ditambahkan atau dihapus dari gudang. Seperti yang terlihat di ERD, ada many-to-many hubungan antara `product` dan`warehouse`. Di sini, one-to-many hubungan dari `product` ke `warehouse` dimodelkan sebagai`warehouseItem`. Nantinya, one-to-many hubungan dari `warehouse` hingga `product` akan dimodelkan juga.

Pola akses 4 dapat ditangani dengan kueri pada `PK=ProductId` dan `SK begins_with “w#“`.

Untuk informasi selengkapnya tentang `begins_with()` dan ekspresi lain yang dapat diterapkan ke kunci urutan, lihat Ekspresi Kondisi Kunci.

Tabel dasar:

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"}}	

Langkah 5: Tangani pola akses 5 (**`getOrderDetailsByOrderId`**) dan 6 (**`getProductByOrderId`**)

Tambahkan beberapa lagi`customer`,`product`, dan `warehouse` item ke tabel dengan mengimpor AnOnlineShop_6.json. Kemudian, impor AnOnlineShop_7.json untuk membuat koleksi item `order` yang dapat mengatasi pola akses 5 (`getOrderDetailsByOrderId`) dan 6 (). `getProductByOrderId` Anda dapat melihat one-to-many hubungan antara `order` dan `product` dimodelkan sebagai entitas OrderItem.

Untuk menangani pola akses 5 (`getOrderDetailsByOrderId`), lakukan kueri tabel dengan `PK=orderId`. Tindakan ini akan memberikan semua informasi tentang pesanan termasuk `customerId` dan produk yang dipesan.

Tabel dasar:

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

Untuk menagani pola akses 6 (`getProductByOrderId`), kita hanya perlu membaca produk dalam `order`. Kueri tabel dengan `PK=orderId` dan `SK begins_with "p#"` untuk mencapai hal ini.

Tabel dasar:

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

Langkah 6: Tangani pola akses 7 (**`getInvoiceByOrderId`**)

Impor AnOnlineShop_8.json untuk menambahkan `invoice` entitas ke koleksi item pesanan untuk menangani pola akses 7 (). `getInvoiceByOrderId` Untuk menangani pola akses ini, operasi `PK=orderId` dan `SK begins_with "i#"` dapat digunakan.

Tabel dasar:

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

Langkah 7: Tangani pola akses 8 (**`getShipmentByOrderId`**)

Impor AnOnlineShop_9.json untuk menambahkan `shipment` entitas ke koleksi item pesanan untuk mengatasi pola akses 8 (). `getShipmentByOrderId` Model yang dipartisi secara vertikal yang sama diperluas dengan menambahkan lebih banyak jenis entitas dalam desain tabel tunggal. Perhatikan bagaimana koleksi item pesanan berisi hubungan yang berbeda dari hubungan entitas `order` dengan entitas`shipment`, `orderItem`, dan `invoice`.

Untuk mendapatkan pengiriman berdasarkan `orderId`, Anda dapat melakukan operasi kueri dengan `PK=orderId` dan `SK begins_with “sh#”`.

Tabel dasar:

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

Langkah 8: Tangani pola akses 9 (**`getOrderByProductIdForDateRange`**)

Kita membuat koleksi item pesanan pada langkah sebelumnya. Pola akses ini memiliki dimensi pencarian baru (`ProductID` dan `Date`) yang mengharuskan Anda memindai seluruh tabel dan memfilter catatan yang relevan untuk mengambil item yang ditargetkan. Untuk menangani pola akses ini, kita harus membuat indeks sekunder global (GSI). Impor AnOnlineShop_10.json untuk membuat koleksi item baru menggunakan GSI yang memungkinkan untuk mengambil *`orderItem`* data dari beberapa koleksi item pesanan. Sekarang data memiliki `GSI1-PK` dan `GSI1-SK` yang masing-masing akan menjadi kunci partisi dan kunci urutan `GSI1`.

DynamoDB otomatis mengisi item yang berisi atribut kunci GSI dari tabel ke GSI. Tidak perlu melakukan penyisipan tambahan secara manual ke GSI.

Untuk menangani pola akses 9, lakukan kueri `GSI1` dengan `GSI1-PK=productId` dan `GSI1SK between (date1, date2)`.

Tabel dasar:

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

Langkah 9: Tangani pola akses 10 (**`getInvoiceByInvoiceId`**) dan 11 (**`getPaymentByInvoiceId`**)

Impor AnOnlineShop_11.json untuk mengatasi pola akses 10 (`getInvoiceByInvoiceId`) dan 11 (`getPaymentByInvoiceId`), keduanya terkait dengan. `invoice` Meskipun berbeda, dua pola akses ini direalisasikan menggunakan kondisi kunci yang sama. `Payments` didefinisikan sebagai atribut dengan jenis data peta pada entitas `invoice`.

> **Note**
>
> `GSI1-PK` dan `GSI1-SK` kelebihan beban untuk menyimpan informasi berbagai entitas sehingga beberapa pola akses dapat dilayani dari GSI yang sama. Untuk informasi selengkapnya tentang kelebihan beban GSI, lihat Membebani Indeks Sekunder Global di DynamoDB.

Untuk menangani pola akses 10 dan 11, lakukan kueri `GSI1` dengan `GSI1-PK=invoiceId` dan `GSI1-SK=invoiceId`.

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

Langkah 10: Tangani pola akses 12 (**`getShipmentDetailsByShipmentId`**) dan 13 (**`getShipmentByWarehouseId`**)

Impor AnOnlineShop_12.json untuk mengatasi pola akses 12 (`getShipmentDetailsByShipmentId`) dan 13 (). `getShipmentByWarehouseId`

Perhatikan bahwa entitas `shipmentItem` ditambahkan ke koleksi item pesanan pada tabel dasar agar dapat mengambil semua detail tentang pesanan dalam satu operasi kueri.

Tabel dasar:

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				

Kunci `GSI1` partisi dan sortir telah digunakan untuk memodelkan one-to-many hubungan antara `shipment` dan`shipmentItem`. Untuk menangani pola akses 12 (`getShipmentDetailsByShipmentId`), lakukan kueri `GSI1` dengan `GSI1-PK=shipmentId` dan `GSI1-SK=shipmentId`.

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

Kita perlu membuat GSI (`GSI2`) lain untuk memodelkan one-to-many hubungan baru antara `warehouse` dan `shipment` untuk pola akses 13 (`getShipmentByWarehouseId`). Untuk

menangani pola akses ini, lakukan kueri `GSI2` dengan `GSI2-PK=warehouseId` dan `GSI2-SK begins_with “sh#”`.

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

Langkah 11: Tangani pola akses 14 (**`getProductInventoryByWarehouseId`**) 15 (**`getInvoiceByCustomerIdForDateRange`**), dan 16 (**`getProductsByCustomerIdForDateRange`**)

Impor AnOnlineShop_13.json untuk menambahkan data yang terkait dengan kumpulan pola akses berikutnya. Untuk menangani pola akses 14 (`getProductInventoryByWarehouseId`), lakukan kueri `GSI2` dengan `GSI2-PK=warehouseId` dan `GSI2-SK begins_with “p#”`.

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	

Untuk menangani pola akses 15 (`getInvoiceByCustomerIdForDateRange`), lakukan kueri `GSI2` dengan `GSI2-PK=customerId` dan `GSI2-SK between (i#date1, i#date2)`.

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	

Untuk menangani pola akses 16 (`getProductsByCustomerIdForDateRange`), lakukan kueri `GSI2` dengan `GSI2-PK=customerId` dan `GSI2-SK between (p#date1, p#date2)`.

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

Di NoSQL Workbench, faset mewakili pola akses data aplikasi yang berbeda untuk DynamoDB. Faset memberi Anda cara untuk melihat subset data dalam tabel tanpa harus melihat rekaman yang tidak memenuhi batasan faset. Faset dianggap sebagai alat pemodelan data visual, dan tidak ada sebagai konstruksi yang dapat digunakan di DynamoDB, karena aspek tersebut murni bantuan untuk memodelkan pola akses. Impor AnOnlineShop_facets.json untuk melihat aspek untuk kasus penggunaan ini.

Semua pola akses dan bagaimana desain skema mengatasinya dirangkum dalam tabel di bawah ini:

Pola akses	Basis table/GSI/LSI	Operasi	Nilai kunci partisi	Nilai kunci urutan
getCustomerByCustomerId	Tabel dasar	GetItem	PK=customerId	SK=customerId
getProductByProductId	Tabel dasar	GetItem	PK=productId	SK=productId
getWarehouseByWarehouseId	Tabel dasar	GetItem	PK=warehouseId	SK=warehouseId
getProductInventoryByProductId	Tabel dasar	Kueri	PK=productId	SK begins_with "w#"
getOrderDetailsByOrderId	Tabel dasar	Kueri	PK=orderId	
getProductByOrderId	Tabel dasar	Kueri	PK=orderId	SK begins_with "p#"
getInvoiceByOrderId	Tabel dasar	Kueri	PK=orderId	SK begins_with "i#"
getShipmentByOrderId	Tabel dasar	Kueri	PK=orderId	SK begins_with "sh#"
getOrderByProductIdForDateRange	GSI1	Kueri	PK=productId	SK antara date1 dan date2
getInvoiceByInvoiceId	GSI1	Kueri	PK=invoiceId	SK=invoiceId

Pola akses	Basis table/GSI/ LSI	Operasi	Nilai kunci partisi	Nilai kunci urutan
getPaymentByInvoiceId	GSI1	Kueri	PK=invoiceId	SK=invoiceId
getShipmentDetailsByShipmentId	GSI1	Kueri	PK=shipmentId	SK=shipmentId
getShipmentByWarehouseId	GSI2	Kueri	PK=warehouseId	SK begins_with "sh#"
getProductInventoryByWarehouseId	GSI2	Kueri	PK=warehouseId	SK begins_with "p#"
getInvoiceByCustomerIdForDateRange	GSI2	Kueri	PK=customerId	SK antara i#date1 dan i#date2
getProductsByCustomerIdForDateRange	GSI2	Kueri	PK=customerId	SK antara p#date1 dan p#date2

Skema akhir toko online

Berikut adalah desain skema akhir. Untuk mengunduh desain skema ini sebagai file JSON, lihat DynamoDB Design Patterns on. GitHub

Tabel dasar

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

Menggunakan NoSQL Workbench dengan desain skema ini

Anda dapat mengimpor skema akhir ini ke NoSQL Workbench, sebuah alat visual yang menyediakan fitur pemodelan data, visualisasi data, dan pengembangan kueri untuk DynamoDB, guna mengeksplorasi dan mengedit proyek baru Anda lebih lanjut. Ikuti langkah-langkah berikut untuk memulai:

1. Unduh NoSQL Workbench. Untuk informasi selengkapnya, lihat the section called “Unduh”.
2. Unduh file skema JSON yang tercantum di atas, yang sudah dalam format model NoSQL Workbench.
3. Impor file skema JSON ke NoSQL Workbench. Untuk informasi selengkapnya, lihat the section called “Mengimpor model yang ada”.
4. Setelah mengimpor model data ke NoSQL Workbench, Anda dapat mengeditnya. Untuk informasi selengkapnya, lihat the section called “Mengedit model yang ada”.
5. Untuk memvisualisasikan model data Anda, menambahkan data sampel, atau mengimpor data sampel dari file CSV, gunakan fitur Data Visualizer di NoSQL Workbench.

Praktik terbaik untuk memodelkan data relasional di DynamoDB

Bagian ini menyediakan praktik terbaik untuk memodelkan data relasional di Amazon DynamoDB. Pertama, kami memperkenalkan konsep pemodelan data tradisional. Kemudian, kami menjelaskan keuntungan menggunakan DynamoDB dibandingkan sistem manajemen basis data relasional tradisional—bagaimana DynamoDB menghilangkan kebutuhan untuk operasi JOIN dan mengurangi overhead.

Kami kemudian menjelaskan cara menyusun tabel DynamoDB yang menskalakan secara efisien. Terakhir, kami memberikan contoh cara memodelkan data relasional di DynamoDB.

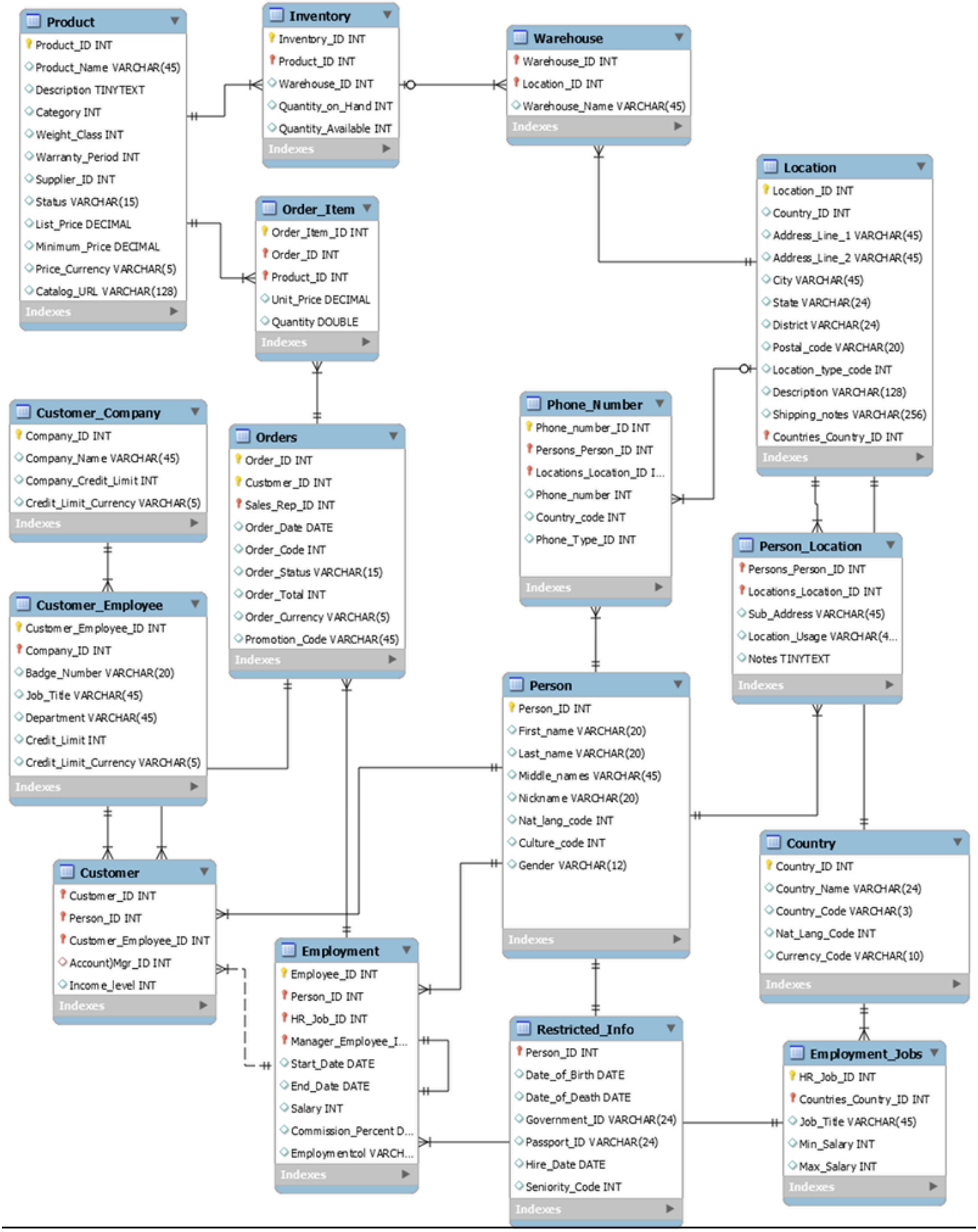
Topik

- Model basis data relasional traditional
- Cara DynamoDB menghilangkan kebutuhan akan operasi JOIN
- Bagaimana transaksi DynamoDB menghilangkan overhead ke proses tulis
- Langkah pertama untuk memodelkan data relasional di DynamoDB
- Contoh memodelkan data relasional di DynamoDB

Model basis data relasional traditional

Sistem manajemen basis data relasional tradisional (RDBMS) menyimpan data dalam struktur relasional yang dinormalisasi. Tujuan dari model data relasional adalah untuk mengurangi duplikasi data (melalui normalisasi) untuk mendukung integritas referensial dan mengurangi anomali data.

Skema berikut adalah contoh dari model data relasional untuk penerapan urutan-entri generik. Penerapan ini mendukung skema sumber daya manusia yang mendukung sistem pendukung operasional dan bisnis dari produsen teoretis.


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

Sebagai layanan basis data non-relasional, DynamoDB menawarkan banyak keunggulan dibandingkan sistem manajemen basis data relasional tradisional.

Cara DynamoDB menghilangkan kebutuhan akan operasi JOIN

Sebuah RDBMS menggunakan bahasa kueri terstruktur (SQL) untuk mengembalikan data ke aplikasi. Karena normalisasi model data, kueri semacam ini biasanya memerlukan penggunaan operator `JOIN` untuk menggabungkan data dari satu atau beberapa tabel.

Misalnya, untuk menghasilkan daftar item pesanan pembelian yang diurutkan berdasarkan jumlah stok di semua gudang yang dapat mengirimkan setiap item, Anda bisa mengeluarkan kueri SQL berikut terhadap skema sebelumnya.

```
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
```

Kueri SQL semacam ini dapat menyediakan API fleksibel untuk mengakses data, tetapi kueri tersebut membutuhkan pemrosesan dalam jumlah besar. Setiap gabungan dalam kueri meningkatkan kompleksitas runtime kueri karena data untuk setiap tabel harus ditahapkan dan kemudian dikumpulkan untuk mengembalikan set hasil.

Faktor lain yang dapat memengaruhi durasi yang dibutuhkan untuk menjalankan kueri adalah ukuran tabel dan apakah kolom yang digabungkan memiliki indeks. Kueri sebelumnya memulai kueri kompleks di beberapa tabel, lalu mengurutkan set hasilnya.

Menghilangkan kebutuhan untuk `JOINs` adalah inti dari pemodelan data NoSQL. Inilah sebabnya kami membangun DynamoDB untuk mendukung Amazon.com, dan mengapa DynamoDB dapat memberikan performa yang konsisten di semua skala. Mengingat kompleksitas runtime kueri SQL dan `JOINs`, performa RBDMS dalam skala besar tidak konstan. Hal ini menyebabkan masalah performa seiring berkembangnya aplikasi pelanggan.

Meskipun normalisasi data mengurangi jumlah data yang disimpan ke disk, sumber daya yang paling terkendala sehingga berdampak performa sering kali adalah waktu CPU dan latensi jaringan.

DynamoDB dibangun untuk meminimalkan kendala tersebut dengan menghilangkan `JOINs` (dan mendorong denormalisasi data) dan mengoptimalkan arsitektur basis data untuk sepenuhnya menjawab kueri aplikasi dengan satu permintaan ke item. Kualitas ini memungkinkan DynamoDB

untuk menghadirkan performa milidetik satu digit di semua skala. Ini karena kompleksitas runtime untuk operasi DynamoDB konstan, terlepas dari ukuran data, untuk pola akses umum.

Bagaimana transaksi DynamoDB menghilangkan overhead ke proses tulis

Faktor lain yang dapat memperlambat RDBMS adalah penggunaan transaksi untuk menulis ke skema yang dinormalisasi. Seperti yang ditunjukkan dalam contoh, struktur data relasional yang digunakan oleh sebagian besar aplikasi pemrosesan transaksi online (OLTP) harus diuraikan dan didistribusikan ke beberapa tabel logis jika disimpan di RDBMS.

Oleh karena itu, kerangka transaksi yang sesuai dengan ACID diperlukan untuk menghindari kondisi balapan dan masalah integritas data yang dapat terjadi jika aplikasi mencoba membaca objek yang sedang dalam proses penulisan. Kerangka transaksi semacam itu, jika digabungkan dengan skema relasional, dapat menambah overhead yang signifikan untuk proses tulis.

Implementasi transaksi di DynamoDB dapat mencegah masalah penskalaan umum yang dialami pada RDBMS. DynamoDB melakukannya dengan menerbitkan transaksi sebagai panggilan API tunggal dan membatasi jumlah item yang dapat diakses dalam transaksi tunggal tersebut. Transaksi yang berjalan lama dapat menyebabkan masalah operasional karena mengunci data dalam jangka waktu yang lama, atau terus-menerus, karena transaksi tidak pernah ditutup.

Untuk mencegah masalah demikian di DynamoDB, transaksi diimplementasikan dengan dua operasi API yang berbeda: `TransactWriteItems` dan `TransactGetItems`. Operasi API ini tidak memiliki semantik awal dan akhir yang lazim di RDBMS. Selanjutnya, DynamoDB memiliki batas akses 100 item dalam transaksi untuk mencegah transaksi yang berjalan lama. Untuk mempelajari selengkapnya tentang transaksi DynamoDB, lihat Bekerja dengan transaksi.

Dengan demikian, ketika bisnis Anda memerlukan respons latensi rendah terhadap kueri dengan lalu lintas tinggi, memanfaatkan sistem NoSQL umumnya masuk akal secara teknis dan ekonomis. Amazon DynamoDB membantu memecahkan masalah yang membatasi skalabilitas sistem relasional dengan menghindari masalah tersebut.

Performa RDBMS biasanya tidak menskalakan dengan baik karena alasan berikut:

- Penggabungan yang mahal digunakan untuk mengumpulkan kembali tampilan yang diperlukan dari hasil kueri.
- Sistem tersebut menormalkan data dan menyimpannya di beberapa tabel yang memerlukan sejumlah kueri untuk menulis ke disk.
- Ini umumnya menimbulkan biaya performa sistem transaksi yang sesuai dengan ACID.

DynamoDB menskalakan dengan baik karena alasan berikut:

- Fleksibilitas skema memungkinkan DynamoDB menyimpan data hierarkis kompleks dalam satu item.
- Desain kunci komposit memungkinkan DynamoDB menyimpan item terkait secara berdekatan di tabel yang sama.
- Transaksi dilakukan dalam satu operasi. Batas jumlah item yang dapat diakses adalah 100, untuk menghindari operasi yang berjalan lama.

Kueri terhadap penyimpanan data menjadi jauh lebih sederhana, sering kali dalam bentuk berikut:

```
SELECT * FROM Table_X WHERE Attribute_Y = "somevalue"
```

DynamoDB dapat memberikan data yang diminta dengan lebih efisien dibandingkan RDBMS dalam contoh sebelumnya.

Langkah pertama untuk memodelkan data relasional di DynamoDB

> **Important**
>
> Desain NoSQL membutuhkan pola pikir yang berbeda dari desain RDBMS. Untuk RDBMS, Anda dapat membuat model data yang dinormalisasi tanpa memikirkan pola akses. Anda kemudian dapat memperluasnya nanti ketika ada pertanyaan dan persyaratan kueri baru. Sebaliknya, di Amazon DynamoDB, Anda tidak boleh mulai merancang skema sampai Anda mengetahui pertanyaan-pertanyaan yang perlu dijawab. Memahami masalah bisnis dan kasus penggunaan aplikasi di awal sangatlah penting.

Untuk mulai merancang tabel DynamoDB yang akan menskalakan secara efisien, Anda harus melakukan beberapa langkah pertama untuk mengidentifikasi pola akses yang diperlukan oleh operasi dan sistem dukungan bisnis (OSS/BSS) yang memerlukan dukungan:

- Untuk aplikasi baru, tinjau cerita pengguna tentang aktivitas dan sasaran. Dokumentasikan berbagai kasus penggunaan yang Anda identifikasi, dan analisis pola akses yang mereka butuhkan.
- Untuk aplikasi yang sudah ada, analisis log kueri untuk mengetahui bagaimana orang-orang saat ini menggunakan sistem tersebut dan apa pola akses utamanya.

Setelah menyelesaikan proses ini, Anda akan mendapatkan daftar seperti berikut.

	Most Common/Import Access Patterns in Our Organization
1	Look up employee details by employee ID
2	Query employee details by employee name
3	Find an employee's phone number(s)
4	Find a customer's phone number(s)
5	Get orders for a given customer within a given date range
6	Show all open orders within a given date range across all customers
7	See all employees hired recently
8	Find all employees working in a given warehouse
9	Get all items on order for a given product
10	Get current inventories for a given product at all warehouses
11	Get customers by account representative
12	Get orders by account representative and date
13	Get all employees with a given job title
14	Get inventory by product and warehouse
15	Get total product inventory
16	Get account representatives ranked by order total and sales period

Dalam aplikasi yang sebenarnya, daftar Anda mungkin lebih panjang. Namun, kumpulan ini menunjukkan berbagai kompleksitas pola kueri yang mungkin Anda temukan di lingkungan produksi.

Pendekatan umum untuk desain skema DynamoDB adalah mengidentifikasi entitas lapisan aplikasi dan menggunakan de-normalisasi dan agregasi kunci komposit untuk mengurangi kompleksitas kueri.

Dalam DynamoDB, ini berarti menggunakan kunci urutan komposit, indeks sekunder global dengan muatan berlebih, tabel/indeks yang dipartisi, dan pola desain lainnya. Anda dapat menggunakan elemen-elemen ini untuk mengatur data sehingga aplikasi dapat mengambil apa pun yang dibutuhkannya untuk pola akses tertentu menggunakan satu kueri pada tabel atau indeks. Pola utama yang dapat Anda gunakan untuk memodelkan skema yang dinormalisasi yang ditampilkan dalam Pemodelan relasional adalah pola daftar kedekatan. Pola lain yang digunakan dalam desain ini dapat meliputi pembagian tulis indeks sekunder global, muatan berlebih indeks sekunder global, kunci komposit, dan agregasi yang terwujud.

> **Important**
>
> Secara umum, Anda harus mempertahankan tabel sesedikit mungkin dalam aplikasi DynamoDB. Pengecualian mencakup kasus yang melibatkan data deret waktu bervolume tinggi, atau set data yang memiliki pola akses yang sangat berbeda. Tabel tunggal dengan indeks terbalik biasanya dapat mengaktifkan kueri sederhana untuk membuat dan mengambil struktur data hierarki kompleks yang diperlukan oleh aplikasi Anda.

Untuk menggunakan NoSQL Workbench untuk DynamoDB guna membantu memvisualisasikan desain kunci partisi Anda, lihat Membangun Model Data dengan NoSQL Workbench.

Contoh memodelkan data relasional di DynamoDB

Contoh ini menjelaskan cara memodelkan data relasional di Amazon DynamoDB. Desain tabel DynamoDB sesuai dengan skema entri urutan relasional yang ditampilkan dalam Pemodelan relasional. Desain ini mengikuti Pola desain daftar kedekatan, yang merupakan cara umum untuk merepresentasikan struktur data relasional di DynamoDB.

Pola desain mengharuskan Anda untuk menentukan set jenis entitas yang biasanya berkorelasi dengan berbagai tabel dalam skema relasional. Item entitas kemudian ditambahkan ke tabel menggunakan kunci primer gabungan (partisi dan urutan). Kunci partisi pada item entitas ini adalah atribut yang mengidentifikasi item secara unik dan disebut secara umum pada semua item sebagai PK. Atribut kunci urutan berisi nilai atribut yang dapat Anda gunakan untuk indeks terbalik atau indeks sekunder global. Hal ini umumnya disebut sebagai SK.

Anda menentukan entitas berikut, yang mendukung skema entri urutan relasional.

1. HR-Employee - PK: EmployeeID, SK: Nama Karyawan
2. HR-Region - PK: RegionID, SK: Nama Wilayah
3. HR-Negara - PK: CountryId, SK: Nama Negara
4. HR-Location - PK: LocationID, SK: Nama Negara
5. HR-Job - PK: JobID, SK: Jabatan
6. Departemen SDM - PK: Departmentid, SK: DepartmentName
7. Pelanggan OE - PK: CustomerID, SK, ID AccountRep
8. OE-Order - PK OrderID, SK: CustomerID
9. OE-Product - PK: ProductID, SK: Nama Produk
10. OE-Warehouse - PK: WarehouseID, SK: Nama Wilayah

Setelah menambahkan item entitas ini ke tabel, Anda dapat menentukan hubungan di antara item entitas tersebut dengan menambahkan item edge ke partisi item entitas. Tabel berikut menunjukkan langkah ini.

Dalam contoh ini, partisi `Employee`, `Order`, dan `Product Entity` pada tabel memiliki item edge lain yang berisi penunjuk ke item entitas lain pada tabel. Selanjutnya, tentukan beberapa indeks

sekunder global (GSIs) untuk mendukung semua pola akses yang ditentukan sebelumnya. Tidak semua item entitas menggunakan jenis nilai yang sama untuk atribut kunci primer atau kunci urutan. Yang diperlukan hanyalah atribut primer utama dan kunci urutan untuk dimasukkan ke dalam tabel.

Fakta bahwa beberapa entitas ini menggunakan nama yang tepat dan yang lain menggunakan entitas lain IDs sebagai nilai kunci pengurutan memungkinkan indeks sekunder global yang sama untuk mendukung beberapa jenis kueri. Teknik ini disebut muatan berlebih GSI. Teknik ini secara efektif menghilangkan batas default 20 indeks sekunder global untuk tabel yang berisi beberapa jenis item. Ini ditunjukkan dalam diagram berikut sebagai GSI 1.

GSI 2 dirancang untuk mendukung pola akses aplikasi yang cukup umum, yaitu untuk mengumpulkan semua item di tabel yang memiliki status tertentu. Untuk tabel besar dengan distribusi item yang tidak merata di seluruh status yang tersedia, pola akses ini dapat menghasilkan hot key, kecuali jika item didistribusikan ke lebih dari satu partisi logis yang dapat dikueri secara paralel. Pola desain ini disebut `write sharding`.

Untuk melakukannya pada GSI 2, aplikasi menambahkan atribut kunci primer GSI 2 ke setiap item Pesanan. Aplikasi mengisinya dengan nomor acak dalam kisaran 0-N, dengan N secara umum dapat dihitung menggunakan rumus berikut, kecuali ada alasan khusus untuk melakukan sebaliknya.

```
ItemsPerRCU = 4KB / AvgItemSize

PartitionMaxReadRate = 3K * ItemsPerRCU

N = MaxRequiredIO / PartitionMaxReadRate
```

Sebagai contoh, misalkan Anda mengharapkan hal berikut:

- Hingga 2 juta pesanan akan berada dalam sistem, yang berkembang hingga 3 juta dalam 5 tahun.
- Hingga 20 persen dari pesanan ini akan berada dalam status OPEN pada waktu tertentu.
- Catatan pesanan rata-rata sekitar 100 byte, dengan tiga `OrderItem` catatan di partisi pesanan yang masing-masing sekitar 50 byte, memberi Anda ukuran entitas pesanan rata-rata 250 byte.

Untuk tabel tersebut, penghitungan faktor N akan seperti berikut ini.

```
ItemsPerRCU = 4KB / 250B = 16

PartitionMaxReadRate = 3K * 16 = 48K
```

```
N = (0.2 * 3M) / 48K = 13
```

Dalam hal ini, Anda perlu mendistribusikan semua pesanan ke setidaknya 13 partisi logis di GSI 2 untuk memastikan bahwa pembacaan semua item `Order` dengan status `OPEN` tidak menyebabkan partisi panas pada lapisan penyimpanan fisik. Memasukkan jumlah ini untuk memungkinkan anomali dalam set data adalah praktik yang baik. Jadi, model yang menggunakan `N = 15` mungkin baik-baik saja. Seperti yang disebutkan sebelumnya, Anda dapat melakukan ini dengan menambahkan nilai 0–N acak ke atribut GSI 2 PK dari setiap data `Order` dan `OrderItem` yang dimasukkan ke tabel.

Perincian ini mengasumsikan bahwa pola akses yang memerlukan pengumpulan semua faktur `OPEN` relatif jarang terjadi sehingga Anda dapat menggunakan kapasitas lonjakan untuk memenuhi permintaan. Anda dapat mengkueri indeks sekunder global berikut menggunakan syarat Kunci Urutan `State` dan `Date Range` untuk menghasilkan sebagian atau semua `Orders` dalam status tertentu sesuai kebutuhan.

Dalam contoh ini, item didistribusikan secara acak ke 15 partisi logis. Struktur ini berfungsi karena pola akses memerlukan sejumlah besar item untuk diambil. Oleh karena itu, kecil kemungkinannya bahwa salah satu dari 15 thread akan mengembalikan set hasil kosong yang berpotensi mewakili kapasitas yang terbuang. Kueri selalu menggunakan 1 unit kapasitas baca (RCU) atau 1 unit kapasitas tulis (WCU), meskipun tidak ada yang dikembalikan atau tidak ada data yang ditulis.

Jika pola akses memerlukan kueri kecepatan tinggi pada indeks sekunder global ini yang mengembalikan set hasil yang jarang, sebaiknya gunakan algoritma hash untuk mendistribusikan item daripada pola acak. Dalam hal ini, Anda dapat memilih atribut yang diketahui ketika kueri dijalankan pada saat runtime dan meng-hash atribut tersebut ke dalam ruang kunci 0–14 saat item dimasukkan. Kemudian, item tersebut dapat dibaca secara efisien dari indeks sekunder global.

Terakhir, Anda dapat meninjau kembali pola akses yang ditetapkan sebelumnya. Berikut ini adalah daftar pola akses dan ketentuan kueri yang akan Anda gunakan dengan versi DynamoDB baru pada aplikasi untuk mengakomodasinya.

S. Tidak.	Pola akses	Ketentuan kueri
1	Cari Detail Karyawan berdasarkan ID Karyawan	Kunci Primer pada tabel, ID="HR-EMPLOYEE"
2	Kueri Detail Karyawan berdasarkan Nama Karyawan	Gunakan GSI-1, PK="Employee Name"

S. Tidak.	Pola akses	Ketentuan kueri
3	Dapatkan detail pekerjaan karyawan saat ini saja	Kunci Primer pada tabel, PK=HR-EMPLOYEE-1, SK dimulai dengan "JH"
4	Dapatkan Pesanan pelanggan untuk rentang tanggal	Gunakan GSI-1, PK=, SK="STATUS-DATE”CUSTOMER1, untuk masing-masing StatusCode
5	Tampilkan semua Pesanan dalam status OPEN untuk rentang tanggal di seluruh pelanggan	Gunakan GSI-2, PK=query secara paralel untuk rentang [0..N], SK antara OPEN-Date1 dan OPEN-Date2
6	Semua Karyawan yang baru direkrut	Gunakan GSI-1, PK="HR-CONFIDENTIAL', SK > date1
7	Temukan semua Karyawan di Gudang tertentu	Gunakan GSI-1, PK= WAREHOUSE1
8	Dapatkan semua Orderitems untuk suatu Produk termasuk inventaris lokasi gudang	Gunakan GSI-1, PK= PRODUCT1
9	Dapatkan pelanggan dengan Perwakilan Akun	Gunakan GSI-1, PK=ACCOUNT-REP
10	Dapatkan pesanan berdasarkan Perwakilan Akun dan tanggal	Gunakan GSI-1, PK=ACCOUNT-REP, SK="STATUS-DATE”, untuk masing-masing StatusCode
11	Dapatkan semua karyawan dengan Jabatan tertentu	Gunakan GSI-1, PK=JOBTITLE

S. Tidak.	Pola akses	Ketentuan kueri
12	Dapatkan inventaris berdasarkan Produk dan Gudang	Kunci Utama di atas meja, PK=OE-, SK= PRODUCT1 PRODUCT1
13	Dapatkan total inventaris produk	Kunci Utama di atas meja, PK=OE-, SK= PRODUCT1 PRODUCT1
14	Dapatkan Perwakilan Akun diberi peringkat berdasarkan Total Pesanan dan Periode Penjualan	Gunakan GSI-1, PK=YYYY-Q1, =Salah scanIndexForward

Migrasi ke DynamoDB dari basis data relasional

Migrasi basis data relasional ke DynamoDB memerlukan perencanaan yang matang untuk memastikan hasil yang sukses. Panduan ini akan membantu Anda memahami cara kerja proses ini, alat yang Anda miliki, lalu cara mengevaluasi strategi migrasi potensial dan memilih salah satu yang sesuai dengan kebutuhan Anda.

Topik

- Alasan untuk bermigrasi ke DynamoDB
- Pertimbangan saat memigrasikan basis data relasional ke DynamoDB
- Memahami cara kerja migrasi ke DynamoDB
- Alat untuk membantu migrasi ke DynamoDB
- Memilih strategi yang tepat untuk migrasi ke DynamoDB
- Melakukan migrasi offline ke DynamoDB
- Melakukan migrasi hibrid ke DynamoDB
- Melakukan migrasi online ke DynamoDB dengan memigrasikan setiap tabel 1:1
- Lakukan migrasi online ke DynamoDB menggunakan tabel penahapan khusus

Alasan untuk bermigrasi ke DynamoDB

Migrasi ke Amazon DynamoDB menghadirkan berbagai manfaat menarik bagi bisnis dan organisasi. Berikut adalah beberapa keuntungan utama yang menjadikan DynamoDB sebagai pilihan menarik untuk migrasi basis data:

- Skalabilitas: DynamoDB dirancang untuk menangani beban kerja yang besar dan menskalakan dengan mulus untuk mengakomodasi lalu lintas dan volume data yang terus bertambah. Dengan DynamoDB, Anda dapat dengan mudah meningkatkan atau menurunkan basis data berdasarkan permintaan sehingga memastikan aplikasi Anda dapat menangani lonjakan lalu lintas secara tiba-tiba tanpa mengorbankan performa.
- Performa: DynamoDB menawarkan akses data latensi rendah sehingga memungkinkan aplikasi untuk mengambil dan memproses data dengan kecepatan luar biasa. Arsitektur terdistribusinya memastikan operasi baca dan tulis didistribusikan di beberapa simpul sehingga memberikan waktu respons milidetik satu digit yang konsisten bahkan pada tingkat permintaan tinggi.

- Terkelola penuh: DynamoDB adalah layanan terkelola penuh yang disediakan oleh AWS. Ini berarti AWS menangani aspek operasional manajemen basis data, termasuk penyediaan, konfigurasi, penambalan, pencadangan, dan penskalaan. Ini memungkinkan Anda untuk lebih fokus pada pengembangan aplikasi dan tidak terlalu fokus pada tugas administrasi basis data.
- Arsitektur nirserver: DynamoDB mendukung model nirserver, yang dikenal sebagai DynamoDB sesuai permintaan, yaitu Anda hanya membayar untuk permintaan baca dan tulis aktual yang dibuat aplikasi Anda tanpa memerlukan penyediaan kapasitas di muka. pay-per-requestModel ini menawarkan efisiensi biaya dan overhead operasional minimal, karena Anda hanya membayar sumber daya yang Anda konsumsi tanpa perlu menyediakan dan memantau kapasitas.
- Fleksibilitas NoSQL: Tidak seperti basis data relasional tradisional, DynamoDB mengikuti model data NoSQL, sehingga memberikan fleksibilitas dalam desain skema. Dengan DynamoDB, Anda dapat menyimpan data terstruktur, semi-terstruktur, dan tidak terstruktur, sehingga cocok untuk menangani jenis data yang beragam dan berkembang. Fleksibilitas ini memungkinkan siklus pengembangan yang lebih cepat dan adaptasi yang lebih mudah terhadap perubahan kebutuhan bisnis.
- Ketersediaan dan daya tahan tinggi: DynamoDB mereplikasi data di beberapa zona ketersediaan dalam suatu Wilayah, sehingga memastikan ketersediaan tinggi dan daya tahan data. Replikasi, failover, dan pemulihan direplikasi secara otomatis, sehingga meminimalkan risiko kehilangan data atau gangguan layanan. DynamoDB menyediakan ketersediaan SLA hingga 99,999%.
- Keamanan dan kepatuhan: DynamoDB terintegrasi dengan AWS Identity and Access Management untuk kontrol akses terperinci. Ini menyediakan enkripsi diam dan bergerak, sehingga memastikan keamanan data Anda. DynamoDB juga mematuhi berbagai standar kepatuhan, termasuk HIPAA, PCI DSS, dan GDPR, sehingga memungkinkan Anda memenuhi persyaratan peraturan.
- Integrasi dengan AWS Ekosistem: Sebagai bagian dari AWS ekosistem, DynamoDB terintegrasi secara mulus dengan layanan AWS lain, seperti,, dan. AWS Lambda AWS CloudFormation AWS AppSync Integrasi ini memungkinkan Anda membangun arsitektur nirserver, memanfaatkan infrastruktur sebagai kode, dan membuat aplikasi berbasis data waktu nyata.

Pertimbangan saat memigrasikan basis data relasional ke DynamoDB

Sistem basis data relasional dan basis data NoSQL memiliki keunggulan dan kelemahan yang berbeda. Perbedaan ini membuat desain basis data menjadi berbeda di antara kedua sistem.

Jenis tugas	Basis data relasional	Basis data NoSQL
Mengkueri basis data	Di basis data relasional, data dapat dikueri secara fleksibel , tetapi kueri relatif mahal dan tidak dapat diskalakan dengan baik dalam situasi lalu lintas tinggi (lihat Langkah pertama untuk memodelkan data relasional di DynamoDB). Aplikasi basis data relasiona l dapat menerapkan logika bisnis dalam prosedur tersimpan, subkueri SQL, kueri pembaruan massal, dan kueri agregasi.	Dalam basis data NoSQL seperti DynamoDB, data dapat dikueri secara efisien dalam sejumlah cara terbatas, di luar itu kueri bisa jadi mahal dan lambat. Penulisan ke DynamoDB merupakan singleton. Logika bisnis aplikasi yang sebelumnya dijalankan dalam prosedur tersimpan harus difaktork an ulang untuk berjalan di luar DynamoDB dalam kode khusus yang berjalan pada host seperti Amazon Amazon atau. EC2 AWS Lambda
Merancang basis data	Anda merancang untuk fleksibilitas tanpa perlu mengkhawatirkan detail penerapan atau performa. Optimasi kueri umumnya tidak memengaruhi desain skema, tetapi normalisasi itu penting.	Anda merancang skema secara spesifik untuk membuat kueri yang paling umum dan penting seefisien dan seterjangkau mungkin. Struktur data Anda disesuaik an dengan kebutuhan spesifik kasus penggunaan bisnis Anda.

Merancang untuk basis data NoSQL membutuhkan pola pikir yang berbeda dari merancang untuk sistem manajemen basis data relasional (RDBMS). Untuk RDBMS, Anda dapat membuat model data yang dinormalisasi tanpa memikirkan pola akses. Anda kemudian dapat memperluasnya nanti ketika ada pertanyaan dan persyaratan kueri baru. Anda dapat mengatur setiap jenis data ke dalam tabelnya sendiri.

Dengan desain NoSQL, Anda dapat mendesain skema Anda untuk DynamoDB ketika Anda mengetahui pertanyaan yang perlu dijawab. Memahami masalah bisnis dan pola baca dan tulis aplikasi sangatlah penting. Anda juga harus berusaha mempertahankan tabel sesedikit mungkin dalam aplikasi DynamoDB. Memiliki lebih sedikit tabel membuat segala sesuatunya lebih terukur, memerlukan lebih sedikit manajemen izin, dan mengurangi overhead untuk aplikasi DynamoDB Anda. Hal ini juga dapat membantu menjaga biaya pencadangan tetap rendah secara keseluruhan.

Tugas memodelkan data relasional untuk DynamoDB dan membangun versi baru aplikasi front-end merupakan topik terpisah. Panduan ini mengasumsikan Anda memiliki versi baru aplikasi yang dibuat untuk menggunakan DynamoDB, tetapi Anda masih perlu menentukan cara terbaik untuk memigrasikan dan menyinkronkan data historis selama cutover.

Pertimbangan Ukuran

Ukuran maksimum setiap item (baris) yang Anda simpan dalam tabel DynamoDB adalah 400KB. Untuk informasi selengkapnya, lihat the section called “Kuota”. Ukuran item ditentukan oleh ukuran total semua nama atribut dan nilai atribut dalam item. Untuk informasi selengkapnya, lihat the section called “Ukuran dan format item”.

Jika aplikasi Anda perlu menyimpan lebih banyak data dalam item daripada batas ukuran DynamoDB yang diizinkan, pisahkan item menjadi koleksi item, kompres data item, atau simpan item sebagai objek di Amazon Simple Storage Service (Amazon S3) Simple Storage Service (Amazon S3) sambil menyimpan pengidentifikasi objek Amazon S3 di item DynamoDB Anda. Lihat the section called “Item besar”. Biaya untuk memperbarui item didasarkan pada ukuran penuh item. Untuk beban kerja yang memerlukan pembaruan yang sering ke item yang ada, memiliki item kecil satu atau dua KB akan lebih murah untuk memperbarui daripada item yang lebih besar. Lihat the section called “Bekerja dengan Kumpulan Item” untuk informasi lebih lanjut tentang koleksi item.

Saat memilih partisi dan mengurutkan atribut kunci, pengaturan tabel lainnya, ukuran dan struktur item, dan apakah akan membuat indeks sekunder, pastikan untuk meninjau dokumentasi Pemodelan DynamoDB serta panduan untuk. the section called “Optimasi biaya” Pastikan untuk menguji rencana migrasi Anda sehingga solusi DynamoDB Anda hemat biaya dan sesuai dengan fitur dan batasan DynamoDB.

Memahami cara kerja migrasi ke DynamoDB

Sebelum meninjau alat migrasi yang tersedia bagi kami, pertimbangkan cara penulisan diproses oleh DynamoDB.

Operasi tulis default dan paling umum adalah operasi API `PutItem` tunggal. Anda dapat melakukan operasi `PutItem` dalam satu putaran untuk memproses kumpulan data. DynamoDB mendukung koneksi bersamaan yang hampir tidak terbatas, jadi dengan asumsi Anda dapat mengonfigurasi dan menjalankan rutinitas pemuatan multi-utas besar-besaran MapReduce seperti atau Spark, kecepatan penulisan hanya dibatasi oleh kapasitas tabel target (yang juga umumnya tidak terbatas).

Saat memuat data ke DynamoDB, penting untuk memahami kecepatan tulis loader Anda. Jika item (baris) yang Anda muat berukuran 1KB atau kurang, kecepatan ini hanyalah jumlah item per detik. Tabel target kemudian dapat disediakan dengan WCU (unit kapasitas tulis) yang memadai untuk menangani tingkat ini. Jika loader Anda melebihi kapasitas yang disediakan dalam detik tertentu, permintaan tambahan dapat mengalami throttling atau ditolak sama sekali. Anda dapat memeriksa throttle di CloudWatch bagan yang ditemukan di tab pemantauan konsol DynamoDB.

Operasi kedua yang dapat dilakukan adalah dengan API terkait yang disebut `BatchWriteItem`. `BatchWriteItem` memungkinkan Anda untuk menggabungkan hingga 25 permintaan tulis ke dalam satu panggilan API. Permintaan ini diterima oleh layanan dan diproses sebagai `PutItem` permintaan terpisah ke tabel. Saat ini, saat memilih`BatchWriteItem`, Anda tidak akan mendapatkan keuntungan dari percobaan ulang otomatis yang disertakan dengan AWS SDK saat melakukan panggilan tunggal dengan. `PutItem` Jadi, jika ada kesalahan (seperti pengecualian throttling), Anda harus mencari daftar penulisan yang gagal pada panggilan respons ke `BatchWriteItem`. Untuk informasi selengkapnya tentang penanganan peringatan pelambatan jika ini terdeteksi di bagan CloudWatch pelambatan, lihat. the section called “Masalah pelambatan”

Jenis impor data ketiga dimungkinkan dengan fitur impor DynamoDB dari S3. Fitur ini memungkinkan Anda untuk menampilkan set data besar di Amazon S3 dan meminta DynamoDB untuk mengimpor data secara otomatis ke tabel baru. Impor ini tidak instan dan memerlukan waktu yang sebanding dengan ukuran set data. Namun, ini nyaman karena tidak memerlukan platform ETL atau kode DynamoDB khusus. DynamoDB memuat data ke dalam tabel baru yang dibuat oleh impor. Saat ini, itu tidak memungkinkan Anda untuk memuat data ke dalam tabel yang ada. DynamoDB mengimpor data apa adanya, tanpa transformasi. Mirip dengan`PutItem`, ini memerlukan proses hulu dan menulis data dalam format yang Anda pilih ke bucket Amazon S3.

Alat untuk membantu migrasi ke DynamoDB

Ada beberapa alat migrasi dan ETL umum yang dapat Anda gunakan untuk memigrasikan data ke DynamoDB.

Amazon menyediakan sejumlah alat data yang dapat digunakan dalam migrasi, termasuk AWS Database Migration Service (DMS), Amazon EMR AWS Glue, dan Amazon Managed Streaming for Apache Kafka. Semua alat ini dapat digunakan untuk melakukan migrasi downtime, dan mereka dapat memanfaatkan fitur Change Data Capture (CDC) database relasional untuk mendukung migrasi online. Saat memilih alat, akan membantu untuk mempertimbangkan keahlian dan pengalaman yang dimiliki organisasi Anda dengan setiap alat bersama dengan fitur, kinerja, dan biaya masing-masing alat.

Banyak pelanggan memilih untuk menulis skrip migrasi dan tugas mereka sendiri untuk membangun transformasi data khusus untuk proses migrasi. Jika Anda berencana untuk mengoperasikan tabel DynamoDB volume tinggi dengan lalu lintas tulis yang padat atau pekerjaan beban massal besar biasa, Anda mungkin ingin menulis kode migrasi sendiri untuk lebih terbiasa dengan perilaku DynamoDB di bawah lalu lintas tulis yang padat. Skenario seperti penanganan throttle dan penyediaan tabel yang efisien dapat dialami di awal proyek saat melakukan migrasi praktik.

Memilih strategi yang tepat untuk migrasi ke DynamoDB

Sebuah aplikasi basis data relasional besar dapat menjangkau seratus atau lebih tabel dan mendukung beberapa fungsi aplikasi yang berbeda. Saat melakukan migrasi besar, pertimbangkan untuk memecah aplikasi Anda menjadi komponen yang lebih kecil atau layanan mikro, dan memigrasikan sekumpulan tabel kecil dalam satu waktu. Anda kemudian dapat memigrasikan komponen lainnya ke DynamoDB secara bertahap.

Saat memilih strategi migrasi, berbagai faktor dapat mengarahkan Anda ke satu solusi atau lainnya. Kami dapat menyajikan opsi-opsi ini dalam pohon keputusan untuk menyederhanakan opsi yang tersedia bagi kami berdasarkan kebutuhan dan sumber daya yang tersedia. Konsep-konsep tersebut disebutkan secara singkat di sini (tetapi akan dibahas lebih mendalam nanti dalam panduan ini):

- Migrasi offline: jika aplikasi Anda dapat mentolerir beberapa waktu henti selama migrasi, itu akan menyederhanakan proses migrasi.
- Migrasi hibrid: pendekatan ini memungkinkan uptime sebagian selama migrasi, seperti mengizinkan pembacaan tetapi tidak menulis, atau mengizinkan pembacaan dan penyisipan tetapi tidak memperbarui dan menghapus.
- Migrasi online: aplikasi yang tidak memerlukan waktu henti selama migrasi tidak mudah dimigrasi, dan dapat memerlukan perencanaan dan pengembangan khusus yang signifikan. Salah satu keputusan utama adalah memperkirakan dan menimbang biaya membangun proses migrasi kustom versus biaya untuk bisnis memiliki jendela downtime selama pemotongan.

Jika	Dan	Maka
Anda boleh menghapus aplikasi selama beberapa waktu selama jendela pemeliharaan untuk melakukan migrasi data. Ini adalah migrasi offline.		Menggunakan AWS DMS dan melakukan migrasi offline menggunakan tugas pemuatan penuh. Bentuk data sumber terlebih dahulu dengan SQL `VIEW` jika diinginkan.
Anda boleh menjalankan aplikasi dalam mode hanya-baca selama migrasi. Ini adalah migrasi hibrida.		Nonaktifkan penulisan di dalam aplikasi atau basis data sumber. Menggunakan AWS DMS dan melakukan migrasi offline menggunakan tugas pemuatan penuh.
Anda boleh menjalankan aplikasi dengan pembacaan dan sisipan catatan baru, tetapi tidak ada pembaruan atau penghapusan, selama migrasi. Ini adalah migrasi hibrida.	Anda memiliki keterampilan pengembangan aplikasi dan dapat memperbarui aplikasi relasional yang ada untuk melakukan penulisan ganda termasuk ke DynamoDB, untuk semua catatan baru	Menggunakan AWS DMS dan melakukan migrasi offline menggunakan tugas pemuatan penuh. Secara bersamaan, deploy versi aplikasi yang ada yang memungkinkan pembacaan dan melakukan penulisan ganda.
Anda memerlukan migrasi dengan waktu henti minimal. Ini adalah migrasi online.	• Anda memigrasi tabel sumber 1-untuk-1 ke DynamoDB tanpa perubahan skema besar.	Gunakan AWS DMS untuk melakukan migrasi data online. Jalankan tugas pemuatan massal diikuti dengan tugas sinkronisasi CDC.
Anda memerlukan migrasi dengan waktu henti minimal. Ini adalah migrasi online.	• Anda menggabungkan tabel sumber ke dalam tabel DynamoDB yang lebih	Buat tabel siap NoSQL dalam basis data SQL. Mengisi dan menyinkronkannya

Jika	Dan	Maka
	sedikit mengikuti skema bertumpuk atau filosofi tabel tunggal. • Anda memiliki keterampilan pengembangan basis data backend dan kapasitas cadangan pada host SQL.	menggunakan JOINs,, UNIONs, pemicu VIEWs, prosedur tersimpan.
Anda memerlukan migrasi dengan waktu henti minimal. Ini adalah migrasi online.	• Anda menggabungkan tabel sumber ke dalam tabel DynamoDB yang lebih sedikit mengikuti filosofi tabel tunggal. Sebagai contoh: • Anda tidak memiliki keterampilan pengembangan basis data backend dan kapasitas cadangan pada host SQL.	Pertimbangkan pendekatan migrasi hybrid atau offline.
Anda memerlukan migrasi dengan waktu henti minimal. Ini adalah migrasi online.	Anda boleh melewatkan migrasi data transaksi historis, atau dapat mengarsipkannya di Amazon S3 sebagai pengganti memigrasinya. Anda hanya perlu memigrasikan beberapa tabel statis kecil.	Tulis skrip atau gunakan alat ETL apa pun untuk memigrasikan tabel. Bentuk data sumber terlebih dahulu dengan SQL `VIEW` jika diinginkan.

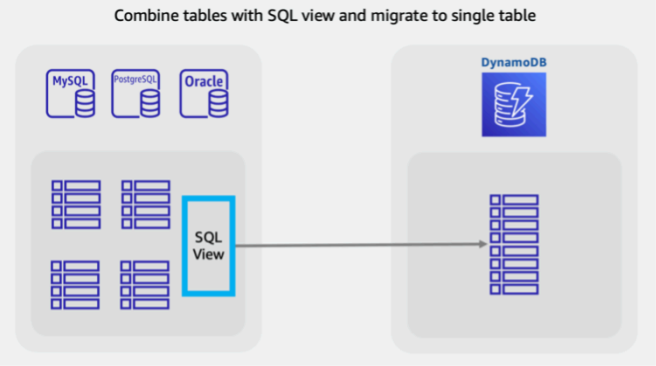
Melakukan migrasi offline ke DynamoDB

Migrasi offline cocok ketika Anda dapat mengizinkan periode waktu henti untuk melakukan migrasi. Basis data relasional biasanya membutuhkan setidaknya beberapa waktu henti setiap bulan untuk

pemeliharaan dan patching, atau mungkin membutuhkan waktu henti lebih lama untuk peningkatan perangkat keras atau peningkatan rilis utama.

Amazon S3 dapat digunakan sebagai area penahapan selama migrasi. Data yang disimpan dalam CSV (nilai dipisahkan koma) atau format DynamoDB JSON dapat diimpor ke tabel DynamoDB baru secara otomatis menggunakan fitur impor DynamoDB dari S3.

Anda mungkin ingin menggabungkan tabel untuk memanfaatkan pola akses NoSQL yang unik (misalnya, mengubah empat tabel lama menjadi tabel DynamoDB tunggal). Permintaan dokumen nilai kunci tunggal atau kueri untuk koleksi item yang dikelompokkan sebelumnya biasanya kembali dengan latensi yang lebih baik daripada database SQL yang melakukan gabungan multi-tabel. Namun, ini membuat tugas migrasi lebih sulit. Tampilan SQL dapat melakukan pekerjaan dalam database sumber untuk menyiapkan kumpulan data tunggal yang mewakili keempat tabel dalam satu set.



Tampilan ini dapat `JOIN` tabel ke dalam bentuk denormalisasi, atau dapat menjaga entitas dinormalisasi dan menumpuk tabel menggunakan SQL. `UNION` Keputusan penting seputar pembentukan kembali data relasional tercakup dalam video ini. Untuk migrasi offline, menggunakan tampilan untuk menggabungkan tabel adalah cara terbaik untuk membentuk data untuk skema tabel tunggal DynamoDB.

Rencana

Lakukan migrasi offline menggunakan Amazon S3

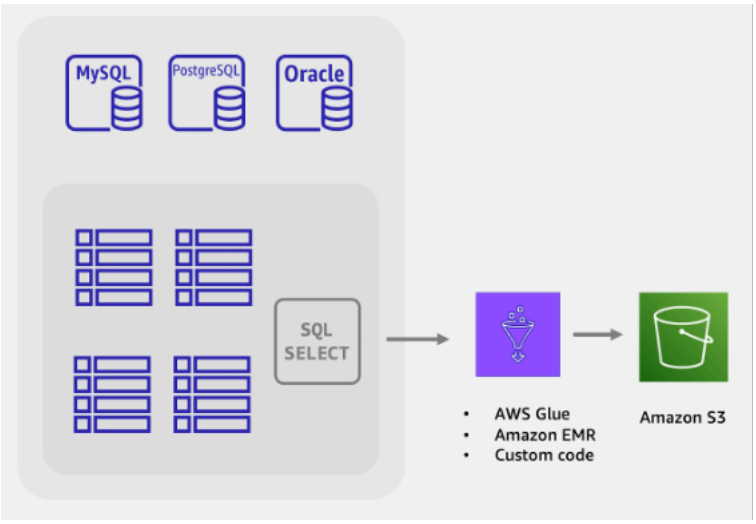
Alat

- Tugas ETL untuk mengekstrak dan mengubah data SQL serta menyimpannya dalam bucket S3 seperti:

  - AWS Database Migration Service, layanan yang dapat memuat data historis secara massal dan juga dapat memproses catatan Change Data Capture (CDC) untuk menyinkronkan tabel sumber dan target.
  - AWS Glue
  - Amazon EMR
  - Kode khusus Anda sendiri
- Fitur impor DynamoDB dari S3

Langkah-langkah migrasi offline:

1. Buat tugas ETL yang dapat menanyakan basis data SQL, mengubah data tabel menjadi DynamoDB JSON atau format CSV, dan menyimpannya ke bucket S3.



2. Fitur Impor DynamoDB dari S3 dipanggil untuk membuat tabel baru dan memuat data secara otomatis dari bucket S3 Anda.

Migrasi offline sangat sederhana dan mudah, tetapi mungkin tidak populer di kalangan pemilik aplikasi dan pengguna. Pengguna akan mendapatkan keuntungan jika aplikasi dapat memberikan tingkat layanan yang lebih rendah selama migrasi, daripada tidak memberikan layanan sama sekali.

Anda dapat menambahkan fungsionalitas untuk menonaktifkan penulisan selama migrasi offline, sambil mengizinkan pembacaan berlanjut seperti biasa. Pengguna aplikasi masih dapat menelusuri dan menanyakan data yang ada dengan aman saat data relasional sedang dimigrasikan. Jika ini yang Anda cari, lanjutkan membaca untuk mempelajari tentang migrasi hibrida.

Melakukan migrasi hibrid ke DynamoDB

Meskipun semua aplikasi basis data melakukan operasi baca dan tulis, jenis operasi tulis yang dilakukan harus dipertimbangkan saat merencanakan migrasi hibrida atau online. Penulisan basis data dapat diklasifikasikan ke dalam tiga bucket: sisipan, pembaruan, dan penghapusan. Beberapa aplikasi mungkin tidak memerlukan pemrosesan penghapusan segera. Aplikasi ini dapat menunda penghapusan ke proses pembersihan massal pada akhir bulan, misalnya. Jenis aplikasi ini bisa jadi lebih mudah untuk dimigrasi sambil memungkinkan waktu aktif parsial.

Rencana

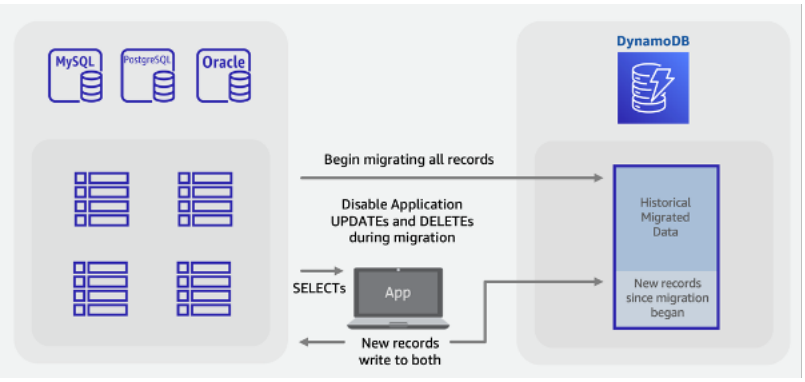
Lakukan migrasi online/offline hibrida dengan penulisan ganda aplikasi

Alat

- Tugas ETL untuk mengekstrak dan mengubah data SQL serta menyimpannya dalam bucket S3 seperti:
  - AWS DMS
  - AWS Glue
  - Amazon EMR
  - Kode khusus Anda sendiri

Langkah-langkah migrasi hibrida:

1. Buat tabel DynamoDB target. Tabel ini akan menerima data massal historis dan data langsung baru
2. Membuat versi aplikasi lama yang penghapusan dan pembaruannya dinonaktifkan saat melakukan semua penyisipan sebagai penulisan ganda ke basis data SQL dan DynamoDB
3. Mulai pekerjaan atau AWS DMS tugas ETL untuk mengisi kembali data yang ada dan menerapkan versi aplikasi baru secara bersamaan
4. Ketika pekerjaan pengurukan selesai, DynamoDB akan memiliki semua catatan yang ada dan baru dan siap untuk pemotongan aplikasi



> **Note**
>
> Pekerjaan backfill menulis langsung dari SQL ke DynamoDB. Kami tidak dapat menggunakan fitur impor S3 seperti pada contoh migrasi offline, karena fitur itu membuat tabel baru yang tidak akan hidup sampai setelah DynamoDB memuat data.

Melakukan migrasi online ke DynamoDB dengan memigrasikan setiap tabel 1:1

Banyak basis data relasional memiliki fitur yang disebut Change Data Capture (CDC), yaitu basis data memungkinkan pengguna untuk meminta daftar perubahan pada tabel yang terjadi sebelum atau setelah titik waktu tertentu. CDC menggunakan log internal untuk mengaktifkan fitur ini dan tidak memerlukan tabel untuk memiliki kolom stempel waktu untuk berfungsi.

Saat memigrasikan skema tabel SQL ke basis data NoSQL, Anda mungkin ingin menggabungkan dan membentuk kembali data Anda menjadi tabel yang lebih sedikit. Melakukannya akan memungkinkan Anda mengumpulkan data di satu tempat dan menghindari keharusan untuk menggabungkan data terkait secara manual dalam operasi baca multi-langkah. Namun, pembentukan data tabel tunggal tidak selalu diperlukan dan terkadang Anda akan memigrasikan tabel 1 per 1 ke DynamoDB. Migrasi tabel 1 per 1 ini tidak terlalu rumit karena Anda dapat memanfaatkan fitur CDC basis data sumber, menggunakan alat ETL umum yang mendukung jenis migrasi ini. Data untuk setiap baris masih dapat diubah ke dalam format baru, tetapi cakupan setiap tabel tetap sama.

Pertimbangkan untuk memigrasikan tabel SQL 1-ke-1 ke DynamoDB, dengan peringatan bahwa DynamoDB tidak mendukung gabungan sisi server. Anda harus menambahkan logika ke aplikasi Anda untuk menggabungkan data dari beberapa tabel.

Rencana

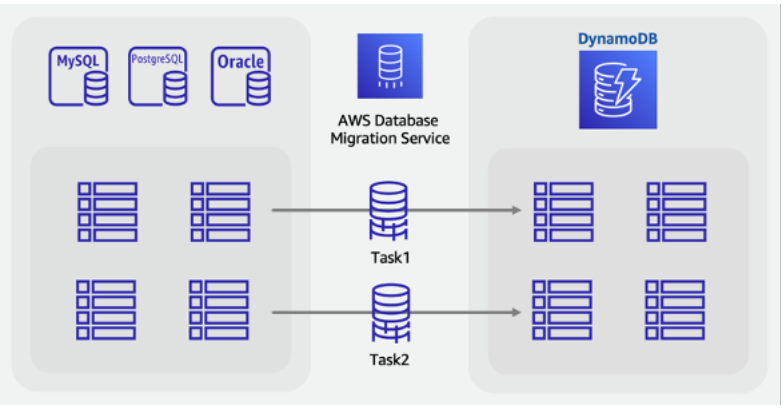
Lakukan migrasi online dari setiap tabel ke DynamoDB menggunakan AWS DMS

Alat

- AWS DMS (DMS)

Langkah-langkah migrasi online:

1. Identifikasi tabel dalam skema sumber yang akan dimigrasikan
2. Buat jumlah tabel yang sama di DynamoDB dengan struktur kunci yang sama seperti di sumber
3. Buat server replikasi AWS DMS dan konfigurasikan titik akhir sumber dan target
4. Tentukan transformasi per baris yang diperlukan (seperti kolom gabungan atau konversi tanggal ke format string ISO-8601)
5. Buat tugas migrasi setiap tabel untuk Beban Penuh dan Change Data Capture
6. Pantau tugas-tugas ini sampai fase replikasi yang sedang berlangsung dimulai
7. Pada titik ini Anda dapat melakukan audit validasi, lalu mengalihkan pengguna ke aplikasi yang membaca dan menulis ke DynamoDB



Lakukan migrasi online ke DynamoDB menggunakan tabel penahapan khusus

Seperti dalam skenario migrasi offline di atas, Anda dapat memilih untuk menggabungkan tabel untuk memanfaatkan pola akses NoSQL yang unik (misalnya, mengubah empat tabel lama menjadi satu

tabel DynamoDB tunggal). SQL `VIEW` dapat melakukan pekerjaan dalam basis data sumber untuk menyiapkan satu set data yang mewakili keempat tabel dalam satu set.

Namun, untuk migrasi online dengan data langsung yang berubah, Anda tidak dapat memanfaatkan fitur CDC karena tidak didukung untuk `VIEW` s. Jika tabel Anda menyertakan kolom stempel waktu terakhir diperbarui, dan tabel tersebut digabungkan ke dalam `VIEW`, Anda kemudian dapat membuat tugas ETL khusus yang menggunakan tabel tersebut untuk mencapai beban massal dengan sinkronisasi.

Pendekatan baru untuk tantangan ini adalah dengan menggunakan fitur SQL standar seperti tampilan, prosedur tersimpan, dan pemicu untuk membuat tabel SQL baru yang berada dalam format DynamoDB NoSQL akhir yang diinginkan.

Jika server database Anda memiliki kapasitas cadangan, Anda dapat membuat tabel pementasan tunggal ini sebelum migrasi dimulai. Anda akan melakukan ini dengan menulis prosedur tersimpan yang akan membaca dari tabel yang ada, mengubah data sesuai kebutuhan, dan menulis ke tabel pementasan baru. Anda dapat menambahkan satu set pemicu untuk mereplikasi perubahan dalam tabel ke dalam tabel pementasan secara real time. Jika pemicu tidak diperbolehkan karena kebijakan perusahaan, perubahan pada prosedur tersimpan dapat mencapai hasil yang sama. Anda akan menambahkan beberapa baris kode ke prosedur apa pun yang menulis data, untuk menambahkan perubahan yang sama ke dalam tabel penahapan.

Memiliki tabel penahapan ini yang sepenuhnya disinkronkan dengan tabel aplikasi lama akan memberi Anda titik awal yang bagus untuk migrasi langsung. Alat yang menggunakan database CDC untuk menyelesaikan migrasi langsung, seperti AWS DMS, sekarang dapat digunakan terhadap tabel ini. Keuntungan dari pendekatan ini adalah penggunaan keterampilan dan fitur SQL terkenal yang tersedia di mesin basis data relasional.

Rencana

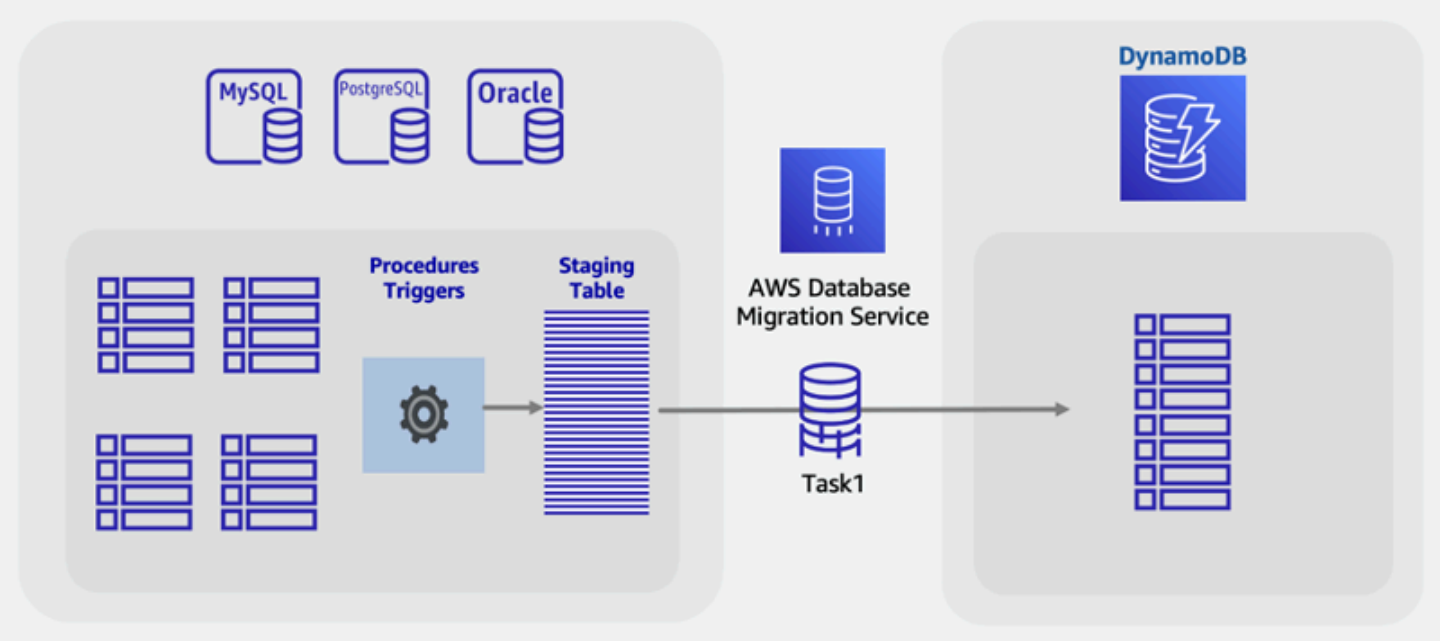
Lakukan migrasi online dengan tabel pementasan SQL menggunakan AWS DMS

Alat

- Prosedur atau pemicu tersimpan SQL khusus
- AWS DMS

Langkah-langkah migrasi online:

1. Dalam mesin basis data relasional sumber, pastikan ada beberapa kapasitas pemrosesan dan ruang disk tambahan.
2. Buat tabel penahapan baru di basis data SQL, dengan stempel waktu atau fitur CDC diaktifkan
3. Tulis dan jalankan prosedur tersimpan untuk menyalin data tabel relasional yang ada ke dalam tabel penahapan
4. Deploy pemicu atau ubah prosedur yang ada untuk penulisan ganda ke tabel penahapan baru saat melakukan penulisan normal ke tabel yang ada
5. Jalankan AWS DMS untuk memigrasi dan menyinkronkan tabel sumber ini ke tabel DynamoDB target



Panduan ini menyajikan beberapa pertimbangan dan pendekatan untuk memigrasikan data basis data relasional ke DynamoDB, dengan fokus pada meminimalkan waktu henti dan menggunakan alat dan teknik basis data umum. Untuk informasi selengkapnya, lihat berikut ini:

- AWS DMS Panduan Pengguna
- AWS Glue Panduan Pengguna
- Praktik Terbaik untuk Bermigrasi dari RDBMS ke DynamoDB

NoSQL Workbench untuk DynamoDB

NoSQL Workbench untuk Amazon DynamoDB adalah aplikasi GUI sisi klien lintas platform yang dapat Anda gunakan untuk pengembangan dan operasi basis data modern. Ini tersedia untuk Windows, macOS, dan Linux. NoSQL Workbench adalah alat pengembangan visual yang menyediakan fitur pemodelan data, visualisasi data, dan pengembangan kueri untuk membantu Anda merancang, membuat, mengkueri, dan mengelola tabel DynamoDB. NoSQL Workbench kini menyertakan DynamoDB lokal sebagai bagian opsional dari proses instalasi, yang mempermudah Anda untuk memodelkan data Anda di DynamoDB lokal. Untuk mempelajari selengkapnya tentang DynamoDB lokal dan persyaratannya, lihat Menyiapkan DynamoDB lokal (versi yang dapat diunduh) .

Pemodelan data

Dengan NoSQL Workbench untuk DynamoDB, Anda dapat membuat model data baru dari, atau mendesain model berdasarkan, model data yang sudah ada yang memenuhi pola akses data aplikasi Anda. Anda juga dapat mengimpor dan mengekspor model data yang didesain pada akhir proses. Untuk informasi selengkapnya, lihat Membangun Model Data dengan NoSQL Workbench.

Visualisasi data

Pemvisualisasi model data menyediakan kanvas tempat Anda dapat memetakan kueri dan memvisualisasikan pola akses (faset) aplikasi tanpa harus menulis kode. Setiap faset berhubungan dengan pola akses yang berbeda di DynamoDB. Anda dapat membuat data sampel secara otomatis untuk digunakan dalam model data Anda. Untuk informasi selengkapnya, lihat Memvisualisasikan pola akses data.

Pembangunan operasi

NoSQL Workbench menyediakan antarmuka pengguna grafis yang kaya bagi Anda untuk mengembangkan dan menguji kueri. Anda dapat menggunakan pembangun operasi untuk melihat, menjelajahi, dan mengkueri dataset langsung. Builder operasi terstruktur mendukung ekspresi proyeksi, ekspresi kondisi, dan menghasilkan kode sampel dalam berbagai bahasa. Anda dapat langsung mengkloning tabel dari satu akun Amazon DynamoDB ke akun lain di Wilayah yang berbeda. Anda juga dapat langsung mengkloning tabel antara DynamoDB lokal dan akun Amazon DynamoDB untuk menyalin skema kunci tabel Anda dengan lebih cepat (dan skema dan item GSI opsional) di antara lingkungan pengembangan Anda. Untuk informasi selengkapnya, lihat Menjelajahi set data dan membangun operasi dengan NoSQL Workbench.

Video di bawah ini merinci konsep pemodelan data dengan NoSQL Workbench.

Topik

- Unduh NoSQL Workbench untuk DynamoDB
- Menginstal NoSQL Workbench untuk DynamoDB
- Membangun Model Data dengan NoSQL Workbench
- Memvisualisasikan pola akses data
- Menjelajahi set data dan membangun operasi dengan NoSQL Workbench
- Model data sampel untuk NoSQL Workbench
- Riwayat rilis untuk NoSQL Workbench

Unduh NoSQL Workbench untuk DynamoDB

Ikuti petunjuk ini untuk mengunduh NoSQL Workbench dan DynamoDB lokal* untuk Amazon DynamoDB.

Prasyarat

Ada dua bagian prasyarat perangkat lunak yang diperlukan untuk menginstal Ubuntu: libfuse2 dan curl.

libfuse2

Pada Ubuntu 22.04, libfuse2 tidak lagi diinstal secara default. Untuk mengatasi ini, jalankan `sudo add-apt-repository universe && sudo apt install libfuse2` untuk menginstal untuk versi Ubuntu terbaru.

keriting

Perbarui Ubuntu, jalankan `sudo apt update && sudo apt upgrade`

Selanjutnya, instalcURL, jalankan: `sudo apt install curl`

Untuk mengunduh NoSQL Workbench dan DynamoDB lokal

1. Unduh versi NoSQL Workbench yang sesuai untuk sistem operasi Anda.

Sistem operasi	Tautan unduhan
macOS (Intel) **	Unduh untuk macOS (Intel)
macOS (silikon Apple)	Unduh untuk macOS (Apple silicon)
Windows	Unduh untuk Windows
Linux***	Unduh untuk Linux

* NoSQL Workbench termasuk DynamoDB lokal sebagai bagian opsional dari proses instalasi.

** Jika pesan peringatan muncul saat Anda mencoba membuka NoSQL Workbench yang menyatakan bahwa aplikasi tidak terdaftar di Apple oleh pengembang yang diidentifikasi, lakukan hal berikut:

1. Temukan aplikasi dan kemudian buka.
2. Kontrol+klik ikon aplikasi, lalu pilih Buka dari menu pintasan.

   Ini menyimpan aplikasi sebagai pengecualian untuk pengaturan keamanan Anda. Buka aplikasi dengan mengklik dua kali sama seperti Anda dapat membuka aplikasi terdaftar apa pun.

*** NoSQL Workbench mendukung Ubuntu 12.04, Fedora 21, dan Debian 8, atau versi yang lebih baru dari distribusi Linux ini.

2. Mulai aplikasi yang Anda unduh, lalu ikuti langkah-langkah di Menginstal NoSQL Workbench.

> **Note**
>
> Java Runtime Environment (JRE) versi 11.x atau yang lebih baru diperlukan untuk menjalankan DynamoDB lokal.

Menginstal NoSQL Workbench untuk DynamoDB

Ikuti langkah-langkah berikut untuk menginstal NoSQL Workbench dan DynamoDB lokal pada platform yang didukung.

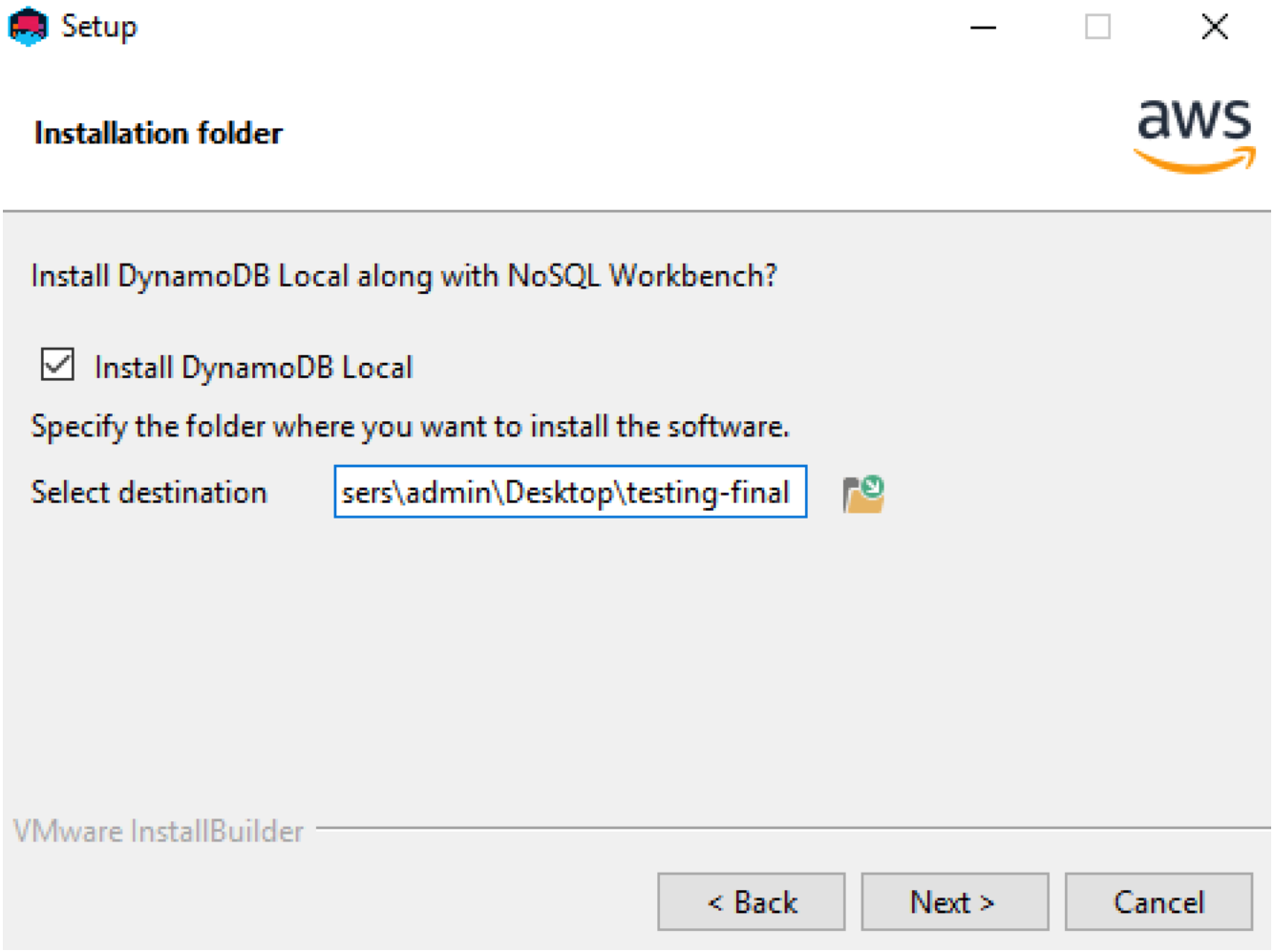
Windows

Untuk menginstal NoSQL Workbench di Windows

1. Jalankan aplikasi penginstal NoSQL Workbench dan pilih bahasa pengaturan. Kemudian pilih OK untuk memulai pengaturan. Untuk informasi selengkapnya tentang mengunduh NoSQL Workbench, lihat Unduh NoSQL Workbench untuk DynamoDB.
2. Pilih Selanjutnya untuk melanjutkan pengaturan, lalu pilih Selanjutnya di layar berikut.
3. Secara default, kotak centang Instal DynamoDB Lokal dipilih untuk menyertakan DynamoDB lokal sebagai bagian dari instalasi. Jika opsi ini tetap dipilih, maka DynamoDB lokal akan diinstal, dan jalur tujuan akan sama dengan jalur instalasi NoSQL Workbench. Mengosongkan kotak centang untuk opsi ini akan melewati instalasi DynamoDB lokal, dan jalur instalasi hanya untuk NoSQL Workbench.

   Pilih tujuan di mana Anda ingin perangkat lunak diinstal, dan pilih Selanjutnya.

   > **Note**
   >
   > Jika Anda memilih untuk tidak menyertakan DynamoDB lokal sebagai bagian dari pengaturan, kosongkan kotak centang Install DynamoDB Local, pilih Berikutnya, dan lewati ke langkah 6. Anda dapat mengunduh DynamoDB lokal secara terpisah sebagai instalasi mandiri di lain waktu. Untuk informasi selengkapnya, lihat Menyiapkan DynamoDB lokal (versi yang dapat diunduh) .



4. Pilih nomor port untuk DynamoDB lokal yang akan digunakan. Port dafult adalah 8000. Setelah Anda memasukkan nomor port, pilih Selanjutnya.
5. Pilih Selanjutnya untuk memulai pengaturan.
6. Setelah pengaturan selesai, pilih Selesai untuk menutup layar pengaturan.
7. Buka aplikasi di jalur instalasi Anda, seperti DBWorkbench/Programs/Dynamo/.

macOS

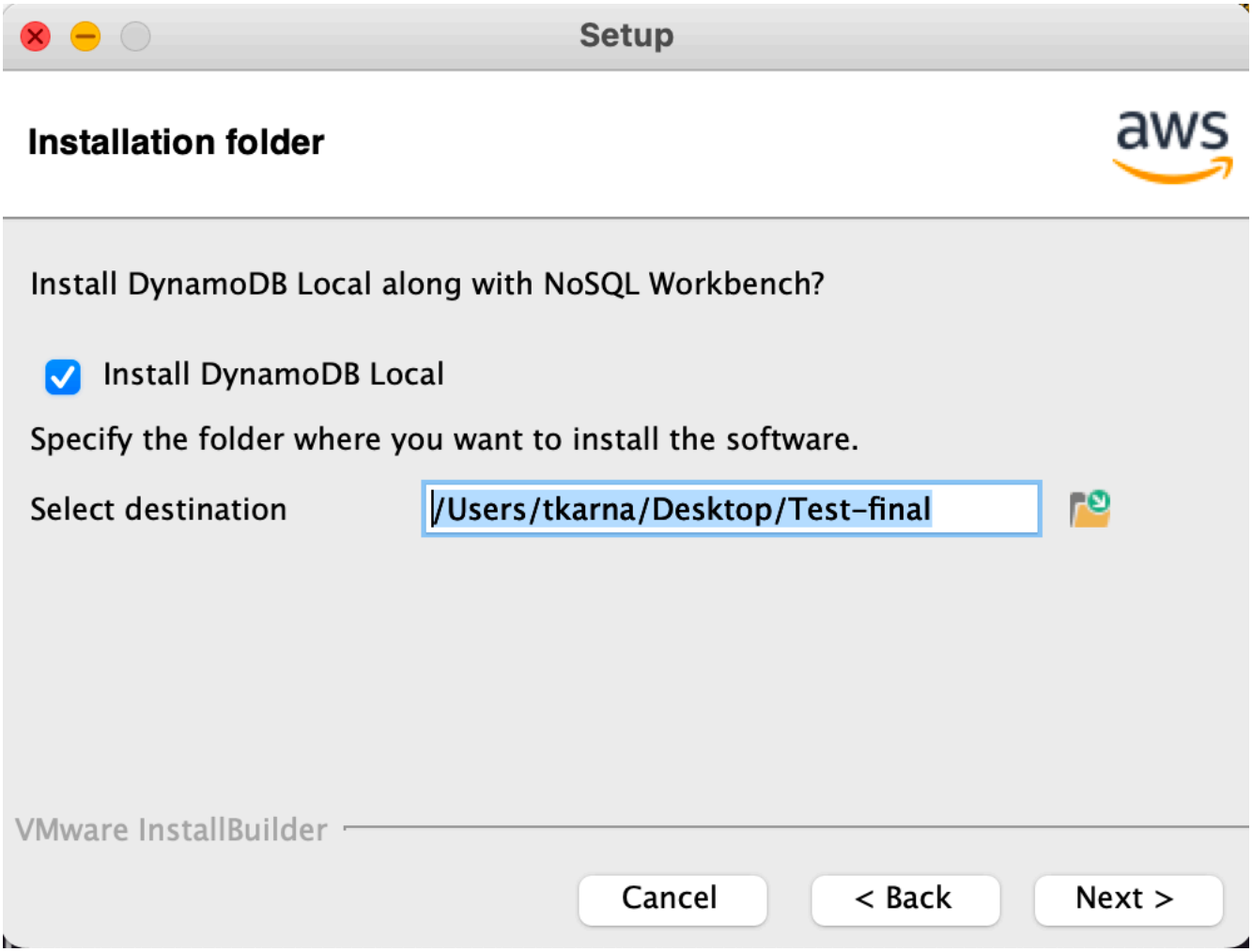
Untuk menginstal NoSQL Workbench di macOS

1. Jalankan aplikasi penginstal NoSQL Workbench dan pilih bahasa pengaturan. Kemudian pilih OK untuk memulai pengaturan. Untuk informasi selengkapnya tentang mengunduh NoSQL Workbench, lihat Unduh NoSQL Workbench untuk DynamoDB.

2. Pilih Selanjutnya untuk melanjutkan pengaturan, lalu pilih Selanjutnya di layar berikut.
3. Secara default, kotak centang Instal DynamoDB lokal dipilih untuk menyertakan DynamoDB lokal sebagai bagian dari instalasi. Jika opsi ini tetap dipilih, maka DynamoDB lokal akan diinstal, dan jalur tujuan akan sama dengan jalur instalasi NoSQL Workbench. Mengosongkan opsi ini akan melewati instalasi DynamoDB lokal, dan jalur instalasi hanya untuk NoSQL Workbench.

   Pilih tujuan di mana Anda ingin perangkat lunak diinstal, dan pilih Selanjutnya.

   > **Note**
   >
   > Jika Anda memilih untuk tidak menyertakan DynamoDB lokal sebagai bagian dari pengaturan, kosongkan kotak centang Instal DynamoDB lokal, pilih Berikutnya, dan lewati ke langkah 6. Anda dapat mengunduh DynamoDB lokal secara terpisah sebagai instalasi mandiri di lain waktu. Untuk informasi selengkapnya, lihat Menyiapkan DynamoDB lokal (versi yang dapat diunduh) .



4. Pilih nomor port untuk DynamoDB lokal yang akan digunakan. Port dafult adalah 8000. Setelah Anda memasukkan nomor port, pilih Selanjutnya.
5. Pilih Selanjutnya untuk memulai pengaturan.
6. Setelah pengaturan selesai, pilih Selesai untuk menutup layar pengaturan.
7. Buka aplikasi di jalur instalasi Anda, seperti /Applications/Dynamo/DBWorkbench.

> **Note**
>
> NoSQL Workbench untuk macOS melakukan pembaruan otomatis. Untuk mendapatkan notifikasi tentang pembaruan, aktifkan akses notifikasi ke NoSQL Workbench di Preferensi Sistem > Notifikasi.

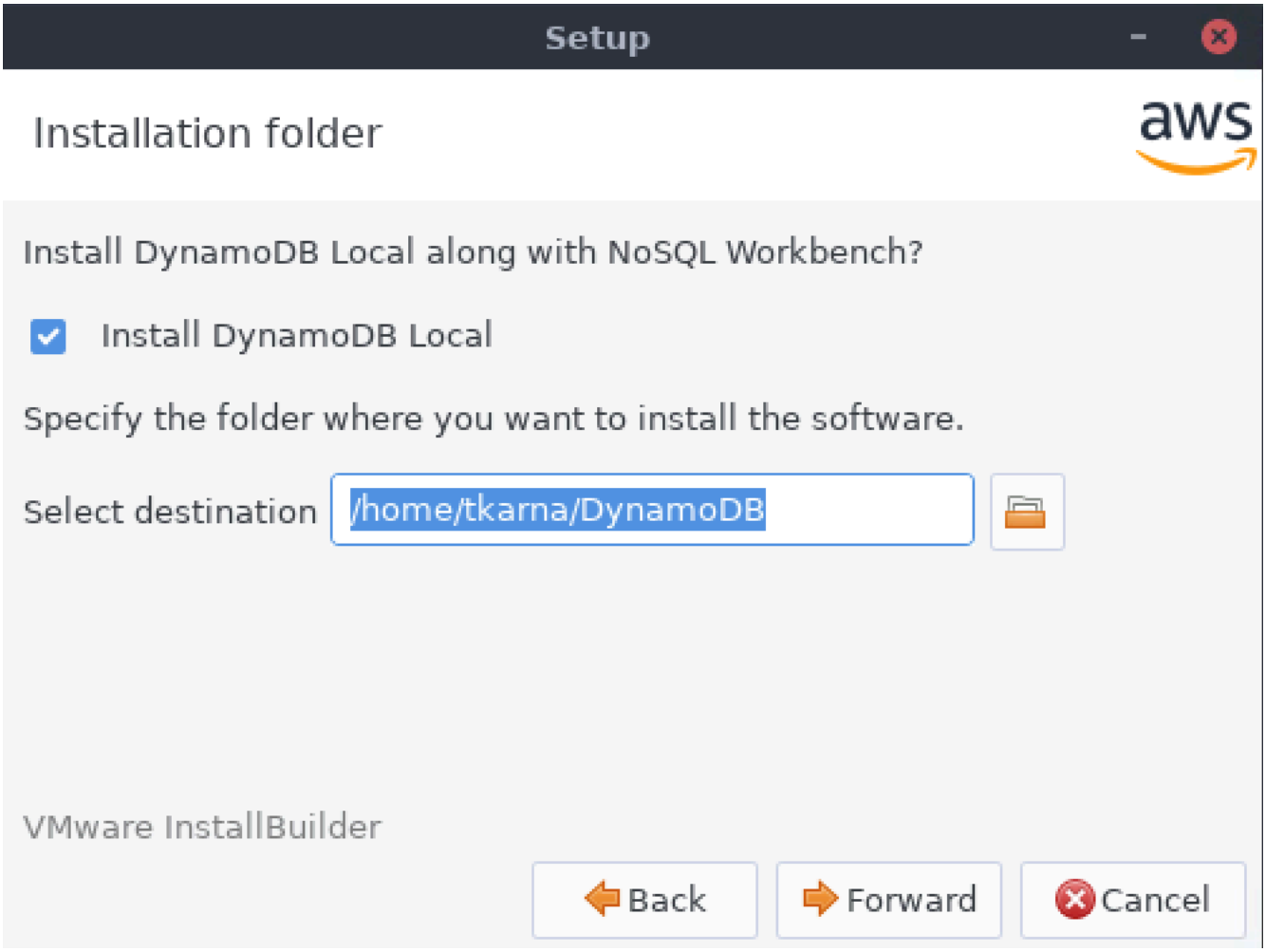
Linux

Untuk menginstal NoSQL Workbench di Linux

1. Jalankan aplikasi penginstal NoSQL Workbench dan pilih bahasa pengaturan. Kemudian pilih OK untuk memulai pengaturan. Untuk informasi selengkapnya tentang mengunduh NoSQL Workbench, lihat Unduh NoSQL Workbench untuk DynamoDB.
2. Pilih Teruskan untuk melanjutkan pengaturan, dan pilih Teruskan di layar berikut.
3. Secara default, kotak centang Instal DynamoDB lokal dipilih untuk menyertakan DynamoDB lokal sebagai bagian dari instalasi. Jika opsi ini tetap dipilih, maka DynamoDB lokal akan diinstal, dan jalur tujuan akan sama dengan jalur instalasi NoSQL Workbench. Mengosongkan opsi ini akan melewati instalasi DynamoDB lokal, dan jalur instalasi hanya untuk NoSQL Workbench.

   Pilih tujuan di mana Anda ingin perangkat lunak diinstal, dan pilih Teruskan.

   > **Note**
   >
   > Jika Anda memilih untuk tidak menyertakan DynamoDB lokal sebagai bagian dari pengaturan, kosongkan kotak centang Instal DynamoDB lokal, pilih Teruskan, dan lewati ke langkah 6. Anda dapat mengunduh DynamoDB lokal secara terpisah sebagai instalasi mandiri di lain waktu. Untuk informasi selengkapnya, lihat Menyiapkan DynamoDB lokal (versi yang dapat diunduh) .



4. Pilih nomor port untuk DynamoDB lokal yang akan digunakan. Port dafult adalah 8000. Setelah Anda memasukkan nomor port yang dimasukkan, pilih Teruskan.
5. Pilih Teruskan untuk memulai pengaturan.
6. Setelah pengaturan selesai, pilih Selesai untuk menutup layar pengaturan.
7. Buka aplikasi di jalur instalasi Anda, seperti/usr/local/programs/DynamoDBWorkbench/.

Untuk memulai AppImage di Linux

1. Jadikan AppImage file dapat dieksekusi:

   chmod noSQL-workbench-linux +x. AppImage

   Ganti noSQL-workbench-linux. AppImage dengan nama file yang sebenarnya dari yang AppImage Anda unduh.

2. Jalankan AppImage:

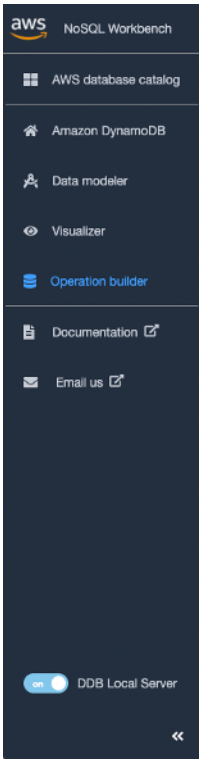
   . /NoSQL-workbench-Linux. AppImage

   Ini akan meluncurkan aplikasi NoSQL Workbench.

> **Note**
>
> Tergantung pada distribusi Linux Anda, Anda mungkin perlu menginstal dependensi tambahan AppImage agar dapat berjalan dengan benar. Jika Anda mengalami masalah, lihat dokumentasi yang disediakan oleh AppImage pengembang atau cari dukungan dari komunitas.

> **Note**
>
> Jika Anda memilih untuk menginstal DynamoDB lokal sebagai bagian dari instalasi NoSQL Workbench, DynamoDB lokal akan dikonfigurasikan sebelumnya dengan opsi default. Untuk mengedit opsi default, ubah skrip DDBLocalMulai yang terletak di direktori /resources/ DDBLocal _scripts/. Anda dapat menemukannya di jalur yang Anda berikan saat instalasi. Untuk mempelajari selengkapnya tentang opsi lokal DynamoDB, lihat Catatan penggunaan DynamoDB lokal .

Jika Anda memilih untuk menginstal DynamoDB lokal sebagai bagian dari instalasi NoSQL Workbench, Anda akan memiliki akses ke tombol untuk mengaktifkan dan menonaktifkan DynamoDB lokal seperti yang ditunjukkan pada gambar berikut.



Membangun Model Data dengan NoSQL Workbench

Anda dapat menggunakan alat pemodel data di NoSQL Workbench untuk Amazon DynamoDB untuk membangun model data baru, atau merancang model berdasarkan model data yang sudah ada yang memenuhi pola akses data aplikasi Anda. Pemodel data menyertakan beberapa model data sampel untuk membantu Anda memulai.

Topik

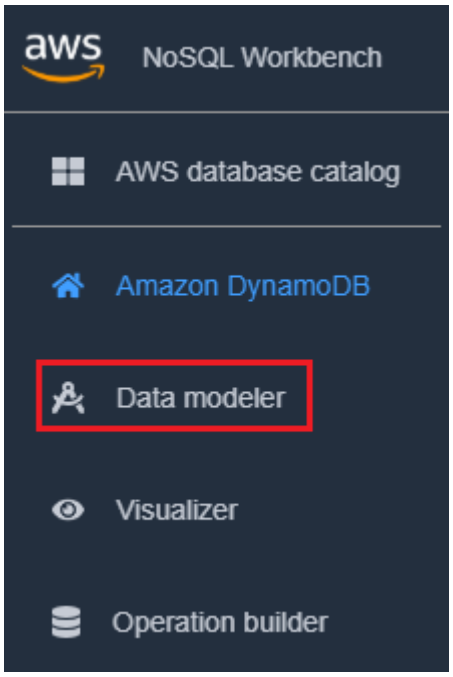
- Membuat model data baru
- Mengimpor model data yang ada
- Mengekspor model data
- Mengedit model data yang ada

Membuat model data baru

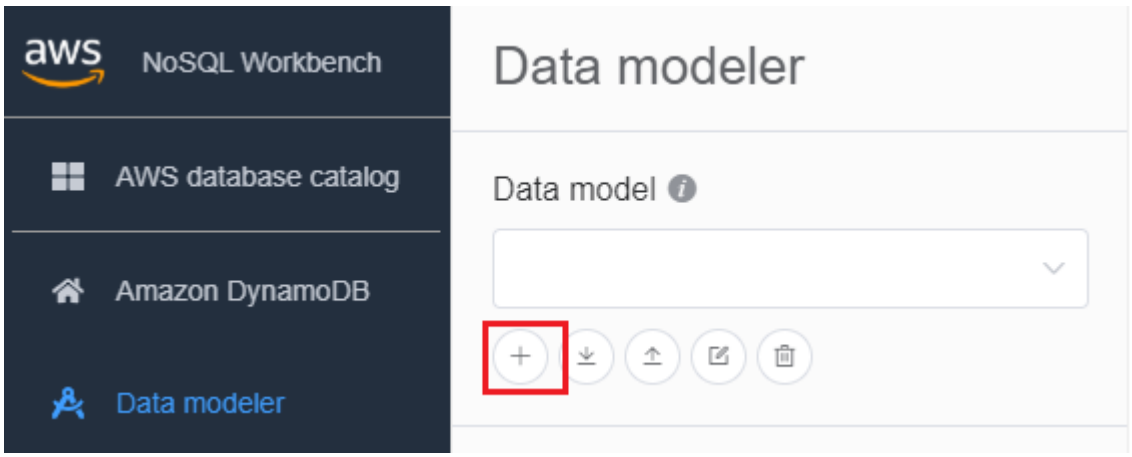
Ikuti langkah-langkah ini untuk membuat model data baru di Amazon DynamoDB menggunakan NoSQL Workbench.

Untuk membuat model data baru

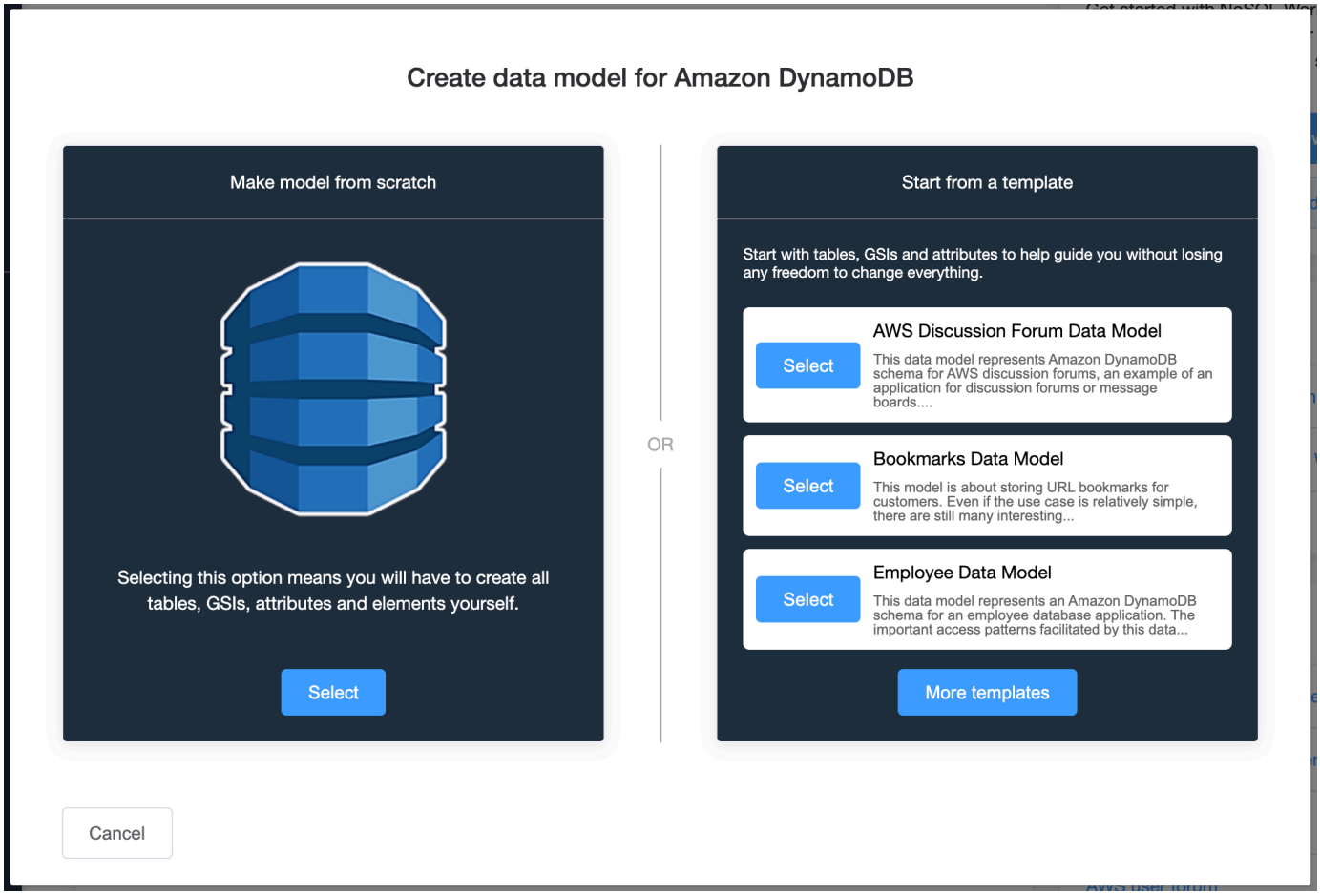
1. Buka NoSQL Workbench, dan di panel navigasi di sisi kiri, pilih ikon Pemodel data.



2. Pilih Buat model data.

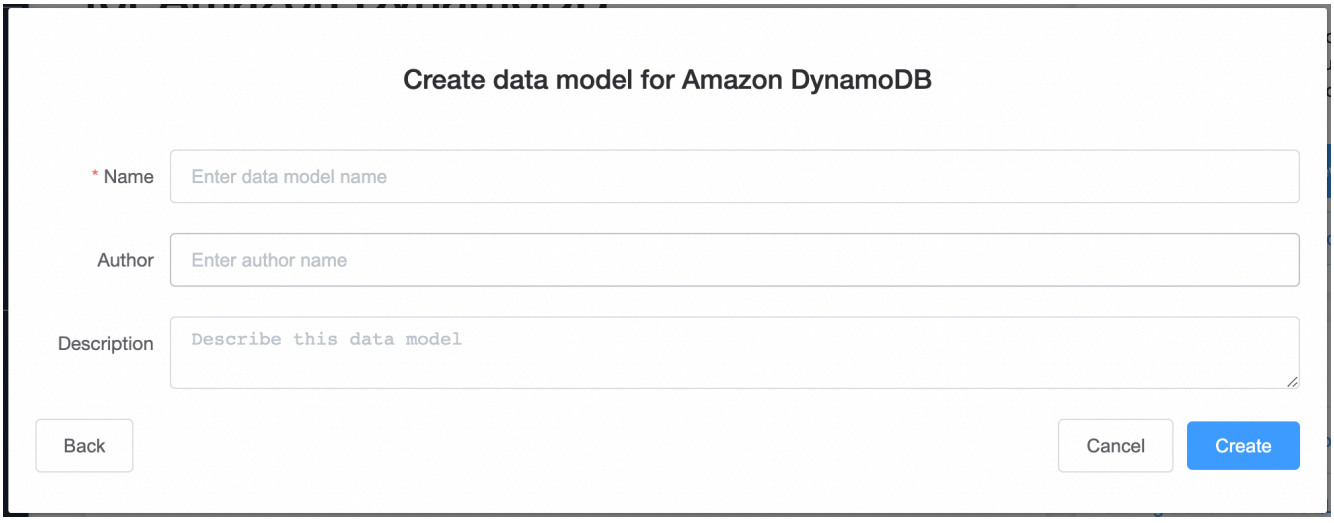


Buat model data memiliki dua pilihan: Membuat model dari awal dan Mulai dari templat.



Make model from scratch

Untuk membuat model dari awal, masukkan nama, penulis, dan deskripsi untuk model data. Pilih Buat setelah selesai.



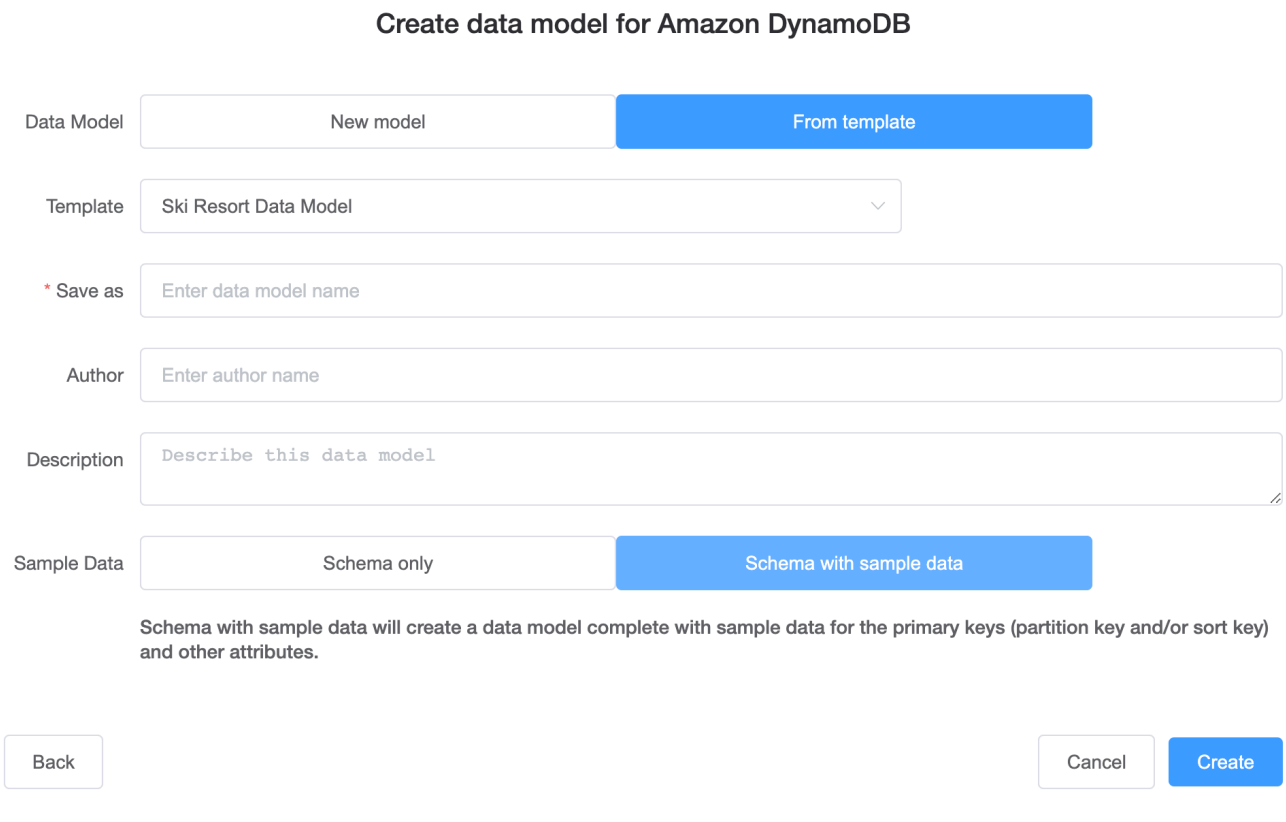
Start from a template

Mulai dari templat memungkinkan Anda memilih model sampel untuk memulai. Pilih Lebih banyak templat untuk melihat opsi templat lainnya. Pilih Pilih untuk templat yang ingin Anda gunakan.

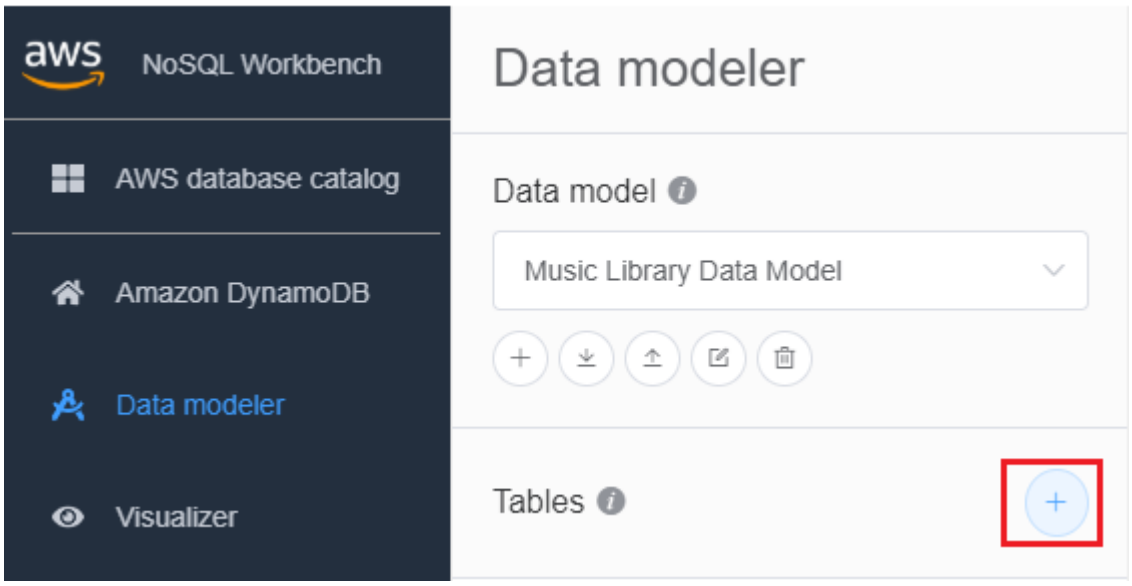
Masukkan nama model data, penulis, dan deskripsi untuk templat yang Anda pilih. Anda dapat memilih antara Skema saja dan Skema dengan data sampel.

- Skema saja membuat model data kosong dengan kunci primer (partisi dan kunci urutan) dan atribut lainnya.
- Skema dengan data sampel akan membuat model data lengkap dengan data sampel untuk kunci primer (kunci partisi dan pengurutan) dan atribut lainnya.

Ketika informasi ini selesai, pilih Buat untuk membuat model.



3. Dengan model yang dibuat, pilih Tambahkan tabel.



Untuk informasi selengkapnya tentang tabel, lihat Bekerja dengan tabel di DynamoDB.

4. Tentukan hal berikut:

   - Nama tabel — Masukkan nama unik untuk tabel.
   - Kunci partisi — Masukkan nama kunci partisi dan tentukan jenisnya. Secara opsional, Anda juga dapat memilih format jenis data yang lebih terperinci untuk pembuatan data sampel.
   - Jika Anda ingin menambahkan kunci urutan:
     1. Pilih Tambah kunci urutan.
     2. Tentukan nama kunci urutan dan jenisnya. Secara opsional, Anda dapat memilih format jenis data yang lebih terperinci untuk pembuatan data sampel.

> **Note**
>
> Untuk mempelajari lebih lanjut tentang desain kunci primer, merancang dan menggunakan kunci partisi secara efektif, dan menggunakan kunci urutan, lihat yang berikut ini:
>
> - Kunci primer
> - Praktik terbaik untuk merancang dan menggunakan kunci partisi secara efektif di DynamoDB
> - Praktik terbaik untuk menggunakan kunci sortir untuk mengatur data di DynamoDB

5. Untuk menambahkan atribut lainnya, lakukan hal berikut untuk setiap atribut:

    1. Pilih Tambahkan atribut.
    2. Tentukan nama atribut dan jenisnya. Secara opsional, Anda dapat memilih format jenis data yang lebih terperinci untuk pembuatan data sampel.

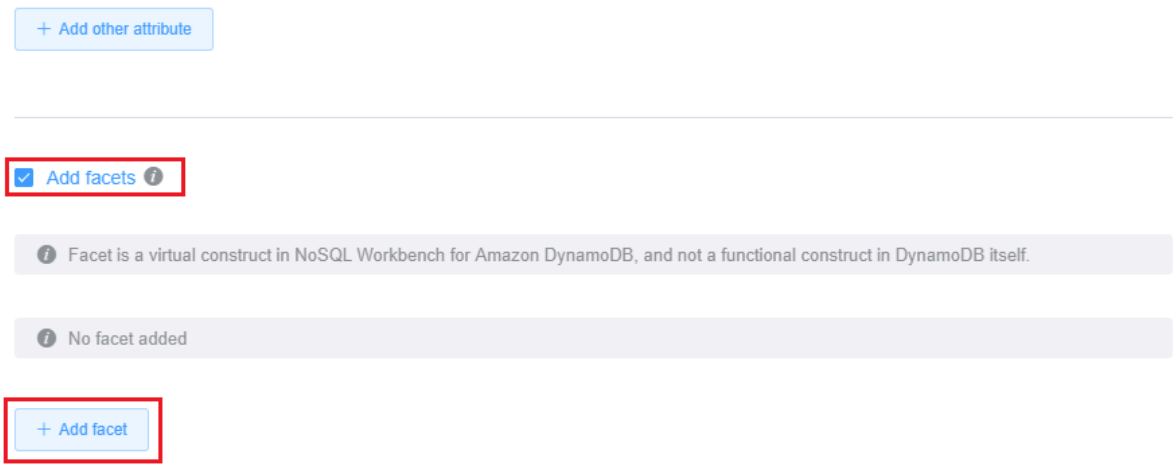
6. Tambahkan faset:

    Anda dapat menambahkan faset secara opsional. Faset adalah konstruksi virtual di NoSQL Workbench. Ini bukan konstruksi fungsional di DynamoDB itu sendiri.

    > **Note**
    >
    > Faset di NoSQL Workbench membantu Anda memvisualisasikan pola akses data aplikasi yang berbeda untuk Amazon DynamoDB dengan hanya sebagian data dalam tabel. Untuk lebih mempelajari faset, lihat Melihat pola akses data.

    Untuk menambahkan faset,

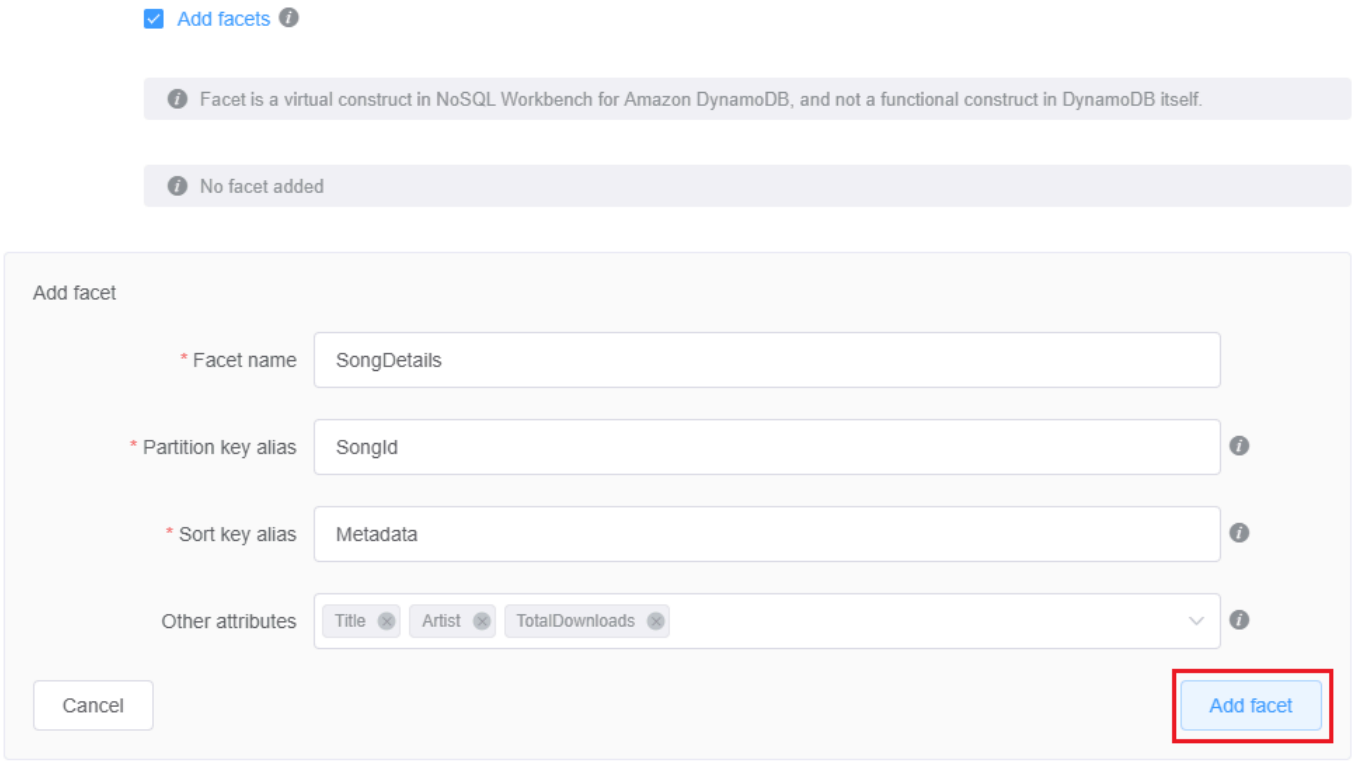
    - Pilih Tambahkan faset.
    - Pilih Tambahkan faset.

    - Tentukan hal berikut:
        - Nama faset.
        - Alias kunci partisi untuk membantu membedakan tampilan faset ini.
        - Alias kunci urutan.

- Pilih Atribut lainnya yang merupakan bagian dari faset ini.

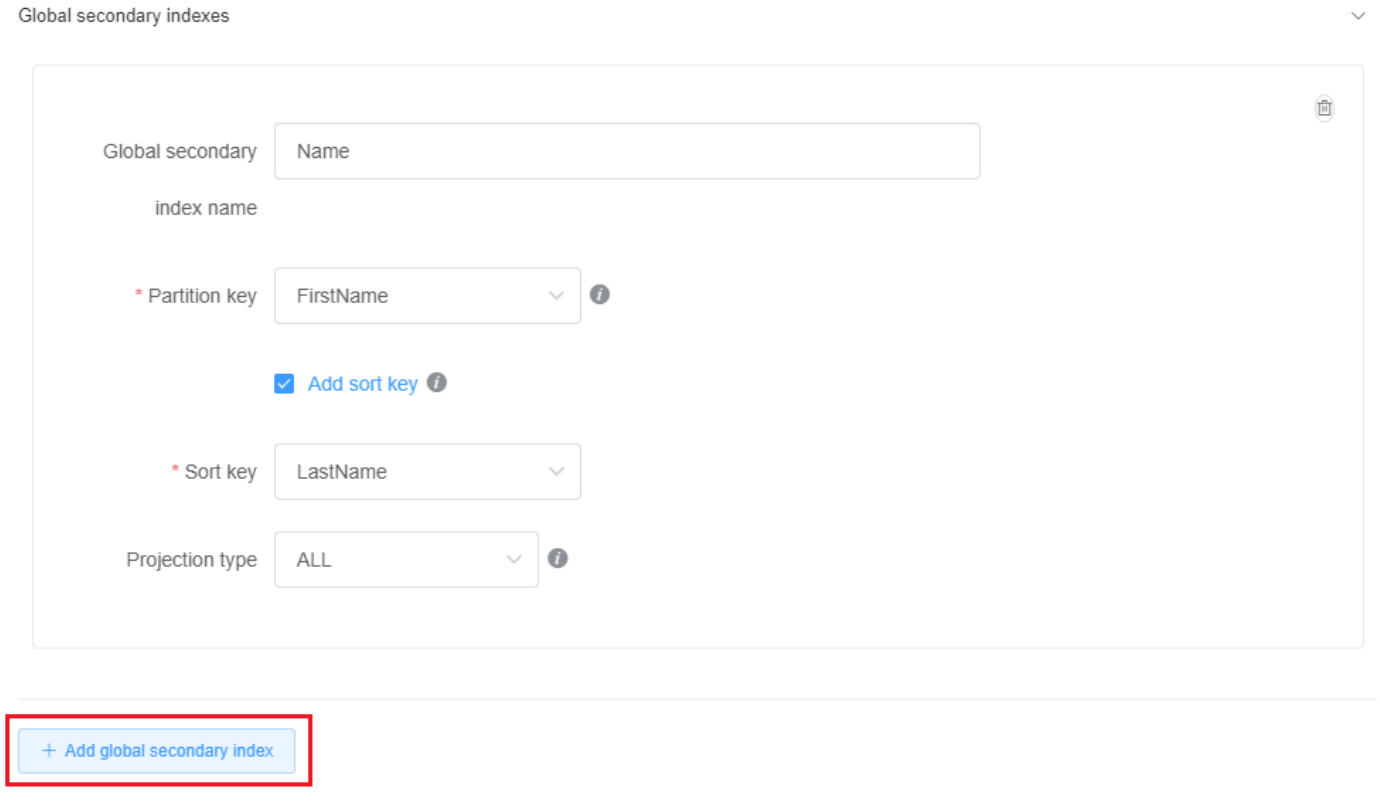
Pilih Tambahkan faset.



Ulangi langkah ini jika Anda ingin menambahkan lebih banyak faset.

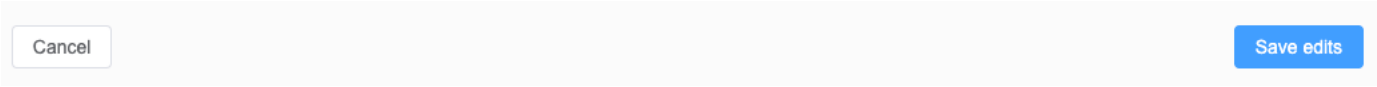
7. Jika Anda ingin menambahkan indeks sekunder global, pilih Tambah indeks sekunder global.

   Tentukan Nama indeks sekunder global, atribut Kunci partisi, dan Jenis proyeksi.



Untuk informasi selengkapnya tentang bekerja dengan indeks sekunder global di DynamoDB, lihat Indeks sekunder global.

8. Simpan hasil edit ke pengaturan tabel Anda..



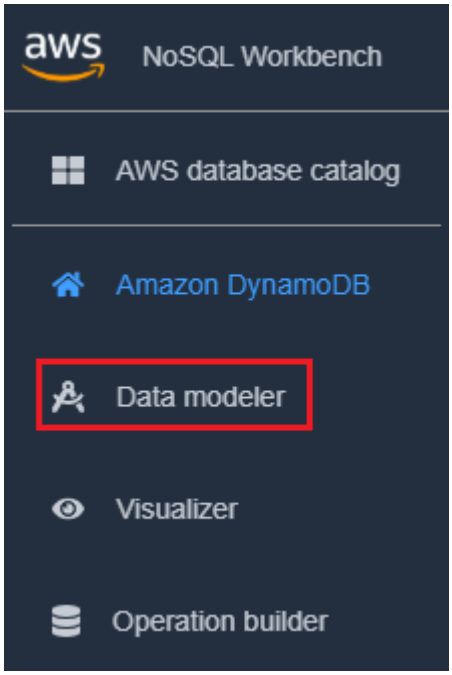
Untuk informasi selengkapnya tentang operasi `CreateTable` API, lihat CreateTabledi Referensi Amazon DynamoDB API.

Mengimpor model data yang ada

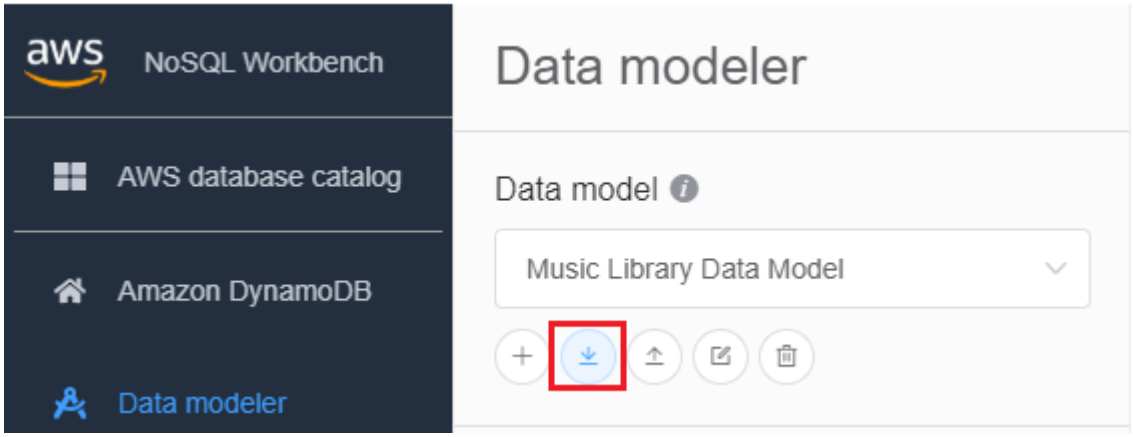
Anda dapat menggunakan NoSQL Workbench untuk Amazon DynamoDB untuk membangun model data dengan mengimpor dan memodifikasi model yang sudah ada. Anda dapat mengimpor model data dalam format model NoSQL Workbench atau dalam format templat JSON AWS CloudFormation.

Untuk mengimpor model data

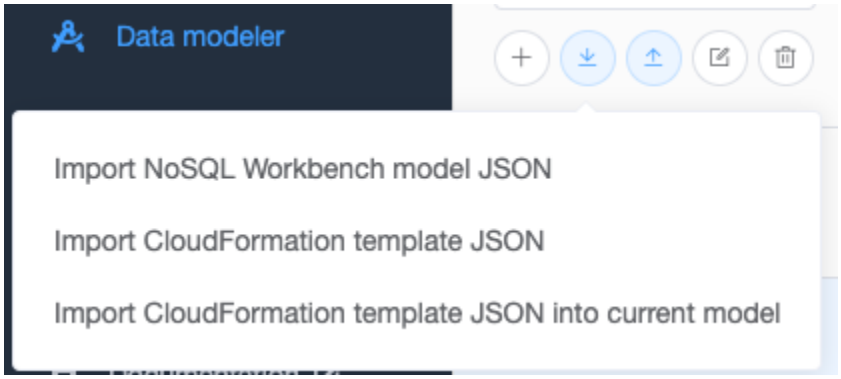
1. Di NoSQL Workbench, dan di panel navigasi di sisi kiri, pilih ikon Pemodel data.



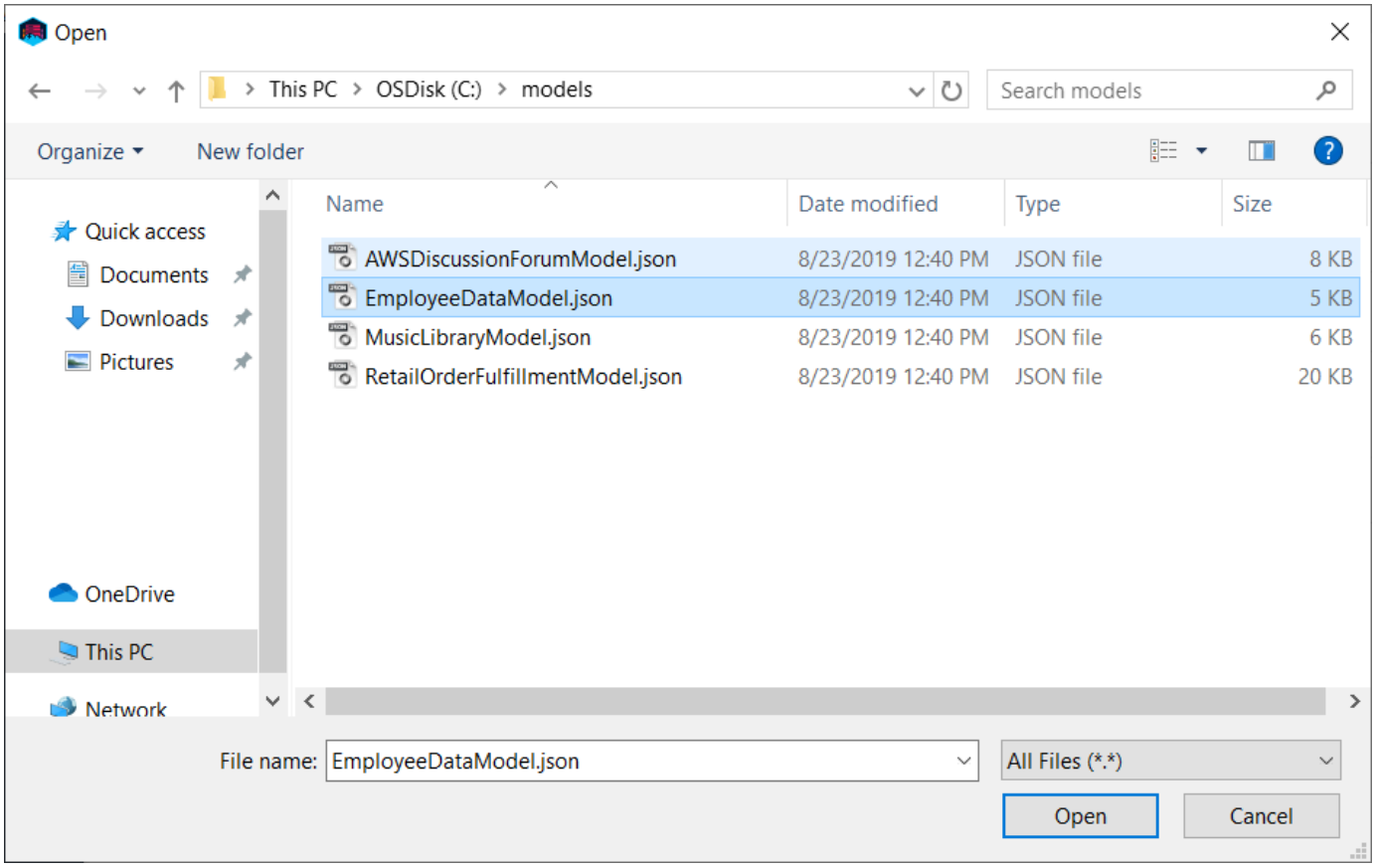
2. Arahkan penunjuk Anda ke Impor model data.



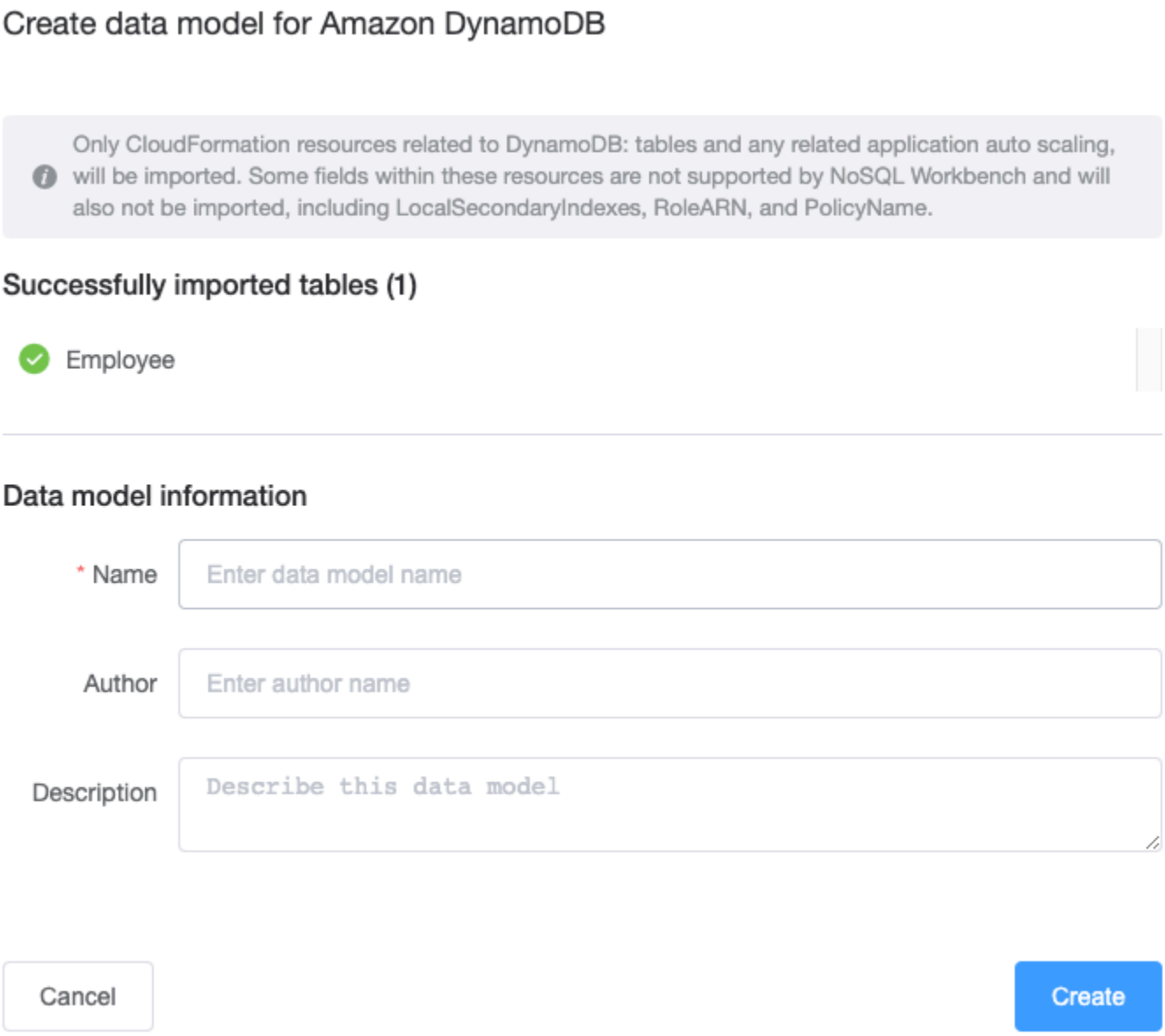
Dalam daftar dropdown, pilih apakah model yang ingin Anda impor dalam format model NoSQL Workbench atau format template JSON. CloudFormation Jika Anda memiliki model data yang ada terbuka di NoSQL Workbench, Anda akan memiliki opsi untuk mengimpor CloudFormation template ke model saat ini.



3. Pilih model yang akan diimpor.



4. Jika model yang Anda impor dalam format CloudFormation templat, Anda akan melihat daftar tabel yang akan diimpor dan memiliki kesempatan untuk menentukan nama model data, penulis, dan deskripsi.

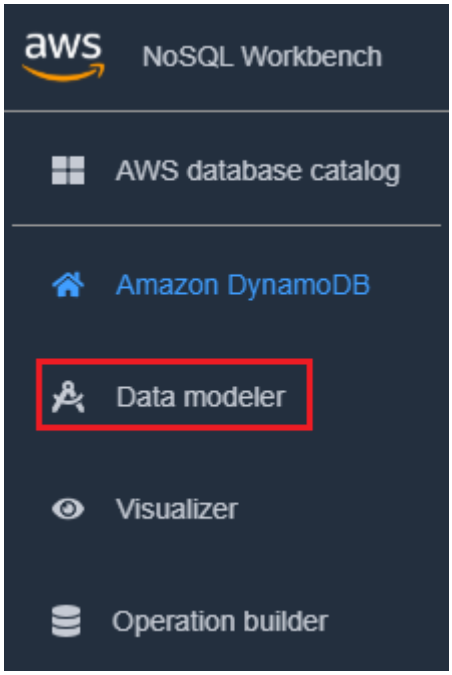


Mengekspor model data

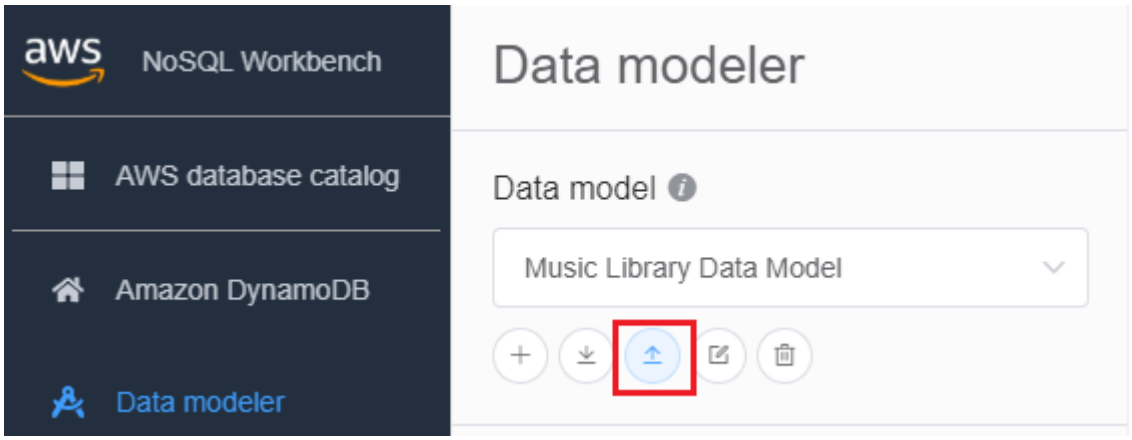
Setelah Anda membuat model data menggunakan NoSQL Workbench untuk Amazon DynamoDB, Anda dapat menyimpan dan mengekspor model dalam format model NoSQL Workbench atau Format templat JSON AWS CloudFormation.

Untuk mengekspor model data

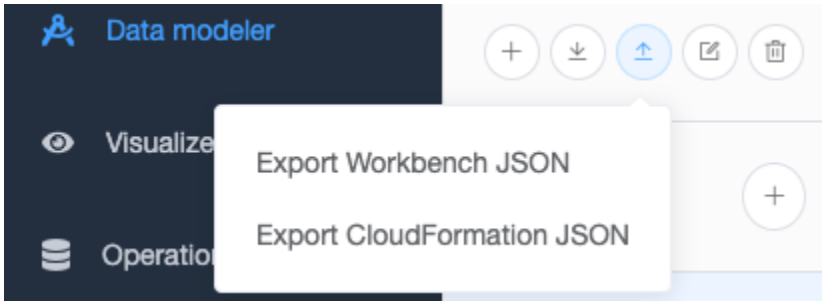
1. Di NoSQL Workbench, dan di panel navigasi di sisi kiri, pilih ikon Pemodel data.



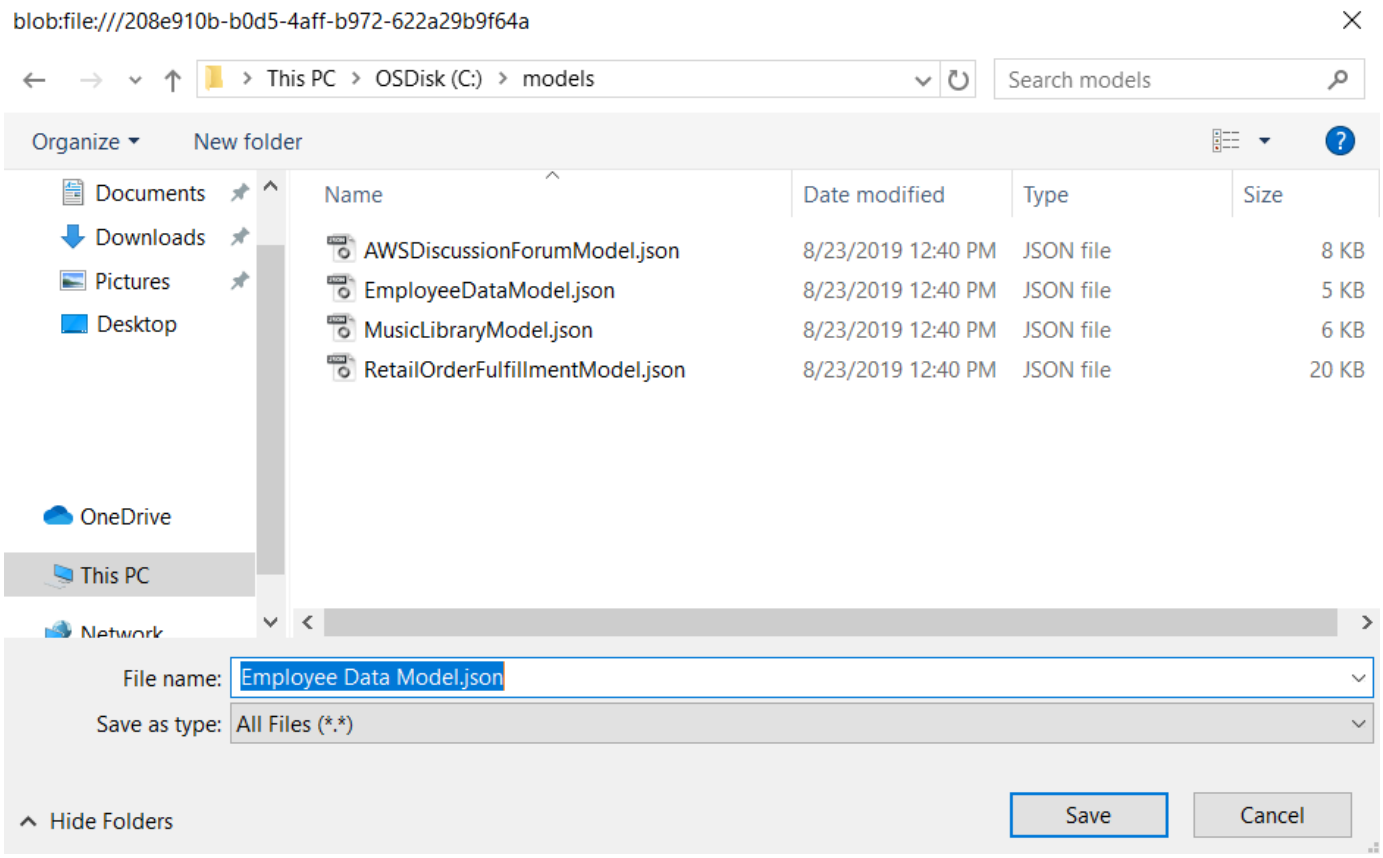
2. Arahkan kursor ke model Ekspor model data.



Dalam daftar dropdown, pilih apakah akan mengekspor model data Anda dalam format model NoSQL Workbench atau format template JSON. CloudFormation



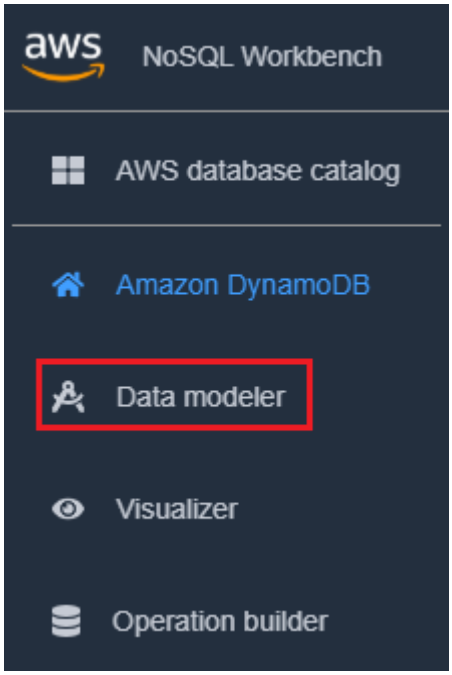
3. Pilih lokasi untuk menyimpan model Anda.



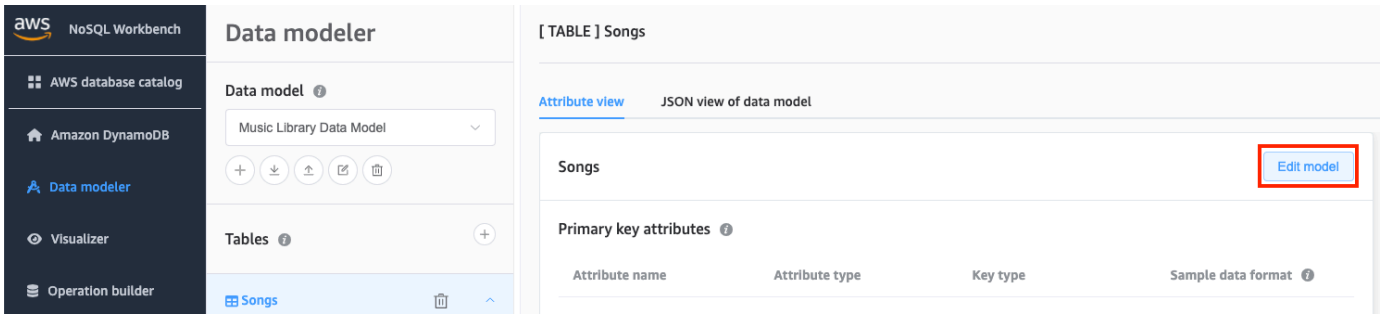
Mengedit model data yang ada

Untuk mengedit model yang ada

1. Di NoSQL Workbench, dan di panel navigasi di sisi kiri, pilih tombol Pemodel data.



2. Pilih model data dan pilih tabel yang ingin Anda edit. Pilih Edit model



3. Lakukan pengeditan yang diperlukan, lalu pilih Simpan pengeditan.

Untuk mengedit model yang sudah ada dan menambahkan faset secara manual

1. Ekspor model Anda. Untuk informasi selengkapnya, lihat Mengekspor model data.
2. Buka file yang diekspor dalam editor.
3. Temukan objek `DataModel` untuk tabel yang ingin Anda buat fasetnya.

   Tambahkan array `TableFacets` yang mewakili semua faset untuk tabel.

   Untuk setiap faset tambahkan objek ke array `TableFacets`. Setiap elemen susunan memiliki sifat berikut:

   - `FacetName` – Nama untuk sisi Anda. Nilai ini harus unik di seluruh model.

- `PartitionKeyAlias` – Nama yang familier untuk kunci partisi tabel. Alias ini ditampilkan saat Anda melihat faset di NoSQL Workbench.
- `SortKeyAlias` – Nama yang familier untuk kunci urutan tabel. Alias ini ditampilkan saat Anda melihat faset di NoSQL Workbench. Properti ini tidak diperlukan jika tabel tidak memiliki kunci urutan yang ditentukan.
- `NonKeyAttributes` – Array nama atribut yang diperlukan untuk pola akses. Nama ini harus dipetakan ke nama atribut yang didefinisikan untuk tabel.

```
{
  "ModelName": "Music Library Data Model",
  "DataModel": [
    {
      "TableName": "Songs",
      "KeyAttributes": {
        "PartitionKey": {
          "AttributeName": "Id",
          "AttributeType": "S"
        },
        "SortKey": {
          "AttributeName": "Metadata",
          "AttributeType": "S"
        }
      },
      "NonKeyAttributes": [
        {
          "AttributeName": "DownloadMonth",
          "AttributeType": "S"
        },
        {
          "AttributeName": "TotalDownloadsInMonth",
          "AttributeType": "S"
        },
        {
          "AttributeName": "Title",
          "AttributeType": "S"
        },
        {
          "AttributeName": "Artist",
          "AttributeType": "S"
        },
        {
```

```
          "AttributeName": "TotalDownloads",
          "AttributeType": "S"
        },
        {
          "AttributeName": "DownloadTimestamp",
          "AttributeType": "S"
        }
      ],
      "TableFacets": [
        {
          "FacetName": "SongDetails",
          "KeyAttributeAlias": {
            "PartitionKeyAlias": "SongId",
            "SortKeyAlias": "Metadata"
          },
          "NonKeyAttributes": [
            "Title",
            "Artist",
            "TotalDownloads"
          ]
        },
        {
          "FacetName": "Downloads",
          "KeyAttributeAlias": {
            "PartitionKeyAlias": "SongId",
            "SortKeyAlias": "Metadata"
          },
          "NonKeyAttributes": [
            "DownloadTimestamp"
          ]
        }
      ]
    }
  ]
}
```

4. Anda sekarang dapat mengimpor model yang dimodifikasi ke NoSQL Workbench. Untuk informasi selengkapnya, lihat Mengimpor model data yang ada.

Memvisualisasikan pola akses data

Anda dapat menggunakan alat pemvisualisasi di NoSQL Workbench untuk Amazon DynamoDB guna memetakan kueri dan memvisualisasikan pola akses yang berbeda (dikenal sebagai faset) dari aplikasi. Setiap faset berhubungan dengan pola akses yang berbeda di DynamoDB. Anda juga dapat secara manual menambahkan data ke model data Anda atau mengimpor data dari MySQL.

Topik

- Menambahkan data sampel ke model data
- Mengimpor data sampel dari file CSV
- Melihat pola akses data
- Melihat semua tabel dalam model data menggunakan tampilan agregat
- Melakukan model data pada DynamoDB

Menambahkan data sampel ke model data

Dengan menambahkan data sampel ke model, Anda dapat menampilkan data saat memvisualisasikan model dan berbagai pola atau faset.

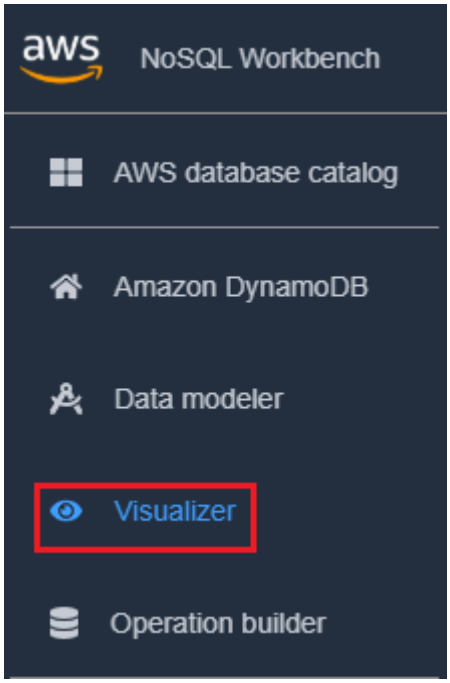
Ada dua cara untuk menambahkan data sampel. Salah satunya adalah menggunakan alat pembuatan auto data sampel kami. Yang lainnya adalah menambahkan data satu per satu.

Ikuti langkah-langkah ini untuk menambahkan data sampel ke model data menggunakan NoSQL Workbench untuk Amazon DynamoDB.

Untuk menghasilkan data sampel secara otomatis

Data sampel yang dibuat secara otomatis membantu Anda menghasilkan antara 1 hingga 5.000 baris data untuk segera digunakan. Anda dapat menentukan jenis data sampel granular untuk membuat data realistis berdasarkan kebutuhan desain dan pengujian Anda. Untuk memanfaatkan kemampuan menghasilkan data realistis, Anda perlu menentukan contoh format jenis data untuk atribut Anda di Pemodel data. Lihat Membuat model data baru untuk menentukan format jenis data sampel.

1. Di panel navigasi di sisi kiri, pilih ikon pemvisualisasi.

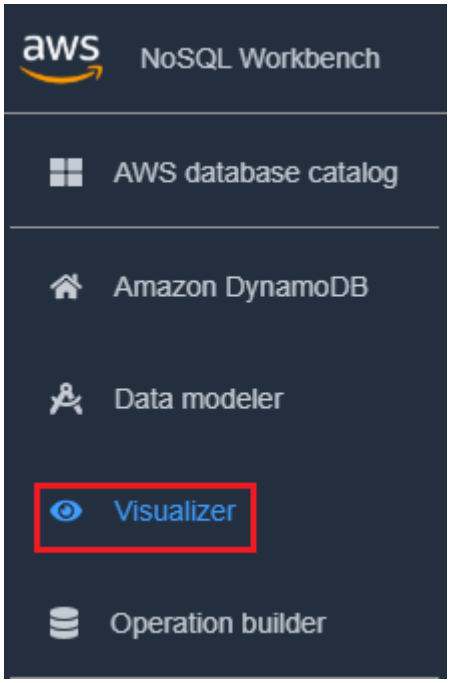


2. Di bagian pemvisualisasi, pilih model data dan pilih tabel.
3. Pilih daftar menurun Tindakan, dan pilih Tambahkan data sampel.

4. Masukkan jumlah atau item data sampel yang ingin Anda hasilkan, lalu pilih Konfirmasi.

Untuk menambahkan data sampel satu per satu

1. Di panel navigasi di sisi kiri, pilih ikon pemvisualisasi.

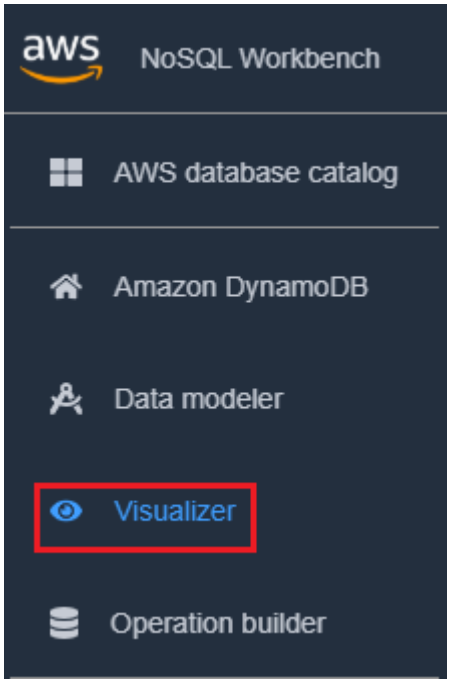


2. Di bagian pemvisualisasi, pilih model data dan pilih tabel.
3. Pilih daftar menurun Tindakan, dan pilih Edit data.

4. Pilih Tambahkan baris baru. Masukkan data sampel ke dalam kotak teks kosong, lalu pilih Tambahkan baris baru lagi untuk menambahkan baris tambahan. Setelah selesai, pilih Simpan perubahan.

Untuk menghapus data sampel

1. Di panel navigasi di sisi kiri, pilih ikon pemvisualisasi.



2. Di bagian pemvisualisasi, pilih model data dan pilih tabel.
3. Pilih daftar menurun Tindakan, dan pilih Edit data.

4. Pilih ikon hapus di samping setiap baris data yang ingin Anda hapus.

Mengimpor data sampel dari file CSV

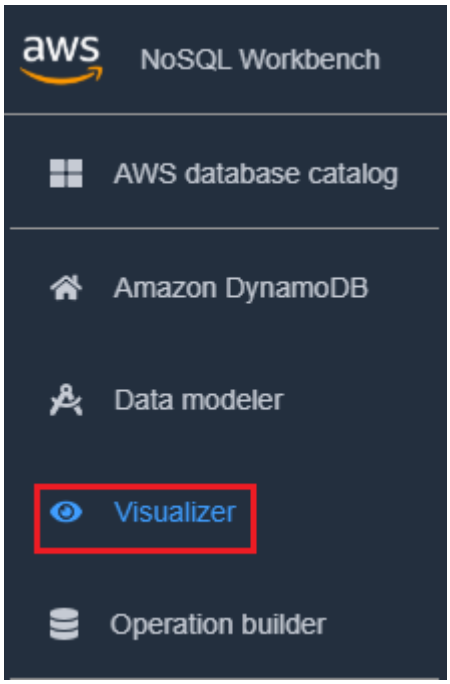
Jika Anda memiliki data sampel yang sudah ada sebelumnya dalam file CSV, Anda dapat mengimpornya ke NoSQL Workbench. Hal ini memungkinkan Anda untuk dengan cepat mengisi model Anda dengan data sampel tanpa harus memasukkannya baris demi baris.

Nama kolom dalam file CSV harus cocok dengan nama atribut dalam model data Anda, tetapi tidak harus dalam urutan yang sama. Misalnya, jika model data Anda memiliki atribut yang disebut `LoginAlias`,`FirstName`, dan`LastName`, kolom CSV Anda dapat berupa`LastName`,`FirstName`, dan`LoginAlias`.

Impor data dari file CSV dibatasi hingga 150 baris sekaligus.

Untuk mengimpor data dari file CSV ke NoSQL Workbench

1. Di panel navigasi di sisi kiri, pilih ikon pemvisualisasi.



2. Di bagian pemvisualisasi, pilih model data dan pilih tabel.
3. Pilih daftar menurun Tindakan, dan pilih Edit Data.
4. Pilih daftar menurun Tindakan lagi, dan pilih Impor file CSV.
5. Pilih file CSV Anda dan pilih Buka. Data dalam file CSV akan ditambahkan ke tabel Anda.

> **Note**
>
> Jika file CSV Anda berisi satu atau beberapa baris yang memiliki kunci yang sama dengan item yang sudah ada di tabel, Anda akan memiliki opsi untuk menimpa item yang sudah ada atau menambahkannya ke akhir tabel. Jika Anda memilih untuk menambahkan item, akhiran "-Copy" akan ditambahkan ke setiap kunci item duplikat untuk membedakan item duplikat dari item yang sudah ada di tabel.

Melihat pola akses data

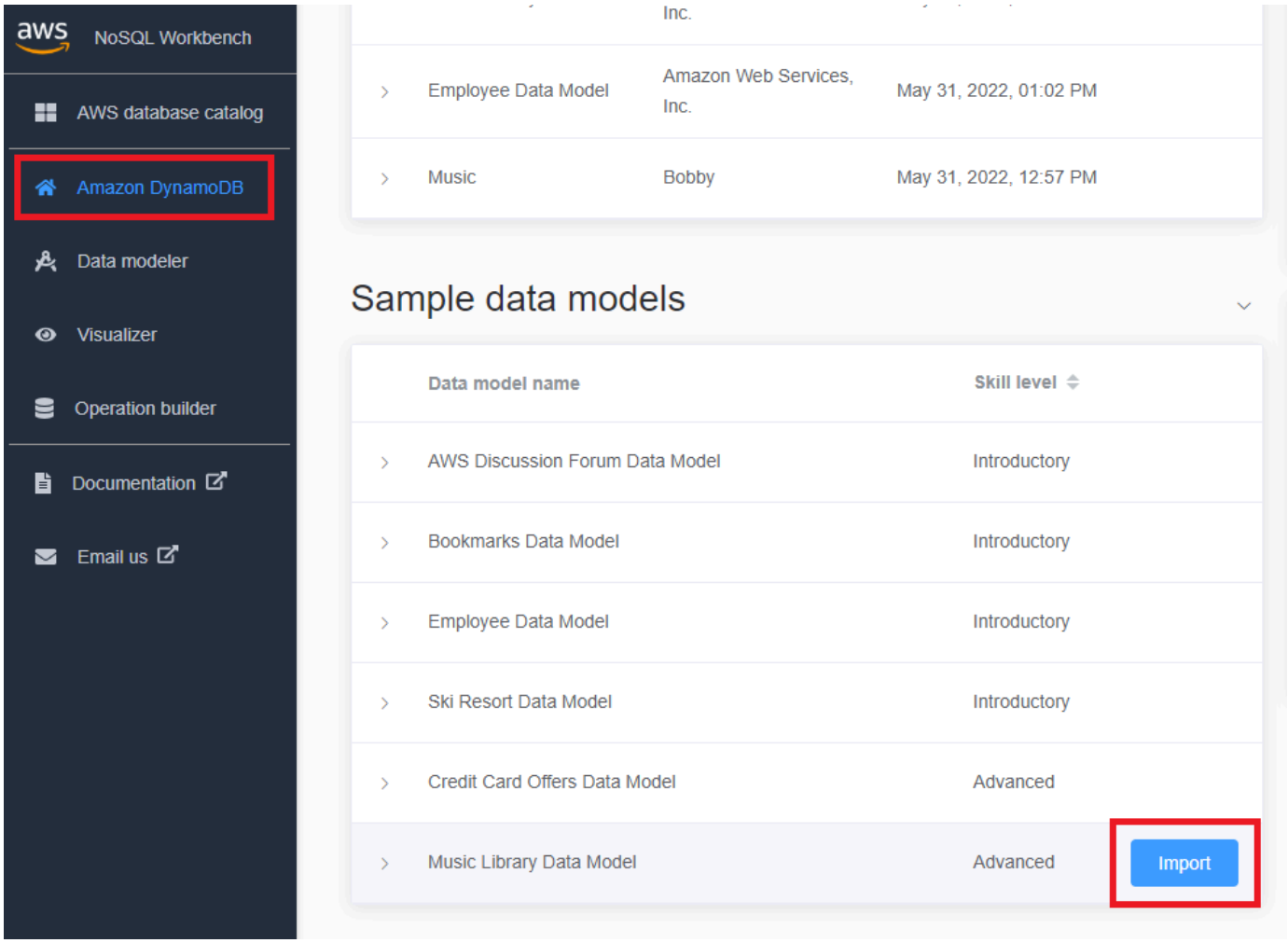
Di NoSQL Workbench, faset mewakili pola akses data aplikasi yang berbeda untuk Amazon DynamoDB. Faset dapat membantu Anda memvisualisasikan model data ketika beberapa jenis data diwakili oleh kunci urutan. Faset memberi Anda cara untuk melihat subset data dalam tabel, tanpa

harus melihat rekaman yang tidak memenuhi batasan faset. Faset dianggap sebagai alat pemodelan data visual, dan tidak ada sebagai konstruksi yang dapat digunakan di DynamoDB, karena aspek tersebut murni bantuan untuk pemodelan pola akses.

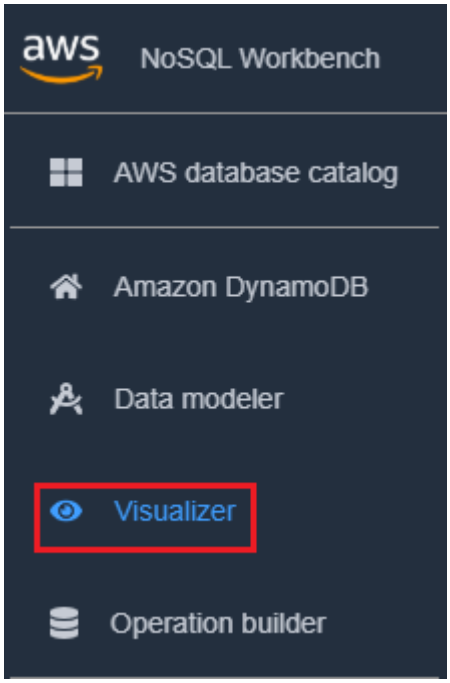
Untuk melihat contoh faset, Anda dapat mengimpor salah satu contoh model data kami dengan faset sebagai bagian dari templat model data.

Impor model data sampel

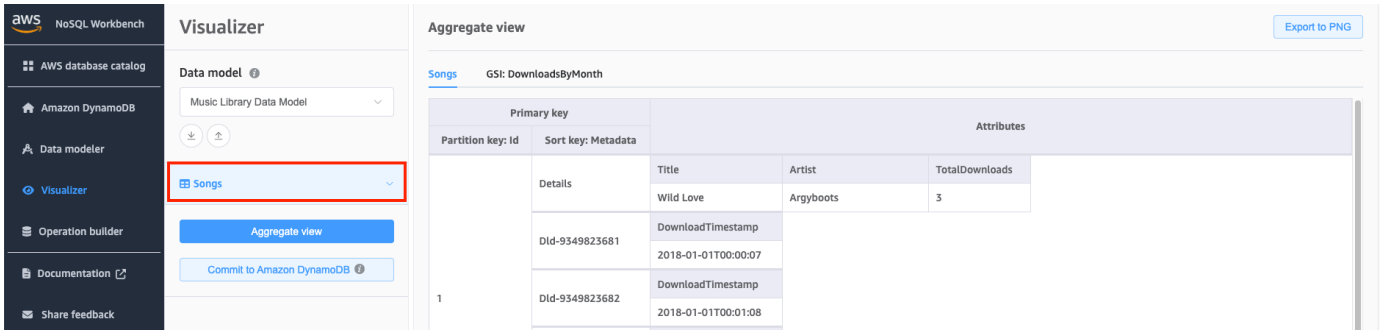
1. Di sebelah kiri, pilih Amazon DynamoDB.
2. Di bagian Model data sampel, arahkan penunjuk Anda ke Model Data Perpustakaan Musik dan pilih Impor.



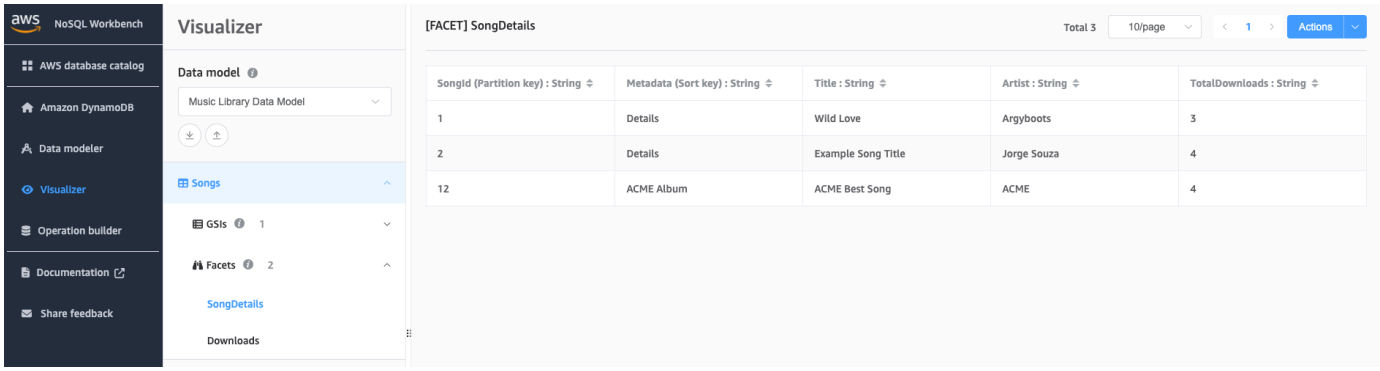
3. Di panel navigasi di sisi kiri, pilih ikon pemvisualisasi.



4. Pilih tabel Lagu untuk memperluasnya. Anda akan diperlihatkan tampilan agregat data Anda.



5. Pilih panah daftar menurun Faset untuk memperluas faset yang tersedia.
6. Pilih SongDetails faset untuk memvisualisasikan data dengan SongDetails faset yang diterapkan.



Anda juga dapat mengedit definisi faset menggunakan Pemodel Data. Untuk informasi selengkapnya, lihat Mengedit model data yang ada.

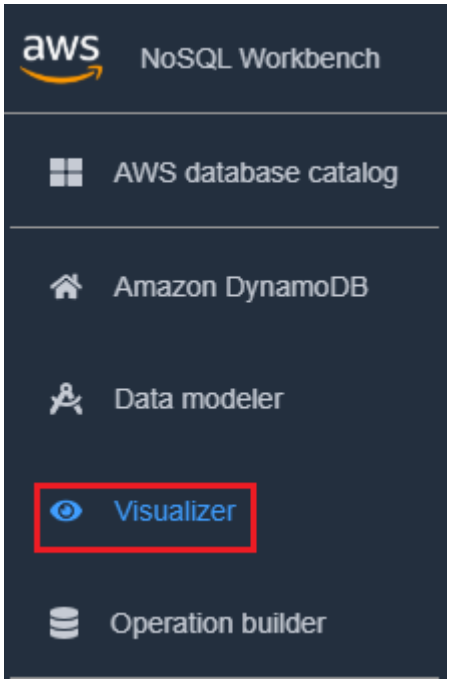
Melihat semua tabel dalam model data menggunakan tampilan agregat

Tampilan agregat di NoSQL Workbench untuk Amazon DynamoDB mewakili semua tabel dalam model data. Untuk setiap tabel, informasi berikut akan muncul:

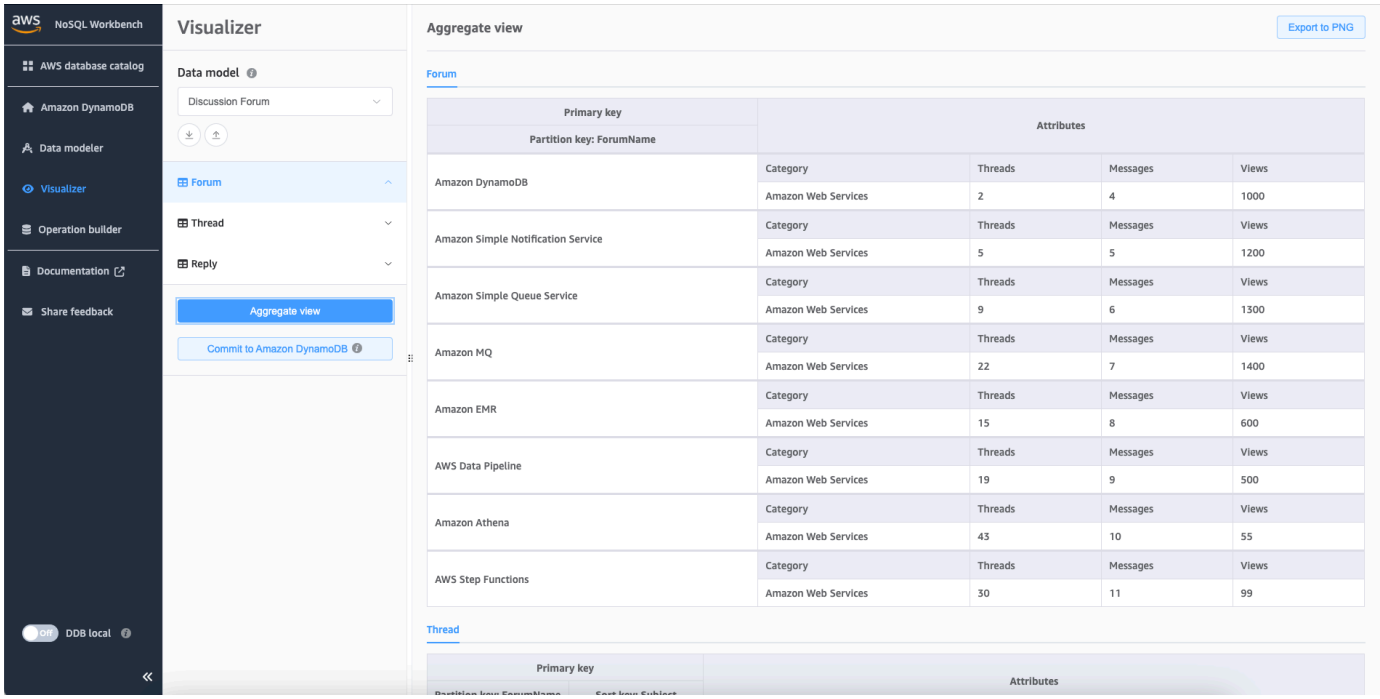
- Nama kolom tabel
- Data sampel
- Semua indeks sekunder global yang terkait dengan tabel. Informasi berikut ditampilkan untuk setiap indeks:
  - Nama kolom indeks
  - Data sampel

Untuk melihat semua informasi tabel

1. Di panel navigasi di sisi kiri, pilih ikon pemvisualisasi.



2. Di pemvisualisasi, pilih Tampilan agregat.



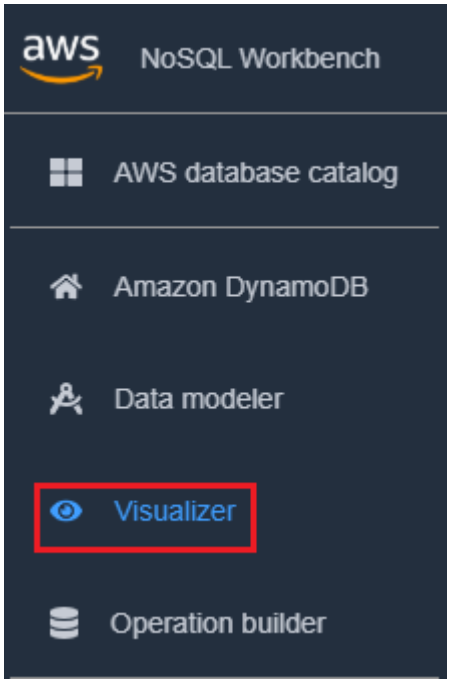
Melakukan model data pada DynamoDB

Bila Anda puas dengan model data, Anda dapat melakukan model ke Amazon DynamoDB.

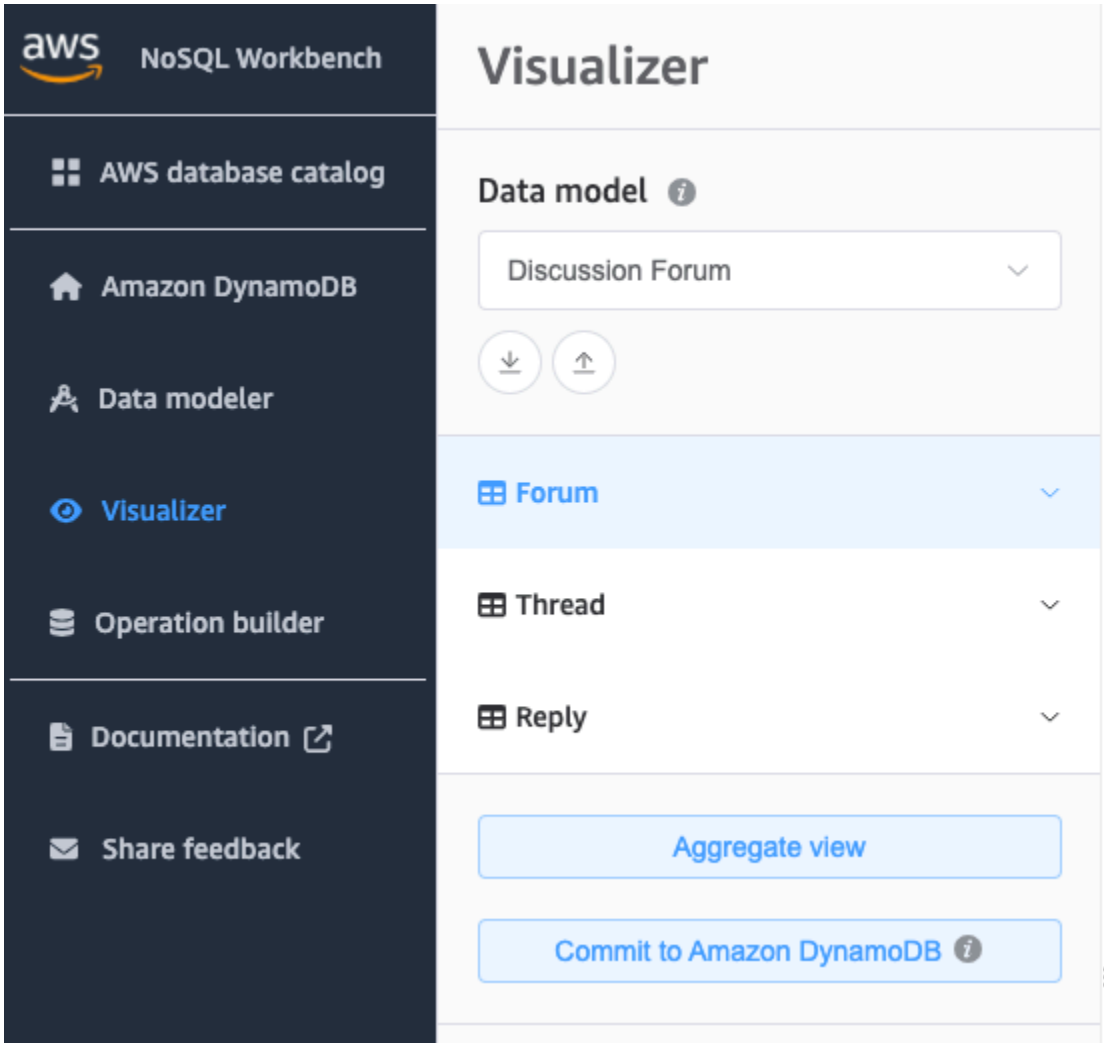
> **Note**
>
> - Tindakan ini menghasilkan pembuatan sumber daya sisi server AWS untuk tabel dan indeks sekunder global yang diwakili dalam model data.
> - Tabel dibuat dengan ciri-ciri sebagai berikut:
>   - Penskalaan otomatis diatur ke 70 persen pemanfaatan target.
>   - Kapasitas yang disediakan diatur ke 5 unit kapasitas baca dan 5 unit kapasitas tulis.
> - Indeks sekunder global dibuat dengan kapasitas yang disediakan sebesar 10 unit kapasitas baca dan 5 unit kapasitas tulis.

Untuk menerapkan model data ke DynamoDB

1. Di panel navigasi di sisi kiri, pilih ikon pemvisualisasi.



2. Pilih Menerapkan pada DynamoDB.



3. Pilih koneksi yang sudah ada, atau buat koneksi baru dengan memilih tab Tambahkan koneksi jarak jauh baru.

   - Untuk menambahkan koneksi baru, tentukan informasi berikut:
     - Alias Akun
     - AWS Wilayah
     - ID kunci akses
     - Kunci akses rahasia

     Untuk informasi selengkapnya tentang cara mendapatkan kunci akses, lihat Mendapatkan kunci AWS akses.
   - Anda dapat menentukan hal berikut secara opsional:
     - Token sesi

- ARN peran IAM
- Jika Anda tidak ingin mendaftar akun tingkat gratis, dan lebih memilih menggunakan DynamoDB lokal (versi yang dapat diunduh):
  1. Pilih tab Tambahkan koneksi lokal DynamoDB baru.
  2. Tentukan Nama koneksi dan Port.

4. Pilih Terapkan.

> **Note**
>
> Jika Anda menginstal DynamoDB lokal sebagai bagian dari pengaturan NoSQL Workbench, Anda harus mengaktifkan DynamoDB lokal dengan menggunakan tombol Server DynamoDB lokal di kiri bawah layar NoSQL Workbench. Lihat Menginstal NoSQL Workbench untuk DynamoDB untuk informasi selengkapnya tentang tombol ini.

Menjelajahi set data dan membangun operasi dengan NoSQL Workbench

NoSQL Workbench for Amazon DynamoDB menyediakan antarmuka pengguna grafis yang kaya bagi Anda untuk mengembangkan dan menguji kueri. Anda dapat menggunakan pembangun operasi di NoSQL Workbench untuk melihat, menjelajahi, dan mengkueri dataset. Builder operasi terstruktur mendukung ekspresi proyeksi, ekspresi kondisi, dan menghasilkan kode sampel dalam berbagai bahasa. Anda dapat langsung mengkloning tabel dari satu akun Amazon DynamoDB ke akun lain di Wilayah yang berbeda. Anda juga dapat langsung mengkloning tabel antara DynamoDB lokal dan akun Amazon DynamoDB untuk menyalin skema kunci tabel Anda dengan lebih cepat (dan skema dan item GSI opsional) di antara lingkungan pengembangan Anda. Anda dapat menyimpan sebanyak 50 operasi data DynamoDB di pembuat operasi.

Topik

- Menghubungkan ke kumpulan data langsung
- Membangun operasi kompleks
- Kloning tabel dengan NoSQL Workbench
- Mengekspor data ke file CSV

Menghubungkan ke kumpulan data langsung

Untuk terhubung ke tabel Amazon DynamoDB Anda dengan NoSQL Workbench, Anda harus terlebih dahulu terhubung ke akun Anda. AWS

Untuk menambahkan koneksi ke basis data Anda

1. Di NoSQL Workbench, di panel navigasi di sisi kiri, pilih ikon Pembangun operasi.
2. Pilih Tambahkan koneksi.
3. Tentukan informasi berikut:

   - Alias akun
   - AWS Wilayah
   - ID kunci akses
   - Kunci akses rahasia

   Untuk informasi selengkapnya tentang cara mendapatkan kunci akses, lihat Mendapatkan kunci AWS akses.

   Anda dapat menentukan hal berikut secara opsional:

   - Token sesi
   - ARN peran IAM
4. Pilih Hubungkan.

   Jika Anda tidak ingin mendaftar akun tingkat gratis, dan lebih memilih menggunakan DynamoDB lokal (versi yang dapat diunduh):

   a. Pilih tab Lokal pada layar koneksi.
   b. Tentukan informasi berikut:

      - Nama koneksi
      - Port

   c. Pilih tombol hubungkan.

> **Note**
>
> Untuk terhubung ke DynamoDB lokal, jalankan DynamoDB lokal secara manual menggunakan terminal Anda (lihat menerapkan DynamoDB lokal di komputer Anda) atau luncurkan DynamoDB lokal secara langsung menggunakan sakelar lokal DDB di menu navigasi NoSQL Workbench. Pastikan port koneksi sama dengan port lokal DynamoDB Anda.

5. Pada koneksi yang dibuat, pilih Buka.

Setelah menghubungkan ke basis data DynamoDB Anda, daftar tabel yang tersedia muncul di panel kiri. Pilih salah satu tabel untuk mengembalikan sampel data yang tersimpan dalam tabel.

Anda sekarang dapat menjalankan kueri terhadap tabel yang dipilih.

Untuk menjalankan kueri pada tabel, lihat bagian selanjutnya tentang operasi bangunan lihatMembangun operasi kompleks.

Membangun operasi kompleks

Pembangun operasi di NoSQL Workbench untuk Amazon DynamoDB menyediakan antarmuka visual di mana Anda dapat melakukan operasi bidang data yang kompleks. Ini termasuk dukungan untuk ekspresi proyeksi dan ekspresi syarat. Setelah Anda membangun operasi, Anda dapat menyimpannya untuk digunakan nanti (hingga 50 operasi dapat disimpan). Anda kemudian dapat menelusuri daftar operasi bidang data yang sering digunakan di menu Operasi Tersimpan, dan menggunakannya untuk mengisi dan membangun operasi baru secara otomatis. Anda juga dapat menghasilkan kode sampel untuk operasi ini, dalam beberapa bahasa.

NoSQL Workbench mendukung pembangunan PartiQL untuk pernyataan DynamoDB, yang memungkinkan Anda untuk berinteraksi dengan DynamoDB menggunakan bahasa kueri yang kompatibel dengan SQL. NoSQL Workbench juga mendukung membangun operasi API CRUD DynamoDB.

Untuk menggunakan NoSQL Workbench untuk membangun operasi, di panel navigasi di sisi kiri, pilih ikon Pembangun operasi.

Topik

- Membangun pernyataan PartiQL
- Membangun operasi API

Membangun pernyataan PartiQL

Untuk menggunakan NoSQL Workbench untuk membangun PartiQL untuk pernyataan DynamoDB, pilih editor PartiQL di dekat bagian atas NoSQL Workbench UI.

Anda dapat membangun jenis pernyataan PartiQL berikut dalam pembangun operasi.

Topik

- Pernyataan singleton
- Transaksi
- Batch

Pernyataan singleton

Untuk menjalankan atau menghasilkan kode untuk pernyataan PartiQL, lakukan hal berikut.

1. Pilih editor PartiQL di dekat bagian atas jendela.
2. Masukkan pernyataan PartiQL yang valid.
3. Jika pernyataan Anda menggunakan parameter:
    a. Pilih Parameter permintaan opsional.
    b. Pilih Tambahkan parameter baru.
    c. Masukkan jenis dan nilai atribut.
    d. Jika Anda ingin menambahkan parameter tambahan, ulangi langkah b dan c.
4. Jika Anda ingin membuat kode, pilih Buat kode.

    Pilih bahasa yang Anda inginkan dari tab yang ditampilkan. Sekarang Anda dapat menyalin kode ini dan menggunakannya dalam aplikasi Anda.
5. Jika Anda ingin operasi segera dijalankan, pilih Jalankan.
6. Jika Anda ingin menyimpan operasi ini untuk digunakan nanti, pilih Simpan operasi. Kemudian masukkan nama untuk operasi Anda dan pilih Simpan.

Transaksi

Untuk menjalankan atau menghasilkan kode untuk transaksi PartiQL, lakukan hal berikut.

1. Pilih Parti QLTransaction dari dropdown Operasi Lainnya.
2. Pilih Tambahkan pernyataan baru.
3. Masukkan pernyataan PartiQL yang valid.

   > **Note**
   >
   > Operasi baca dan tulis tidak didukung dalam permintaan transaksi PartiQL yang sama. Pernyataan PILIH tidak dapat berada dalam permintaan yang sama dengan pernyataan SISIPKAN, PERBARUI, dan HAPUS. Lihat Melakukan transaksi dengan PartiQL untuk DynamoDB untuk detail selengkapnya.

4. Jika pernyataan Anda menggunakan parameter
   a. Pilih Parameter permintaan opsional.
   b. Pilih Tambahkan parameter baru.
   c. Masukkan jenis dan nilai atribut.
   d. Jika Anda ingin menambahkan parameter tambahan, ulangi langkah b dan c.
5. Jika Anda ingin menambahkan lebih banyak pernyataan, ulangi langkah 2 sampai 4.
6. Jika Anda ingin membuat kode, pilih Buat kode.

   Pilih bahasa yang Anda inginkan dari tab yang ditampilkan. Sekarang Anda dapat menyalin kode ini dan menggunakannya dalam aplikasi Anda.
7. Jika Anda ingin operasi segera dijalankan, pilih Jalankan.
8. Jika Anda ingin menyimpan operasi ini untuk digunakan nanti, pilih Simpan operasi. Kemudian masukkan nama untuk operasi Anda dan pilih Simpan.

Batch

Untuk menjalankan atau menghasilkan kode untuk batch PartiQL, lakukan hal berikut.

1. Pilih Parti QLBatch dari dropdown Operasi Lainnya.
2. Pilih Tambahkan pernyataan baru.

3. Masukkan pernyataan PartiQL yang valid.

> **Note**
>
> Operasi baca dan tulis tidak didukung dalam permintaan batch PartiQL yang sama, yang berarti pernyataan PILIH tidak dapat berada dalam permintaan yang sama dengan pernyataan SISIPKAN, PERBARUI, dan HAPUS. Operasi tulis untuk item yang sama tidak diperbolehkan. Seperti halnya BatchGetItem operasi, hanya operasi baca tunggal yang didukung. Operasi pemindaian dan kueri tidak didukung. Lihat Menjalankan operasi batch dengan PartiQL untuk DynamoDB untuk detail selengkapnya.

4. Jika pernyataan Anda menggunakan parameter:

   a. Pilih Parameter permintaan opsional.

   b. Pilih Tambahkan parameter baru.

   c. Masukkan jenis dan nilai atribut.

   d. Jika Anda ingin menambahkan parameter tambahan, ulangi langkah b dan c.

5. Jika Anda ingin menambahkan lebih banyak pernyataan, ulangi langkah 2 sampai 4.

6. Jika Anda ingin membuat kode, pilih Buat kode.

   Pilih bahasa yang Anda inginkan dari tab yang ditampilkan. Sekarang Anda dapat menyalin kode ini dan menggunakannya dalam aplikasi Anda.

7. Jika Anda ingin operasi segera dijalankan, pilih Jalankan.

8. Jika Anda ingin menyimpan operasi ini untuk digunakan nanti, pilih Simpan operasi. Kemudian masukkan nama untuk operasi Anda dan pilih Simpan.

Membangun operasi API

Untuk menggunakan NoSQL Workbench untuk membangun APIs DynamoDB CRUD, pilih Operation builder dari sebelah kiri antarmuka pengguna NoSQL Workbench.

Kemudian pilih Buka dan pilih koneksi.

Anda dapat melakukan operasi berikut di pembangun operasi.

- Hapus Tabel
- Buat Tabel

- Perbarui Tabel

- Put Item
- Perbarui Item
- Hapus Item
- Kueri
- Pemindaian
- Transaksi Get Item
- Transaksi Write Item

Hapus tabel

Untuk menjalankan `Delete Table` operasi, lakukan hal berikut.

1. Temukan tabel yang ingin Anda hapus dari bagian Tabel.
2. Pilih Hapus Tabel dari menu elipsis horizontal.
3. Konfirmasikan bahwa Anda ingin menghapus tabel dengan memasukkan nama Tabel.
4. Pilih Hapus.

Untuk informasi selengkapnya tentang operasi ini, lihat Menghapus tabel di Referensi API Amazon DynamoDB.

Hapus GSI

Untuk menjalankan `Delete GSI` operasi, lakukan hal berikut.

1. Temukan GSI dari tabel yang ingin Anda hapus dari bagian Tabel.
2. Pilih Hapus GSI dari menu elipsis horizontal.
3. Konfirmasikan bahwa Anda ingin menghapus GSI dengan memasukkan nama GSI.
4. Pilih Hapus.

Untuk informasi selengkapnya tentang operasi ini, lihat Menghapus tabel di Referensi API Amazon DynamoDB.

Membuat tabel

Untuk menjalankan `Create Table` operasi, lakukan hal berikut.

1. Pilih ikon + di sebelah bagian Tabel.
2. Masukkan nama tabel yang diinginkan.
3. Buat kunci partisi.
4. Opsional: buat kunci sortir.
5. Untuk menyesuaikan pengaturan kapasitas, dan hapus centang pada kotak di samping Gunakan pengaturan kapasitas default.
   - Anda sekarang dapat memilih Kapasitas sesuai permintaan atau Yang disediakan.

     Dengan Provisioned dipilih, Anda dapat mengatur unit kapasitas baca dan tulis minimum dan maksimum. Anda juga dapat mengaktifkan atau menonaktifkan penskalaan otomatis.
   - Jika tabel saat ini disetel ke On-Demand, Anda tidak akan dapat menentukan throughput yang disediakan.
   - Jika Anda beralih dari On-Demand ke Provisioned throughput, maka Autoscaling akan secara otomatis diterapkan ke semua GSIs dengan: min: 1, max: 10; target: 70%.
6. Pilih Lewati GSIs dan buat untuk membuat tabel ini tanpa GSI. Secara opsional, Anda dapat memilih Berikutnya untuk membuat GSI dengan tabel baru ini.

Untuk informasi selengkapnya tentang operasi ini, lihat Membuat tabel dalam Referensi API Amazon DynamoDB.

Buat GSI

Untuk menjalankan `Create GSI` operasi, lakukan hal berikut.

1. Temukan tabel yang ingin Anda tambahkan GSI.
2. Dari menu elipsis horizontal, pilih Buat GSI.
3. Beri nama GSI Anda di bawah nama Indeks.
4. Buat kunci partisi.
5. Opsional: buat kunci sortir.
6. Pilih opsi jenis proyeksi dari dropdown.
7. Pilih Buat GSI.

Untuk informasi selengkapnya tentang operasi ini, lihat Membuat tabel dalam Referensi API Amazon DynamoDB.

Perbarui Tabel

Untuk memperbarui pengaturan kapasitas untuk tabel dengan `Update Table` operasi, lakukan hal berikut.

1. Temukan tabel yang ingin Anda perbarui pengaturan kapasitas.
2. Dari menu elipsis horizontal, pilih Perbarui pengaturan kapasitas.
3. Pilih kapasitas Provisioned atau On-Demand.

   Dengan Provisioned dipilih, Anda dapat mengatur unit kapasitas baca dan tulis minimum dan maksimum. Anda juga dapat mengaktifkan atau menonaktifkan penskalaan otomatis.
4. Pilih Perbarui.

Untuk informasi selengkapnya tentang operasi ini, lihat Memperbarui tabel di Referensi API Amazon DynamoDB.

Perbarui GSI

Untuk memperbarui pengaturan kapasitas untuk GSI dengan `Update Table` operasi, lakukan hal berikut.

> **Note**
>
> Secara default, indeks sekunder global mewarisi pengaturan kapasitas tabel dasar. Indeks sekunder global dapat memiliki mode kapasitas yang berbeda hanya ketika tabel dasar dalam mode kapasitas yang disediakan. Saat membuat indeks sekunder global pada tabel mode yang disediakan, Anda harus menentukan unit kapasitas baca dan tulis untuk beban kerja yang diharapkan pada indeks tersebut. Untuk informasi selengkapnya, lihat Pertimbangan Throughput yang disediakan untuk Indeks Sekunder Global.

1. Temukan GSI yang ingin Anda perbarui pengaturan kapasitas.
2. Dari menu elipsis horizontal, pilih Perbarui pengaturan kapasitas.
3. Anda sekarang dapat memilih Kapasitas sesuai permintaan atau Yang disediakan.

Dengan Provisioned dipilih, Anda dapat mengatur unit kapasitas baca dan tulis minimum dan maksimum. Anda juga dapat mengaktifkan atau menonaktifkan penskalaan otomatis.

4. Pilih Perbarui.

Untuk informasi selengkapnya tentang operasi ini, lihat Memperbarui tabel di Referensi API Amazon DynamoDB.

Put item

Anda membuat item dengan menggunakan `Put Item` operasi. Untuk menjalankan atau menghasilkan kode untuk suatu operasi `Put Item`, lakukan hal berikut.

1. Temukan tabel tempat Anda ingin membuat item.
2. Dari dropdown Tindakan, pilih Buat item.
3. Masukkan nilai kunci partisi.
4. Masukkan nilai kunci urutan, jika ada.
5. Jika Anda ingin menambahkan atribut non-kunci, lakukan hal berikut:
   a. Pilih + Tambahkan atribut lainnya.
   b. Tentukan Nama atribut, Jenis, dan Nilai.
6. Jika ekspresi kondisi harus dipenuhi agar operasi `Put Item` berhasil, lakukan hal berikut:
   a. Pilih Kondisi.
   b. Tentukan nama atribut, operator perbandingan, jenis atribut, dan nilai atribut.
   c. Jika kondisi lain dibutuhkan, pilih Kondisi lagi.

   Untuk informasi selengkapnya, lihat Contoh CLI ekspresi kondisi DynamoDB.
7. Jika Anda ingin membuat kode, pilih Buat kode.

   Pilih bahasa yang Anda inginkan dari tab yang ditampilkan. Sekarang Anda dapat menyalin kode ini dan menggunakannya dalam aplikasi Anda.
8. Jika Anda ingin operasi segera dijalankan, pilih Jalankan.
9. Jika Anda ingin menyimpan operasi ini untuk digunakan nanti, pilih Simpan operasi, lalu masukkan nama untuk operasi Anda dan pilih Simpan.

Untuk informasi selengkapnya tentang operasi ini, lihat PutItemdi Referensi Amazon DynamoDB API.

Perbarui item

Untuk menjalankan atau menghasilkan kode untuk operasi `Update Item`, lakukan hal berikut:

1. Temukan tabel tempat Anda ingin memperbarui item.
2. Pilih item.
3. Masukkan nama atribut dan nilai atribut untuk ekspresi yang dipilih.
4. Jika Anda ingin menambahkan lebih banyak ekspresi, pilih ekspresi lain di daftar tarik-turun Perbarui Ekspresi, lalu pilih ikon +.
5. Jika ekspresi kondisi harus dipenuhi agar operasi `Update Item` berhasil, lakukan hal berikut:

   a. Pilih Kondisi.

   b. Tentukan nama atribut, operator perbandingan, jenis atribut, dan nilai atribut.

   c. Jika kondisi lain dibutuhkan, pilih Kondisi lagi.

   Untuk informasi selengkapnya, lihat Contoh CLI ekspresi kondisi DynamoDB.
6. Jika Anda ingin membuat kode, pilih Buat kode.

   Pilih tab untuk bahasa yang Anda inginkan. Sekarang Anda dapat menyalin kode ini dan menggunakannya dalam aplikasi Anda.
7. Jika Anda ingin operasi segera dijalankan, pilih Jalankan.
8. Jika Anda ingin menyimpan operasi ini untuk digunakan nanti, pilih Simpan operasi, lalu masukkan nama untuk operasi Anda dan pilih Simpan.

Untuk informasi selengkapnya tentang operasi ini, lihat UpdateItemdi Referensi Amazon DynamoDB API.

Hapus item

Untuk menjalankan `Delete Item` operasi, lakukan hal berikut.

1. Temukan tabel tempat Anda ingin menghapus item.
2. Pilih item.
3. Dari dropdown Tindakan, pilih Hapus item.

4. Konfirmasikan bahwa Anda ingin menghapus item dengan memilih Hapus.

Untuk informasi selengkapnya tentang operasi ini, lihat DeleteItemdi Referensi Amazon DynamoDB API.

Item duplikat

Anda dapat menduplikasi item dengan membuat item baru dengan atribut yang sama. Untuk menduplikasi item, lakukan hal berikut.

1. Temukan tabel tempat Anda ingin menduplikasi item.
2. Pilih item.
3. Dari dropdown Tindakan, pilih item Duplikat.
4. Tentukan kunci partisi baru.
5. Tentukan kunci sortir baru (jika perlu).
6. Pilih Jalankan.

Untuk informasi selengkapnya tentang operasi ini, lihat DeleteItemdi Referensi Amazon DynamoDB API.

Kueri

Untuk menjalankan atau menghasilkan kode untuk suatu operasi `Query`, lakukan hal berikut.

1. Pilih Query dari bagian atas NoSQL Workbench UI.
2. Tentukan nilai kunci partisi.
3. Jika kunci urutan diperlukan untuk operasi `Query`:
    a. Pilih Kunci urutan.
    b. Tentukan operator perbandingan, dan nilai atribut.
4. Pilih Kueri untuk menjalankan operasi kueri ini. Jika lebih banyak opsi diperlukan, centang kotak centang Opsi lainnya dan lanjutkan dengan langkah-langkah berikut.
5. Jika tidak semua atribut harus dikembalikan bersama hasil operasi, pilih Ekspresi proyeksi.
6. Pilih ikon +.
7. Masukkan atribut yang akan dikembalikan dengan hasil kueri.

8. Jika diperlukan lebih banyak atribut, pilih + .
9. Jika ekspresi kondisi harus dipenuhi agar operasi `Query` berhasil, lakukan hal berikut:

    a. Pilih Kondisi.

    b. Tentukan nama atribut, operator perbandingan, jenis atribut, dan nilai atribut.

    c. Jika kondisi lain dibutuhkan, pilih Kondisi lagi.

    Untuk informasi selengkapnya, lihat Contoh CLI ekspresi kondisi DynamoDB.
10. Jika Anda ingin membuat kode, pilih Buat kode.

    Pilih tab untuk bahasa yang Anda inginkan. Sekarang Anda dapat menyalin kode ini dan menggunakannya dalam aplikasi Anda.
11. Jika Anda ingin operasi segera dijalankan, pilih Jalankan.
12. Jika Anda ingin menyimpan operasi ini untuk digunakan nanti, pilih Simpan operasi, lalu masukkan nama untuk operasi Anda dan pilih Simpan.

Untuk informasi selengkapnya tentang operasi ini, lihat Kueri dalam Referensi API Amazon DynamoDB.

Scan

Untuk menjalankan atau menghasilkan kode untuk suatu operasi `Scan`, lakukan hal berikut.

1. Pilih Pindai dari bagian atas NoSQL Workbench UI.
2. Pilih tombol Pindai untuk melakukan operasi pemindaian dasar ini. Jika lebih banyak opsi diperlukan, centang kotak centang Opsi lainnya dan lanjutkan dengan langkah-langkah berikut.
3. Tentukan nama atribut untuk memfilter hasil pemindaian Anda.
4. Jika tidak semua atribut harus dikembalikan bersama hasil operasi, pilih Ekspresi proyeksi.
5. Jika ekspresi kondisi harus dipenuhi agar operasi pemindaian berhasil, lakukan hal berikut:

    a. Pilih Kondisi.

    b. Tentukan nama atribut, operator perbandingan, jenis atribut, dan nilai atribut.

    c. Jika kondisi lain dibutuhkan, pilih Kondisi lagi.

    Untuk informasi selengkapnya, lihat Contoh CLI ekspresi kondisi DynamoDB.

6. Jika Anda ingin membuat kode, pilih Buat kode.

   Pilih tab untuk bahasa yang Anda inginkan. Sekarang Anda dapat menyalin kode ini dan menggunakannya dalam aplikasi Anda.
7. Jika Anda ingin operasi segera dijalankan, pilih Jalankan.
8. Jika Anda ingin menyimpan operasi ini untuk digunakan nanti, pilih Simpan operasi, lalu masukkan nama untuk operasi Anda dan pilih Simpan.

TransactGetItems

Untuk menjalankan atau menghasilkan kode untuk suatu operasi `TransactGetItems`, lakukan hal berikut.

1. Dari dropdown More operations di bagian atas NoSQL Workbench UI, pilih. TransactGetItems
2. Pilih ikon + di dekat TransactGetItem.
3. Tentukan kunci partisi.
4. Tentukan kunci sortir (jika perlu).
5. Pilih Jalankan untuk melakukan operasi, Simpan operasi untuk menyimpannya, atau Hasilkan kode untuk menghasilkan kode untuknya.

Untuk informasi selengkapnya tentang transaksi, lihat Amazon DynamoDB Transactions.

TransactWriteItems

Untuk menjalankan atau menghasilkan kode untuk operasi `TransactWriteItems`, lakukan hal berikut.

1. Dari dropdown More operations di bagian atas NoSQL Workbench UI, pilih. TransactWriteItems
2. Pilih operasi dari dropdown Tindakan.
3. Pilih ikon + di dekat TransactWriteItem.
4. Di dropdown Tindakan, pilih operasi yang ingin Anda lakukan.
   - Untuk `DeleteItem`, ikuti petunjuk untuk operasi Hapus item.
   - Untuk `PutItem`, ikuti petunjuk untuk operasi Put item.
   - Untuk `UpdateItem`, ikuti petunjuk untuk operasi Perbarui item.

Untuk mengubah urutan tindakan, pilih tindakan dalam daftar di sisi kiri, lalu pilih panah atas atau bawah untuk memindahkannya ke atas atau bawah dalam daftar.

Untuk menghapus tindakan, pilih tindakan dalam daftar, kemudian pilih ikon Hapus (tempat sampah).

5. Pilih Jalankan untuk melakukan operasi, Simpan operasi untuk menyimpannya, atau Hasilkan kode untuk menghasilkan kode untuknya.

Untuk informasi selengkapnya tentang transaksi, lihat Amazon DynamoDB Transactions.

Kloning tabel dengan NoSQL Workbench

Tabel kloning akan menyalin skema kunci tabel (dan skema dan item GSI opsional) di antara lingkungan pengembangan Anda. Anda dapat mengkloning tabel antara DynamoDB lokal ke akun Amazon DynamoDB, dan bahkan mengkloning tabel dari satu akun ke akun lain di Wilayah yang berbeda untuk eksperimen yang lebih cepat.

Untuk mengkloning tabel

1. Di Operation Builder, pilih koneksi Anda dan Wilayah (Pilihan wilayah tidak tersedia untuk DynamoDB lokal).
2. Setelah Anda terhubung ke DynamoDB, telusuri tabel Anda dan pilih tabel yang ingin Anda kloning.
3. Dari menu elipsis horizontal, pilih opsi Clone.
4. Masukkan detail tujuan klon Anda:
    a. Pilih koneksi.
    b. Pilih Wilayah (Wilayah tidak tersedia untuk DynamoDB lokal).
    c. Masukkan nama tabel baru.
    d. Pilih opsi klon:
        i. Skema kunci dipilih secara default dan tidak dapat tidak dipilih. Secara default, kloning tabel akan menyalin kunci utama Anda dan mengurutkan kunci jika tersedia.
        ii. Skema GSI dipilih secara default jika tabel Anda yang akan dikloning memiliki GSI. Mengkloning tabel akan menyalin kunci utama GSI Anda dan kunci pengurutan jika

tersedia. Anda memiliki opsi untuk membatalkan pilihan skema GSI untuk melewati kloning skema GSI. Mengkloning tabel akan menyalin pengaturan kapasitas tabel dasar Anda sebagai pengaturan kapasitas GSI. Anda dapat menggunakan `UpdateTable` operasi di Operation Builder untuk memperbarui pengaturan kapasitas GSI tabel setelah kloning selesai.

5. Masukkan jumlah item yang akan dikloning. Untuk hanya mengkloning skema kunci dan secara opsional skema GSI, Anda dapat menyimpan Item untuk mengkloning nilai pada 0. Jumlah maksimum item yang dapat dikloning adalah 5000.
6. Pilih mode kapasitas:
   a. Mode sesuai permintaan dipilih secara default. DynamoDB on-demand pay-per-request menawarkan harga untuk permintaan baca dan tulis sehingga Anda hanya membayar untuk apa yang Anda gunakan. Untuk mempelajari lebih lanjut, lihat Mode DynamoDB On-Demand.
   b. Mode yang disediakan memungkinkan Anda menentukan jumlah pembacaan dan penulisan per detik yang Anda perlukan untuk aplikasi Anda. Anda dapat menggunakan penskalaan otomatis untuk menyesuaikan kapasitas yang disediakan tabel Anda secara otomatis sebagai respons terhadap perubahan lalu lintas. Untuk mempelajari selengkapnya, lihat DynamoDB Provisioned mode.
7. Pilih Clone untuk memulai kloning.
8. Proses kloning akan berjalan di latar belakang. Tab Operation builder akan menampilkan notifikasi ketika ada perubahan dalam status tabel kloning. Anda dapat mengakses status ini dengan memilih tab Operation builder dan kemudian memilih tombol panah. Tombol panah terletak di widget status tabel kloning yang terletak di dekat bagian bawah bilah sisi menu.

Mengekspor data ke file CSV

Anda dapat mengekspor hasil kueri dari Pembangun Operasi ke file CSV. Tindakan ini memungkinkan Anda untuk memuat data ke dalam spreadsheet atau memprosesnya menggunakan bahasa pemrograman pilihan Anda.

Mengekspor ke CSV

1. Di Pembangun Operasi, jalankan operasi pilihan Anda, seperti Pindai atau Kueri.

> **Note**
>
> - Anda hanya dapat mengekspor hasil dari operasi API baca dan pernyataan PartiQL ke file CSV. Anda tidak dapat mengekspor hasil dari pernyataan baca transaksi.
> - Saat ini, Anda dapat mengekspor hasil satu halaman pada satu waktu ke file CSV. Jika ada beberapa halaman hasil, Anda harus mengekspor setiap halaman satu per satu.

2. Pilih item yang ingin Anda ekspor dari hasil.
3. Di dropdown Tindakan, pilih Ekspor sebagai CSV.
4. Pilih nama file dan lokasi untuk file ZIP, lalu pilih Simpan.

Model data sampel untuk NoSQL Workbench

Beranda untuk pemodel dan pemvisualisasi menampilkan sejumlah model sampel yang dikirimkan dengan NoSQL Workbench. Bagian ini menjelaskan model tersebut dan potensi penggunaannya.

Topik

- Model data karyawan
- Model data forum diskusi
- Model data pustaka musik
- Model data resor ski
- Model data penawaran kartu kredit
- Model data bookmark

Model data karyawan

Model data ini adalah model perkenalan. Hal ini mewakili detail dasar karyawan seperti alias unik, nama depan, nama belakang, penunjukan, manajer, dan keterampilan.

Model data ini menggambarkan beberapa teknik seperti penanganan atribut kompleks seperti memiliki lebih dari satu keterampilan. Model ini juga merupakan contoh one-to-many hubungan melalui manajer dan karyawan pelaporan mereka yang telah dicapai oleh indeks sekunder DirectReports.

Pola akses yang difasilitasi oleh model data ini adalah:

- Pengambilan catatan karyawan menggunakan alias login karyawan, difasilitasi oleh tabel yang disebut `Employee`.
- Mencari karyawan berdasarkan nama, difasilitasi oleh indeks sekunder global tabel Karyawan yang disebut `Name`.
- Pengambilan semua laporan langsung dari manajer menggunakan alias login manajer, difasilitasi oleh indeks sekunder global tabel Karyawan yang disebut `DirectReports`.

Model data forum diskusi

Model data ini mewakili forum diskusi. Menggunakan model ini, pelanggan dapat terlibat dengan komunitas pengembang, mengajukan pertanyaan, dan menanggapi postingan pelanggan lain. Setiap layanan AWS memiliki forum khusus. Siapapun dapat memulai thread diskusi baru dengan memposting pesan di forum, dan setiap thread menerima sejumlah balasan.

Pola akses yang difasilitasi oleh model data ini adalah:

- Pengambilan catatan forum menggunakan nama forum, difasilitasi oleh tabel yang disebut `Forum`.
- Pengambilan thread tertentu atau semua thread untuk forum, difasilitasi oleh tabel yang disebut `Thread`.
- Cari balasan menggunakan alamat email pengguna postingan, yang difasilitasi oleh indeks sekunder global tabel Balasan yang disebut `PostedBy-Message-Index`.

Model data pustaka musik

Model data ini mewakili pustaka musik yang memiliki banyak koleksi lagu dan menampilkan lagu yang paling banyak diunduh hampir secara waktu nyata.

Pola akses yang difasilitasi oleh model data ini adalah:

- Pengambilan catatan lagu, difasilitasi oleh sebuah tabel yang disebut `Songs`.
- Pengambilan catatan unduhan tertentu atau seluruh catatan unduhan untuk sebuah lagu, difasilitasi oleh tabel yang disebut `Songs`.
- Pengambilan catatan jumlah unduhan bulanan tertentu atau seluruh catatan jumlah unduhan bulanan untuk sebuah lagu, difasilitasi oleh tabel yang disebut `Song`.

- Pengambilan semua catatan (termasuk catatan lagu, catatan unduhan, dan catatan jumlah unduhan bulanan) untuk sebuah lagu, difasilitasi oleh tabel yang disebut `Songs`.
- Pencarian lagu yang paling banyak diunduh, difasilitasi oleh indeks sekunder global tabel Lagu yang disebut `DownloadsByMonth`.

Model data resor ski

Model data ini mewakili resor ski yang memiliki banyak kumpulan data untuk setiap lift ski yang dikumpulkan setiap hari.

Pola akses yang difasilitasi oleh model data ini adalah:

- Pengambilan semua data untuk lift ski tertentu atau resor secara keseluruhan, dinamis dan statis, difasilitasi oleh tabel yang disebut `SkiLifts`.
- Pengambilan semua data dinamis (termasuk penunggang lift unik, cakupan salju, bahaya longsoran salju, dan status lift) untuk lift ski atau keseluruhan resor pada tanggal tertentu, difasilitasi oleh tabel yang disebut `SkiLifts`.
- Pengambilan semua data statis (termasuk apakah lift hanya untuk pengendara berpengalaman, kaki vertikal lift naik, dan waktu berkendara lift) untuk lift ski tertentu, difasilitasi oleh tabel yang disebut `SkiLifts`.
- Pengambilan tanggal data yang direkam untuk lift ski tertentu atau resor keseluruhan yang diurutkan berdasarkan total pengendara unik, difasilitasi oleh indeks sekunder global SkiLifts tabel yang disebut. `SkiLiftsByRiders`

Model data penawaran kartu kredit

Model data ini digunakan oleh Aplikasi Penawaran Kartu Kredit.

Penyedia kartu kredit menghasilkan penawaran seiring waktu. Penawaran ini mencakup transfer saldo tanpa biaya, peningkatan batas kredit, penurunan suku bunga, uang kembali, dan miles maskapai. Setelah pelanggan menerima atau menolak penawaran ini, status penawaran masing-masing akan diperbarui.

Pola akses yang difasilitasi oleh model data ini adalah:

- Pengambilan catatan akun menggunakan `AccountId`, yang difasilitasi oleh tabel utama.

- Pengambilan semua akun dengan beberapa item yang diproyeksikan, yang difasilitasi oleh indeks sekunder `AccountIndex`.
- Pengambilan akun dan semua catatan penawaran yang terkait dengan akun tersebut dengan menggunakan `AccountId`, seperti yang difasilitasi oleh tabel utama.
- Pengambilan akun dan catatan penawaran khusus yang terkait dengan akun tersebut dengan menggunakan `AccountId` dan `OfferId`, sebagaimana difasilitasi oleh tabel utama.
- Pengambilan semua catatan penawaran `ACCEPTED/DECLINED` dari `OfferType` spesifik terkait dengan akun yang menggunakan `AccountId`, `OfferType`, dan `Status`, seperti yang difasilitasi oleh indeks sekunder `GSI1`.
- Pengambilan catatan penawaran dan item penawaran terkait menggunakan `OfferId`, seperti yang difasilitasi oleh tabel utama.

Model data bookmark

Model data ini digunakan untuk menyimpan bookmark pelanggan.

Sebuah pelanggan dapat memiliki banyak bookmark dan sebuah bookmark dapat dimiliki oleh banyak pelanggan. Model data ini mewakili suatu many-to-many hubungan.

Pola akses yang difasilitasi oleh model data ini adalah:

- Satu kueri berdasarkan `customerId` kini dapat mengembalikan data pelanggan serta bookmark.
- Indeks `ByEmail` kueri mengembalikan data pelanggan berdasarkan alamat email. Perhatikan bahwa bookmark tidak diambil oleh indeks ini.
- Indeks `ByUrl` kueri mendapatkan data bookmark berdasarkan URL. Perhatikan bahwa kami memiliki customerId sebagai kunci urutan untuk indeks karena URL yang sama dapat ditandai oleh banyak pelanggan.
- Indeks `ByCustomerFolder` kueri mendapatkan bookmark berdasarkan folder untuk setiap pelanggan.

Riwayat rilis untuk NoSQL Workbench

Tabel berikut menjelaskan perubahan penting dalam setiap rilis alat klien NoSQL Workbench.

Versi	Perubahan	Deskripsi	Tanggal
3.13.5	Mode kapasitas untuk pengaturan tabel default sekarang sesuai permintaan	Saat Anda membuat tabel dengan pengaturan default, DynamoDB membuat tabel yang menggunakan mode kapasitas sesuai permintaan, bukan mode kapasitas yang disediakan.	Februari 24, 2025
3.13.0	Peningkatan pembuat operasi NoSQL Workbench	NoSQL Workbench sekarang menyertakan dukungan asli untuk mode gelap. Operasi tabel dan item yang ditingkatkan di pembuat operasi. Hasil item dan informasi permintaan pembuat operasi tersedia dalam format JSON.	April 24, 2024
3.12.0	Kloning tabel dengan NoSQL Workbench dan mengembalikan kapasitas yang dikonsumsi	Anda sekarang dapat mengkloning tabel antara DynamoDB lokal dan akun layanan web DynamoDB atau antara akun layanan web DynamoDB untuk iterasi pengembangan yang lebih cepat. Lihat RCU atau WCU	Februari 26, 2024

Versi	Perubahan	Deskripsi	Tanggal
		yang dikonsumsi setelah menjalankan operasi menggunak an Operations Builder. Kami memperbaiki masalah data timpa saat mengimpor dari file CSV.	
3.11.0	DynamoDB perbaikan lokal	Anda sekarang dapat menentukan port saat meluncurkan instance lokal DynamoDB bawaan. NoSQL Workbench sekarang dapat diinstal pada Windows tanpa hak admin. Kami telah memperbarui template model data.	Januari 17, 2024
3.10.0	Dukungan asli untuk silikon Apple	NoSQL Workbench sekarang menyertak an dukungan asli untuk Mac dengan silikon Apple. Anda sekarang dapat mengkonfigurasi format pembuatan data sampel untuk atribut tipe`Number`.	5 Desember 2023

Versi	Perubahan	Deskripsi	Tanggal
3.9.0	Perbaikan pemodel data	Pemvisualisasi sekarang mendukung melakukan model data ke DynamoDB lokal dengan opsi untuk menimpa tabel yang ada.	3 November 2023
3.8.0	Pembuatan data sampel	NoSQL Workbench kini mendukung pembuatan data sampel untuk model data DynamoDB Anda.	25 September 2023
3.6.0	Peningkatan dalam Pembangun operasi	Peningkatan manajemen koneksi di pembuat Operasi. Nama atribut di Pemodel Data sekarang dapat diubah tanpa menghapus data. Perbaikan bug lainnya.	11 April 2023
3.5.0	Support untuk AWS Wilayah baru	NoSQL Workbench sekarang mendukung wilayah ap-south-2, ap-southeast-3, ap-southeast-4, eu-central-2, eu-south-2, me-central-1, and me-west-1.	23 Februari 2023

Versi	Perubahan	Deskripsi	Tanggal
3.4.0	Dukungan untuk DynamoDB lokal	NoSQL Workbench sekarang mendukung instalasi DynamoDB lokal sebagai bagian dari proses instalasi.	6 Desember 2022
3.3.0	Dukungan untuk operasi bidang kontrol	Pembangun Operasi sekarang mendukung operasi bidang kontrol.	1 Juni 2022
3.2.0	Impor dan ekspor CSV	Anda sekarang dapat mengimpor data sampel dari file CSV di alat Visualizer, dan juga mengekspor hasil kueri Operation Builder ke file CSV.	11 Oktober 2021
3.1.0	Simpan operasi	Operation Builder di NoSQL Workbench sekarang mendukung operasi penyimpan an untuk digunakan nanti.	12 Juli 2021

Versi	Perubahan	Deskripsi	Tanggal
3.0.0	Pengaturan kapasitas dan CloudFormation impor/ekspor	NoSQL Workbench untuk Amazon DynamoDB sekarang mendukung penentuan mode kapasitas baca/tulis untuk tabel serta dapat mengimpor dan mengekspor model data dalam format AWS CloudFormation .	21 April 2021
2.2.0	Dukungan untuk PartiQL	NoSQL Workbench untuk Amazon DynamoDB menambahkan dukungan untuk membangun pernyataan PartiQL untuk DynamoDB.	4 Desember 2020
1.1.0	Dukungan untuk Linux.	NoSQL Workbench untuk Amazon DynamoDB didukung pada Linux—Ubuntu, Fedora dan Debian.	4 Mei 2020
1.0.0	NoSQL Workbench untuk Amazon DynamoDB — GA.	NoSQL Workbench untuk Amazon DynamoDB tersedia secara umum.	2 Maret 2020

Versi	Perubahan	Deskripsi	Tanggal
0.4.1	Dukungan untuk peran IAM dan kredensial keamanan sementara.	NoSQL Workbench untuk Amazon DynamoDB menambahkan dukungan untuk peran AWS Identity and Access Management (IAM) dan kredensial keamanan sementara.	19 Desember 2019
0.3.1	Dukungan untuk DynamoDB lokal (Versi Dapat Diunduh).	NoSQL Workbench kini mendukung koneksi ke DynamoDB lokal (Versi yang Dapat Diunduh) untuk merancang, membuat, mengkueri , dan mengelola tabel DynamoDB.	8 November, 2019
0.2.1	Pratinjau NoSQL Workbench dirilis.	Ini adalah rilis awal NoSQL Workbench untuk DynamoDB. Gunakan NoSQL Workbench untuk merancang, membuat, mengkueri , dan mengelola tabel DynamoDB.	16 September 2019

Backup dan restore untuk DynamoDB

DynamoDB menawarkan cadangan point-in-time dan pemulihan sesuai permintaan (PITR) untuk membantu melindungi data DynamoDB Anda dari peristiwa bencana dan menawarkan pengarsipan data untuk retensi jangka panjang. Anda dapat mencadangkan tabel dari beberapa megabyte hingga ratusan terabyte data, tanpa berdampak pada kinerja dan ketersediaan aplikasi produksi Anda. Semua backup secara otomatis dienkripsi, dikatalogkan, dan mudah ditemukan.

Dengan cadangan sesuai permintaan, Anda dapat membuat cadangan snapshot dari tabel Anda yang disimpan dan dikelola DynamoDB. Anda dikenakan biaya berdasarkan ukuran dan durasi backup Anda. Menggunakan cadangan sesuai permintaan, Anda dapat memulihkan seluruh tabel DynamoDB Anda ke keadaan yang tepat saat cadangan dibuat.

Ada dua pilihan untuk membuat dan mengelola DynamoDB on-demand backup:

- DynamoDB
- AWS Backup

Anda dapat menggunakan kemampuan sesuai permintaan DynamoDB untuk membuat pencadangan penuh dari tabel Anda untuk retensi jangka panjang, dan pengarsipan untuk kebutuhan kepatuhan terhadap peraturan. Anda dapat mencadangkan dan memulihkan data tabel kapan saja dari AWS Management Console atau dengan satu panggilan API.

Point-in-time backup recovery (PITR) sepenuhnya dikelola oleh DynamoDB dan menyediakan hingga 35 hari titik pemulihan pada granularitas per detik. Untuk menggunakan point-in-time pemulihan, yang merupakan backup berkelanjutan, aktifkan point-in-time pemulihan (PITR) pada tabel DynamoDB Anda. Anda dikenakan biaya berdasarkan ukuran tabel DynamoDB Anda selama durasi Anda mengaktifkan PITR di atas meja. Mengaktifkan Point-in-Time Pemulihan (PITR) pada tabel DynamoDB Anda terus mencadangkan data Anda. Ini membantu Anda mengembalikan tabel Anda ke titik waktu tertentu dalam periode pemulihan PITR dengan membuat tabel DynamoDB baru dengan status yang tepat dari tabel asli Anda pada saat itu.

Point-in-time pemulihan membantu melindungi tabel DynamoDB Anda dari operasi tulis atau hapus yang tidak disengaja. Dengan point-in-time pemulihan, Anda tidak perlu khawatir tentang membuat, memelihara, atau menjadwalkan cadangan sesuai permintaan. Misalnya, skrip pengujian tidak sengaja menulis ke tabel DynamoDB produksi.

Dengan point-in-time pemulihan, Anda dapat mengembalikan tabel Anda ke titik waktu mana pun selama 35 hari terakhir. Anda dapat mengatur periode pemulihan ke nilai apa pun antara 1 dan 35 hari. Setelah Anda mengaktifkan point-in-time pemulihan, Anda dapat mengembalikan ke titik waktu dari lima menit sebelum waktu saat ini hingga periode pemulihan yang dikonfigurasi. DynamoDB memelihara cadangan tambahan tabel Anda.

Selain itu, point-in-time operasi tidak memengaruhi kinerja atau latensi API.

Anda dapat mengembalikan tabel DynamoDB ke titik waktu menggunakan AWS Management Console,AWS CLI(), AWS Command Line Interface atau DynamoDB API. Proses point-in-time pemulihan mengembalikan ke tabel baru.

Untuk informasi selengkapnya tentang ketersediaan dan harga AWS Wilayah, lihat harga Amazon DynamoDB.

> **Note**
>
> - Penandaan dan kontrol akses berbasis atribut (ABAC) tidak didukung untuk backup DynamoDB. Untuk menggunakan ABAC dengan cadangan, kami sarankan Anda menggunakannya. AWS Backup
> - Tag tidak disimpan dalam tabel yang dipulihkan. Anda perlu menambahkan tag ke tabel yang dipulihkan sebelum dapat menggunakan kondisi berbasis tag dalam kebijakan Anda.

Video berikut akan memberi Anda tampilan pengantar konsep pencadangan dan pemulihan dan berbicara lebih banyak tentang point-in-time pemulihan.

Pencadangan dan pemulihan

Topik

- Point-in-time backup untuk DynamoDB
- Menggunakan cadangan dan pemulihan DynamoDB sesuai permintaan
- Memahami penagihan Amazon DynamoDB untuk backup
- Kembalikan tabel di DynamoDB
- Menggunakan AWS Backup dengan DynamoDB

Point-in-time backup untuk DynamoDB

Amazon point-in-time DynamoDB Recovery (PITR) menyediakan backup kontinu otomatis dari data tabel DynamoDB Anda. Point-in-timebackup recovery (PITR) sepenuhnya dikelola oleh DynamoDB dan menyediakan hingga 35 hari titik pemulihan pada granularitas per detik. Dengan point-in-time pemulihan, Anda tidak perlu khawatir tentang membuat, memelihara, atau menjadwalkan cadangan sesuai permintaan. Bagian ini memberikan gambaran umum tentang cara kerja proses di DynamoDB.

Sebelum Anda mulai

Sebelum Anda mengaktifkan point-in-time pemulihan (PITR) pada tabel Amazon DynamoDB, pertimbangkan hal berikut:

- Mengatur `RecoveryPeriodinDays` memungkinkan Anda untuk mempersingkat periode waktu pencadangan berkelanjutan diambil. Secara default, Anda `RecoveryPeriodinDays` adalah 35. Namun, Anda dapat mengaturnya menjadi nilai antara 1 dan 35. Memperpendek tidak `RecoveryPeriodinDays` berdampak pada harga PITR karena harga didasarkan pada ukuran tabel dan indeks sekunder lokal.
- Jika Anda menonaktifkan point-in-time pemulihan dan kemudian mengaktifkannya kembali di atas meja, Anda mengatur ulang waktu mulai di mana Anda dapat memulihkan tabel itu. Akibatnya, Anda hanya dapat segera memulihkan tabel tersebut menggunakan `LatestRestorableDateTime`.
- Anda dapat mengaktifkan point-in-time pemulihan pada setiap replika lokal tabel global. Saat Anda memulihkan tabel, cadangan akan dikembalikan ke tabel independen yang bukan bagian dari tabel global. Jika Anda menggunakan Tabel Global versi 2019.11.21 (Saat ini) dari tabel global, Anda dapat membuat tabel global baru dari tabel yang dipulihkan. Untuk informasi selengkapnya, lihat Tabel global DynamoDB: Cara kerjanya.
- Anda juga dapat memulihkan data tabel DynamoDB AWS di seluruh Wilayah sehingga tabel yang dipulihkan dibuat di Wilayah yang berbeda dari tempat tabel sumber berada. Anda dapat melakukan pemulihan lintas wilayah antara Wilayah AWS komersial, Wilayah AWS Tiongkok, dan Wilayah AWS GovCloud (AS). Anda hanya membayar untuk data yang Anda transfer keluar dari Wilayah sumber dan untuk memulihkan ke tabel baru di Wilayah tujuan.
- AWS CloudTrail mencatat semua tindakan konsol dan API untuk point-in-time pemulihan guna mengaktifkan pencatatan, pemantauan berkelanjutan, dan audit. Untuk informasi selengkapnya, lihat Pencatatan log operasi DynamoDB menggunakan AWS CloudTrail.

Topik

- Aktifkan point-in-time pemulihan di DynamoDB

Aktifkan point-in-time pemulihan di DynamoDB

Amazon point-in-time DynamoDB recovery (PITR) menyediakan backup otomatis data tabel DynamoDB Anda. Bagian ini memberikan gambaran umum tentang cara kerja proses di DynamoDB.

> **Note**
>
> DynamoDB mengenakan biaya untuk PITR berdasarkan ukuran setiap tabel DynamoDB, termasuk data tabel dan indeks sekunder lokal. Mengubah jendela pemulihan (misalnya, dari 35 hari menjadi 1 hari) tidak mengurangi harga. Biaya tetap sama terlepas dari periode pemulihan yang Anda pilih. Periode pemulihan maksimum yang dikonfigurasi tidak memengaruhi harga yang dikenakan untuk menyalakan PITR. Untuk menentukan biaya cadangan Anda, DynamoDB terus memantau ukuran tabel yang mengaktifkan PITR. Anda ditagih untuk penggunaan PITR sampai Anda mematikan PITR untuk setiap tabel.

Topik

- Mengaktifkan pemulihan point-in-time
- Aktifkan PITR (konsol)
- Aktifkan PITR ()AWS CLI
- Aktifkan PITR ()AWS CloudFormation
- Aktifkan PITR (API)
- Periode Pemulihan
- Sunting PITR
- Hapus tabel dengan PITR diaktifkan

Mengaktifkan pemulihan point-in-time

Anda dapat mengaktifkan point-in-time pemulihan menggunakan AWS Management Console, AWS Command Line Interface (AWS CLI), atau DynamoDB API. Saat diaktifkan, point-in-time pemulihan menyediakan pencadangan berkelanjutan hingga Anda mematikannya secara eksplisit.

Setelah Anda mengaktifkan point-in-time pemulihan, Anda dapat mengembalikan ke titik waktu mana pun di dalam `EarliestRestorableDateTime` dan`LatestRestorableDateTime`. `LatestRestorableDateTime`Biasanya lima menit sebelum waktu saat ini. Untuk informasi selengkapnya, lihat Memulihkan tabel DynamoDB ke titik waktu.

> **Note**
>
> Proses point-in-time pemulihan selalu kembali ke tabel baru.

Aktifkan PITR (konsol)

Untuk mengaktifkan PITR menggunakan konsol DynamoDB

1. Navigasikan ke konsol DynamoDB.
2. Pilih Tabel dari navigasi kiri dan pilih tabel DynamoDB Anda.
3. Dari tab Backup, untuk opsi Point in Time Recovery, pilih Edit.
4. Pilih Aktifkan point-in-time pemulihan.
5. Pilih nilai antara 1 dan 35 untuk periode pemulihan cadangan Anda. Ini menunjukkan periode waktu maksimum di mana cadangan berkelanjutan dapat dipulihkan.

Aktifkan PITR ()AWS CLI

> **Note**
>
> Jika Anda menerima kesalahan saat menjalankan AWS CLI perintah, lihat Memecahkan masalah AWS CLI kesalahan. Pastikan Anda menggunakan AWS CLI versi terbaru.

Jalankan update-continuous-backupsperintah dengan pengaturan point-in-time-recovery-spesifikasi diaktifkan:

```
aws dynamodb update-continuous-backups \
--table-name Music \
--point-in-time-recovery-specification
 PointInTimeRecoveryEnabled=true,RecoveryPeriodInDays=35
```

Aktifkan PITR ()AWS CloudFormation

Gunakan AWS::DynamoDB::Tablesumber daya dengan `PointInTimeRecoverySpecification` properti dihidupkan:

```
Resources:
  iotCatalog:
    Type: AWS::DynamoDB::Table
      Properties:
      ...
      PointInTimeRecoverySpecification:
        PointInTimeRecoveryEnabled: true
        RecoveryPeriodInDays: 35
```

Minta contoh sintaks:

```
{
   "PointInTimeRecoverySpecification": {
      "PointInTimeRecoveryEnabled": boolean,
      "RecoveryPeriodInDays: number
   },
   "TableName": "string"
}
```

Aktifkan PITR (API)

Jalankan operasi UpdateContinuousBackupsAPI dengan `PointInTimeRecoverySpecification` parameter dihidupkan.

Minta contoh sintaks:

```
{
   "PointInTimeRecoverySpecification": {
      "PointInTimeRecoveryEnabled": boolean,
      "RecoveryPeriodInDays" : number
   },
   "TableName": "string"
}
```

Contoh sintaks respons:

```
{
   "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
    }
)
```

Periode Pemulihan

Anda dapat mengatur periode pemulihan untuk pencadangan berkelanjutan menjadi angka apa pun antara 1 dan 35 hari. Ini `RecoveryPeriodInDays` menentukan periode waktu di mana backup berkelanjutan Anda dipertahankan. Misalnya, jika Anda menetapkan nilai ini menjadi 30 hari, Anda hanya akan memiliki kemampuan untuk mengembalikan tabel Anda ke titik waktu dari 30 hari terakhir.

> **Note**
>
> DynamoDB mengenakan biaya untuk PITR berdasarkan ukuran setiap tabel DynamoDB, termasuk data tabel dan indeks sekunder lokal. Periode pemulihan maksimum yang dikonfigurasi tidak memengaruhi harga yang dikenakan untuk menyalakan PITR. Untuk detail tentang harga, lihat harga DynamoDB.

Sunting PITR

Anda dapat mengedit pengaturan PITR di meja Anda dan mengubah periode pemulihan. Jika Anda mengubah periode pemulihan dan meningkatkannya ke nilai yang lebih tinggi dari yang ditetapkan sebelumnya, Anda tidak `EarliestRestorePoint` akan segera berubah. Karena periode pemulihan adalah jendela bergulir, DynamoDB akan terus mengambil cadangan otomatis hingga periode peningkatan baru tercapai. Jika Anda mengubah periode pemulihan dan menurunkannya ke nilai yang lebih rendah dari yang ditetapkan sebelumnya, Anda `EarliestRestorePoint` akan segera berkurang agar sesuai dengan periode pemulihan Anda, dan pencadangan berkelanjutan apa pun yang berada di luar nilai yang ditetapkan baru tidak akan dapat dipulihkan.

Hapus tabel dengan PITR diaktifkan

Saat Anda menghapus tabel yang mengaktifkan point-in-time pemulihan, DynamoDB secara otomatis membuat snapshot cadangan yang disebut cadangan sistem dan mempertahankannya selama 35 hari (tanpa biaya tambahan). Anda dapat menggunakan cadangan sistem untuk mengembalikan tabel yang dihapus ke keadaan sebelum penghapusan. Semua cadangan sistem mengikuti konvensi penamaan standar. *`table-name`* `$DeletedTableBackup`

> **Note**
>
> Setelah tabel dengan point-in-time pemulihan diaktifkan dihapus, Anda dapat menggunakan cadangan sistem untuk mengembalikan tabel itu ke satu titik waktu. Cadangan sistem akan dibuat setelah penghapusan tabel, dan merupakan snapshot dari tabel tepat sebelum tabel dihapus.

Menggunakan cadangan dan pemulihan DynamoDB sesuai permintaan

Amazon DynamoDB mendukung fitur pencadangan dan pemulihan sesuai permintaan yang berdiri sendiri. Fitur-fitur tersebut tersedia untuk Anda terlepas dari apakah Anda menggunakan AWS Backup.

Anda dapat menggunakan kemampuan sesuai permintaan DynamoDB untuk membuat pencadangan penuh dari tabel Anda untuk retensi jangka panjang dan pengarsipan untuk kebutuhan kepatuhan terhadap peraturan. Anda dapat mencadangkan dan memulihkan data tabel kapan saja dengan satu

klik pada AWS Management Console atau dengan satu panggilan API. Tindakan pencadangan dan pemulihan dijalankan tanpa memengaruhi ketersediaan atau performa tabel sama sekali.

Anda dapat membuat backup tabel menggunakan konsol, AWS Command Line Interface (AWS CLI), atau DynamoDB API. Untuk informasi selengkapnya, lihat Mencadangkan tabel DynamoDB.

Untuk informasi tentang memulihkan tabel dari cadangan, lihat Memulihkan tabel DynamoDB dari cadangan.

Mencadangkan dan memulihkan tabel DynamoDB dengan DynamoDB: Cara kerjanya

Anda dapat menggunakan fitur pencadangan sesuai permintaan DynamoDB untuk membuat pencadangan lengkap tabel Amazon DynamoDB Anda. Fitur ini tersedia secara independen dari AWS cadangan. Bagian ini memberikan gambaran umum tentang apa yang terjadi selama proses pencadangan dan pemulihan DynamoDB.

Pencadangan

Saat Anda membuat cadangan sesuai permintaan dengan DynamoDB, penanda waktu permintaan dikatalogkan. Cadangan dibuat secara asinkron dengan menerapkan semua perubahan hingga waktu permintaan ke snapshot tabel lengkap terakhir. Permintaan pencadangan DynamoDB diproses seketika dan tersedia untuk pemulihan dalam beberapa menit.

> **Note**
>
> Setiap kali Anda membuat cadangan sesuai permintaan, seluruh data tabel dicadangkan. Tidak ada batasan untuk jumlah pencadangan sesuai permintaan.

Semua pencadangan di DynamoDB bekerja tanpa mengonsumsi throughput apa pun yang disediakan pada tabel.

Pencadangan DynamoDB tidak menjamin konsistensi kausal pada item-item; tetapi, selisih antara pembaruan dalam cadangan biasanya jauh lebih sedikit dari satu detik.

Saat pencadangan sedang berlangsung, Anda tidak dapat melakukan hal berikut:

- Menjeda atau membatalkan operasi pencadangan.

- Menghapus tabel sumber cadangan.
- Nonaktifkan pencadangan pada tabel jika pencadangan untuk tabel tersebut sedang berlangsung.

Jika Anda tidak ingin membuat skrip penjadwalan dan pekerjaan pembersihan, Anda dapat menggunakannya AWS Backup untuk membuat rencana cadangan dengan jadwal dan kebijakan retensi untuk tabel DynamoDB Anda. AWS Backup menjalankan cadangan dan menghapusnya saat kedaluwarsa. Lihat informasi selengkapnya di Panduan Developer AWS Backup.

Selain itu AWS Backup, Anda dapat menjadwalkan backup periodik atau future dengan menggunakan AWS Lambda fungsi. Untuk informasi selengkapnya, lihat posting blog Solusi nirserver untuk menjadwalkan Pencadangan Sesuai Permintaan Amazon DynamoDB Anda.

Jika Anda menggunakan konsol, cadangan apa pun yang dibuat menggunakan AWS Backup tercantum di tab Cadangan dengan jenis Cadangan yang disetel ke. `AWS`

> **Note**
>
> Anda tidak dapat menghapus cadangan yang ditandai dengan Jenis cadangan AWS menggunakan konsol DynamoDB. Untuk mengelola cadangan ini, gunakan konsol. AWS Backup

Untuk mempelajari cara melakukan pencadangan, lihat Mencadangkan tabel DynamoDB.

Memulihkan

Anda memulihkan tabel tanpa mengonsumsi throughput apa pun yang disediakan pada tabel. Anda dapat melakukan pemulihan tabel penuh dari cadangan DynamoDB Anda, atau Anda dapat mengonfigurasi pengaturan tabel tujuan. Saat melakukan pemulihan, Anda dapat mengubah pengaturan tabel berikut:

- Indeks sekunder global () GSIs
- Indeks sekunder lokal () LSIs
- Mode penagihan
- Kapasitas baca dan tulis yang disediakan
- Pengaturan enkripsi

> **Important**
>
> Ketika Anda melakukan pemulihan tabel penuh, tabel tujuan diatur dengan unit kapasitas baca dan unit kapasitas tulis yang sama seperti yang dimiliki tabel sumber, seperti yang tercatat pada saat cadangan diminta. Proses pemulihan juga memulihkan indeks sekunder lokal dan indeks sekunder global.

Anda juga dapat memulihkan data tabel DynamoDB Anda AWS di seluruh Wilayah sehingga tabel yang dipulihkan dibuat di Wilayah yang berbeda dari tempat cadangan berada. Anda dapat melakukan pemulihan lintas wilayah antara Wilayah AWS komersial, Wilayah AWS Tiongkok, dan Wilayah AWS GovCloud (AS). Anda hanya membayar untuk data yang Anda transfer keluar dari Wilayah sumber dan untuk memulihkan ke tabel baru di Wilayah tujuan.

Pemulihan dapat lebih cepat dan lebih hemat biaya jika Anda memilih untuk mengecualikan beberapa atau semua indeks sekunder dari pembuatan pada tabel yang baru dipulihkan.

Anda harus menyiapkan berikut ini secara manual pada tabel yang dipulihkan:

- Kebijakan penskalaan otomatis
- AWS Identity and Access Management Kebijakan (IAM)
- CloudWatch Metrik dan alarm Amazon
- Tanda
- Pengaturan aliran
- Pengaturan Waktu untuk Tayang (TTL)
- Pengaturan perlindungan penghapusan
- Pengaturan pemulihan titik waktu (PITR)

Anda hanya dapat memulihkan seluruh data tabel ke tabel baru dari cadangan. Anda dapat menulis ke tabel yang dipulihkan hanya setelah tabel tersebut aktif.

> **Note**
>
> Anda tidak dapat menimpa tabel yang ada selama operasi pemulihan.

Metrik layanan menunjukkan bahwa 95 persen pemulihan tabel pelanggan selesai dalam waktu kurang dari satu jam. Namun, waktu pemulihan terkait langsung dengan konfigurasi tabel Anda (seperti ukuran tabel dan jumlah partisi yang mendasarinya) dan variabel terkait lainnya. Praktik terbaik saat merencanakan pemulihan bencana adalah mendokumentasikan waktu penyelesaian pemulihan rata-rata secara teratur dan menetapkan bagaimana waktu tersebut memengaruhi Sasaran Waktu Pemulihan Anda secara keseluruhan.

Untuk mempelajari cara melakukan pemulihan, lihat Memulihkan tabel DynamoDB dari cadangan.

Anda dapat menggunakan kebijakan IAM untuk kontrol akses. Untuk informasi selengkapnya, lihat Menggunakan IAM dengan pencadangan dan pemulihan DynamoDB.

Semua konsol cadangan dan pemulihan serta tindakan API ditangkap dan direkam di AWS CloudTrail untuk pencatatan, pemantauan berkelanjutan, dan audit.

Mencadangkan tabel DynamoDB

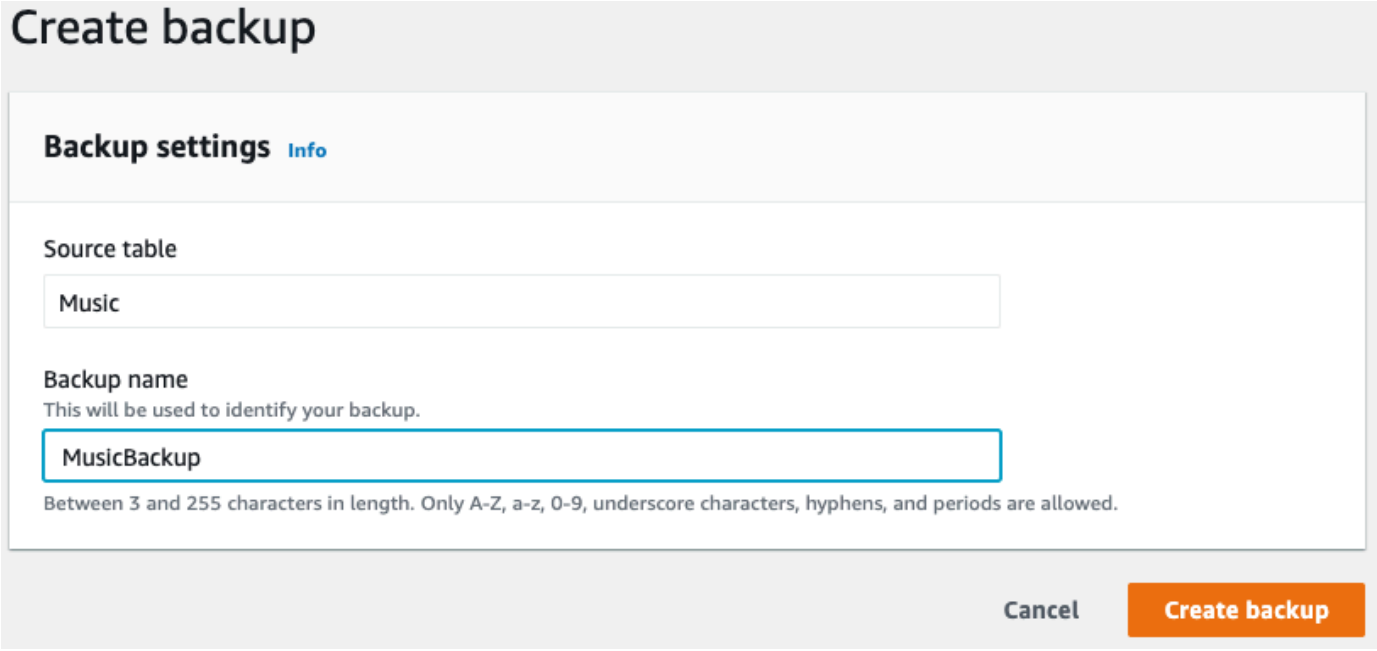
Bagian ini menjelaskan cara menggunakan konsol Amazon DynamoDB atau untuk AWS Command Line Interface mencadangkan tabel.

Membuat cadangan tabel (konsol)

Ikuti langkah-langkah ini untuk membuat cadangan bernama `MusicBackup` untuk tabel `Music` yang sudah ada menggunakan AWS Management Console.

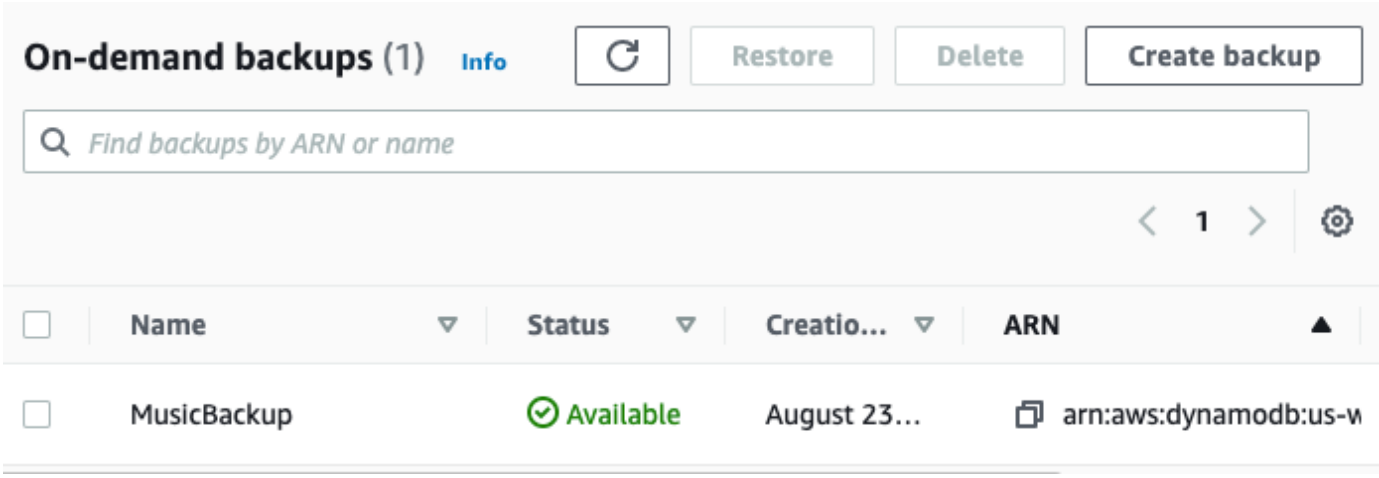
Untuk membuat cadangan tabel

1. Masuk ke AWS Management Console dan buka konsol DynamoDB di. https://console.aws.amazon.com/dynamodb/
2. Anda dapat membuat cadangan dengan melakukan salah satu hal berikut:
   - Pada tab Cadangan di tabel `Music`, pilih Buat cadangan.
   - Di panel navigasi pada sisi kiri konsol, pilih Cadangan. Kemudian pilih Buat cadangan.
3. Pastikan bahwa `Music` adalah nama tabel, dan masukkan **`MusicBackup`** sebagai nama cadangan. Kemudian, pilih Buat cadangan untuk membuat cadangan.



> **Note**
>
> Jika Anda membuat cadangan menggunakan bagian Cadangan di panel navigasi, tabel tidak dipilihkan untuk Anda terlebih dahulu. Anda harus memilih sendiri nama tabel sumber untuk cadangan.

Selagi cadangan sedang dibuat, status cadangan diatur ke Sedang dibuat. Setelah pencadangan selesai, status cadangan berubah menjadi Tersedia.



Membuat cadangan tabel (AWS CLI)

Ikuti langkah-langkah ini guna membuat cadangan untuk tabel `Music` yang sudah ada menggunakan AWS CLI.

Untuk membuat cadangan tabel

- Buat cadangan dengan nama `MusicBackup` untuk tabel `Music`.

```
aws dynamodb create-backup --table-name Music \
 --backup-name MusicBackup
```

Selagi cadangan sedang dibuat, status cadangan diatur ke `CREATING`.

```
{
    "BackupDetails": {
        "BackupName": "MusicBackup",
        "BackupArn": "arn:aws:dynamodb:us-east-1:123456789012:table/Music/
backup/01489602797149-73d8d5bc",
        "BackupStatus": "CREATING",
        "BackupCreationDateTime": 1489602797.149
    }
}
```

Setelah pencadangan selesai, `BackupStatus` akan berubah menjadi `AVAILABLE`. Untuk mengonfirmasi hal ini, gunakan perintah `describe-backup`. Anda bisa mendapatkan nilai input `backup-arn` dari output langkah sebelumnya atau dengan menggunakan perintah `list-backups`.

```
aws dynamodb describe-backup \
--backup-arn arn:aws:dynamodb:us-east-1:123456789012:table/Music/backup/01489173575360-
b308cd7d
```

Untuk melacak cadangan, Anda dapat menggunakan perintah `list-backups`. Perintah ini menampilkan daftar semua cadangan yang berstatus `CREATING` atau `AVAILABLE`.

```
aws dynamodb list-backups
```

Perintah `list-backups` dan perintah `describe-backup` berguna untuk memeriksa informasi tentang tabel sumber dari cadangan.

Memulihkan tabel DynamoDB dari cadangan

Bagian ini menjelaskan cara memulihkan tabel dari cadangan menggunakan konsol Amazon DynamoDB atau AWS Command Line Interface ().AWS CLI

> **Note**
>
> Jika Anda ingin menggunakan AWS CLI, Anda harus mengkonfigurasinya terlebih dahulu. Untuk informasi selengkapnya, lihat Mengakses DynamoDB.

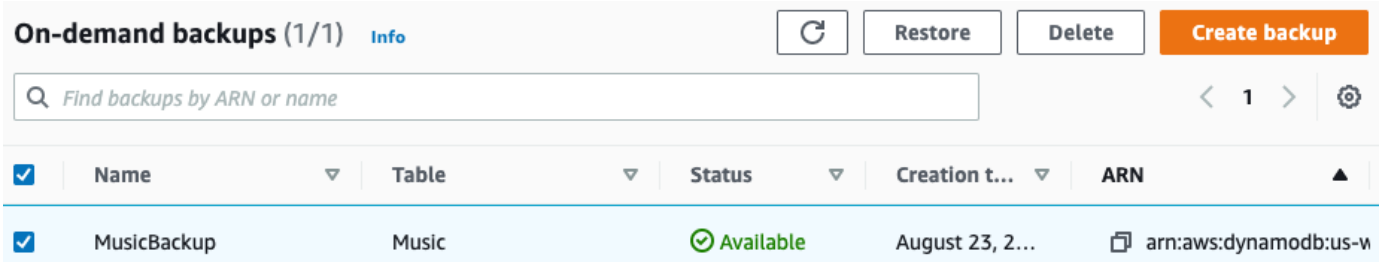
Memulihkan tabel dari cadangan (konsol)

Prosedur berikut menunjukkan cara memulihkan tabel `Music` dengan menggunakan file `MusicBackup` yang dibuat di tutorial Mencadangkan tabel DynamoDB.

> **Note**
>
> Prosedur ini mengasumsikan bahwa tabel `Music` tidak ada lagi sebelum memulihkannya menggunakan file `MusicBackup`.

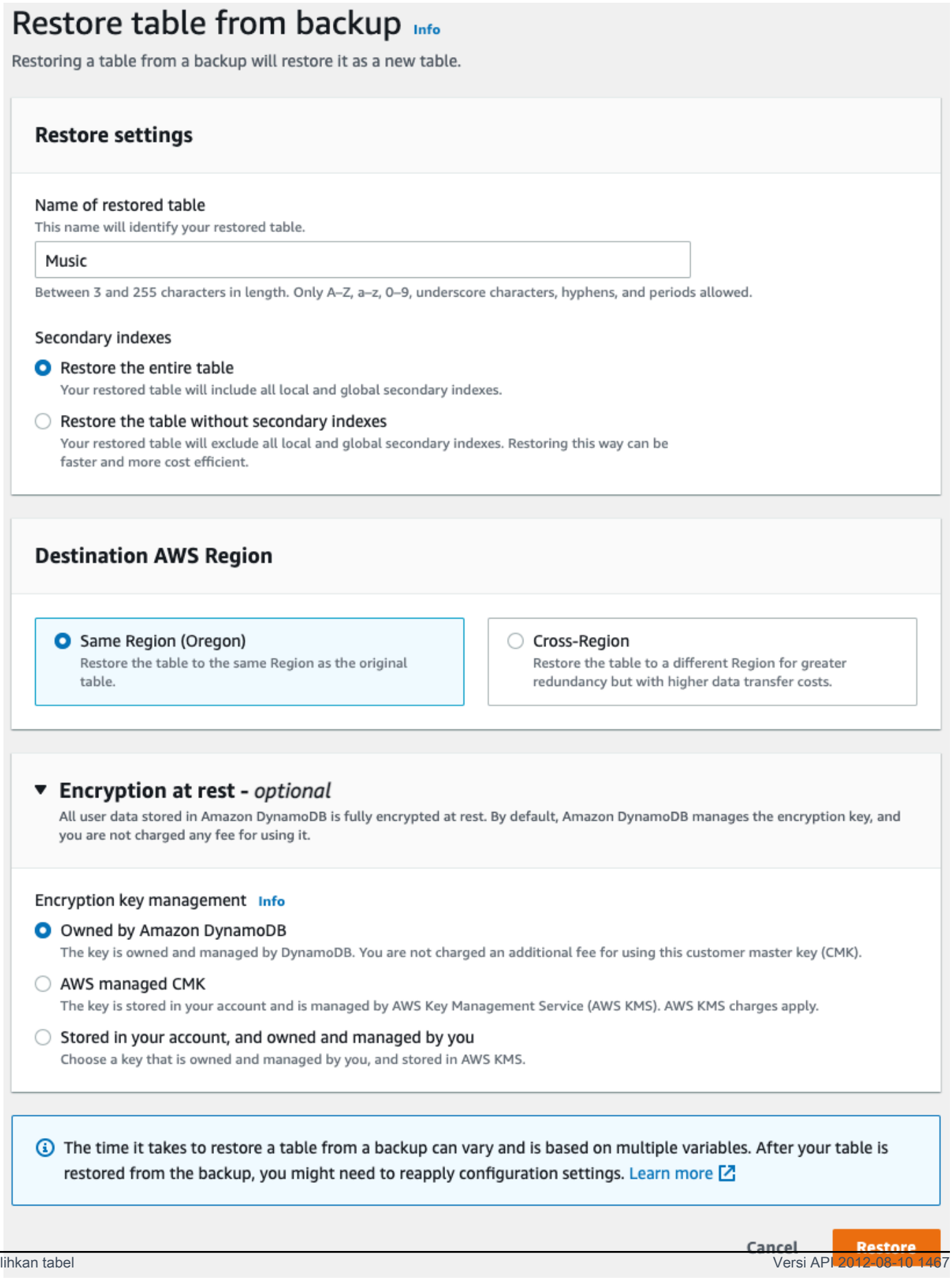
Untuk memulihkan tabel dari cadangan

1. Masuk ke AWS Management Console dan buka konsol DynamoDB di. https://console.aws.amazon.com/dynamodb/
2. Di panel navigasi pada sisi kiri konsol, pilih Cadangan.
3. Dalam daftar cadangan, pilih `MusicBackup`.



4. Pilih Pulihkan.
5. Masukkan **`Music`** sebagai nama tabel baru. Konfirmasikan nama cadangan dan detail cadangan lainnya. Kemudian pilih Pulihkan tabel untuk memulai proses pemulihan.

> **Note**
>
> Anda dapat mengembalikan tabel ke AWS Wilayah yang sama atau ke Wilayah yang berbeda dari tempat cadangan berada. Anda juga dapat mengecualikan indeks sekunder dari yang dibuat pada tabel yang baru dipulihkan. Selain itu, Anda dapat menentukan mode enkripsi yang berbeda.
>
> Tabel yang dipulihkan dari cadangan selalu dibuat menggunakan kelas tabel DynamoDB Standard.


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

Tabel yang sedang dipulihkan ditampilkan dengan status Membuat. Setelah proses pemulihan selesai, status tabel `Music` berubah menjadi Aktif.

Memulihkan tabel dari cadangan (AWS CLI)

Ikuti langkah-langkah ini untuk menggunakan AWS CLI untuk mengembalikan `Music` tabel menggunakan `MusicBackup` yang dibuat dalam Mencadangkan tabel DynamoDB tutorial.

Untuk memulihkan tabel dari cadangan

1. Konfirmasikan pencadangan yang ingin Anda pulihkan dengan menggunakan perintah `list-backups`. Contoh ini menggunakan `MusicBackup`.

   ```
   aws dynamodb list-backups
   ```

   Untuk mendapatkan detail tambahan tentang pencadangan, gunakan perintah `describe-backup`. Anda bisa mendapatkan input `backup-arn` dari langkah sebelumnya.

   ```
   aws dynamodb describe-backup \
   --backup-arn arn:aws:dynamodb:us-east-1:123456789012:table/Music/
   backup/01489173575360-b308cd7d
   ```

2. Pulihkan tabel dari cadangan. Dalam hal ini, `MusicBackup` mengembalikan `Music` tabel ke AWS Wilayah yang sama.

   ```
   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. Memulihkan tabel dari cadangan dengan pengaturan tabel kustom. Dalam hal ini, `MusicBackup` memulihkan tabel `Music` dan menentukan mode enkripsi untuk tabel yang dipulihkan.

   > **Note**
   >
   > Parameter `sse-specification-override` mengambil nilai yang sama seperti yang digunakan parameter `sse-specification-override` dalam perintah `CreateTable`.

Untuk mempelajari informasi lebih lanjut, lihat Mengelola tabel yang dienkripsi di 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
```

Anda dapat mengembalikan tabel ke AWS Wilayah yang berbeda dari tempat cadangan berada.

Note

- Parameter `sse-specification-override` bersifat wajib untuk pemulihan lintas Wilayah, namun opsional untuk pemulihan di Wilayah yang sama dengan tabel sumber.
- Saat melakukan pemulihan lintas wilayah dari baris perintah, Anda harus mengatur AWS Wilayah default ke wilayah tujuan yang diinginkan. Untuk mempelajari selengkapnya, lihat Opsi baris perintah di Panduan Pengguna AWS Command Line Interface .

```
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
```

Anda dapat menimpa modus penagihan dan throughput yang disediakan untuk tabel yang dipulihkan.

```
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
```

Anda dapat mengecualikan beberapa atau semua indeks sekunder dari yang dibuat pada tabel yang baru dipulihkan.

> **Note**
>
> Pemulihan bisa lebih cepat dan hemat biaya jika Anda mengecualikan beberapa atau semua indeks sekunder agar tidak dibuat pada tabel yang dipulihkan.

```
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**
>
> Indeks sekunder yang disediakan harus sesuai indeks yang ada. Anda tidak dapat membuat indeks baru pada saat pemulihan.

Anda dapat menggunakan kombinasi dari penimpaan yang berbeda. Misalnya, Anda dapat menggunakan satu indeks sekunder global dan mengubah throughput yang disediakan secara bersamaan, sebagai berikut.

```
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
```

Untuk memverifikasi pemulihan, gunakan perintah `describe-table` untuk menggambarkan tabel `Music`.

```
aws dynamodb describe-table --table-name Music
```

Tabel yang sedang dipulihkan dari cadangan ditampilkan dengan status Membuat. Setelah proses pemulihan selesai, status tabel `Music` berubah menjadi Aktif.

> **Important**
>
> Saat pemulihan sedang berlangsung, jangan mengubah atau menghapus kebijakan peran IAM Anda; jika tidak, perilaku yang tidak diharapkan dapat terjadi. Misalnya, Anda menghapus izin menulis untuk sebuah tabel saat tabel tersebut sedang dipulihkan. Dalam kasus ini, operasi `RestoreTableFromBackup` yang mendasarinya tidak akan mampu menulis data apa pun yang dipulihkan ke tabel.
> Setelah operasi pemulihan selesai, Anda dapat mengubah atau menghapus kebijakan peran IAM Anda.
> Kebijakan IAM yang melibatkan pembatasan IP sumber untuk mengakses tabel pemulihan target harus memiliki kunci `aws:ViaAWSService` yang disetel ke `false` untuk memastikan bahwa pembatasan hanya berlaku untuk permintaan yang dibuat langsung oleh pengguna utama. Jika tidak, pemulihan akan dibatalkan.
> Jika cadangan Anda dienkripsi dengan Kunci yang dikelola AWS atau kunci yang dikelola pelanggan, jangan menonaktifkan atau menghapus kunci saat pemulihan sedang berlangsung, atau pemulihan akan gagal.
> Setelah operasi pemulihan selesai, Anda dapat mengubah kunci enkripsi untuk tabel yang dipulihkan dan menonaktifkan atau menghapus kunci lama.

Menghapus cadangan tabel DynamoDB

Bagian ini menjelaskan cara menggunakan AWS Management Console atau AWS Command Line Interface (AWS CLI) untuk menghapus cadangan tabel Amazon DynamoDB.

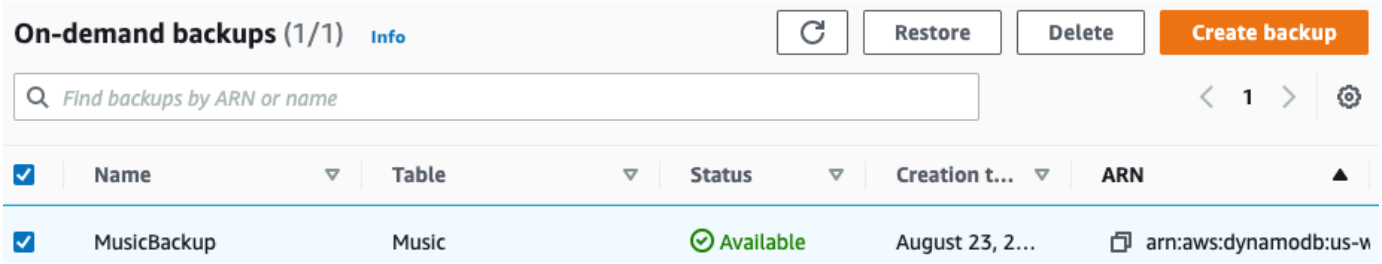
> **Note**
>
> Jika Anda ingin menggunakan AWS CLI, Anda harus mengkonfigurasinya terlebih dahulu. Untuk informasi selengkapnya, lihat Menggunakan AWS CLI.

Menghapus cadangan tabel (konsol)

Prosedur berikut menunjukkan cara menggunakan konsol untuk menghapus `MusicBackup` yang dibuat di tutorial Mencadangkan tabel DynamoDB.

Cara menghapus cadangan

1. Masuk ke AWS Management Console dan buka konsol DynamoDB di. https://console.aws.amazon.com/dynamodb/
2. Di panel navigasi pada sisi kiri konsol, pilih Cadangan.
3. Dalam daftar cadangan, pilih `MusicBackup`.



4. Pilih Hapus. Konfirmasikan bahwa Anda ingin menghapus cadangan dengan mengetik **`delete`** dan mengeklik Hapus.

Menghapus cadangan tabel (AWS CLI)

Contoh berikut menghapus cadangan untuk tabel `Music` yang ada menggunakan AWS CLI.

```
aws dynamodb delete-backup \
--backup-arn arn:aws:dynamodb:us-east-1:123456789012:table/Music/
backup/01489602797149-73d8d5bc
```

Menggunakan IAM dengan pencadangan dan pemulihan DynamoDB

Anda dapat menggunakan AWS Identity and Access Management (IAM) untuk membatasi tindakan pencadangan dan pemulihan Amazon DynamoDB untuk beberapa sumber daya. Itu `CreateBackup` dan `RestoreTableFromBackup` APIs beroperasi berdasarkan per-tabel.

Untuk informasi selengkapnya tentang menggunakan kebijakan IAM dalam DynamoDB, lihat Kebijakan berbasis identitas untuk DynamoDB.

Berikut ini adalah contoh kebijakan IAM yang dapat Anda gunakan untuk mengonfigurasi fungsionalitas pencadangan dan pemulihan tertentu di DynamoDB.

Contoh 1: Izinkan CreateBackup dan RestoreTableFromBackup tindakan

Kebijakan IAM berikut memberikan izin untuk memungkinkan tindakan `CreateBackup` dan `RestoreTableFromBackup` DynamoDB pada semua tabel:

```
{
    "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**
>
> Izin RestoreTableFromBackup DynamoDB diperlukan pada cadangan sumber, dan izin baca dan tulis DynamoDB pada tabel target diperlukan untuk memulihkan fungsionalitas.
> Izin RestoreTableToPointInTime DynamoDB diperlukan pada tabel sumber, dan izin baca dan tulis DynamoDB pada tabel target diperlukan untuk memulihkan fungsionalitas.

Contoh 2: Izinkan CreateBackup dan tolak RestoreTableFromBackup

Kebijakan IAM berikut memberikan izin tindakan `CreateBackup` dan menolak tindakan `RestoreTableFromBackup`:

```
{
```

```
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": ["dynamodb:CreateBackup"],
            "Resource": "*"
        },
        {
            "Effect": "Deny",
            "Action": ["dynamodb:RestoreTableFromBackup"],
            "Resource": "*"
        }

    ]
}
```

Contoh 3: Izinkan ListBackups dan tolak CreateBackup dan RestoreTableFromBackup

Kebijakan IAM berikut memberikan izin tindakan `ListBackups` serta menolak tindakan `CreateBackup` dan `RestoreTableFromBackup`:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": ["dynamodb:ListBackups"],
            "Resource": "*"
        },
        {
            "Effect": "Deny",
            "Action": [
                "dynamodb:CreateBackup",
                "dynamodb:RestoreTableFromBackup"
            ],
            "Resource": "*"
        }

    ]
}
```

Contoh 4: Izinkan ListBackups dan tolak DeleteBackup

Kebijakan IAM berikut memberikan izin tindakan `ListBackups` dan menolak tindakan `DeleteBackup`:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": ["dynamodb:ListBackups"],
            "Resource": "*"
        },
        {
            "Effect": "Deny",
            "Action": ["dynamodb:DeleteBackup"],
            "Resource": "*"
        }

    ]
}
```

Contoh 5: Izinkan RestoreTableFromBackup dan DescribeBackup untuk semua sumber daya dan tolak DeleteBackup untuk cadangan tertentu

Kebijakan IAM berikut memberikan izin untuk tindakan `RestoreTableFromBackup` dan `DescribeBackup` serta menolak tindakan `DeleteBackup` untuk sumber daya cadangan tertentu:

```
{
    "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**
>
> Izin RestoreTableFromBackup DynamoDB diperlukan pada cadangan sumber, dan izin baca dan tulis DynamoDB pada tabel target diperlukan untuk memulihkan fungsionalitas.
> Izin RestoreTableToPointInTime DynamoDB diperlukan pada tabel sumber, dan izin baca dan tulis DynamoDB pada tabel target diperlukan untuk memulihkan fungsionalitas.

Contoh 6: Izinkan CreateBackup tabel tertentu

Kebijakan IAM berikut memberikan izin untuk tindakan `CreateBackup` hanya pada tabel `Movies`:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": ["dynamodb:CreateBackup"],
            "Resource": [
                "arn:aws:dynamodb:us-east-1:123456789012:table/Movies"
```

```
            ]
        }
    ]
}
```

Contoh 7: Izinkan ListBackups

Kebijakan IAM berikut memberikan izin tindakan `ListBackups`:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": ["dynamodb:ListBackups"],
            "Resource": "*"
        }
    ]
}
}
```

> **Important**
>
> Anda tidak dapat memberikan izin untuk `ListBackups` tindakan pada tabel tertentu.

Contoh 8: Izinkan akses ke AWS Backup fitur

Anda akan memerlukan izin API untuk tindakan `StartAwsBackupJob` agar pencadangan berhasil dengan fitur-fitur canggih, dan tindakan `dynamodb:RestoreTableFromAwsBackup` agar berhasil memulihkan cadangan tersebut.

Kebijakan IAM berikut memberikan AWS Backup izin untuk memicu pencadangan dengan fitur-fitur canggih dan pemulihan. Perhatikan juga bahwa jika tabel dienkripsi, kebijakan akan memerlukan akses ke kunci AWS KMS.

```
{
    "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:account-id:table/Books"
        },
        {
            "Sid": "AllowRestoreFromAwsBackup",
            "Effect": "Allow",
            "Action": ["dynamodb:RestoreTableFromAwsBackup"],
            "Resource": "*"
        },

    ]
}
```

Contoh 9: Tolak RestoreTableToPointInTime untuk Tabel Sumber Tertentu

Kebijakan IAM berikut menolak izin tindakan `RestoreTableToPointInTime` untuk tabel sumber tertentu:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Deny",
            "Action": [
                "dynamodb:RestoreTableToPointInTime"
            ],
            "Resource": "arn:aws:dynamodb:us-east-1:123456789012:table/Music"
        }
    ]
}
```

Contoh 10: Tolak RestoreTableFromBackup untuk semua Cadangan untuk Tabel Sumber Tertentu

Kebijakan IAM berikut menolak izin tindakan `RestoreTableToPointInTime` untuk semua cadangan untuk tabel sumber tertentu:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Deny",
            "Action": [
                "dynamodb:RestoreTableFromBackup"
            ],
            "Resource": "arn:aws:dynamodb:us-east-1:123456789012:table/Music/backup/*"
        }
    ]
}
```

Memahami penagihan Amazon DynamoDB untuk backup

Panduan ini memberikan rincian tentang cara kerja penagihan DynamoDB untuk backup. Kami akan memecah berbagai komponen yang berkontribusi pada biaya keseluruhan, memberikan penjelasan yang jelas dan contoh praktis.

DynamoDB menawarkan cadangan point-in-time dan pemulihan sesuai permintaan (PITR) untuk membantu melindungi data DynamoDB Anda dari peristiwa bencana dan menawarkan pengarsipan data untuk retensi jangka panjang.

Cara kerjanya

DynamoDB on-demand backup ditagih setiap bulan. Jika Anda mengambil cadangan pada hari tertentu dalam sebulan, Anda akan melihat satu tagihan untuk cadangan itu dihitung untuk sisa hari dalam sebulan (contoh: membuat cadangan pada tanggal 27, Anda hanya akan dikenakan biaya untuk beberapa hari tersisa di bulan itu, diterapkan sebagai biaya tunggal pada tanggal 27).

Jika Anda menyimpan cadangan yang diambil sebelumnya untuk bulan-bulan berikutnya, Anda akan selalu melihat biaya sebulan penuh untuk cadangan yang diterapkan pada tanggal 1. Jika cadangan dihapus sebelum akhir bulan, biaya akan disesuaikan berdasarkan penggunaan aktual.

Sebagai contoh, jika Anda membuat cadangan pada 27 Juli, dan dipertahankan hingga bulan Agustus, Anda akan melihat biaya berikut untuk cadangan itu:

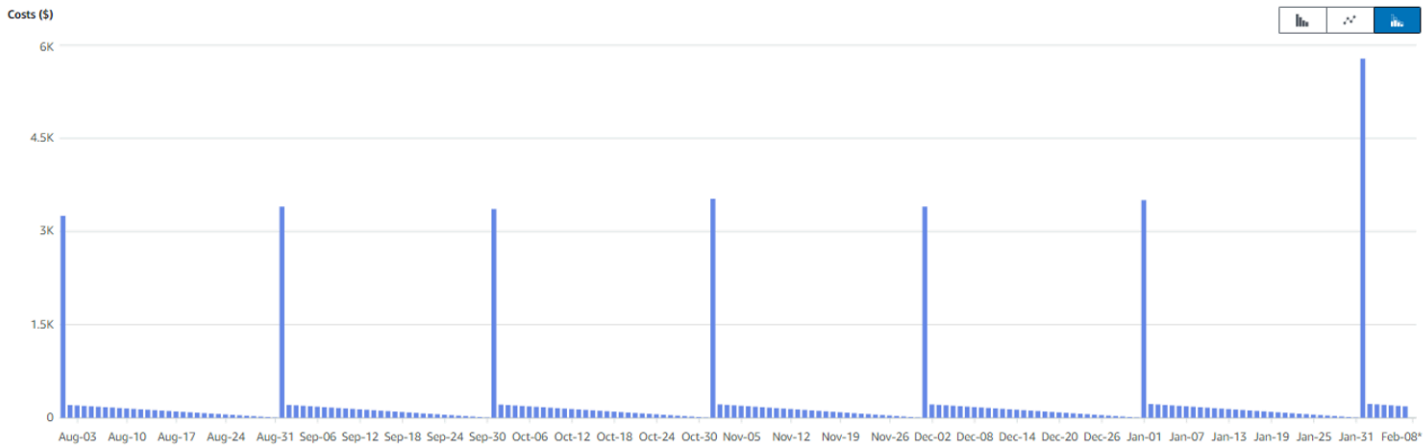
- Biaya pada 27 Juli untuk hari-hari yang tersisa di bulan Juli
- Biaya pada 1 Agustus untuk seluruh bulan Agustus
- Biaya pada tanggal 1 setiap bulan berikutnya bahwa cadangan ada

Ketika cadangan dipertahankan untuk tabel DynamoDB, Anda mungkin mengamati bahwa biaya untuk metrik `DynamoDB (Region)-TimedBackupStorage-ByteHrs` penggunaan tampaknya sangat tinggi pada tanggal 1 bulan itu. Selain itu, jika Anda memeriksa metrik ini di awal bulan baru dan membandingkannya dengan siklus penagihan sebelumnya, Anda dapat mengamati apa yang tampaknya merupakan lonjakan besar dalam penggunaan. Ini dengan desain. Pada tanggal 1 setiap bulan, setiap backup DynamoDB yang ada akan dikenakan biaya penggunaan untuk seluruh bulan yang diterapkan. Setiap cadangan DynamoDB yang dihapus selama sebulan akan memiliki biaya penggunaannya yang diprorata untuk mencerminkan penggunaan aktual. Akibatnya, Anda mungkin melihat biaya (diterapkan pada tanggal 1) menurun sepanjang bulan. Ini karena kebijakan retensi menerapkan kedaluwarsa atau penghapusan manual untuk pencadangan bawaan terjadi. Ini akan dieksplorasi dalam skenario di bawah ini.

Demikian pula, Anda akan melihat lonjakan yang lebih kecil sepanjang bulan saat cadangan baru dibuat, dengan biayanya diterapkan pada hari pembuatan untuk sisa bulan itu.

Contoh penagihan cadangan DynamoDB

Berikut adalah contoh dari apa yang mungkin Anda lihat di Cost Explorer pada awal bulan:



Perhatikan bagaimana 1 Februari tampaknya memiliki lonjakan yang jauh lebih besar dibandingkan bulan-bulan sebelumnya. Mari kita uraikan mengapa ini terjadi.

Dari Halaman Harga DynamoDB:

“Total ukuran penyimpanan cadangan yang ditagih setiap bulan adalah jumlah dari semua cadangan tabel DynamoDB. DynamoDB memantau ukuran backup on-demand secara terus menerus sepanjang bulan untuk menentukan biaya cadangan Anda.

Ini menjelaskan mengapa RUU tersebut secara konsisten menunjukkan lonjakan besar dalam penggunaan pada tanggal 1 setiap bulan. Setiap cadangan yang ada yang masuk ke bulan baru memiliki biaya sebulan penuh yang diterapkan pada tanggal 1. Dengan kata lain, jika Anda memasuki bulan dengan 300 cadangan DynamoDB, Anda akan melihat biaya penggunaan sebulan penuh diterapkan pada hari pertama bulan itu untuk semua 300 cadangan.

Pencadangan baru yang diambil sepanjang bulan akan dikenakan biaya penggunaan dari hari pembuatan hingga akhir bulan.

Bagaimana jika cadangan dihapus pertengahan bulan?

Berikut adalah beberapa skenario yang perlu dipertimbangkan:

1. Jika cadangan dari bulan sebelumnya dihapus pada tanggal 15 bulan berjalan, biaya penggunaan untuk cadangan itu, yang masih diterapkan pada tanggal 1, akan disesuaikan dengan penggunaan aktual alih-alih penggunaan bulan penuh yang diasumsikan sebelumnya diterapkan. Contoh di bawah ini menjelaskan hal ini secara lebih rinci.
2. Saat Anda membuat cadangan selama sebulan, biaya penggunaan untuk sisa bulan diterapkan pada hari pembuatannya. Namun, jika Anda menghapus cadangan ini sebelum bulan berakhir, biaya penggunaan Anda akan disesuaikan untuk menyertakan hanya hari-hari pencadangan aktif, masih berlaku pada tanggal pembuatannya.

Mengapa penggunaan bulan berjalan tampaknya jauh lebih tinggi pada tanggal 1 dari bulan sebelumnya, dan apa yang terjadi jika saya menghapus cadangan?

Untuk menjawab pertanyaan 2 bagian yang penting ini, mari kita buat contoh skenario menggunakan informasi berikut:

- Panjang Bulan: 30 Hari
- Frekuensi Cadangan DynamoDB: 10/hari, 300/bln
- Kebijakan Penyimpanan Cadangan DynamoDB: 30 Hari
- DynamoDB Per-Backup Biaya: $2/hari, $60/bln
- Total 1 Bulan Sebelumnya (`TimedBackupStorage-ByteHrs`, diperiksa pada tanggal 1 Bulan Berjalan): $9.300
- Total Bulan Sebelumnya (`TimedBackupStorage-ByteHrs`): $18,600
- Total 1 Bulan Saat Ini (`TimedBackupStorage-ByteHrs`, Diperiksa pada tanggal 1): $18.000
- Perubahan Penggunaan DynamoDB: Tidak Month-to-Month Ada

Dengan menggunakan informasi di atas, kita dapat melihat bahwa 300 cadangan dibuat pada bulan sebelumnya dengan kebijakan untuk mempertahankannya selama 30 hari. Pada tanggal 1 bulan baru, semua cadangan ini masih tetap ada karena belum mencapai akhir periode pemulihan mereka. Namun, setiap harinya, set cadangan tertua akan mulai turun, seperti yang ditunjukkan di sini:

Tabel dropoff cadangan DynamoDB

Bulan baru	Hari 1	Hari 2	Hari 3	Hari 4	Hari 5
Total backup bulan sebelumnya terbawa	300	290	280	270	260

- Pada tanggal 1, kita dapat melihat 300 cadangan @ $60/bln per cadangan, dengan total $18.000 diterapkan. `TimedBackupStorage-ByteHrs` Ini berbeda dengan bulan sebelumnya, di mana total bulan adalah $18.600.
- Pada tanggal 2, 10 dari cadangan tersebut akan kedaluwarsa dan drop off. Ketika ini terjadi, biaya yang diterapkan untuk cadangan tersebut akan disesuaikan dengan Penggunaan Aktual, bukan Penggunaan yang Diasumsikan. Ini menghasilkan 10 cadangan tersebut, sebelumnya dengan biaya yang diterapkan pada tanggal 1 dari $600 (10 Backup x 30 Hari) yang disesuaikan menjadi $20 (10 Backup x 1 Hari).
- Hari berikutnya, blok 10 berikutnya akan kedaluwarsa dan turun, mengalihkan penggunaannya dari 30 hari ke 2 hari, mengurangi biaya mereka menjadi $40 (10 Backup x 2 Hari).

Dengan berlalunya hari, kita akan melihat larger-than-previous-month lonjakan itu mulai menyusut. Jika kami memperluas ini untuk mencakup seluruh bulan, kami akan mengamati hal berikut:

Progresi Biaya Cadangan DynamoDB (1 Bulan)

300 cadangan di blok 10	1	ke-10	20	ke-30
Blok 1	$600	$20	$20	$20
Blok 2	$600	$40	$40	$40
Blok 3	$600	$60	$60	$60

300 cadangan di blok 10	1	ke-10	20	ke-30
Blok 4	$600	$80	$80	$80
Blok 5	$600	$100	$100	$100
Blok 6	$600	$120	$120	$120
Blok 7	$600	$140	$140	$140
Blok 8	$600	$160	$160	$160
Blok 9	$600	$180	$180	$180
Blok 10	$600	$600	$200	$200
Blok 11	$600	$600	$220	$220
Blok 12	$600	$600	$240	$240
Blok 13	$600	$600	$260	$260
Blok 14	$600	$600	$280	$280
Blok 15	$600	$600	$300	$300
Blok 16	$600	$600	$320	$320
Blok 17	$600	$600	$340	$340
Blok 18	$600	$600	$360	$360
Blok 19	$600	$600	$380	$380
Blok 20	$600	$600	$600	$400
Blok 21	$600	$600	$600	$420
Blok 22	$600	$600	$600	$440
Blok 23	$600	$600	$600	$460

300 cadangan di blok 10	1	ke-10	20	ke-30
Blok 24	$600	$600	$600	$480
Blok 25	$600	$600	$600	$500
Blok 26	$600	$600	$600	$520
Blok 27	$600	$600	$600	$540
Blok 28	$600	$600	$600	$560
Blok 29	$600	$600	$600	$580
Blok 30	$600	$600	$600	$600
Total 1 bulan ($)	$18.000	$13.500	$10.400	$9.300

Ketika blok baru turun setiap hari, penggunaannya disesuaikan dengan berapa hari itu ada, versus jumlah bulan penuh. Akibatnya, pada akhir bulan biaya yang diamati pada tanggal 1 akan turun dari $18.000 awal ke $9.300 yang diharapkan. Angka ini, dikombinasikan dengan cadangan yang baru dibuat sepanjang bulan (yang akan memiliki tabel penagihan yang mirip dengan yang di atas, tetapi terbalik), akan menghasilkan pengeluaran bulanan sesuai dengan $18.600 bulan lalu.

Kembalikan tabel di DynamoDB

Anda dapat memulihkan tabel DynamoDB dari cadangan PITR atau cadangan sesuai permintaan menggunakan AWS Management Console, Command Line Interface ()AWS CLI, AWS atau DynamoDB API. Proses pemulihan mengembalikan ke tabel DynamoDB baru.

Memulihkan tabel menggunakan point-in-time pemulihan

Anda dapat mengembalikan meja Anda ke titik waktu mana pun hingga`EarliestRestoreableDateTime`.

> **Important**
>
> Jika Anda menonaktifkan point-in-time pemulihan dan kemudian mengaktifkannya di atas meja, Anda mengatur ulang waktu mulai di mana Anda dapat memulihkan tabel

itu. Akibatnya, Anda hanya dapat segera memulihkan tabel tersebut menggunakan `LatestRestorableDateTime`.

Saat Anda memulihkan menggunakan point-in-time pemulihan, DynamoDB mengembalikan data tabel Anda ke status berdasarkan tanggal dan waktu yang dipilih (day:hour:minute:second) ke tabel baru. Anda memulihkan tabel tanpa mengonsumsi throughput apa pun yang disediakan pada tabel. Anda dapat melakukan pemulihan tabel lengkap menggunakan point-in-time pemulihan, atau Anda dapat mengonfigurasi pengaturan tabel tujuan. Anda dapat mengubah pengaturan tabel berikut pada tabel yang dipulihkan:

- Indeks sekunder global () GSIs
- Indeks sekunder lokal () LSIs
- Mode penagihan
- Kapasitas baca dan tulis yang disediakan
- Pengaturan enkripsi

Important

Saat Anda melakukan pemulihan tabel penuh, tabel tujuan diatur dengan unit kapasitas baca dan tulis yang disediakan yang sama seperti yang dimiliki tabel sumber saat cadangan diminta. Misalnya, throughput tabel yang disediakan baru-baru ini diturunkan menjadi 50 unit kapasitas baca dan 50 unit kapasitas tulis. Anda kemudian memulihkan status tabel ke tiga minggu lalu, saat throughput yang disediakan ditetapkan ke 100 unit kapasitas baca dan 100 unit kapasitas tulis. Dalam hal ini, DynamoDB memulihkan data tabel Anda ke titik waktu tersebut dengan throughput yang disediakan sejak saat itu (100 unit kapasitas baca dan 100 unit kapasitas tulis).

Anda juga dapat memulihkan data Wilayah AWS tabel DynamoDB Anda sedemikian rupa sehingga tabel yang dipulihkan dibuat di Wilayah yang berbeda dari tempat tabel sumber berada. Anda dapat melakukan Pemulihan lintas wilayah antara Wilayah AWS komersial, Wilayah AWS Tiongkok, dan. AWS GovCloud (US) Anda hanya membayar untuk data yang Anda transfer keluar dari Wilayah sumber dan untuk memulihkan ke tabel baru di Wilayah tujuan.

> **Note**
>
> Pemulihan Lintas Wilayah tidak didukung jika wilayah sumber atau tujuan adalah Asia Pasifik (Hong Kong) atau Timur Tengah (Bahrain).

Pemulihan bisa lebih cepat dan hemat biaya jika Anda mengecualikan beberapa atau semua indeks agar tidak dibuat pada tabel yang dipulihkan. Anda harus mengatur hal berikut secara manual pada tabel yang dipulihkan:

- Kebijakan penskalaan otomatis
- AWS Identity and Access Management kebijakan
- Metrik dan alarm CloudWatch Acara Amazon
- Tanda
- Pengaturan aliran
- Pengaturan Waktu untuk Tayang (TTL)
- Point-in-time pengaturan pemulihan

Waktu yang dibutuhkan untuk mengembalikan tabel bervariasi berdasarkan beberapa faktor dan tidak selalu berkorelasi dengan ukuran tabel.

Memulihkan tabel DynamoDB ke titik waktu

Amazon point-in-time DynamoDB recovery (PITR) menyediakan backup berkelanjutan dari data tabel DynamoDB Anda. Anda dapat memulihkan tabel ke titik waktu menggunakan konsol DynamoDB atau AWS Command Line Interface (AWS CLI). Proses point-in-time pemulihan mengembalikan ke tabel baru.

Jika Anda ingin menggunakan AWS CLI, Anda harus mengkonfigurasinya terlebih dahulu. Untuk informasi selengkapnya, lihat Mengakses DynamoDB.

Topik

- Memulihkan tabel DynamoDB ke suatu titik waktu (konsol)
- Memulihkan tabel ke titik waktu (AWS CLI)

Memulihkan tabel DynamoDB ke suatu titik waktu (konsol)

Contoh berikut menunjukkan cara menggunakan konsol DynamoDB untuk memulihkan tabel yang ada yang bernama `Music` ke titik waktu tertentu.

> **Note**
>
> Prosedur ini mengasumsikan bahwa Anda telah mengaktifkan point-in-time pemulihan. Untuk mengaktifkannya untuk **`Music`** tabel, pada tab Cadangan, di bagian Point-in-time pemulihan (PITR), pilih Edit dan kemudian centang kotak di sebelah Aktifkan. point-in-time-recovery

Untuk memulihkan tabel ke suatu titik waktu

1. Masuk ke AWS Management Console dan buka konsol DynamoDB di. https://console.aws.amazon.com/dynamodb/
2. Di panel navigasi di sisi kiri konsol, pilih Tabel.
3. Dalam daftar tabel, pilih tabel `Music`.
4. Pada tab Backup **`Music`** tabel, di bagian Point-in-time pemulihan (PITR), pilih Restore.
5. Untuk nama tabel baru, masukkan **`MusicMinutesAgo`**.

   > **Note**
   >
   > Anda dapat mengembalikan tabel ke AWS Wilayah yang sama atau ke Wilayah yang berbeda dari tempat tabel sumber berada. Anda juga dapat mengecualikan indeks sekunder dari yang dibuat pada tabel yang dipulihkan. Selain itu, Anda dapat menentukan mode enkripsi yang berbeda.

6. Untuk mengonfirmasi waktu yang dapat dipulihkan, atur tanggal dan waktu pemulihan ke Paling awal. Kemudian pilih Pulihkan untuk memulai proses pemulihan.

   Tabel yang sedang dipulihkan ditampilkan dengan status Memulihkan. Setelah proses pemulihan selesai, status tabel `MusicMinutesAgo` berubah menjadi Aktif.

Memulihkan tabel ke titik waktu (AWS CLI)

Prosedur berikut menunjukkan bagaimana menggunakan AWS CLI untuk mengembalikan tabel yang ada bernama `Music` ke titik waktu.

> **Note**
>
> Prosedur ini mengasumsikan bahwa Anda telah mengaktifkan point-in-time pemulihan. Untuk mengaktifkannya untuk tabel `Music`, jalankan perintah berikut.
>
> ```
> aws dynamodb update-continuous-backups \
>     --table-name Music \
>     --point-in-time-recovery-specification PointInTimeRecoveryEnabled=True
> ```

Untuk memulihkan tabel ke suatu titik waktu

1. Konfirmasikan bahwa point-in-time pemulihan diaktifkan untuk `Music` tabel dengan menggunakan `describe-continuous-backups` perintah.

   ```
   aws dynamodb describe-continuous-backups \
       --table-name Music
   ```

   Pencadangan berkelanjutan (diaktifkan secara otomatis pada pembuatan tabel) dan point-in-time pemulihan diaktifkan.

   ```
   {
       "ContinuousBackupsDescription": {
           "PointInTimeRecoveryDescription": {
               "PointInTimeRecoveryStatus": "ENABLED",
               "EarliestRestorableDateTime": 1519257118.0,
               "LatestRestorableDateTime": 1520018653.01
           },
           "ContinuousBackupsStatus": "ENABLED"
       }
   }
   ```

2. Memulihkan tabel ke titik waktu. Dalam hal ini, tabel `Music` dipulihkan ke `LatestRestorableDateTime` (~5 menit lalu) ke Wilayah AWS yang sama.

   ```
   aws dynamodb restore-table-to-point-in-time \
       --source-table-name Music \
       --target-table-name MusicMinutesAgo \
       --use-latest-restorable-time
   ```

> **Note**
>
> Anda juga dapat memulihkan ke titik waktu tertentu. Untuk melakukannya, jalankan perintah menggunakan argumen `--restore-date-time`, dan tentukan stempel waktu. Anda dapat menentukan titik waktu dalam periode pemulihan yang dikonfigurasi, yang dapat diatur ke nilai apa pun antara 1 dan 35 hari. Sebagai contoh, perintah berikut memulihkan tabel ke `EarliestRestorableDateTime`.
>
> ```
> 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
> ```
>
> Menentukan argumen `--no-use-latest-restorable-time` bersifat opsional ketika memulihkan ke titik waktu tertentu.

3. Memulihkan tabel ke titik waktu dengan pengaturan tabel kustom. Dalam hal ini, tabel `Music` dipulihkan ke `LatestRestorableDateTime` (~5 menit lalu).

   Anda dapat menentukan mode enkripsi yang berbeda untuk tabel dipulihkan, sebagai berikut.

   > **Note**
   >
   > Parameter `sse-specification-override` mengambil nilai yang sama seperti yang digunakan parameter `sse-specification-override` dalam perintah `CreateTable`. Untuk mempelajari informasi lebih lanjut, lihat Mengelola tabel yang dienkripsi di 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
   ```

Anda dapat mengembalikan tabel ke AWS Wilayah yang berbeda dari tempat tabel sumber berada.

> **Note**
>
> - Parameter `sse-specification-override` bersifat wajib untuk pemulihan lintas Wilayah, namun opsional untuk pemulihan ke Wilayah yang sama dengan tabel sumber.
> - Parameter `source-table-arn` harus disediakan untuk pemulihan lintas-wilayah.
> - Saat melakukan pemulihan lintas wilayah dari baris perintah, Anda harus mengatur AWS Wilayah default ke wilayah tujuan yang diinginkan. Untuk mempelajari selengkapnya, lihat Opsi baris perintah di Panduan Pengguna AWS Command Line Interface .

```
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
```

Anda dapat menimpa modus penagihan dan throughput yang disediakan untuk tabel yang dipulihkan.

```
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
```

Anda dapat mengecualikan beberapa atau semua indeks sekunder dari yang dibuat pada tabel yang baru dipulihkan.

> **Note**
>
> Pemulihan dapat lebih cepat dan lebih hemat biaya jika Anda mengecualikan beberapa atau semua indeks sekunder dari pembuatan pada tabel yang baru dipulihkan.

```
aws dynamodb restore-table-to-point-in-time \
    --source-table-name Music \
    --target-table-name MusicMinutesAgo \
    --use-latest-restorable-time \
    --global-secondary-index-override '[]'
```

Anda dapat menggunakan kombinasi dari penimpaan yang berbeda. Misalnya, Anda dapat menggunakan satu indeks sekunder global dan mengubah throughput yang disediakan secara bersamaan, sebagai berikut.

```
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
```

Untuk memverifikasi pemulihan, gunakan perintah `describe-table` untuk menggambarkan tabel `MusicEarliestRestorableDateTime`.

```
aws dynamodb describe-table --table-name MusicEarliestRestorableDateTime
```

Tabel yang sedang dipulihkan ditampilkan dengan status Membuat dan pemulihan dalam proses sebagai true. Setelah proses pemulihan selesai, status tabel `MusicEarliestRestorableDateTime` berubah menjadi Aktif.

> **Important**
>
> Saat pemulihan sedang berlangsung, jangan mengubah atau menghapus kebijakan AWS Identity and Access Management (IAM) yang memberikan izin kepada entitas IAM (misalnya, pengguna, grup, atau peran) untuk melakukan pemulihan. Jika tidak, perilaku tak terduga dapat terjadi. Misalnya, Anda menghapus izin menulis untuk sebuah tabel saat tabel tersebut sedang dipulihkan. Dalam kasus ini, operasi `RestoreTableToPointInTime` yang mendasarinya tidak dapat menulis data apa pun yang dipulihkan ke tabel. Kebijakan IAM yang melibatkan pembatasan IP sumber untuk mengakses tabel pemulihan target juga dapat menyebabkan masalah.
> Anda dapat mengubah atau menghapus izin hanya setelah operasi pemulihan selesai.

Menggunakan AWS Backup dengan DynamoDB

Amazon DynamoDB dapat membantu Anda memenuhi kepatuhan peraturan dan persyaratan kelangsungan bisnis melalui fitur pencadangan yang disempurnakan di. AWS Backup AWS Backup adalah layanan perlindungan data yang dikelola sepenuhnya yang memudahkan untuk memusatkan dan mengotomatiskan pencadangan di seluruh AWS layanan, di cloud, dan di tempat. Dengan menggunakan layanan ini, Anda dapat mengonfigurasi kebijakan pencadangan dan memantau aktivitas AWS sumber daya Anda di satu tempat. Untuk menggunakannya AWS Backup, Anda harus secara tegas ikut serta. Pilihan keikutsertaan berlaku untuk akun dan AWS Wilayah tertentu, jadi Anda mungkin harus memilih beberapa Wilayah menggunakan akun yang sama. Untuk informasi selengkapnya, lihat Panduan Developer AWS Backup.

Amazon DynamoDB terintegrasi secara native dengan. AWS Backup Anda dapat menggunakan AWS Backup untuk menjadwalkan, menyalin, menandai, dan siklus hidup backup DynamoDB sesuai permintaan Anda secara otomatis. Anda dapat terus melihat dan memulihkan cadangan ini dari konsol DynamoDB. Anda dapat menggunakan konsol DynamoDB, API, AWS dan Command Line Interface (AWS CLI) untuk mengaktifkan pencadangan otomatis untuk tabel DynamoDB Anda.

> **Note**
>
> Setiap pencadangan yang dilakukan melalui DynamoDB tidak akan berubah. Anda masih dapat membuat cadangan melalui alur kerja DynamoDB saat ini.

Fitur cadangan yang disempurnakan tersedia melalui AWS Backup meliputi:

Pencadangan terjadwal - Anda dapat mengatur pencadangan terjadwal secara teratur dari tabel DynamoDB Anda menggunakan paket pencadangan.

Penyalinan lintas akun dan lintas wilayah - Anda dapat secara otomatis menyalin cadangan ke brankas cadangan lain di AWS Wilayah atau akun lain, yang memungkinkan Anda untuk mendukung persyaratan perlindungan data Anda.

Tingkat penyimpanan dingin - Anda dapat mengonfigurasi cadangan untuk menerapkan aturan siklus hidup guna menghapus atau mentransisikan cadangan ke penyimpanan yang lebih dingin. Hal ini dapat membantu Anda mengoptimalkan biaya cadangan Anda.

Tanda - Anda dapat secara otomatis menandai cadangan Anda untuk tujuan penagihan dan alokasi biaya.

Enkripsi — Pencadangan berdasarkan permintaan DynamoDB yang dikelola AWS Backup sekarang disimpan di brankas. AWS Backup Ini memungkinkan Anda untuk mengenkripsi dan mengamankan backup Anda dengan menggunakan yang independen dari kunci enkripsi AWS KMS key tabel DynamoDB Anda.

Pencadangan audit — Anda dapat menggunakan AWS Backup Audit Manager untuk mengaudit kepatuhan AWS Backup kebijakan Anda dan untuk menemukan aktivitas cadangan dan sumber daya yang belum sesuai dengan kontrol yang Anda tetapkan. Anda juga dapat menggunakannya untuk secara otomatis membuat jejak audit laporan harian dan sesuai permintaan untuk tujuan tata kelola cadangan Anda.

Cadangan aman menggunakan model WORM - Anda dapat menggunakan AWS Backup Vault Lock untuk mengaktifkan pengaturan write-once-read-many (WORM) untuk cadangan Anda. Dengan AWS Backup Vault Lock, Anda dapat menambahkan lapisan pertahanan tambahan yang melindungi cadangan dari operasi penghapusan yang tidak disengaja atau berbahaya, perubahan pada periode pemulihan cadangan, dan pembaruan pengaturan siklus hidup. Untuk mempelajari lebih lanjut, lihat AWS Backup Vault Lock.

Fitur cadangan yang disempurnakan ini tersedia di semua AWS Wilayah. Untuk mempelajari lebih lanjut tentang fitur-fitur ini, lihat Panduan Developer AWS Backup.

Topik

- Mencadangkan dan memulihkan tabel DynamoDB dengan: Cara kerjanya AWS Backup
- Membuat cadangan tabel DynamoDB dengan AWS Backup
- Menyalin cadangan tabel DynamoDB dengan AWS Backup

- Memulihkan cadangan tabel DynamoDB dari AWS Backup
- Menghapus cadangan tabel DynamoDB dengan AWS Backup
- Perbedaan catatan penggunaan antara backup on-demand yang dikelola oleh dan DynamoDB AWS Backup

Mencadangkan dan memulihkan tabel DynamoDB dengan: Cara kerjanya AWS Backup

Anda dapat menggunakan fitur pencadangan sesuai permintaan untuk mencadangkan penuh tabel Amazon DynamoDB Anda. Bagian ini memberikan gambaran umum tentang apa yang terjadi selama proses pencadangan dan pemulihan.

Pencadangan

Saat Anda membuat cadangan sesuai permintaan AWS Backup, penanda waktu permintaan dikatalogkan. Cadangan dibuat secara asinkron dengan menerapkan semua perubahan hingga waktu permintaan ke snapshot tabel lengkap terakhir.

Setiap kali Anda membuat cadangan sesuai permintaan, seluruh data tabel dicadangkan. Tidak ada batasan untuk jumlah pencadangan sesuai permintaan.

> **Note**
>
> Tidak seperti DynamoDB Backups, backup yang dibuat dengan tidak instan. AWS Backup

Saat pencadangan sedang berlangsung, Anda tidak dapat melakukan hal berikut:

- Menjeda atau membatalkan operasi pencadangan.
- Menghapus tabel sumber cadangan.
- Nonaktifkan pencadangan pada tabel jika pencadangan untuk tabel tersebut sedang berlangsung.

AWS Backup menyediakan jadwal pencadangan otomatis, manajemen retensi, dan manajemen siklus hidup. Ini menghilangkan kebutuhan akan skrip khusus dan proses manual. AWS Backup menjalankan cadangan dan menghapusnya saat kedaluwarsa. Lihat informasi selengkapnya di Panduan Developer AWS Backup.

Jika Anda menggunakan konsol, cadangan apa pun yang dibuat menggunakan AWS Backup tercantum di tab Cadangan dengan jenis Cadangan yang disetel ke. `AWS_BACKUP`

> **Note**
>
> Anda tidak dapat menghapus cadangan yang ditandai dengan Jenis cadangan `AWS_BACKUP` menggunakan konsol DynamoDB. Untuk mengelola cadangan ini, gunakan konsol. AWS Backup

Untuk mempelajari cara melakukan pencadangan, lihat Mencadangkan tabel DynamoDB.

Memulihkan

Anda memulihkan tabel tanpa mengonsumsi throughput apa pun yang disediakan pada tabel. Anda dapat melakukan pemulihan tabel penuh dari cadangan DynamoDB Anda, atau Anda dapat mengonfigurasi pengaturan tabel tujuan. Saat melakukan pemulihan, Anda dapat mengubah pengaturan tabel berikut:

- Indeks sekunder global () GSIs
- Indeks sekunder lokal () LSIs
- Mode penagihan
- Kapasitas baca dan tulis yang disediakan
- Pengaturan enkripsi

> **Important**
>
> Saat Anda melakukan pemulihan tabel penuh, tabel tujuan diatur dengan unit kapasitas baca dan tulis yang disediakan yang sama seperti yang dimiliki tabel sumber saat cadangan diminta. Proses pemulihan juga memulihkan indeks sekunder lokal dan indeks sekunder global.

Anda dapat menyalin cadangan data tabel DynamoDB Anda ke Wilayah AWS lain dan kemudian mengembalikannya di Wilayah baru tersebut. Anda dapat menyalin dan kemudian memulihkan cadangan antara Wilayah AWS komersial, Wilayah AWS Tiongkok, dan Wilayah AWS GovCloud

(AS). Anda hanya membayar data yang Anda salin dari Wilayah sumber dan data yang Anda pulihkan ke tabel baru di Wilayah tujuan.

AWS Backup akan mengembalikan tabel dengan semua indeks asli.

Anda harus menyiapkan berikut ini secara manual pada tabel yang dipulihkan:

- Kebijakan penskalaan otomatis
- AWS Identity and Access Management Kebijakan (IAM)
- CloudWatch Metrik dan alarm Amazon
- Tanda
- Pengaturan aliran
- Pengaturan Waktu untuk Tayang (TTL)
- Pengaturan perlindungan penghapusan
- Pengaturan pemulihan titik waktu (PITR)

Anda hanya dapat memulihkan seluruh data tabel ke tabel baru dari cadangan. Anda dapat menulis ke tabel yang dipulihkan hanya setelah tabel tersebut aktif.

> **Note**
>
> AWS Backup pemulihan tidak merusak. Anda tidak dapat menimpa tabel yang ada selama operasi pemulihan.

Metrik layanan menunjukkan bahwa 95 persen pemulihan tabel pelanggan selesai dalam waktu kurang dari satu jam. Namun, waktu pemulihan terkait langsung dengan konfigurasi tabel Anda (seperti ukuran tabel dan jumlah partisi yang mendasarinya) dan variabel terkait lainnya. Praktik terbaik saat merencanakan pemulihan bencana adalah mendokumentasikan waktu penyelesaian pemulihan rata-rata secara teratur dan menetapkan bagaimana waktu tersebut memengaruhi Sasaran Waktu Pemulihan Anda secara keseluruhan.

Untuk mempelajari cara melakukan pemulihan, lihat Memulihkan tabel DynamoDB dari cadangan.

Anda dapat menggunakan kebijakan IAM untuk kontrol akses. Untuk informasi selengkapnya, lihat Menggunakan IAM dengan pencadangan dan pemulihan DynamoDB.

Semua konsol cadangan dan pemulihan serta tindakan API ditangkap dan direkam di AWS CloudTrail untuk pencatatan, pemantauan berkelanjutan, dan audit.

Membuat cadangan tabel DynamoDB dengan AWS Backup

Bagian ini menjelaskan cara mengaktifkan AWS Backup untuk membuat cadangan sesuai permintaan dan terjadwal dari tabel DynamoDB Anda.

Topik

- Menghidupkan AWS Backup fitur
- Pencadangan sesuai permintaan
- Pencadangan terjadwal

Menghidupkan AWS Backup fitur

Anda harus mengaktifkan AWS Backup untuk menggunakannya dengan DynamoDB.

Untuk menghidupkan AWS Backup, lanjutkan langkah-langkah berikut:

1. Masuk ke AWS Management Console dan buka konsol DynamoDB di. https://console.aws.amazon.com/dynamodb/
2. Di panel navigasi pada sisi kiri konsol, pilih Cadangan.
3. Di jendela Pengaturan Cadangan, pilih Aktifkan.
4. Layar konfirmasi akan muncul. Pilih Aktifkan fitur.

AWS Backup fitur sekarang tersedia untuk tabel DynamoDB Anda.

Jika Anda memilih untuk menonaktifkan AWS Backup fitur setelah diaktifkan, ikuti langkah-langkah berikut:

1. Masuk ke AWS Management Console dan buka konsol DynamoDB di. https://console.aws.amazon.com/dynamodb/
2. Di panel navigasi pada sisi kiri konsol, pilih Cadangan.
3. Di jendela Pengaturan Cadangan, pilih Nonaktifkan.
4. Layar konfirmasi akan muncul. Pilih Nonaktifkan fitur.

Jika Anda tidak dapat mengaktifkan atau menonaktifkan AWS Backup fitur, AWS admin Anda mungkin perlu melakukan tindakan tersebut.

Pencadangan sesuai permintaan

Untuk membuat cadangan sesuai permintaan tabel DynamoDB, ikuti langkah-langkah berikut:

1. Masuk ke AWS Management Console dan buka konsol DynamoDB di. https://console.aws.amazon.com/dynamodb/
2. Di panel navigasi pada sisi kiri konsol, pilih Cadangan.
3. Pilih Buat cadangan.
4. Dari menu dropdown yang muncul, pilih Buat cadangan sesuai permintaan.
5. Untuk membuat cadangan yang dikelola AWS Backup dengan penyimpanan hangat dan fitur dasar lainnya, pilih Pengaturan Default. Untuk membuat cadangan yang dapat dialihkan ke penyimpanan dingin, atau untuk membuat cadangan dengan fitur DynamoDB, bukan, pilih Sesuaikan pengaturan. AWS Backup

   Jika Anda ingin membuat cadangan ini dengan fitur DynamoDB sebelumnya, pilih Sesuaikan pengaturan lalu pilih Cadangkan dengan DynamoDB.
6. Setelah Anda menyelesaikan pengaturan, pilih Buat cadangan.

Pencadangan terjadwal

Untuk menjadwalkan cadangan, ikuti langkah-langkah ini.

1. Masuk ke AWS Management Console dan buka konsol DynamoDB di. https://console.aws.amazon.com/dynamodb/
2. Di panel navigasi pada sisi kiri konsol, pilih Cadangan.
3. Dari menu tarik-turun yang muncul, pilih Jadwalkan pencadangan dengan. AWS Backup
4. Anda akan dibawa AWS Backup untuk membuat rencana cadangan.

Menyalin cadangan tabel DynamoDB dengan AWS Backup

Anda dapat membuat salinan dari cadangan saat ini. Anda dapat menyalin cadangan ke beberapa AWS akun atau AWS Wilayah sesuai permintaan atau secara otomatis sebagai bagian dari rencana

pencadangan terjadwal. Anda juga dapat mengotomatiskan urutan salinan lintas akun dan lintas Wilayah untuk Amazon DynamoDB Encryption Client.

Replikasi Lintas Wilayah sangat berharga jika Anda memiliki kelangsungan bisnis atau persyaratan kepatuhan untuk menyimpan cadangan dengan jarak minimum dari data produksi Anda.

Pencadangan lintas akun berguna untuk menyalin cadangan Anda dengan aman ke satu atau beberapa AWS akun di organisasi Anda untuk alasan operasional atau keamanan. Jika cadangan asli Anda terhapus, Anda dapat menyalin cadangan dari akun tujuannya ke akun sumbernya, lalu memulai pemulihan. Sebelum Anda dapat melakukan ini, Anda harus memiliki dua akun yang dimiliki oleh organisasi yang sama dalam layanan Organizations.

Salinan mewarisi konfigurasi cadangan sumber kecuali Anda menentukan sebaliknya, dengan satu pengecualian: jika Anda menentukan bahwa salinan baru Anda "Tidak Pernah" kedaluwarsa. Dengan pengaturan ini, salinan baru masih mewarisi tanggal kedaluwarsa sumbernya. Jika Anda ingin salinan cadangan baru Anda bersifat permanen, atur cadangan sumber Anda agar tidak pernah kedaluwarsa, atau tentukan salinan baru Anda untuk kedaluwarsa 100 tahun setelah pembuatannya.

> **Note**
>
> Jika Anda menyalin ke akun lain, Anda harus memiliki izin dari akun tersebut terlebih dahulu.

Untuk menyalin cadangan, lakukan hal berikut:

1. Masuk ke AWS Management Console dan buka konsol DynamoDB di. https://console.aws.amazon.com/dynamodb/
2. Di panel navigasi pada sisi kiri konsol, pilih Cadangan.
3. Pilihlah kotak centang di samping cadangan yang Anda salin.
   - Jika cadangan yang ingin Anda salin berwarna abu-abu, Anda harus mengaktifkan fitur-fitur lanjutan. AWS Backup Kemudian, buat cadangan baru. Anda sekarang dapat menyalin cadangan baru ini ke Wilayah dan akun lain, serta menyalin cadangan baru lain nantinya.
4. Pilih Salin.
5. Jika Anda ingin menyalin cadangan ke akun atau Wilayah lain, pilih kotak centang di sebelah Salin titik pemulihan ke tujuan lain. Kemudian pilih apakah Anda akan menyalin ke Wilayah lain di akun Anda, atau ke akun lain di Wilayah yang berbeda.

> **Note**
>
> Untuk memulihkan cadangan ke Wilayah atau akun lain, Anda harus terlebih dahulu menyalin cadangan ke Wilayah atau akun tersebut.

6. Pilih brankas untuk tempat menyimpan file salinan. Anda juga dapat membuat brankas cadangan baru jika diinginkan.
7. Pilih Salin cadangan.

Memulihkan cadangan tabel DynamoDB dari AWS Backup

Bagian ini menjelaskan cara mengembalikan cadangan tabel DynamoDB dari. AWS Backup

Topik

- Memulihkan tabel DynamoDB dari AWS Backup
- Memulihkan tabel DynamoDB ke Wilayah atau akun lain

Memulihkan tabel DynamoDB dari AWS Backup

Untuk memulihkan tabel DynamoDB Anda, ikuti langkah-langkah AWS Backup berikut:

1. Masuk ke AWS Management Console dan buka konsol DynamoDB di https://console.aws.amazon.com/dynamodb/
2. Di panel navigasi di sisi kiri konsol, pilih Tabel.
3. Pilih tab Backup.
4. Centang kotak centang di samping cadangan sebelumnya yang ingin Anda pulihkan.
5. Pilih Pulihkan. Anda akan dibawa ke tabel Pulihkan dari layar cadangan.
6. Masukkan nama untuk tabel yang baru dipulihkan, enkripsi yang dimiliki tabel baru ini, kunci yang ingin Anda gunakan untuk mengenkripsi pemulihan, dan opsi lainnya.
7. Setelah selesai, pilih Pulihkan.

Memulihkan tabel DynamoDB ke Wilayah atau akun lain

Untuk mengembalikan tabel DynamoDB ke Wilayah atau akun lain, Anda harus terlebih dahulu menyalin cadangan ke Wilayah atau akun baru tersebut. Untuk menyalin ke akun lain, akun tersebut

harus memberi Anda izin terlebih dahulu. Setelah Anda menyalin cadangan DynamoDB Anda ke Wilayah atau akun baru, cadangan tersebut dapat dipulihkan dengan proses di bagian sebelumnya.

Menghapus cadangan tabel DynamoDB dengan AWS Backup

Bagian ini menjelaskan cara menghapus cadangan tabel DynamoDB dengan. AWS Backup

Cadangan DynamoDB yang dibuat melalui fitur AWS Backup disimpan di brankas Backup. AWS

Untuk menghapus cadangan semacam ini, lakukan hal berikut:

1. Masuk ke AWS Management Console dan buka konsol DynamoDB di. https://console.aws.amazon.com/dynamodb/
2. Di panel navigasi pada sisi kiri konsol, pilih Cadangan.
3. Pada layar berikut, pilih Continue to AWS Backup.

   Anda akan dibawa ke Konsol AWS Backup. Untuk mempelajari lebih lanjut tentang cara menghapus cadangan di Konsol AWS Backup, lihat Menghapus cadangan.

   Untuk informasi selengkapnya, AWS Backup lihat Backup dan recovery menggunakan AWS Backup dalam Panduan AWS Preskriptif.

Perbedaan catatan penggunaan antara backup on-demand yang dikelola oleh dan DynamoDB AWS Backup

Bagian ini menjelaskan perbedaan teknis antara pencadangan sesuai permintaan yang dikelola oleh AWS Backup dan DynamoDB.

AWS Backup memiliki beberapa alur kerja dan perilaku yang berbeda dari DynamoDB. Ini termasuk:

Enkripsi - Cadangan yang dibuat dengan AWS Backup paket disimpan dalam brankas terenkripsi dengan kunci yang dikelola oleh layanan. AWS Backup Vault memiliki kebijakan kontrol akses untuk keamanan tambahan.

Backup ARN - File cadangan yang dibuat oleh sekarang AWS Backup akan memiliki AWS Backup ARN, yang dapat memengaruhi model izin pengguna. Nama sumber daya cadangan (ARNs) akan berubah dari `arn:aws:dynamodb` `kearn:aws:backup`.

Menghapus cadangan - Cadangan yang dibuat dengan hanya AWS Backup dapat dihapus dari brankas. AWS Backup Anda tidak akan dapat menghapus AWS Backup file dari konsol DynamoDB.

Proses pencadangan - Tidak seperti pencadangan DynamoDB, cadangan yang dibuat dengan AWS Backup tidak terjadi secara instan.

Penagihan - Cadangan tabel DynamoDB dengan fitur ditagih dari. AWS Backup AWS Backup

Peran IAM - Jika Anda mengelola akses melalui peran IAM, Anda juga perlu mengonfigurasi peran IAM baru dengan izin baru ini:

```
"dynamodb:StartAwsBackupJob",
"dynamodb:RestoreTableFromAwsBackup"
```

`dynamodb:StartAwsBackupJob`diperlukan untuk pencadangan yang sukses dengan AWS Backup fitur, dan `dynamodb:RestoreTableFromAwsBackup` diperlukan untuk memulihkan dari cadangan yang dibuat dengan AWS Backup fitur.

Untuk melihat izin ini dalam kebijakan IAM lengkap, lihat Contoh 8 dalam Menggunakan IAM.

Contoh kode untuk DynamoDB menggunakan AWS SDKs

Contoh kode berikut menunjukkan cara menggunakan DynamoDB dengan kit pengembangan perangkat lunak (AWS SDK).

Dasar-dasar adalah contoh kode yang menunjukkan kepada Anda bagaimana melakukan operasi penting dalam suatu layanan.

Tindakan merupakan kutipan kode dari program yang lebih besar dan harus dijalankan dalam konteks. Sementara tindakan menunjukkan cara memanggil fungsi layanan individual, Anda dapat melihat tindakan dalam konteks dalam skenario terkait.

Skenario adalah contoh kode yang menunjukkan kepada Anda bagaimana menyelesaikan tugas tertentu dengan memanggil beberapa fungsi dalam layanan atau dikombinasikan dengan yang lain Layanan AWS.

AWS kontribusi komunitas adalah contoh yang dibuat dan dikelola oleh banyak tim AWS. Untuk memberikan umpan balik, gunakan mekanisme yang disediakan di repositori terkait.

Untuk daftar lengkap panduan pengembang AWS SDK dan contoh kode, lihatMenggunakan DynamoDB dengan SDK AWS. Topik ini juga mencakup informasi tentang memulai dan detail tentang versi SDK sebelumnya.

Memulai

Halo DynamoDB

Contoh kode berikut ini menunjukkan cara untuk mulai menggunakan DynamoDB.

.NET

SDK untuk .NET

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
using Amazon.DynamoDBv2;
```

```
using Amazon.DynamoDBv2.Model;

namespace DynamoDB_Actions;

public static class HelloDynamoDB
{
    static async Task Main(string[] args)
    {
        var dynamoDbClient = new AmazonDynamoDBClient();

        Console.WriteLine($"Hello Amazon Dynamo DB! Following are some of your
 tables:");
        Console.WriteLine();

        // You can use await and any of the async methods to get a response.
        // Let's get the first five tables.
        var response = await dynamoDbClient.ListTablesAsync(
            new ListTablesRequest()
            {
                Limit = 5
            });

        foreach (var table in response.TableNames)
        {
            Console.WriteLine($"\tTable: {table}");
            Console.WriteLine();
        }
    }
}
```

- Untuk detail API, lihat ListTablesdi Referensi AWS SDK untuk .NET API.

C++

SDK untuk C++

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

Kode untuk CMake file 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})
```

Kode untuk file sumber 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;
}
```

- Untuk detail API, lihat ListTablesdi Referensi AWS SDK untuk C++ API.

Java

SDK untuk Java 2.x

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
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!");
    }
}
```

- Untuk detail API, lihat ListTablesdi Referensi AWS SDK for Java 2.x API.

JavaScript

SDK untuk JavaScript (v3)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

Untuk detail lebih lanjut tentang bekerja dengan DynamoDB di, lihat Pemrograman DynamoDB AWS SDK untuk JavaScript dengan. 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;
```

```
};
```

- Untuk detail API, lihat ListTablesdi Referensi AWS SDK untuk JavaScript API.

Python

SDK untuk Python (Boto3)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
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.")
```

- Untuk detail API, lihat ListTablesdi AWS SDK for Python (Boto3) Referensi API.

Ruby

SDK untuk Ruby

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
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
```

- Untuk detail API, lihat ListTablesdi Referensi AWS SDK untuk Ruby API.

Contoh kode

- Contoh dasar untuk DynamoDB menggunakan AWS SDKs
  - Halo DynamoDB
  - Pelajari dasar-dasar DynamoDB dengan SDK AWS
  - Tindakan untuk DynamoDB menggunakan AWS SDKs
    - Gunakan BatchExecuteStatement dengan AWS SDK
    - Gunakan BatchGetItem dengan AWS SDK atau CLI
    - Gunakan BatchWriteItem dengan AWS SDK atau CLI
    - Gunakan CreateTable dengan AWS SDK atau CLI
    - Gunakan DeleteItem dengan AWS SDK atau CLI
    - Gunakan DeleteTable dengan AWS SDK atau CLI
    - Gunakan DescribeTable dengan AWS SDK atau CLI
    - Gunakan DescribeTimeToLive dengan AWS SDK atau CLI
    - Gunakan ExecuteStatement dengan AWS SDK
    - Gunakan GetItem dengan AWS SDK atau CLI
    - Gunakan ListTables dengan AWS SDK atau CLI
    - Gunakan PutItem dengan AWS SDK atau CLI
    - Gunakan Query dengan AWS SDK atau CLI

    - Gunakan Scan dengan AWS SDK atau CLI
    - Gunakan UpdateItem dengan AWS SDK atau CLI
    - Gunakan UpdateTable dengan AWS SDK atau CLI
    - Gunakan UpdateTimeToLive dengan AWS SDK atau CLI
- Skenario untuk DynamoDB menggunakan AWS SDKs
    - Mempercepat pembacaan DynamoDB dengan DAX menggunakan SDK AWS
    - Bekerja dengan skenario Indeks Sekunder Global DynamoDB tingkat lanjut menggunakan v2 AWS Command Line Interface
    - Membangun aplikasi untuk mengirimkan data ke tabel DynamoDB
    - Bandingkan beberapa nilai dengan satu atribut di DynamoDB dengan SDK AWS
    - Perbarui item DynamoDB secara kondisional dengan TTL menggunakan SDK AWS
    - Connect ke instance DynamoDB lokal menggunakan SDK AWS
    - Hitung operator ekspresi di DynamoDB dengan SDK AWS
    - Membuat API REST Gateway API untuk melacak data COVID-19
    - Membuat aplikasi messenger dengan Step Functions
    - Membuat aplikasi manajemen aset foto yang memungkinkan pengguna mengelola foto menggunakan label
    - Membuat tabel DynamoDB dengan Indeks Sekunder Global menggunakan SDK AWS
    - Buat tabel DynamoDB dengan pengaturan throughput hangat menggunakan SDK AWS
    - Membuat aplikasi web untuk melacak data DynamoDB
    - Membuat aplikasi obrolan websocket dengan API Gateway
    - Buat item DynamoDB dengan TTL menggunakan SDK AWS
    - Hapus data DynamoDB menggunakan pernyataan PartiQL DELETE dengan SDK AWS
    - Mendeteksi APD dalam gambar dengan Amazon AWS Rekognition menggunakan SDK
    - Masukkan data DynamoDB menggunakan pernyataan PartiQL INSERT dengan SDK AWS
    - Menginvokasi fungsi Lambda dari browser
    - Kelola Indeks Sekunder Global DynamoDB menggunakan v2 AWS Command Line Interface
    - Mengelola kebijakan berbasis sumber daya DynamoDB menggunakan v2 AWS Command Line Interface
    - Memantau kinerja Amazon DynamoDB menggunakan SDK AWS
    - Melakukan operasi kueri DynamoDB lanjutan menggunakan SDK AWS

- Lakukan operasi daftar di DynamoDB dengan SDK AWS
- Melakukan operasi peta di DynamoDB dengan SDK AWS
- Lakukan operasi set di DynamoDB dengan SDK AWS
- Kueri tabel DynamoDB dengan menggunakan kumpulan pernyataan PartiQL dan SDK AWS
- Kueri tabel DynamoDB menggunakan PartiQL dan SDK AWS
- Kueri tabel DynamoDB menggunakan Global Secondary Index dengan SDK AWS
- Kueri tabel DynamoDB menggunakan kondisi begins_with dengan SDK AWS
- Kueri tabel DynamoDB menggunakan rentang tanggal dalam kunci pengurutan dengan SDK AWS
- Kueri tabel DynamoDB dengan ekspresi filter kompleks dengan SDK AWS
- Kueri tabel DynamoDB dengan ekspresi filter dinamis dengan SDK AWS
- Kueri tabel DynamoDB dengan ekspresi filter dan batas dengan SDK AWS
- Kueri tabel DynamoDB dengan atribut bersarang menggunakan SDK AWS
- Kueri tabel DynamoDB dengan pagination menggunakan SDK AWS
- Kueri tabel DynamoDB dengan pembacaan yang sangat konsisten menggunakan SDK AWS
- Kueri data DynamoDB menggunakan pernyataan PartiQL SELECT dengan SDK AWS
- Kueri tabel DynamoDB untuk item TTL menggunakan SDK AWS
- Kueri tabel DynamoDB menggunakan pola tanggal dan waktu dengan SDK AWS
- Simpan EXIF dan informasi gambar lainnya menggunakan SDK AWS
- Mengatur Kontrol Akses Berbasis Atribut untuk DynamoDB menggunakan v2 AWS Command Line Interface
- Memahami urutan ekspresi pembaruan di DynamoDB dengan SDK AWS
- Memperbarui setelan tabel DynamoDB dengan throughput hangat menggunakan SDK AWS
- Memperbarui item DynamoDB dengan TTL menggunakan SDK AWS
- Perbarui data DynamoDB menggunakan pernyataan PartiQL UPDATE dengan SDK AWS
- Menggunakan API Gateway untuk menginvokasi fungsi Lambda
- Menggunakan Step Functions untuk menginvokasi fungsi Lambda
- Menggunakan model dokumen untuk DynamoDB menggunakan SDK AWS
- Menggunakan model persistensi objek tingkat tinggi untuk DynamoDB menggunakan SDK AWS

- Gunakan operasi penghitung atom di DynamoDB dengan SDK AWS
- Menggunakan operasi bersyarat di DynamoDB dengan SDK AWS

- Gunakan nama atribut ekspresi di DynamoDB dengan SDK AWS
- Menggunakan peristiwa terjadwal untuk menginvokasi fungsi Lambda
- Bekerja dengan DynamoDB Local Secondary Indexes menggunakan v2 AWS Command Line Interface
- Bekerja dengan DynamoDB Streams dan menggunakan v2 Time-to-Live AWS Command Line Interface
- Bekerja dengan tabel global DynamoDB dan replikasi multi-wilayah menggunakan v2 AWS Command Line Interface
- Bekerja dengan penandaan sumber daya DynamoDB menggunakan v2 AWS Command Line Interface
- Bekerja dengan enkripsi tabel DynamoDB menggunakan v2 AWS Command Line Interface

- Contoh tanpa server untuk DynamoDB
  - Memanggil fungsi Lambda dari pemicu DynamoDB
  - Melaporkan kegagalan item batch untuk fungsi Lambda dengan pemicu DynamoDB
- AWS kontribusi komunitas untuk DynamoDB
  - Membangun dan menguji aplikasi tanpa server

Contoh dasar untuk DynamoDB menggunakan AWS SDKs

Contoh kode berikut menunjukkan cara menggunakan dasar-dasar Amazon AWS SDKs DynamoDB dengan.

Contoh

- Halo DynamoDB
- Pelajari dasar-dasar DynamoDB dengan SDK AWS
- Tindakan untuk DynamoDB menggunakan AWS SDKs
  - Gunakan BatchExecuteStatement dengan AWS SDK
  - Gunakan BatchGetItem dengan AWS SDK atau CLI
  - Gunakan BatchWriteItem dengan AWS SDK atau CLI
  - Gunakan CreateTable dengan AWS SDK atau CLI
  - Gunakan DeleteItem dengan AWS SDK atau CLI
  - Gunakan DeleteTable dengan AWS SDK atau CLI

- Gunakan DescribeTable dengan AWS SDK atau CLI
- Gunakan DescribeTimeToLive dengan AWS SDK atau CLI
- Gunakan ExecuteStatement dengan AWS SDK
- Gunakan GetItem dengan AWS SDK atau CLI
- Gunakan ListTables dengan AWS SDK atau CLI
- Gunakan PutItem dengan AWS SDK atau CLI
- Gunakan Query dengan AWS SDK atau CLI
- Gunakan Scan dengan AWS SDK atau CLI
- Gunakan UpdateItem dengan AWS SDK atau CLI
- Gunakan UpdateTable dengan AWS SDK atau CLI
- Gunakan UpdateTimeToLive dengan AWS SDK atau CLI

Halo DynamoDB

Contoh kode berikut ini menunjukkan cara untuk mulai menggunakan DynamoDB.

.NET

SDK untuk .NET

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
using Amazon.DynamoDBv2;
using Amazon.DynamoDBv2.Model;

namespace DynamoDB_Actions;

public static class HelloDynamoDB
{
    static async Task Main(string[] args)
    {
        var dynamoDbClient = new AmazonDynamoDBClient();
```

```
        Console.WriteLine($"Hello Amazon Dynamo DB! Following are some of your
 tables:");
        Console.WriteLine();

        // You can use await and any of the async methods to get a response.
        // Let's get the first five tables.
        var response = await dynamoDbClient.ListTablesAsync(
            new ListTablesRequest()
            {
                Limit = 5
            });

        foreach (var table in response.TableNames)
        {
            Console.WriteLine($"\tTable: {table}");
            Console.WriteLine();
        }
    }
}
```

- Untuk detail API, lihat ListTablesdi Referensi AWS SDK untuk .NET API.

C++

SDK untuk C++

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

Kode untuk CMake file 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})
```

Kode untuk file sumber 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;
}
```

- Untuk detail API, lihat ListTablesdi Referensi AWS SDK untuk C++ API.

Java

SDK untuk Java 2.x

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
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!");
    }
}
```

- Untuk detail API, lihat ListTablesdi Referensi AWS SDK for Java 2.x API.

JavaScript

SDK untuk JavaScript (v3)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

Untuk detail lebih lanjut tentang bekerja dengan DynamoDB di, lihat Pemrograman DynamoDB AWS SDK untuk JavaScript dengan. 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;
};
```

- Untuk detail API, lihat ListTablesdi Referensi AWS SDK untuk JavaScript API.

Python

SDK untuk Python (Boto3)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
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.")
```

- Untuk detail API, lihat ListTablesdi AWS SDK for Python (Boto3) Referensi API.

Ruby

SDK untuk Ruby

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
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
```

- Untuk detail API, lihat ListTablesdi Referensi AWS SDK untuk Ruby API.

Untuk daftar lengkap panduan pengembang AWS SDK dan contoh kode, lihatMenggunakan DynamoDB dengan SDK AWS. Topik ini juga mencakup informasi tentang memulai dan detail tentang versi SDK sebelumnya.

Pelajari dasar-dasar DynamoDB dengan SDK AWS

Contoh-contoh kode berikut menunjukkan cara:

- Buat tabel yang dapat menyimpan data film.
- Masukkan, dapatkan, dan perbarui satu film dalam tabel tersebut.
- Tulis data film ke tabel dari file JSON sampel.
- Kueri untuk film yang dirilis pada tahun tertentu.
- Pindai film yang dirilis dalam suatu rentang tahun.
- Hapus film dari tabel, lalu hapus tabel tersebut.

.NET

SDK untuk .NET

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
// This example application performs the following basic Amazon DynamoDB
```

```
// functions:
//
//     CreateTableAsync
//     PutItemAsync
//     UpdateItemAsync
//     BatchWriteItemAsync
//     GetItemAsync
//     DeleteItemAsync
//     Query
//     Scan
//     DeleteItemAsync
//
using Amazon.DynamoDBv2;
using DynamoDB_Actions;

public class DynamoDB_Basics
{
    // Separator for the console display.
    private static readonly string SepBar = new string('-', 80);

    public static async Task Main()
    {
        var client = new AmazonDynamoDBClient();

        var tableName = "movie_table";

        // Relative path to moviedata.json in the local repository.
        var movieFileName = @"..\..\..\..\..\..\..\resources\sample_files
\movies.json";

        DisplayInstructions();

        // Create a new table and wait for it to be active.
        Console.WriteLine($"Creating the new table: {tableName}");

        var success = await DynamoDbMethods.CreateMovieTableAsync(client,
 tableName);

        if (success)
        {
            Console.WriteLine($"\nTable: {tableName} successfully created.");
        }
        else
        {
```

```
            Console.WriteLine($"\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 DynamoDbMethods.PutItemAsync(client, 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 DynamoDbMethods.UpdateItemAsync(client, 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 DynamoDbMethods.BatchWriteItemsAsync(client,
movieFileName);
        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 DynamoDbMethods.GetItemAsync(client, lookupMovie,
tableName);
        if (item.Count > 0)
        {
            DynamoDbMethods.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 DynamoDbMethods.DeleteItemAsync(client, 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($"Movies released in {findYear}");
        var queryCount = await DynamoDbMethods.QueryMoviesAsync(client,
tableName, findYear);
        Console.WriteLine($"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 DynamoDbMethods.ScanTableAsync(client, tableName,
startYear, endYear);
        Console.WriteLine($"Found {scanCount} movies released between {startYear}
and {endYear}");

        WaitForEnter();

        // Delete the table.
        success = await DynamoDbMethods.DeleteTableAsync(client, tableName);

        if (success)
        {
            Console.WriteLine($"Successfully deleted {tableName}");
        }
        else
        {
            Console.WriteLine($"Could not delete {tableName}");
        }

        Console.WriteLine("The DynamoDB Basics example application is done.");
```

```
        WaitForEnter();
    }

    /// <summary>
    /// Displays the description of the application on the console.
    /// </summary>
    private static void DisplayInstructions()
    {
        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()
    {
        Console.WriteLine("\nPress <Enter> to continue.");
        Console.WriteLine(SepBar);
        _ = Console.ReadLine();
    }
```

```
}
```

Membuat tabel yang akan berisi data film.

```
        /// <summary>
        /// Creates a new Amazon DynamoDB table and then waits for the new
        /// table to become active.
        /// </summary>
        /// <param name="client">An initialized Amazon DynamoDB client object.</
param>
        /// <param name="tableName">The name of the table to create.</param>
        /// <returns>A Boolean value indicating the success of the operation.</
returns>
        public static async Task<bool> CreateMovieTableAsync(AmazonDynamoDBClient
 client, string tableName)
        {
            var response = await client.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
            {
                System.Threading.Thread.Sleep(sleepDuration);

                var describeTableResponse = await
client.DescribeTableAsync(request);
                status = describeTableResponse.Table.TableStatus;

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

            return status == TableStatus.ACTIVE;
        }
```

Menambahkan satu film ke tabel.

```
        /// <summary>
        /// Adds a new item to the table.
        /// </summary>
```

```
        /// <param name="client">An initialized Amazon DynamoDB client object.</
param>
        /// <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 static async Task<bool> PutItemAsync(AmazonDynamoDBClient client,
 Movie newMovie, string tableName)
        {
            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,
            };

            var response = await client.PutItemAsync(request);
            return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
        }
```

Memperbarui satu item dalam tabel.

```
        /// <summary>
        /// Updates an existing item in the movies table.
        /// </summary>
        /// <param name="client">An initialized Amazon DynamoDB client object.</
param>
        /// <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 static async Task<bool> UpdateItemAsync(
            AmazonDynamoDBClient client,
            Movie newMovie,
            MovieInfo newInfo,
            string tableName)
        {
            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,
            };

            var response = await client.UpdateItemAsync(request);

            return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
        }
```

Mengambil satu item dari tabel film.

```
        /// <summary>
        /// Gets information about an existing movie from the table.
        /// </summary>
        /// <param name="client">An initialized Amazon DynamoDB client object.</
param>
        /// <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 static async Task<Dictionary<string, AttributeValue>>
 GetItemAsync(AmazonDynamoDBClient client, Movie newMovie, string tableName)
        {
            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 client.GetItemAsync(request);
            return response.Item;
        }
```

Menulis batch item ke tabel film.

```
        /// <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 = JsonSerializer.Deserialize<List<Movie>>(
                json,
                new JsonSerializerOptions
                {
                    PropertyNameCaseInsensitive = true
                });

            // Now return the first 250 entries.
            return allMovies.GetRange(0, 250);
        }

        /// <summary>
        /// Writes 250 items to the movie table.
        /// </summary>
        /// <param name="client">The initialized DynamoDB client object.</param>
        /// <param name="movieFileName">A string containing the full path to
        /// the JSON file containing movie data.</param>
        /// <returns>A long integer value representing the number of movies
        /// imported from the JSON file.</returns>
        public static async Task<long> BatchWriteItemsAsync(
            AmazonDynamoDBClient client,
            string movieFileName)
        {
            var movies = ImportMovies(movieFileName);
            if (movies is null)
            {
                Console.WriteLine("Couldn't find the JSON file with movie
data.");
                return 0;
            }

            var context = new DynamoDBContext(client);

            var movieBatch = context.CreateBatchWrite<Movie>();
            movieBatch.AddPutItems(movies);
```

```
            Console.WriteLine("Adding imported movies to the table.");
            await movieBatch.ExecuteAsync();

            return movies.Count;
        }
```

Menghapus satu item dari tabel.

```
        /// <summary>
        /// Deletes a single item from a DynamoDB table.
        /// </summary>
        /// <param name="client">The initialized DynamoDB client object.</param>
        /// <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 static async Task<bool> DeleteItemAsync(
            AmazonDynamoDBClient client,
            string tableName,
            Movie movieToDelete)
        {
            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,
            };

            var response = await client.DeleteItemAsync(request);
            return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
        }
```

Melakukan kueri tabel untuk film yang dirilis pada tahun tertentu.

```
        /// <summary>
        /// Queries the table for movies released in a particular year and
        /// then displays the information for the movies returned.
        /// </summary>
        /// <param name="client">The initialized DynamoDB client object.</param>
        /// <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 static async Task<int> QueryMoviesAsync(AmazonDynamoDBClient
client, string tableName, int year)
        {
            var movieTable = Table.LoadTable(client, tableName);
            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;

            Search 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;
        }
```

Memindai tabel untuk film yang dirilis dalam suatu rentang tahun.

```
        public static async Task<int> ScanTableAsync(
            AmazonDynamoDBClient client,
            string tableName,
            int startYear,
            int endYear)
        {
            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 client.ScanAsync(request);
                foundCount += response.Items.Count;
                response.Items.ForEach(i => DisplayItem(i));
                request.ExclusiveStartKey = response.LastEvaluatedKey;
            }
            while (response.LastEvaluatedKey.Count > 0);
            return foundCount;
        }
```

Menghapus tabel film.

```
        public static async Task<bool> DeleteTableAsync(AmazonDynamoDBClient
client, string tableName)
        {
            var request = new DeleteTableRequest
            {
                TableName = tableName,
            };

            var response = await client.DeleteTableAsync(request);
            if (response.HttpStatusCode == System.Net.HttpStatusCode.OK)
            {
                Console.WriteLine($"Table {response.TableDescription.TableName}
successfully deleted.");
                return true;
            }
            else
            {
                Console.WriteLine("Could not delete table.");
                return false;
            }
        }
```

- Untuk detail API, lihat topik berikut di Referensi API AWS SDK untuk .NET .

- BatchWriteItem
- CreateTable
- DeleteItem
- DeleteTable
- DescribeTable
- GetItem
- PutItem
- Kueri
- Scan
- UpdateItem

Bash

AWS CLI dengan skrip Bash

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

Skenario mulai DynamoDB.

```
###############################################################################
# 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
}
```

Fungsi DynamoDB yang digunakan dalam skenario ini.

```
##############################################################################
# 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
}
```

Fungsi utilitas yang digunakan dalam skenario ini.

```
###############################################################################
# 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
}
```

- Untuk detail API, lihat topik berikut di Referensi Perintah AWS CLI .
  - BatchWriteItem
  - CreateTable
  - DeleteItem
  - DeleteTable
  - DescribeTable
  - GetItem
  - PutItem
  - Kueri
  - Scan
  - UpdateItem

C++

SDK untuk C++

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkap dan pelajari cara menyiapkan dan menjalankan di Repositori Contoh Kode AWS.

```
    {
        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;
}
```

- Untuk detail API, lihat topik berikut di Referensi API AWS SDK untuk C++ .
  - BatchWriteItem
  - CreateTable
  - DeleteItem
  - DeleteTable
  - DescribeTable
  - GetItem
  - PutItem
  - Kueri
  - Scan
  - UpdateItem

Go

SDK untuk Go V2

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

Jalankan skenario interaktif untuk membuat tabel dan melakukan tindakan pada tabel tersebut.

```
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))
}
```

Tentukan struct Movie yang digunakan dalam contoh ini.

```
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"])
}
```

Buat struct dan metode yang memanggil tindakan DynamoDB.

```
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
}
```

- Untuk detail API, lihat topik berikut di Referensi API AWS SDK untuk Go .

- BatchWriteItem
- CreateTable
- DeleteItem
- DeleteTable
- DescribeTable
- GetItem
- PutItem
- Kueri
- Scan
- UpdateItem

Java

SDK untuk Java 2.x

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

Buat tabel DynamoDB.

```
// 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);
        }
    }
```

Buat fungsi pembantu untuk mengunduh dan mengekstrak file JSON sampel.

```
   // 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++;
       }
   }
```

Dapatkan item dari tabel.

```
   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);
        }
    }
```

Contoh lengkap.

```
/**
 * 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);
        }
    }
}
```

- Untuk detail API, lihat topik berikut di Referensi API AWS SDK for Java 2.x .
  - BatchWriteItem
  - CreateTable
  - DeleteItem
  - DeleteTable

- DescribeTable
- GetItem
- PutItem
- Kueri
- Scan
- UpdateItem

JavaScript

SDK untuk JavaScript (v3)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkap dan pelajari cara menyiapkan dan menjalankan di Repositori Contoh Kode AWS.

```
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 client.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.");
};
```

- Untuk detail API, lihat topik berikut di Referensi API AWS SDK untuk JavaScript .
  - BatchWriteItem
  - CreateTable
  - DeleteItem
  - DeleteTable
  - DescribeTable
  - GetItem

- PutItem
- Kueri
- Scan
- UpdateItem

Kotlin

SDK untuk Kotlin

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

Buat tabel DynamoDB.

```
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 { 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}")
    }
}
```

Buat fungsi pembantu untuk mengunduh dan mengekstrak file JSON sampel.

```
// 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 { region = "us-east-1" }.use { ddb ->
        ddb.putItem(request)
        println("Added $title to the Movie table.")
    }
}
```

Dapatkan item dari tabel.

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

Contoh lengkap.

```
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 { 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 { 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 { 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 { 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 { 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 { 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]}")
            }
        }
    }
}
```

- Untuk detail API, lihat topik berikut di Referensi API AWS SDK untuk Kotlin.

- BatchWriteItem
- CreateTable
- DeleteItem
- DeleteTable
- DescribeTable
- GetItem
- PutItem
- Kueri
- Scan
- UpdateItem

PHP

SDK untuk PHP

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkap dan pelajari cara menyiapkan dan menjalankan di Repositori Contoh Kode AWS.

```
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);
    }
}
```

- Untuk detail API, lihat topik berikut di Referensi API AWS SDK untuk PHP .
  - BatchWriteItem
  - CreateTable
  - DeleteItem
  - DeleteTable
  - DescribeTable
  - GetItem
  - PutItem
  - Kueri
  - Scan
  - UpdateItem

Python

SDK untuk Python (Boto3)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

Buat kelas yang merangkum tabel DynamoDB.

```
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
```

Buat fungsi pembantu untuk mengunduh dan mengekstrak file JSON sampel.

```
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]
```

Jalankan skenario interaktif untuk membuat tabel dan melakukan tindakan pada tabel tersebut.

```
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}")
```

Skenario ini menggunakan kelas pembantu berikut untuk mengajukan pertanyaan pada prompt perintah.

```
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
```

- Untuk detail API, lihat topik berikut di Referensi API AWS SDK untuk Python (Boto3).
  - BatchWriteItem
  - CreateTable
  - DeleteItem
  - DeleteTable
  - DescribeTable
  - GetItem

- PutItem
- Kueri
- Scan
- UpdateItem

Ruby

SDK untuk Ruby

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

Buat kelas yang merangkum tabel DynamoDB.

```
# 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
```

Buat fungsi pembantu untuk mengunduh dan mengekstrak file JSON sampel.

```
  # 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
```

Jalankan skenario interaktif untuk membuat tabel dan melakukan tindakan pada tabel tersebut.

```
  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
```

- Untuk detail API, lihat topik berikut di Referensi API AWS SDK untuk Ruby .
  - BatchWriteItem
  - CreateTable
  - DeleteItem
  - DeleteTable
  - DescribeTable
  - GetItem
  - PutItem
  - Kueri
  - Scan
  - UpdateItem

SAP ABAP

SDK untuk SAP ABAP

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
    " 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 ZCL_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.
```

- Untuk mengetahui hal detail mengenai API, silakan lihat topik-topik berikut di referensi API AWS SDK untuk ABAP SAP.
  - BatchWriteItem
  - CreateTable
  - DeleteItem
  - DeleteTable
  - DescribeTable
  - GetItem
  - PutItem
  - Kueri
  - Scan
  - UpdateItem

Swift

SDK untuk Swift

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

Kelas Swift yang menangani panggilan DynamoDB ke SDK untuk Swift.

```
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
        }
    }
}
```

Struktur yang digunakan oleh MovieTable kelas untuk mewakili film.

```
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
    }
}
```

Sebuah program yang menggunakan MovieTable kelas untuk mengakses database DynamoDB.

```
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)
        }
    }
}
```

- Untuk detail API, lihat topik berikut di Referensi API AWS SDK untuk Swift.
  - BatchWriteItem
  - CreateTable

- DeleteItem
- DeleteTable
- DescribeTable
- GetItem
- PutItem
- Kueri
- Scan
- UpdateItem

Untuk daftar lengkap panduan pengembang AWS SDK dan contoh kode, lihatMenggunakan DynamoDB dengan SDK AWS. Topik ini juga mencakup informasi tentang memulai dan detail tentang versi SDK sebelumnya.

Tindakan untuk DynamoDB menggunakan AWS SDKs

Contoh kode berikut menunjukkan bagaimana melakukan tindakan DynamoDB individu dengan. AWS SDKs Setiap contoh menyertakan tautan ke GitHub, di mana Anda dapat menemukan instruksi untuk mengatur dan menjalankan kode.

Kutipan ini memanggil API DynamoDB dan merupakan kutipan kode dari program yang lebih besar yang harus dijalankan dalam konteks. Anda dapat melihat tindakan dalam konteks diSkenario untuk DynamoDB menggunakan AWS SDKs .

Contoh berikut hanya mencakup tindakan yang paling umum digunakan. Untuk daftar lengkapnya, lihat Referensi API Amazon DynamoDB.

Contoh

- Gunakan BatchExecuteStatement dengan AWS SDK
- Gunakan BatchGetItem dengan AWS SDK atau CLI
- Gunakan BatchWriteItem dengan AWS SDK atau CLI
- Gunakan CreateTable dengan AWS SDK atau CLI
- Gunakan DeleteItem dengan AWS SDK atau CLI
- Gunakan DeleteTable dengan AWS SDK atau CLI
- Gunakan DescribeTable dengan AWS SDK atau CLI
- Gunakan DescribeTimeToLive dengan AWS SDK atau CLI

- Gunakan ExecuteStatement dengan AWS SDK
- Gunakan GetItem dengan AWS SDK atau CLI
- Gunakan ListTables dengan AWS SDK atau CLI
- Gunakan PutItem dengan AWS SDK atau CLI
- Gunakan Query dengan AWS SDK atau CLI
- Gunakan Scan dengan AWS SDK atau CLI
- Gunakan UpdateItem dengan AWS SDK atau CLI
- Gunakan UpdateTable dengan AWS SDK atau CLI
- Gunakan UpdateTimeToLive dengan AWS SDK atau CLI

Gunakan **`BatchExecuteStatement`** dengan AWS SDK

Contoh kode berikut menunjukkan cara menggunakan`BatchExecuteStatement`.

Contoh tindakan adalah kutipan kode dari program yang lebih besar dan harus dijalankan dalam konteks. Anda dapat melihat tindakan ini dalam konteks dalam contoh kode berikut:

- Hapus data menggunakan PartiQL DELETE
- Masukkan data menggunakan PartiQL INSERT
- Melakukan kueri pada tabel menggunakan batch pernyataan PartiQL
- Kueri data menggunakan PartiQL SELECT
- Perbarui data menggunakan PartiQL UPDATE

.NET

SDK untuk .NET

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

Gunakan batch pernyataan INSERT untuk menambahkan item.

```
        /// <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!;
            }
        }
```

Gunakan batch pernyataan SELECT untuk mendapatkan item.

```
        /// <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;
            }

        }
```

Gunakan batch pernyataan UPDATE untuk memperbarui item.

```
        /// <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;
        }
```

Gunakan batch DELETE untuk menghapus item.

```
        /// <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;
        }
```

- Untuk detail API, lihat BatchExecuteStatementdi Referensi AWS SDK untuk .NET API.

C++

SDK untuk C++

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

Gunakan batch pernyataan INSERT untuk menambahkan item.

```
    // 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;
        }
    }
```

Gunakan batch pernyataan SELECT untuk mendapatkan item.

```
    // 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;
        }
    }
```

Gunakan batch pernyataan UPDATE untuk memperbarui item.

```
    // 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;
        }
    }
```

Gunakan batch DELETE untuk menghapus item.

```
    // 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;
        }
    }
```

- Untuk detail API, lihat BatchExecuteStatementdi Referensi AWS SDK untuk C++ API.

Go

SDK untuk Go V2

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

Tentukan struct penerima fungsi untuk contoh.

```
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
}
```

Gunakan batch pernyataan INSERT untuk menambahkan item.

```
// 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
}
```

Gunakan batch pernyataan SELECT untuk mendapatkan item.

```
// 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
}
```

Gunakan batch pernyataan UPDATE untuk memperbarui item.

```
// 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
}
```

Gunakan batch DELETE untuk menghapus item.

```
// 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
}
```

Tentukan struct Movie yang digunakan dalam contoh ini.

```
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"])
}
```

- Untuk detail API, lihat BatchExecuteStatementdi Referensi AWS SDK untuk Go API.

JavaScript

SDK untuk JavaScript (v3)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

Buat batch item menggunakan PartiQL.

```
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;
```

```
};
```

Dapatkan batch item menggunakan PartiQL.

```
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;
};
```

Perbarui batch item menggunakan PartiQL.

```
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;
};
```

Hapus batch item menggunakan PartiQL.

```
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;
};
```

- Untuk detail API, lihat BatchExecuteStatementdi Referensi AWS SDK untuk JavaScript API.

PHP

SDK untuk PHP

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
    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,
                ],
            ],
        ]);
    }
```

- Untuk detail API, lihat BatchExecuteStatementdi Referensi AWS SDK untuk PHP API.

Python

SDK untuk Python (Boto3)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
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
```

- Untuk detail API, lihat BatchExecuteStatementdi AWS SDK for Python (Boto3) Referensi API.

Ruby

SDK untuk Ruby

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

Baca batch item menggunakan 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
```

Hapus batch item menggunakan 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

  # 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
```

- Untuk detail API, lihat BatchExecuteStatementdi Referensi AWS SDK untuk Ruby API.

Untuk daftar lengkap panduan pengembang AWS SDK dan contoh kode, lihatMenggunakan DynamoDB dengan SDK AWS. Topik ini juga mencakup informasi tentang memulai dan detail tentang versi SDK sebelumnya.

Gunakan **`BatchGetItem`** dengan AWS SDK atau CLI

Contoh kode berikut menunjukkan cara menggunakan`BatchGetItem`.

.NET

SDK untuk .NET

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
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);
        }
    }
}
```

- Untuk detail API, lihat BatchGetItemdi Referensi AWS SDK untuk .NET API.

Bash

AWS CLI dengan skrip Bash

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
###############################################################################
# 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
}
```

Fungsi utilitas yang digunakan dalam contoh ini.

```
###############################################################################
# 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
}
```

- Untuk detail API, lihat BatchGetItemdi Referensi AWS CLI Perintah.

C++

SDK untuk C++

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
//! 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;
}
```

- Untuk detail API, lihat BatchGetItemdi Referensi AWS SDK untuk C++ API.

CLI

AWS CLI

Untuk mengambil beberapa item dari tabel

`batch-get-items`Contoh berikut membaca beberapa item dari `MusicCollection` tabel menggunakan batch tiga `GetItem` permintaan, dan meminta jumlah unit kapasitas baca yang dikonsumsi oleh operasi. Perintah hanya mengembalikan `AlbumTitle` atribut.

```
aws dynamodb batch-get-item \
    --request-items file://request-items.json \
    --return-consumed-capacity TOTAL
```

Isi dari `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"
```

```
    }
}
```

Output:

```
{
    "Responses": {
        "MusicCollection": [
            {
                "AlbumTitle": {
                    "S": "Somewhat Famous"
                }
            },
            {
                "AlbumTitle": {
                    "S": "Blue Sky Blues"
                }
            },
            {
                "AlbumTitle": {
                    "S": "Louder Than Ever"
                }
            }
        ]
    },
    "UnprocessedKeys": {},
    "ConsumedCapacity": [
        {
            "TableName": "MusicCollection",
            "CapacityUnits": 1.5
        }
    ]
}
```

Untuk informasi selengkapnya, lihat Operasi Batch di Panduan Pengembang Amazon DynamoDB.

- Untuk detail API, lihat BatchGetItemdi Referensi AWS CLI Perintah.

Java

SDK untuk Java 2.x

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

Menunjukkan cara mendapatkan item batch menggunakan klien layanan.

```
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.");
```

```
        }
    }
}
```

Menunjukkan cara mendapatkan item batch menggunakan klien layanan dan paginator.

```
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());
            });
    }
}
```

- Untuk detail API, lihat BatchGetItemdi Referensi AWS SDK for Java 2.x API.

JavaScript

SDK untuk JavaScript (v3)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

Contoh ini menggunakan klien dokumen untuk menyederhanakan penggunaan item di DynamoDB. Untuk detail API, lihat 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;
};
```

- Untuk informasi selengkapnya, silakan lihat Panduan Developer AWS SDK untuk JavaScript.
- Untuk detail API, lihat BatchGetItemdi Referensi AWS SDK untuk JavaScript API.

SDK untuk JavaScript (v2)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
// 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);
    });
  }
});
```

- Untuk informasi selengkapnya, silakan lihat Panduan Developer AWS SDK untuk JavaScript.

- Untuk detail API, lihat BatchGetItemdi Referensi AWS SDK untuk JavaScript API.

PowerShell

Alat untuk PowerShell V4

Contoh 1: Mendapatkan item dengan SongTitle “Somewhere Down The Road” dari tabel DynamoDB 'Music' dan 'Songs'.

```
$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
```

Output:

```
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
```

- Untuk detail API, lihat BatchGetItemdi Referensi Alat AWS untuk PowerShell Cmdlet (V4).

Python

SDK untuk Python (Boto3)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
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
```

- Untuk detail API, lihat BatchGetItemdi AWS SDK for Python (Boto3) Referensi API.

Swift

SDK untuk Swift

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
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
        }
    }
```

- Untuk detail API, lihat referensi BatchGetItem AWSSDK untuk Swift API.

Untuk daftar lengkap panduan pengembang AWS SDK dan contoh kode, lihatMenggunakan DynamoDB dengan SDK AWS. Topik ini juga mencakup informasi tentang memulai dan detail tentang versi SDK sebelumnya.

Gunakan **`BatchWriteItem`** dengan AWS SDK atau CLI

Contoh kode berikut menunjukkan cara menggunakan`BatchWriteItem`.

Contoh tindakan adalah kutipan kode dari program yang lebih besar dan harus dijalankan dalam konteks. Anda dapat melihat tindakan ini dalam konteks dalam contoh kode berikut:

- Pelajari dasar-dasarnya

.NET

SDK untuk .NET

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

Menulis batch item ke tabel film.

```
        /// <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 = JsonSerializer.Deserialize<List<Movie>>(
                json,
                new JsonSerializerOptions
                {
                    PropertyNameCaseInsensitive = true
                });

            // Now return the first 250 entries.
            return allMovies.GetRange(0, 250);
        }

        /// <summary>
        /// Writes 250 items to the movie table.
        /// </summary>
        /// <param name="client">The initialized DynamoDB client object.</param>
        /// <param name="movieFileName">A string containing the full path to
        /// the JSON file containing movie data.</param>
        /// <returns>A long integer value representing the number of movies
        /// imported from the JSON file.</returns>
        public static async Task<long> BatchWriteItemsAsync(
            AmazonDynamoDBClient client,
            string movieFileName)
        {
            var movies = ImportMovies(movieFileName);
            if (movies is null)
            {
```

```
                Console.WriteLine("Couldn't find the JSON file with movie
 data.");
                return 0;
            }

            var context = new DynamoDBContext(client);

            var movieBatch = context.CreateBatchWrite<Movie>();
            movieBatch.AddPutItems(movies);

            Console.WriteLine("Adding imported movies to the table.");
            await movieBatch.ExecuteAsync();

            return movies.Count;
        }
```

- Untuk detail API, lihat BatchWriteItemdi Referensi AWS SDK untuk .NET API.

Bash

AWS CLI dengan skrip Bash

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
###############################################################################
# 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
}
```

Fungsi utilitas yang digunakan dalam contoh ini.

```
###############################################################################
# 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
}
```

- Untuk detail API, lihat BatchWriteItemdi Referensi AWS CLI Perintah.

C++

SDK untuk C++

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
//! 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;
}
```

- Untuk detail API, lihat BatchWriteItemdi Referensi AWS SDK untuk C++ API.

CLI

AWS CLI

Untuk menambahkan beberapa item ke tabel

`batch-write-item`Contoh berikut menambahkan tiga item baru ke `MusicCollection` tabel menggunakan batch tiga `PutItem` permintaan. Ini juga meminta informasi tentang jumlah unit kapasitas tulis yang dikonsumsi oleh operasi dan koleksi item apa pun yang dimodifikasi oleh operasi.

```
aws dynamodb batch-write-item \
    --request-items file://request-items.json \
    --return-consumed-capacity INDEXES \
    --return-item-collection-metrics SIZE
```

Isi dari `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"}
                }
            }
        }
    ]
}
```

Output:

```
{
    "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
                }
            }
        }
    ]
}
```

Untuk informasi selengkapnya, lihat Operasi Batch di Panduan Pengembang Amazon DynamoDB.

- Untuk detail API, lihat BatchWriteItemdi Referensi AWS CLI Perintah.

Go

SDK untuk Go V2

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
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
}
```

Tentukan struct Movie yang digunakan dalam contoh ini.

```
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"])
}
```

- Untuk detail API, lihat BatchWriteItemdi Referensi AWS SDK untuk Go API.

Java

SDK untuk Java 2.x

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

Menyisipkan banyak item ke dalam tabel dengan menggunakan klien layanan.

```
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);
        }
    }
}
```

Menyisipkan banyak item ke dalam tabel menggunakan klien yang disempurnakan.

```
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);
                }
        }
}
```

- Untuk detail API, lihat BatchWriteItemdi Referensi AWS SDK for Java 2.x API.

JavaScript

SDK untuk JavaScript (v3)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

Contoh ini menggunakan klien dokumen untuk menyederhanakan penggunaan item di DynamoDB. Untuk detail API, lihat 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);
  }
};
```

- Untuk detail API, lihat BatchWriteItemdi Referensi AWS SDK untuk JavaScript API.

SDK untuk JavaScript (v2)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
// 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);
  }
});
```

- Untuk informasi selengkapnya, silakan lihat Panduan Developer AWS SDK untuk JavaScript.
- Untuk detail API, lihat BatchWriteItemdi Referensi AWS SDK untuk JavaScript API.

PHP

SDK untuk PHP

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
    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;
            }
        }
    }
```

- Untuk detail API, lihat BatchWriteItemdi Referensi AWS SDK untuk PHP API.

PowerShell

Alat untuk PowerShell V4

Contoh 1: Membuat item baru, atau mengganti item yang ada dengan item baru di tabel DynamoDB Musik dan Lagu.

```
$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
```

Output:

```
$requestItem = @{
    'Music' = [Amazon.DynamoDBv2.Model.WriteRequest]($writeRequest)
    'Songs' = [Amazon.DynamoDBv2.Model.WriteRequest]($writeRequest)
}

Set-DDBBatchItem -RequestItem $requestItem
```

- Untuk detail API, lihat BatchWriteItemdi Referensi Alat AWS untuk PowerShell Cmdlet (V4).

Python

SDK untuk Python (Boto3)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
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
```

- Untuk detail API, lihat BatchWriteItemdi AWS SDK for Python (Boto3) Referensi API.

Ruby

SDK untuk Ruby

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
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
```

- Untuk detail API, lihat BatchWriteItemdi Referensi AWS SDK untuk Ruby API.

Swift

SDK untuk Swift

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
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
        }
    }
```

- Untuk detail API, lihat referensi BatchWriteItem AWSSDK untuk Swift API.

Untuk daftar lengkap panduan pengembang AWS SDK dan contoh kode, lihatMenggunakan DynamoDB dengan SDK AWS. Topik ini juga mencakup informasi tentang memulai dan detail tentang versi SDK sebelumnya.

Gunakan **`CreateTable`** dengan AWS SDK atau CLI

Contoh kode berikut menunjukkan cara menggunakan`CreateTable`.

Contoh tindakan adalah kutipan kode dari program yang lebih besar dan harus dijalankan dalam konteks. Anda dapat melihat tindakan ini dalam konteks dalam contoh kode berikut:

- Pelajari dasar-dasarnya
- Mempercepat pembacaan dengan DAX
- Skenario Indeks Sekunder Global Tingkat Lanjut

- Buat tabel dengan indeks sekunder global
- Buat tabel dengan throughput hangat diaktifkan
- Kelola Indeks Sekunder Global
- Mengelola kebijakan berbasis sumber daya
- Mengatur Kontrol Akses Berbasis Atribut
- Bekerja dengan tabel global dan replikasi multi-wilayah
- Bekerja dengan Indeks Sekunder Lokal
- Bekerja dengan penandaan sumber daya
- Bekerja dengan Streams dan Time-to-Live
- Bekerja dengan enkripsi tabel

.NET

SDK untuk .NET

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
        /// <summary>
        /// Creates a new Amazon DynamoDB table and then waits for the new
        /// table to become active.
        /// </summary>
        /// <param name="client">An initialized Amazon DynamoDB client object.</
param>
        /// <param name="tableName">The name of the table to create.</param>
        /// <returns>A Boolean value indicating the success of the operation.</
returns>
        public static async Task<bool> CreateMovieTableAsync(AmazonDynamoDBClient
 client, string tableName)
        {
            var response = await client.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
            {
                System.Threading.Thread.Sleep(sleepDuration);
```

```
                var describeTableResponse = await
 client.DescribeTableAsync(request);
                status = describeTableResponse.Table.TableStatus;

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

            return status == TableStatus.ACTIVE;
        }
```

- Untuk detail API, lihat CreateTabledi Referensi AWS SDK untuk .NET API.

Bash

AWS CLI dengan skrip Bash

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
###############################################################################
# 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
}
```

Fungsi utilitas yang digunakan dalam contoh ini.

```
###############################################################################
# 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
}
```

- Untuk detail API, lihat CreateTabledi Referensi AWS CLI Perintah.

C++

SDK untuk C++

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
//! 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);
}
```

Kode yang menunggu tabel menjadi aktif.

```
//! 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;
}
```

- Untuk detail API, lihat CreateTabledi Referensi AWS SDK untuk C++ API.

CLI

AWS CLI

Contoh 1: Untuk membuat tabel dengan tag

`create-table`Contoh berikut menggunakan atribut tertentu dan skema kunci untuk membuat tabel bernama`MusicCollection`. Tabel ini menggunakan throughput yang disediakan dan dienkripsi saat istirahat menggunakan CMK yang dimiliki default. AWS Perintah ini juga menerapkan tag ke tabel, dengan kunci dari `Owner` dan nilai`blueTeam`.

```
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
```

Output:

```
{
    "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"
    }
}
```

Untuk informasi selengkapnya, lihat Operasi Dasar untuk Tabel dalam Panduan Developer Amazon DynamoDB.

Contoh 2: Untuk membuat tabel dalam Mode On-Demand

Contoh berikut membuat tabel yang disebut `MusicCollection` menggunakan mode on-demand, bukan mode throughput yang disediakan. Ini berguna untuk tabel dengan beban kerja yang tidak terduga.

```
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
```

Output:

```
{
    "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"
        }
    }
}
```

Untuk informasi selengkapnya, lihat Operasi Dasar untuk Tabel dalam Panduan Developer Amazon DynamoDB.

Contoh 3: Untuk membuat tabel dan mengenkripsi dengan Customer Managed CMK

Contoh berikut membuat tabel bernama `MusicCollection` dan mengenkripsi menggunakan CMK yang dikelola pelanggan.

```
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
```

Output:

```
{
    "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"
        }
    }
}
```

Untuk informasi selengkapnya, lihat Operasi Dasar untuk Tabel dalam Panduan Developer Amazon DynamoDB.

Contoh 4: Untuk membuat tabel dengan Indeks Sekunder Lokal

Contoh berikut menggunakan atribut tertentu dan skema kunci untuk membuat tabel bernama `MusicCollection` dengan Indeks Sekunder Lokal bernama`AlbumTitleIndex`.

```
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\"]
                }
            }
```

```
        ]"
```

Output:

```
{
    "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"
            }
        ]
    }
}
```

Untuk informasi selengkapnya, lihat Operasi Dasar untuk Tabel dalam Panduan Developer Amazon DynamoDB.

Contoh 5: Untuk membuat tabel dengan Indeks Sekunder Global

Contoh berikut membuat tabel bernama `GameScores` dengan Global Secondary Index disebut`GameTitleIndex`. Tabel dasar memiliki kunci partisi `UserId` dan kunci urutan `GameTitle`, sehingga Anda dapat menemukan skor terbaik pengguna individu untuk game tertentu secara efisien, sedangkan GSI memiliki kunci partisi `GameTitle` dan kunci urutan `TopScore`, memungkinkan Anda untuk cepat menemukan skor tertinggi secara keseluruhan untuk game tertentu.

```
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
                }
            }
        ]"
```

Output:

```
{
    "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"
            }
        ]
    }
}
```

Untuk informasi selengkapnya, lihat Operasi Dasar untuk Tabel dalam Panduan Developer Amazon DynamoDB.

Contoh 6: Untuk membuat tabel dengan beberapa Indeks Sekunder Global sekaligus

Contoh berikut membuat tabel bernama `GameScores` dengan dua Global Secondary Indexes. Skema GSI diteruskan melalui file, bukan pada baris perintah.

```
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
```

Isi dari `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
        }
    }
]
```

Output:

```
{
    "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"
            }
```

```
        ]
    }
}
```

Untuk informasi selengkapnya, lihat Operasi Dasar untuk Tabel dalam Panduan Developer Amazon DynamoDB.

Contoh 7: Untuk membuat tabel dengan Streams diaktifkan

Contoh berikut membuat tabel yang disebut `GameScores` dengan DynamoDB Streams diaktifkan. Baik gambar baru dan lama dari setiap item akan ditulis ke aliran.

```
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
```

Output:

```
{
    "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"
    }
}
```

Untuk informasi selengkapnya, lihat Operasi Dasar untuk Tabel dalam Panduan Developer Amazon DynamoDB.

Contoh 8: Untuk membuat tabel dengan Keys-Only Stream diaktifkan

Contoh berikut membuat tabel yang disebut `GameScores` dengan DynamoDB Streams diaktifkan. Hanya atribut kunci dari item yang dimodifikasi yang ditulis ke aliran.

```
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
```

Output:

```
{
    "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
```

```
    }
}
```

Untuk informasi selengkapnya, lihat Mengubah pengambilan data untuk DynamoDB Streams di Panduan Pengembang Amazon DynamoDB.

Contoh 9: Untuk membuat tabel dengan kelas Standard Infrequent Access

Contoh berikut membuat tabel yang disebut `GameScores` dan menetapkan kelas tabel Standard-Infrequent Access (DynamoDB Standard-IA). Kelas tabel ini dioptimalkan untuk penyimpanan menjadi biaya dominan.

```
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
```

Output:

```
{
    "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
    }
}
```

Untuk informasi selengkapnya, lihat Kelas tabel di Panduan Pengembang Amazon DynamoDB.

Contoh 10: Membuat tabel dengan Hapus Perlindungan diaktifkan

Contoh berikut membuat tabel yang disebut `GameScores` dan memungkinkan perlindungan penghapusan.

```
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
```

Output:

```
{
    "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
    }
}
```

Untuk informasi selengkapnya, lihat Menggunakan perlindungan penghapusan di Panduan Pengembang Amazon DynamoDB.

- Untuk detail API, lihat CreateTabledi Referensi AWS CLI Perintah.

Go

SDK untuk Go V2

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
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
}
```

- Untuk detail API, lihat CreateTabledi Referensi AWS SDK untuk Go API.

Java

SDK untuk Java 2.x

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
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 "";
    }
}
```

- Untuk detail API, lihat CreateTabledi Referensi AWS SDK for Java 2.x API.

JavaScript

SDK untuk JavaScript (v3)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
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;
};
```

- Untuk informasi selengkapnya, silakan lihat Panduan Developer AWS SDK untuk JavaScript.
- Untuk detail API, lihat CreateTabledi Referensi AWS SDK untuk JavaScript API.

SDK untuk JavaScript (v2)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
// 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);
  }
});
```

- Untuk informasi selengkapnya, silakan lihat Panduan Developer AWS SDK untuk JavaScript.
- Untuk detail API, lihat CreateTabledi Referensi AWS SDK untuk JavaScript API.

Kotlin

SDK untuk Kotlin

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
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 { 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
    }
}
```

- Untuk detail API, lihat CreateTabledi AWS SDK untuk referensi API Kotlin.

PHP

SDK untuk PHP

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

Buat tabel.

```
        $tableName = "ddb_demo_table_$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],
        ]);
    }
```

- Untuk detail API, lihat CreateTabledi Referensi AWS SDK untuk PHP API.

PowerShell

Alat untuk PowerShell V4

Contoh 1: Contoh ini membuat tabel bernama Thread yang memiliki kunci utama yang terdiri dari 'ForumName' (hash tipe kunci) dan 'Subject' (rentang tipe kunci). Skema yang digunakan untuk membangun tabel dapat disalurkan ke setiap cmdlet seperti yang ditunjukkan atau ditentukan menggunakan parameter -Schema.

```
$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
```

Output:

```
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  : {}
```

Contoh 2: Contoh ini membuat tabel bernama Thread yang memiliki kunci utama yang terdiri dari 'ForumName' (hash tipe kunci) dan 'Subject' (rentang tipe kunci). Indeks sekunder lokal juga didefinisikan. Kunci indeks sekunder lokal akan diatur secara otomatis dari kunci hash utama pada tabel (ForumName). Skema yang digunakan untuk membangun tabel dapat disalurkan ke setiap cmdlet seperti yang ditunjukkan atau ditentukan menggunakan parameter -Schema.

```
$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
```

Output:

```
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}
```

Contoh 3: Contoh ini menunjukkan cara menggunakan pipeline tunggal untuk membuat tabel bernama Thread yang memiliki kunci utama yang terdiri dari 'ForumName' (hash tipe kunci) dan 'Subjek' (rentang tipe kunci) dan indeks sekunder lokal. DDBKeyAdd-Schema dan Add-DDBIndex Schema membuat TableSchema objek baru untuk Anda jika tidak disediakan dari pipeline atau parameter -Schema.

```
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
```

Output:

```
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}
```

- Untuk detail API, lihat CreateTabledi Referensi Alat AWS untuk PowerShell Cmdlet (V4).

Python

SDK untuk Python (Boto3)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

Buat tabel untuk menyimpan data film.

```
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
```

- Untuk detail API, lihat CreateTabledi AWS SDK for Python (Boto3) Referensi API.

Ruby

SDK untuk Ruby

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
# 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
```

- Untuk detail API, lihat CreateTabledi Referensi AWS SDK untuk Ruby API.

Rust

SDK untuk Rust

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
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))
        }
    }
}
```

- Untuk detail API, lihat CreateTablereferensi AWS SDK for Rust API.

SAP ABAP

SDK untuk SAP ABAP

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
    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.
```

- Untuk detail API, lihat CreateTabledi AWS SDK untuk referensi SAP ABAP API.

Swift

SDK untuk Swift

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
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
        }
    }
```

- Untuk detail API, lihat referensi CreateTable AWSSDK untuk Swift API.

Untuk daftar lengkap panduan pengembang AWS SDK dan contoh kode, lihatMenggunakan DynamoDB dengan SDK AWS. Topik ini juga mencakup informasi tentang memulai dan detail tentang versi SDK sebelumnya.

Gunakan **`DeleteItem`** dengan AWS SDK atau CLI

Contoh kode berikut menunjukkan cara menggunakan`DeleteItem`.

Contoh tindakan adalah kutipan kode dari program yang lebih besar dan harus dijalankan dalam konteks. Anda dapat melihat tindakan ini dalam konteks dalam contoh kode berikut:

- Pelajari dasar-dasarnya
- Gunakan operasi bersyarat

.NET

SDK untuk .NET

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
        /// <summary>
        /// Deletes a single item from a DynamoDB table.
        /// </summary>
        /// <param name="client">The initialized DynamoDB client object.</param>
        /// <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 static async Task<bool> DeleteItemAsync(
            AmazonDynamoDBClient client,
            string tableName,
            Movie movieToDelete)
        {
            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,
            };

            var response = await client.DeleteItemAsync(request);
            return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
        }
```

- Untuk detail API, lihat DeleteItemdi Referensi AWS SDK untuk .NET API.

Bash

AWS CLI dengan skrip Bash

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

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

```
}
```

Fungsi utilitas yang digunakan dalam contoh ini.

```
###############################################################################
# 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
}
```

- Untuk detail API, lihat DeleteItemdi Referensi AWS CLI Perintah.

C++

SDK untuk C++

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
//! 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);
}
```

Kode yang menunggu tabel menjadi aktif.

```
//! 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;
}
```

- Untuk detail API, lihat DeleteItemdi Referensi AWS SDK untuk C++ API.

CLI

AWS CLI

Contoh 1: Untuk menghapus item

`delete-item`Contoh berikut menghapus item dari `MusicCollection` tabel dan meminta rincian tentang item yang telah dihapus dan kapasitas yang digunakan oleh permintaan.

```
aws dynamodb delete-item \
    --table-name MusicCollection \
    --key file://key.json \
    --return-values ALL_OLD \
    --return-consumed-capacity TOTAL \
    --return-item-collection-metrics SIZE
```

Isi dari `key.json`:

```
{
    "Artist": {"S": "No One You Know"},
    "SongTitle": {"S": "Scared of My Shadow"}
}
```

Output:

```
{
    "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
        ]
```

```
    }
}
```

Untuk informasi selengkapnya, lihat Menulis Item di Panduan Pengembang Amazon DynamoDB.

Contoh 2: Untuk menghapus item secara kondisional

Contoh berikut menghapus item dari `ProductCatalog` tabel hanya jika salah satu `Sporting Goods` atau `Gardening Supplies` dan harganya antara 500 dan 600. `ProductCategory` la mengembalikan rincian tentang item yang telah dihapus.

```
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
```

Isi dari `names.json`:

```
{
    "#P": "Price"
}
```

Isi dari `values.json`:

```
{
    ":cat1": {"S": "Sporting Goods"},
    ":cat2": {"S": "Gardening Supplies"},
    ":lo": {"N": "500"},
    ":hi": {"N": "600"}
}
```

Output:

```
{
    "Attributes": {
        "Id": {
            "N": "456"
```

```
        },
        "Price": {
            "N": "550"
        },
        "ProductCategory": {
            "S": "Sporting Goods"
        }
    }
}
```

Untuk informasi selengkapnya, lihat Menulis Item di Panduan Pengembang Amazon DynamoDB.

- Untuk detail API, lihat DeleteItemdi Referensi AWS CLI Perintah.

Go

SDK untuk Go V2

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
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
}
```

Tentukan struct Movie yang digunakan dalam contoh ini.

```
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"])
}
```

- Untuk detail API, lihat DeleteItemdi Referensi AWS SDK untuk Go API.

Java

SDK untuk Java 2.x

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
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);
        }
    }
}
```

- Untuk detail API, lihat DeleteItemdi Referensi AWS SDK for Java 2.x API.

JavaScript

SDK untuk JavaScript (v3)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

Contoh ini menggunakan klien dokumen untuk menyederhanakan penggunaan item di DynamoDB. Untuk detail API, lihat 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;
};
```

- Untuk informasi selengkapnya, silakan lihat Panduan Developer AWS SDK untuk JavaScript.
- Untuk detail API, lihat DeleteItemdi Referensi AWS SDK untuk JavaScript API.

SDK untuk JavaScript (v2)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

Hapus item dari tabel.

```
// 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);
  }
});
```

Hapus item dari tabel menggunakan klien dokumen DynamoDB.

```
// 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);
  }
```

```
});
```

- Untuk informasi selengkapnya, silakan lihat Panduan Developer AWS SDK untuk JavaScript.
- Untuk detail API, lihat DeleteItemdi Referensi AWS SDK untuk JavaScript API.

Kotlin

SDK untuk Kotlin

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
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 { region = "us-east-1" }.use { ddb ->
        ddb.deleteItem(request)
        println("Item with key matching $keyVal was deleted")
    }
}
```

- Untuk detail API, lihat DeleteItemdi AWS SDK untuk referensi API Kotlin.

PHP

SDK untuk PHP

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
        $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,
        ]);
    }
```

- Untuk detail API, lihat DeleteItemdi Referensi AWS SDK untuk PHP API.

PowerShell

Alat untuk PowerShell V4

Contoh 1: Menghapus item DynamoDB yang cocok dengan kunci yang disediakan.

```
$key = @{
    SongTitle = 'Somewhere Down The Road'
    Artist = 'No One You Know'
} | ConvertTo-DDBItem
Remove-DDBItem -TableName 'Music' -Key $key -Confirm:$false
```

- Untuk detail API, lihat DeleteItemdi Referensi Alat AWS untuk PowerShell Cmdlet (V4).

Python

SDK untuk Python (Boto3)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
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
```

Anda dapat menentukan kondisi sehingga item dihapus hanya ketika memenuhi kriteria tertentu.

```
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
```

- Untuk detail API, lihat DeleteItemdi AWS SDK for Python (Boto3) Referensi API.

Ruby

SDK untuk Ruby

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
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
```

- Untuk detail API, lihat DeleteItemdi Referensi AWS SDK untuk Ruby API.

Rust

SDK untuk Rust

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
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)),
    }
}
```

- Untuk detail API, lihat DeleteItemreferensi AWS SDK for Rust API.

SAP ABAP

SDK untuk SAP ABAP

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
    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.
```

- Untuk detail API, lihat DeleteItemdi AWS SDK untuk referensi SAP ABAP API.

Swift

SDK untuk Swift

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
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
        }
    }
```

- Untuk detail API, lihat referensi DeleteItem AWSSDK untuk Swift API.

Untuk daftar lengkap panduan pengembang AWS SDK dan contoh kode, lihatMenggunakan DynamoDB dengan SDK AWS. Topik ini juga mencakup informasi tentang memulai dan detail tentang versi SDK sebelumnya.

Gunakan **`DeleteTable`** dengan AWS SDK atau CLI

Contoh kode berikut menunjukkan cara menggunakan`DeleteTable`.

Contoh tindakan adalah kutipan kode dari program yang lebih besar dan harus dijalankan dalam konteks. Anda dapat melihat tindakan ini dalam konteks dalam contoh kode berikut:

- Pelajari dasar-dasarnya
- Mempercepat pembacaan dengan DAX
- Kelola Indeks Sekunder Global

.NET

SDK untuk .NET

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
        public static async Task<bool> DeleteTableAsync(AmazonDynamoDBClient
client, string tableName)
        {
            var request = new DeleteTableRequest
            {
                TableName = tableName,
            };

            var response = await client.DeleteTableAsync(request);
            if (response.HttpStatusCode == System.Net.HttpStatusCode.OK)
            {
                Console.WriteLine($"Table {response.TableDescription.TableName}
successfully deleted.");
```

```
                return true;
            }
            else
            {
                Console.WriteLine("Could not delete table.");
                return false;
            }
        }
```

- Untuk detail API, lihat DeleteTabledi Referensi AWS SDK untuk .NET API.

Bash

AWS CLI dengan skrip Bash

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

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

```
}
```

Fungsi utilitas yang digunakan dalam contoh ini.

```
###############################################################################
# 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
}
```

- Untuk detail API, lihat DeleteTabledi Referensi AWS CLI Perintah.

C++

SDK untuk C++

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
//! 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();
}
```

- Untuk detail API, lihat DeleteTabledi Referensi AWS SDK untuk C++ API.

CLI

AWS CLI

Untuk menghapus tabel

`delete-table`Contoh berikut menghapus `MusicCollection` tabel.

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

Output:

```
{
    "TableDescription": {
        "TableStatus": "DELETING",
```

```
        "TableSizeBytes": 0,
        "ItemCount": 0,
        "TableName": "MusicCollection",
        "ProvisionedThroughput": {
            "NumberOfDecreasesToday": 0,
            "WriteCapacityUnits": 5,
            "ReadCapacityUnits": 5
        }
    }
}
```

Untuk informasi selengkapnya, lihat Menghapus Tabel di Panduan Pengembang Amazon DynamoDB.

- Untuk detail API, lihat DeleteTabledi Referensi AWS CLI Perintah.

Go

SDK untuk Go V2

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
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
}
```

- Untuk detail API, lihat DeleteTabledi Referensi AWS SDK untuk Go API.

Java

SDK untuk Java 2.x

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
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!");
    }
}
```

- Untuk detail API, lihat DeleteTabledi Referensi AWS SDK for Java 2.x API.

JavaScript

SDK untuk JavaScript (v3)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
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;
};
```

- Untuk detail API, lihat DeleteTabledi Referensi AWS SDK untuk JavaScript API.

SDK untuk JavaScript (v2)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
// 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);
  }
});
```

- Untuk informasi selengkapnya, silakan lihat Panduan Developer AWS SDK untuk JavaScript.
- Untuk detail API, lihat DeleteTabledi Referensi AWS SDK untuk JavaScript API.

Kotlin

SDK untuk Kotlin

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
suspend fun deleteDynamoDBTable(tableNameVal: String) {
    val request =
        DeleteTableRequest {
            tableName = tableNameVal
        }

    DynamoDbClient { region = "us-east-1" }.use { ddb ->
        ddb.deleteTable(request)
        println("$tableNameVal was deleted")
    }
}
```

- Untuk detail API, lihat DeleteTabledi AWS SDK untuk referensi API Kotlin.

PHP

SDK untuk PHP

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
    public function deleteTable(string $TableName)
    {
        $this->customWaiter(function () use ($TableName) {
            return $this->dynamoDbClient->deleteTable([
                'TableName' => $TableName,
```

```
            ]);
        });
    }
```

- Untuk detail API, lihat DeleteTabledi Referensi AWS SDK untuk PHP API.

PowerShell

Alat untuk PowerShell V4

Contoh 1: Menghapus tabel yang ditentukan. Anda diminta untuk konfirmasi sebelum operasi berlangsung.

```
Remove-DDBTable -TableName "myTable"
```

Contoh 2: Menghapus tabel yang ditentukan. Anda tidak diminta untuk konfirmasi sebelum operasi berlangsung.

```
Remove-DDBTable -TableName "myTable" -Force
```

- Untuk detail API, lihat DeleteTabledi Referensi Alat AWS untuk PowerShell Cmdlet (V4).

Python

SDK untuk Python (Boto3)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
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
```

- Untuk detail API, lihat DeleteTabledi AWS SDK for Python (Boto3) Referensi API.

Ruby

SDK untuk Ruby

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
# 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
```

- Untuk detail API, lihat DeleteTabledi Referensi AWS SDK untuk Ruby API.

Rust

SDK untuk Rust

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
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())),
    }
}
```

- Untuk detail API, lihat DeleteTablereferensi AWS SDK for Rust API.

SAP ABAP

SDK untuk SAP ABAP

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
    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.
```

- Untuk detail API, lihat DeleteTabledi AWS SDK untuk referensi SAP ABAP API.

Swift

SDK untuk Swift

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
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
        }
```

```
}
```

- Untuk detail API, lihat referensi DeleteTable AWSSDK untuk Swift API.

Untuk daftar lengkap panduan pengembang AWS SDK dan contoh kode, lihatMenggunakan DynamoDB dengan SDK AWS. Topik ini juga mencakup informasi tentang memulai dan detail tentang versi SDK sebelumnya.

Gunakan `DescribeTable` dengan AWS SDK atau CLI

Contoh kode berikut menunjukkan cara menggunakan`DescribeTable`.

Contoh tindakan adalah kutipan kode dari program yang lebih besar dan harus dijalankan dalam konteks. Anda dapat melihat tindakan ini dalam konteks dalam contoh kode berikut:

- Pelajari dasar-dasarnya
- Bekerja dengan Streams dan Time-to-Live
- Bekerja dengan enkripsi tabel

.NET

SDK untuk .NET

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
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}");

    }
```

- Untuk detail API, lihat DescribeTabledi Referensi AWS SDK untuk .NET API.

Bash

AWS CLI dengan skrip Bash

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
###############################################################################
# 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
}
```

Fungsi utilitas yang digunakan dalam contoh ini.

```
###############################################################################
# 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
}
```

- Untuk detail API, lihat DescribeTabledi Referensi AWS CLI Perintah.

C++

SDK untuk C++

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
//! 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();
}
```

- Untuk detail API, lihat DescribeTabledi Referensi AWS SDK untuk C++ API.

CLI

AWS CLI

Untuk menggambarkan tabel

`describe-table`Contoh berikut menjelaskan `MusicCollection` tabel.

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

Output:

```
{
    "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
    }
}
```

Untuk informasi selengkapnya, lihat Menjelaskan Tabel di Panduan Pengembang Amazon DynamoDB.

- Untuk detail API, lihat DescribeTabledi Referensi AWS CLI Perintah.

Go

SDK untuk Go V2

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
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
}
```

- Untuk detail API, lihat DescribeTabledi Referensi AWS SDK untuk Go API.

Java

SDK untuk Java 2.x

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
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!");
    }
}
```

- Untuk detail API, lihat DescribeTabledi Referensi AWS SDK for Java 2.x API.

JavaScript

SDK untuk JavaScript (v3)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
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;
};
```

- Untuk informasi selengkapnya, silakan lihat Panduan Developer AWS SDK untuk JavaScript.
- Untuk detail API, lihat DescribeTabledi Referensi AWS SDK untuk JavaScript API.

SDK untuk JavaScript (v2)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
// 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);
  }
});
```

- Untuk informasi selengkapnya, silakan lihat Panduan Developer AWS SDK untuk JavaScript.
- Untuk detail API, lihat DescribeTabledi Referensi AWS SDK untuk JavaScript API.

PowerShell

Alat untuk PowerShell V4

Contoh 1: Mengembalikan rincian tabel yang ditentukan.

```
Get-DDBTable -TableName "myTable"
```

- Untuk detail API, lihat DescribeTabledi Referensi Alat AWS untuk PowerShell Cmdlet (V4).

Python

SDK untuk Python (Boto3)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
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
```

- Untuk detail API, lihat DescribeTabledi AWS SDK for Python (Boto3) Referensi API.

Ruby

SDK untuk Ruby

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
# 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
```

- Untuk detail API, lihat DescribeTabledi Referensi AWS SDK untuk Ruby API.

SAP ABAP

SDK untuk SAP ABAP

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
    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.
```

- Untuk detail API, lihat DescribeTabledi AWS SDK untuk referensi SAP ABAP API.

Untuk daftar lengkap panduan pengembang AWS SDK dan contoh kode, lihatMenggunakan DynamoDB dengan SDK AWS. Topik ini juga mencakup informasi tentang memulai dan detail tentang versi SDK sebelumnya.

Gunakan **`DescribeTimeToLive`** dengan AWS SDK atau CLI

Contoh kode berikut menunjukkan cara menggunakan`DescribeTimeToLive`.

Contoh tindakan adalah kutipan kode dari program yang lebih besar dan harus dijalankan dalam konteks. Anda dapat melihat tindakan ini dalam konteks dalam contoh kode berikut:

- Bekerja dengan Streams dan Time-to-Live

CLI

AWS CLI

Untuk melihat pengaturan Waktu ke Langsung untuk tabel

`describe-time-to-live`Contoh berikut menampilkan Pengaturan Time to Live untuk `MusicCollection` tabel.

```
aws dynamodb describe-time-to-live \
    --table-name MusicCollection
```

Output:

```
{
    "TimeToLiveDescription": {
        "TimeToLiveStatus": "ENABLED",
        "AttributeName": "ttl"
    }
}
```

Untuk informasi selengkapnya, lihat Waktu untuk Hidup di Panduan Pengembang Amazon DynamoDB.

- Untuk detail API, lihat DescribeTimeToLivedi Referensi AWS CLI Perintah.

Java

SDK untuk Java 2.x

Jelaskan konfigurasi TTL pada tabel DynamoDB yang ada menggunakan. 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;
        }
    }
```

- Untuk detail API, lihat DescribeTimeToLivedi Referensi AWS SDK for Java 2.x API.

JavaScript

SDK untuk JavaScript (v3)

Jelaskan konfigurasi TTL pada tabel DynamoDB yang ada menggunakan. AWS SDK untuk 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');
```

- Untuk detail API, lihat DescribeTimeToLivedi Referensi AWS SDK untuk JavaScript API.

Python

SDK untuk Python (Boto3)

Jelaskan konfigurasi TTL pada tabel DynamoDB yang ada menggunakan. AWS SDK untuk 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")
```

- Untuk detail API, lihat DescribeTimeToLivedi AWS SDK for Python (Boto3) Referensi API.

Untuk daftar lengkap panduan pengembang AWS SDK dan contoh kode, lihatMenggunakan DynamoDB dengan SDK AWS. Topik ini juga mencakup informasi tentang memulai dan detail tentang versi SDK sebelumnya.

Gunakan **`ExecuteStatement`** dengan AWS SDK

Contoh kode berikut menunjukkan cara menggunakan`ExecuteStatement`.

Contoh tindakan adalah kutipan kode dari program yang lebih besar dan harus dijalankan dalam konteks. Anda dapat melihat tindakan ini dalam konteks dalam contoh kode berikut:

- Hapus data menggunakan PartiQL DELETE
- Masukkan data menggunakan PartiQL INSERT
- Melakukan kueri tabel menggunakan PartiQL
- Kueri data menggunakan PartiQL SELECT
- Perbarui data menggunakan PartiQL UPDATE

.NET

SDK untuk .NET

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

Gunakan pernyataan INSERT untuk menambahkan item.

```
        /// <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;
        }
```

Gunakan pernyataan SELECT untuk mendapatkan item.

```
        /// <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;
        }
```

Gunakan pernyataan SELECT untuk mendapatkan daftar item.

```
        /// <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;
        }
```

Gunakan pernyataan UPDATE untuk memperbarui item.

```
        /// <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;
        }
```

Gunakan pernyataan DELETE untuk menghapus satu film.

```
        /// <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;
        }
```

- Untuk detail API, lihat ExecuteStatementdi Referensi AWS SDK untuk .NET API.

C++

SDK untuk C++

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

Gunakan pernyataan INSERT untuk menambahkan item.

```
    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;
        }
    }
```

Gunakan pernyataan SELECT untuk mendapatkan item.

```
    //  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;
            }
        }
    }
```

Gunakan pernyataan UPDATE untuk memperbarui item.

```
    //  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;
        }
    }
```

Gunakan pernyataan DELETE untuk menghapus sebuah item.

```
    // 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;
        }
    }
```

- Untuk detail API, lihat ExecuteStatementdi Referensi AWS SDK untuk C++ API.

Go

SDK untuk Go V2

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

Tentukan struct penerima fungsi untuk contoh.

```
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
}
```

Gunakan pernyataan INSERT untuk menambahkan item.

```
// 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
}
```

Gunakan pernyataan SELECT untuk mendapatkan item.

```
// 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
}
```

Gunakan pernyataan SELECT untuk mendapatkan daftar item dan memproyeksikan hasilnya.

```
// 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
}
```

Gunakan pernyataan UPDATE untuk memperbarui item.

```
// 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
}
```

Gunakan pernyataan DELETE untuk menghapus sebuah item.

```
// 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
}
```

Tentukan struct Movie yang digunakan dalam contoh ini.

```
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"])
}
```

- Untuk detail API, lihat ExecuteStatementdi Referensi AWS SDK untuk Go API.

JavaScript

SDK untuk JavaScript (v3)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

Buat item menggunakan 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;
};
```

Dapatkan item menggunakan 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: "SELECT * FROM CloudTypes WHERE IsStorm=?",
    Parameters: [false],
    ConsistentRead: true,
  });

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

```
  return response;
};
```

Perbarui item menggunakan 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;
};
```

Hapus item menggunakan 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;
};
```

- Untuk detail API, lihat ExecuteStatementdi Referensi AWS SDK untuk JavaScript API.

PHP

SDK untuk PHP

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
    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,
        ]);
    }
```

- Untuk detail API, lihat ExecuteStatementdi Referensi AWS SDK untuk PHP API.

Python

SDK untuk Python (Boto3)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
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
```

- Untuk detail API, lihat ExecuteStatementdi AWS SDK for Python (Boto3) Referensi API.

Ruby

SDK untuk Ruby

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

Pilih satu item menggunakan 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

  # 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
```

Perbarui satu item menggunakan 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
```

Tambahkan satu item menggunakan 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

  # 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
```

Hapus satu item menggunakan 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
```

- Untuk detail API, lihat ExecuteStatementdi Referensi AWS SDK untuk Ruby API.

Untuk daftar lengkap panduan pengembang AWS SDK dan contoh kode, lihatMenggunakan DynamoDB dengan SDK AWS. Topik ini juga mencakup informasi tentang memulai dan detail tentang versi SDK sebelumnya.

Gunakan **`GetItem`** dengan AWS SDK atau CLI

Contoh kode berikut menunjukkan cara menggunakan`GetItem`.

Contoh tindakan adalah kutipan kode dari program yang lebih besar dan harus dijalankan dalam konteks. Anda dapat melihat tindakan ini dalam konteks dalam contoh kode berikut:

- Pelajari dasar-dasarnya
- Mempercepat pembacaan dengan DAX
- Bekerja dengan tabel global dan replikasi multi-wilayah

.NET

SDK untuk .NET

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
        /// <summary>
        /// Gets information about an existing movie from the table.
        /// </summary>
        /// <param name="client">An initialized Amazon DynamoDB client object.</
param>
        /// <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 static async Task<Dictionary<string, AttributeValue>>
 GetItemAsync(AmazonDynamoDBClient client, Movie newMovie, string tableName)
        {
            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 client.GetItemAsync(request);
            return response.Item;
        }
```

- Untuk detail API, lihat GetItemdi Referensi AWS SDK untuk .NET API.

Bash

AWS CLI dengan skrip Bash

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
###############################################################################
# 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
}
```

Fungsi utilitas yang digunakan dalam contoh ini.

```
###############################################################################
# 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
}
```

- Untuk detail API, lihat GetItemdi Referensi AWS CLI Perintah.

C++

SDK untuk C++

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
//! 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();
}
```

- Untuk detail API, lihat GetItemdi Referensi AWS SDK untuk C++ API.

CLI

AWS CLI

Contoh 1: Untuk membaca item dalam tabel

`get-item`Contoh berikut mengambil item dari `MusicCollection` tabel. Tabel memiliki kunci hash-and-range utama (`Artist`dan`SongTitle`), jadi Anda harus menentukan kedua atribut ini. Perintah tersebut juga meminta informasi tentang kapasitas baca yang dikonsumsi oleh operasi.

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

Isi dari `key.json`:

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

Output:

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

Untuk informasi selengkapnya, lihat Membaca Item di Panduan Pengembang Amazon DynamoDB.

Contoh 2: Untuk membaca item menggunakan pembacaan yang konsisten

Contoh berikut mengambil item dari `MusicCollection` tabel menggunakan pembacaan yang sangat konsisten.

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

Isi dari `key.json`:

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

Output:

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

Untuk informasi selengkapnya, lihat Membaca Item di Panduan Pengembang Amazon DynamoDB.

Contoh 3: Untuk mengambil atribut tertentu dari suatu item

Contoh berikut menggunakan ekspresi proyeksi untuk mengambil hanya tiga atribut dari item yang diinginkan.

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

Isi dari `names.json`:

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

Output:

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

Untuk informasi selengkapnya, lihat Membaca Item di Panduan Pengembang Amazon DynamoDB.

- Untuk detail API, lihat GetItemdi Referensi AWS CLI Perintah.

Go

SDK untuk Go V2

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
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
}
```

Tentukan struct Movie yang digunakan dalam contoh ini.

```
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"])
}
```

- Untuk detail API, lihat GetItemdi Referensi AWS SDK untuk Go API.

Java

SDK untuk Java 2.x

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

Mendapat item dari tabel dengan menggunakan 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);
        }
    }
}
```

- Untuk detail API, lihat GetItemdi Referensi AWS SDK for Java 2.x API.

JavaScript

SDK untuk JavaScript (v3)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

Contoh ini menggunakan klien dokumen untuk menyederhanakan penggunaan item di DynamoDB. Untuk detail API, lihat 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;
};
```

- Untuk detail API, lihat GetItemdi Referensi AWS SDK untuk JavaScript API.

SDK untuk JavaScript (v2)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

Dapatkan item dari tabel.

```
// 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);
  }
});
```

Dapatkan item dari tabel menggunakan klien dokumen DynamoDB.

```
// 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);
  }
});
```

- Untuk informasi selengkapnya, silakan lihat Panduan Developer AWS SDK untuk JavaScript.
- Untuk detail API, lihat GetItemdi Referensi AWS SDK untuk JavaScript API.

Kotlin

SDK untuk Kotlin

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

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

- Untuk detail API, lihat GetItemdi AWS SDK untuk referensi API Kotlin.

PHP

SDK untuk PHP

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
        $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,
        ]);
    }
```

- Untuk detail API, lihat GetItemdi Referensi AWS SDK untuk PHP API.

PowerShell

Alat untuk PowerShell V4

Contoh 1: Mengembalikan item DynamoDB dengan SongTitle kunci partisi dan kunci sort Artist.

```
$key = @{
  SongTitle = 'Somewhere Down The Road'
  Artist = 'No One You Know'
} | ConvertTo-DDBItem

Get-DDBItem -TableName 'Music' -Key $key | ConvertFrom-DDBItem
```

Output:

```
Name                            Value
----                            -----
Genre                           Country
SongTitle                       Somewhere Down The Road
Price                           1.94
Artist                          No One You Know
CriticRating                    9
AlbumTitle                      Somewhat Famous
```

- Untuk detail API, lihat GetItemdi Referensi Alat AWS untuk PowerShell Cmdlet (V4).

Python

SDK untuk Python (Boto3)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
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"]
```

- Untuk detail API, lihat GetItemdi AWS SDK for Python (Boto3) Referensi API.

Ruby

SDK untuk Ruby

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
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
```

- Untuk detail API, lihat GetItemdi Referensi AWS SDK untuk Ruby API.

SAP ABAP

SDK untuk SAP ABAP

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
    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.
```

- Untuk detail API, lihat GetItemdi AWS SDK untuk referensi SAP ABAP API.

Swift

SDK untuk Swift

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
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
        }
    }
```

- Untuk detail API, lihat referensi GetItem AWSSDK untuk Swift API.

Untuk daftar lengkap panduan pengembang AWS SDK dan contoh kode, lihatMenggunakan DynamoDB dengan SDK AWS. Topik ini juga mencakup informasi tentang memulai dan detail tentang versi SDK sebelumnya.

Gunakan **`ListTables`** dengan AWS SDK atau CLI

Contoh kode berikut menunjukkan cara menggunakan`ListTables`.

Contoh tindakan adalah kutipan kode dari program yang lebih besar dan harus dijalankan dalam konteks. Anda dapat melihat tindakan ini dalam konteks dalam contoh kode berikut:

- Mengatur Kontrol Akses Berbasis Atribut

.NET

SDK untuk .NET

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
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);
    }
```

- Untuk detail API, lihat ListTablesdi Referensi AWS SDK untuk .NET API.

Bash

AWS CLI dengan skrip Bash

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

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

```
}
```

Fungsi utilitas yang digunakan dalam contoh ini.

```
###############################################################################
# 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
}
```

- Untuk detail API, lihat ListTablesdi Referensi AWS CLI Perintah.

C++

SDK untuk C++

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
//! 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;
}
```

- Untuk detail API, lihat ListTablesdi Referensi AWS SDK untuk C++ API.

CLI

AWS CLI

Contoh 1: Untuk daftar tabel

`list-tables`Contoh berikut mencantumkan semua tabel yang terkait dengan AWS akun saat ini dan Wilayah.

```
aws dynamodb list-tables
```

Output:

```
{
    "TableNames": [
        "Forum",
        "ProductCatalog",
        "Reply",
        "Thread"
    ]
}
```

Untuk informasi selengkapnya, lihat Daftar Nama Tabel di Panduan Pengembang Amazon DynamoDB.

Contoh 2: Untuk membatasi ukuran halaman

Contoh berikut mengembalikan daftar semua tabel yang ada, tetapi mengambil hanya satu item dalam setiap panggilan, melakukan beberapa panggilan jika perlu untuk mendapatkan seluruh daftar. Membatasi ukuran halaman berguna saat menjalankan perintah daftar pada sejumlah besar sumber daya, yang dapat mengakibatkan kesalahan "waktu habis" saat menggunakan ukuran halaman default 1000.

```
aws dynamodb list-tables \
    --page-size 1
```

Output:

```
{
    "TableNames": [
        "Forum",
        "ProductCatalog",
        "Reply",
        "Thread"
    ]
}
```

Untuk informasi selengkapnya, lihat Daftar Nama Tabel di Panduan Pengembang Amazon DynamoDB.

Contoh 3: Untuk membatasi jumlah item yang dikembalikan

Contoh berikut membatasi jumlah item yang dikembalikan ke 2. Respons mencakup `NextToken` nilai yang dapat digunakan untuk mengambil halaman hasil berikutnya.

```
aws dynamodb list-tables \
    --max-items 2
```

Output:

```
{
    "TableNames": [
        "Forum",
        "ProductCatalog"
    ],
    "NextToken":
 "abCDeFGhiJKlmnOPqrSTuvwxYZ1aBCdEFghijK7LM51nOpqRSTuv3WxY3ZabC5dEFGhI2Jk3LmnoPQ6RST9"
}
```

Untuk informasi selengkapnya, lihat Daftar Nama Tabel di Panduan Pengembang Amazon DynamoDB.

Contoh 4: Untuk mengambil halaman berikutnya dari hasil

Perintah berikut menggunakan `NextToken` nilai dari panggilan sebelumnya ke `list-tables` perintah untuk mengambil halaman lain dari hasil. Karena respons dalam kasus ini tidak termasuk `NextToken` nilai, kami tahu bahwa kami telah mencapai akhir hasil.

```
aws dynamodb list-tables \
    --starting-
token abCDeFGhiJKlmnOPqrSTuvwxYZ1aBCdEFghijK7LM51nOpqRSTuv3WxY3ZabC5dEFGhI2Jk3LmnoPQ6RST9
```

Output:

```
{
    "TableNames": [
        "Reply",
        "Thread"
    ]
}
```

Untuk informasi selengkapnya, lihat Daftar Nama Tabel di Panduan Pengembang Amazon DynamoDB.

- Untuk detail API, lihat ListTablesdi Referensi AWS CLI Perintah.

Go

SDK untuk Go V2

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
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
}
```

- Untuk detail API, lihat ListTablesdi Referensi AWS SDK untuk Go API.

Java

SDK untuk Java 2.x

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
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!");
    }
}
```

- Untuk detail API, lihat ListTablesdi Referensi AWS SDK for Java 2.x API.

JavaScript

SDK untuk JavaScript (v3)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
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;
};
```

- Untuk informasi selengkapnya, silakan lihat Panduan Developer AWS SDK untuk JavaScript.
- Untuk detail API, lihat ListTablesdi Referensi AWS SDK untuk JavaScript API.

SDK untuk JavaScript (v2)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
// 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);
  }
});
```

- Untuk informasi selengkapnya, silakan lihat Panduan Developer AWS SDK untuk JavaScript.
- Untuk detail API, lihat ListTablesdi Referensi AWS SDK untuk JavaScript API.

Kotlin

SDK untuk Kotlin

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
suspend fun listAllTables() {
    DynamoDbClient { region = "us-east-1" }.use { ddb ->
        val response = ddb.listTables(ListTablesRequest {})
        response.tableNames?.forEach { tableName ->
            println("Table name is $tableName")
        }
    }
}
```

- Untuk detail API, lihat ListTablesdi AWS SDK untuk referensi API Kotlin.

PHP

SDK untuk PHP

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
public function listTables($exclusiveStartTableName = "", $limit = 100)
{
    $this->dynamoDbClient->listTables([
        'ExclusiveStartTableName' => $exclusiveStartTableName,
        'Limit' => $limit,
    ]);
}
```

- Untuk detail API, lihat ListTablesdi Referensi AWS SDK untuk PHP API.

PowerShell

Alat untuk PowerShell V4

Contoh 1: Mengembalikan rincian semua tabel, secara otomatis iterasi sampai layanan menunjukkan tidak ada tabel lebih lanjut.

```
Get-DDBTableList
```

- Untuk detail API, lihat ListTablesdi Referensi Alat AWS untuk PowerShell Cmdlet (V4).

Python

SDK untuk Python (Boto3)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
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
```

- Untuk detail API, lihat ListTablesdi AWS SDK for Python (Boto3) Referensi API.

Ruby

SDK untuk Ruby

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

Tentukan apakah tabel ada.

```
# 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
```

- Untuk detail API, lihat ListTablesdi Referensi AWS SDK untuk Ruby API.

Rust

SDK untuk Rust

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
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)
}
```

Tentukan apakah tabel ada.

```
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()),
    }
}
```

- Untuk detail API, lihat ListTablesreferensi AWS SDK for Rust API.

SAP ABAP

SDK untuk SAP ABAP

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
    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.
```

- Untuk detail API, lihat ListTablesdi AWS SDK untuk referensi SAP ABAP API.

Swift

SDK untuk Swift

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
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)
    }
```

- Untuk detail API, lihat referensi ListTables AWSSDK untuk Swift API.

Untuk daftar lengkap panduan pengembang AWS SDK dan contoh kode, lihatMenggunakan DynamoDB dengan SDK AWS. Topik ini juga mencakup informasi tentang memulai dan detail tentang versi SDK sebelumnya.

Gunakan **`PutItem`** dengan AWS SDK atau CLI

Contoh kode berikut menunjukkan cara menggunakan`PutItem`.

Contoh tindakan adalah kutipan kode dari program yang lebih besar dan harus dijalankan dalam konteks. Anda dapat melihat tindakan ini dalam konteks dalam contoh kode berikut:

- Pelajari dasar-dasarnya
- Mempercepat pembacaan dengan DAX
- Skenario Indeks Sekunder Global Tingkat Lanjut
- Buat item dengan TTL
- Gunakan operasi bersyarat
- Bekerja dengan tabel global dan replikasi multi-wilayah
- Bekerja dengan Streams dan Time-to-Live

.NET

SDK untuk .NET

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
        /// <summary>
        /// Adds a new item to the table.
        /// </summary>
        /// <param name="client">An initialized Amazon DynamoDB client object.</
param>
        /// <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 static async Task<bool> PutItemAsync(AmazonDynamoDBClient client,
Movie newMovie, string tableName)
        {
            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,
            };

            var response = await client.PutItemAsync(request);
            return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
        }
```

- Untuk detail API, lihat PutItemdi Referensi AWS SDK untuk .NET API.

Bash

AWS CLI dengan skrip Bash

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

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

}
```

Fungsi utilitas yang digunakan dalam contoh ini.

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

```
}
```

- Untuk detail API, lihat PutItemdi Referensi AWS CLI Perintah.

C++

SDK untuk C++

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
//! 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);
}
```

Kode yang menunggu tabel menjadi aktif.

```
//! 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;
}
```

- Untuk detail API, lihat PutItemdi Referensi AWS SDK untuk C++ API.

CLI

AWS CLI

Contoh 1: Untuk menambahkan item ke tabel

`put-item`Contoh berikut menambahkan item baru ke MusicCollectiontabel.

```
aws dynamodb put-item \
    --table-name MusicCollection \
    --item file://item.json \
    --return-consumed-capacity TOTAL \
    --return-item-collection-metrics SIZE
```

Isi dari `item.json`:

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

Output:

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

Untuk informasi selengkapnya, lihat Menulis Item di Panduan Pengembang Amazon DynamoDB.

Contoh 2: Untuk menimpa item secara kondisional dalam tabel

`put-item`Contoh berikut menimpa item yang ada dalam `MusicCollection` tabel hanya jika item yang ada memiliki `AlbumTitle` atribut dengan nilai. `Greatest Hits` Perintah mengembalikan nilai item sebelumnya.

```
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
```

Isi dari `item.json`:

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

Isi dari `names.json`:

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

Isi dari `values.json`:

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

Output:

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

```
        },
        "SongTitle": {
            "S": "Call Me Today"
        }
    }
}
```

Jika kunci sudah ada, Anda akan melihat output berikut:

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

Untuk informasi selengkapnya, lihat Menulis Item di Panduan Pengembang Amazon DynamoDB.

- Untuk detail API, lihat PutItemdi Referensi AWS CLI Perintah.

Go

SDK untuk Go V2

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
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
}
```

Tentukan struct Movie yang digunakan dalam contoh ini.

```
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"])
}
```

- Untuk detail API, lihat PutItemdi Referensi AWS SDK untuk Go API.

Java

SDK untuk Java 2.x

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

Menempatkan item ke dalam tabel menggunakan 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);
        }
    }
}
```

- Untuk detail API, lihat PutItemdi Referensi AWS SDK for Java 2.x API.

JavaScript

SDK untuk JavaScript (v3)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

Contoh ini menggunakan klien dokumen untuk menyederhanakan penggunaan item di DynamoDB. Untuk detail API, lihat 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;
};
```

- Untuk detail API, lihat PutItemdi Referensi AWS SDK untuk JavaScript API.

SDK untuk JavaScript (v2)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

Menempatkan item dalam tabel.

```
// 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);
  }
});
```

Tempatkan item dalam tabel menggunakan klien dokumen DynamoDB.

```
// 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);
  }
});
```

- Untuk informasi selengkapnya, silakan lihat Panduan Developer AWS SDK untuk JavaScript.
- Untuk detail API, lihat PutItemdi Referensi AWS SDK untuk JavaScript API.

Kotlin

SDK untuk Kotlin

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

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

- Untuk detail API, lihat PutItemdi AWS SDK untuk referensi API Kotlin.

PHP

SDK untuk PHP

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
        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);
    }
```

- Untuk detail API, lihat PutItemdi Referensi AWS SDK untuk PHP API.

PowerShell

Alat untuk PowerShell V4

Contoh 1: Membuat item baru, atau mengganti item yang sudah ada dengan item baru.

```
$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
```

- Untuk detail API, lihat PutItemdi Referensi Alat AWS untuk PowerShell Cmdlet (V4).

Python

SDK untuk Python (Boto3)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
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
```

- Untuk detail API, lihat PutItemdi AWS SDK for Python (Boto3) Referensi API.

Ruby

SDK untuk Ruby

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
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
```

- Untuk detail API, lihat PutItemdi Referensi AWS SDK untuk Ruby API.

Rust

SDK untuk Rust

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
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,
    })
}
```

- Untuk detail API, lihat PutItemreferensi AWS SDK for Rust API.

SAP ABAP

SDK untuk SAP ABAP

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
    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.
```

- Untuk detail API, lihat PutItemdi AWS SDK untuk referensi SAP ABAP API.

Swift

SDK untuk Swift

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
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
    }
```

- Untuk detail API, lihat referensi PutItem AWSSDK untuk Swift API.

Untuk daftar lengkap panduan pengembang AWS SDK dan contoh kode, lihatMenggunakan DynamoDB dengan SDK AWS. Topik ini juga mencakup informasi tentang memulai dan detail tentang versi SDK sebelumnya.

Gunakan **`Query`** dengan AWS SDK atau CLI

Contoh kode berikut menunjukkan cara menggunakan`Query`.

Contoh tindakan adalah kutipan kode dari program yang lebih besar dan harus dijalankan dalam konteks. Anda dapat melihat tindakan ini dalam konteks dalam contoh kode berikut:

- Pelajari dasar-dasarnya
- Mempercepat pembacaan dengan DAX
- Skenario Indeks Sekunder Global Tingkat Lanjut
- Bandingkan beberapa nilai dengan satu atribut
- Kelola Indeks Sekunder Global
- Lakukan operasi kueri lanjutan

- Kueri tabel menggunakan kondisi begins_with
- Kueri tabel menggunakan rentang tanggal
- Kueri tabel menggunakan Global Secondary Index
- Kueri tabel dengan ekspresi filter yang kompleks
- Kueri tabel dengan ekspresi filter dinamis
- Kueri tabel dengan ekspresi filter dan batas
- Kueri tabel dengan atribut bersarang
- Kueri tabel dengan pagination
- Kueri tabel dengan pembacaan yang sangat konsisten
- Kueri untuk item TTL
- Tabel kueri menggunakan pola tanggal dan waktu
- Gunakan nama atribut ekspresi
- Bekerja dengan Indeks Sekunder Lokal

.NET

SDK untuk .NET

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
        /// <summary>
        /// Queries the table for movies released in a particular year and
        /// then displays the information for the movies returned.
        /// </summary>
        /// <param name="client">The initialized DynamoDB client object.</param>
        /// <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 static async Task<int> QueryMoviesAsync(AmazonDynamoDBClient
client, string tableName, int year)
```

```
        {
            var movieTable = Table.LoadTable(client, tableName);
            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;

            Search 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;
        }
```

- Untuk detail API, lihat Kueri di Referensi API AWS SDK untuk .NET .

Bash

AWS CLI dengan skrip Bash

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
###############################################################################
# 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
}
```

Fungsi utilitas yang digunakan dalam contoh ini.

```
###############################################################################
# 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
}
```

- Untuk detail API, lihat Kueri di Referensi Perintah AWS CLI .

C++

SDK untuk C++

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
//! 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;
}
```

- Untuk detail API, lihat Kueri di Referensi API AWS SDK untuk C++ .

CLI

AWS CLI

Contoh 1: Untuk menanyakan tabel

`query`Contoh berikut query item dalam `MusicCollection` tabel. Tabel memiliki kunci hash-and-range utama (`Artist`dan`SongTitle`), tetapi kueri ini hanya menentukan nilai kunci hash. Ini mengembalikan judul lagu oleh artis bernama “No One You Know”.

```
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
```

Isi dari `expression-attributes.json`:

```
{
    ":v1": {"S": "No One You Know"}
}
```

Output:

```
{
    "Items": [
        {
            "SongTitle": {
                "S": "Call Me Today"
            },
            "SongTitle": {
                "S": "Scared of My Shadow"
            }
        }
    ],
    "Count": 2,
    "ScannedCount": 2,
    "ConsumedCapacity": {
        "TableName": "MusicCollection",
        "CapacityUnits": 0.5
    }
}
```

Untuk informasi selengkapnya, lihat Bekerja dengan Kueri di DynamoDB di Panduan Pengembang Amazon DynamoDB.

Contoh 2: Untuk menanyakan tabel menggunakan pembacaan yang sangat konsisten dan melintasi indeks dalam urutan menurun

Contoh berikut melakukan kueri yang sama dengan contoh pertama, tetapi mengembalikan hasil dalam urutan terbalik dan menggunakan pembacaan yang sangat konsisten.

```
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
```

Isi dari `expression-attributes.json`:

```
{
    ":v1": {"S": "No One You Know"}
}
```

Output:

```
{
    "Items": [
        {
            "SongTitle": {
                "S": "Scared of My Shadow"
            }
        },
        {
            "SongTitle": {
                "S": "Call Me Today"
            }
        }
    ],
    "Count": 2,
    "ScannedCount": 2,
    "ConsumedCapacity": {
        "TableName": "MusicCollection",
        "CapacityUnits": 1.0
    }
}
```

Untuk informasi selengkapnya, lihat Bekerja dengan Kueri di DynamoDB di Panduan Pengembang Amazon DynamoDB.

Contoh 3: Untuk menyaring hasil tertentu

Contoh berikut query `MusicCollection` tetapi mengecualikan hasil dengan nilai-nilai tertentu dalam atribut. `AlbumTitle` Perhatikan bahwa ini tidak mempengaruhi `ScannedCount` atau`ConsumedCapacity`, karena filter diterapkan setelah item telah dibaca.

```
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
```

Isi dari `values.json`:

```
{
    ":v1": {"S": "No One You Know"},
    ":v2": {"S": "Blue Sky Blues"},
    ":v3": {"S": "Greatest Hits"}
}
```

Isi dari `names.json`:

```
{
    "#n1": "Artist",
    "#n2": "AlbumTitle"
}
```

Output:

```
{
    "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
    }
}
```

Untuk informasi selengkapnya, lihat Bekerja dengan Kueri di DynamoDB di Panduan Pengembang Amazon DynamoDB.

Contoh 4: Untuk mengambil hanya jumlah item

Contoh berikut mengambil hitungan item yang cocok dengan query, tetapi tidak mengambil salah satu item itu sendiri.

```
aws dynamodb query \
    --table-name MusicCollection \
    --select COUNT \
    --key-condition-expression "Artist = :v1" \
    --expression-attribute-values file://expression-attributes.json
```

Isi dari `expression-attributes.json`:

```
{
    ":v1": {"S": "No One You Know"}
}
```

Output:

```
{
    "Count": 2,
    "ScannedCount": 2,
    "ConsumedCapacity": null
}
```

Untuk informasi selengkapnya, lihat Bekerja dengan Kueri di DynamoDB di Panduan Pengembang Amazon DynamoDB.

Contoh 5: Untuk menanyakan indeks

Contoh berikut query indeks `AlbumTitleIndex` sekunder lokal. Query mengembalikan semua atribut dari tabel dasar yang telah diproyeksikan ke indeks sekunder lokal. Perhatikan bahwa saat menanyakan indeks sekunder lokal atau indeks sekunder global, Anda juga harus memberikan nama tabel dasar menggunakan `table-name` parameter.

```
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
```

Isi dari `expression-attributes.json`:

```
{
    ":v1": {"S": "No One You Know"}
}
```

Output:

```
{
    "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
            }
        }
    }
}
```

Untuk informasi selengkapnya, lihat Bekerja dengan Kueri di DynamoDB di Panduan Pengembang Amazon DynamoDB.

- Untuk detail API, lihat Kueri di Referensi Perintah AWS CLI .

Go

SDK untuk Go V2

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
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
}
```

Tentukan struct Movie yang digunakan dalam contoh ini.

```
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"])
}
```

- Untuk detail API, lihat Kueri di Referensi API AWS SDK untuk Go .

Java

SDK untuk Java 2.x

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

Kueri tabel dengan menggunakan 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;
```

```
    }
}
```

Melakukan tabel menggunakan `DynamoDbClient` dan indeks sekunder.

```
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);
        }
    }
}
```

- Untuk detail API, lihat Kueri di Referensi API AWS SDK for Java 2.x .

JavaScript

SDK untuk JavaScript (v3)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

Contoh ini menggunakan klien dokumen untuk menyederhanakan penggunaan item di DynamoDB. Untuk detail API, lihat 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;
};
```

- Untuk informasi selengkapnya, silakan lihat Panduan Developer AWS SDK untuk JavaScript.
- Untuk detail API, lihat Kueri di Referensi API AWS SDK untuk JavaScript .

SDK untuk JavaScript (v2)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
// 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);
  }
});
```

- Untuk informasi selengkapnya, silakan lihat Panduan Developer AWS SDK untuk JavaScript.
- Untuk detail API, lihat Kueri di Referensi API AWS SDK untuk JavaScript .

Kotlin

SDK untuk Kotlin

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
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 { region = "us-east-1" }.use { ddb ->
        val response = ddb.query(request)
        return response.count
    }
}
```

- Untuk detail API, lihat Kueri di Referensi API AWS SDK untuk Kotlin.

PHP

SDK untuk PHP

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
        $birthKey = [
            'Key' => [
                'year' => [
                    'N' => "$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);
    }
```

- Untuk detail API, lihat Kueri di Referensi API AWS SDK untuk PHP .

PowerShell

Alat untuk PowerShell V4

Contoh 1: Memanggil query yang mengembalikan item DynamoDB dengan yang ditentukan dan Artist. SongTitle

```
$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
```

Output:

```
Name                           Value
----                           -----
Genre                          Country
Artist                         No One You Know
Price                          1.94
CriticRating                   9
SongTitle                      Somewhere Down The Road
AlbumTitle                     Somewhat Famous
```

- Untuk detail API, lihat Kueri di Referensi Alat AWS untuk PowerShell Cmdlet (V4).

Python

SDK untuk Python (Boto3)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

Kueri item menggunakan ekspresi kondisi kunci.

```
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"]
```

Kueri dan proyeksikan item untuk mengembalikan subset data.

```
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"]
```

- Untuk detail API, lihat Kueri di Referensi API AWS SDK untuk Python (Boto3).

Ruby

SDK untuk Ruby

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
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
```

- Untuk detail API, lihat Kueri di Referensi API AWS SDK untuk Ruby .

Rust

SDK untuk Rust

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

Temukan film yang dibuat pada tahun tertentu.

```
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![])
    }
```

```
}
```

- Untuk detail API, lihat Kueri di referensi API AWS SDK untuk Rust.

SAP ABAP

SDK untuk SAP ABAP

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
    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.
```

- Untuk detail API, lihat Kueri di referensi API AWS SDK untuk SAP ABAP.

Swift

SDK untuk Swift

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
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
        }
    }
```

- Untuk detail API, lihat Kueri di Referensi API AWS SDK untuk Swift.

Untuk daftar lengkap panduan pengembang AWS SDK dan contoh kode, lihatMenggunakan DynamoDB dengan SDK AWS. Topik ini juga mencakup informasi tentang memulai dan detail tentang versi SDK sebelumnya.

Gunakan **`Scan`** dengan AWS SDK atau CLI

Contoh kode berikut menunjukkan cara menggunakan`Scan`.

Contoh tindakan adalah kutipan kode dari program yang lebih besar dan harus dijalankan dalam konteks. Anda dapat melihat tindakan ini dalam konteks dalam contoh kode berikut:

- Pelajari dasar-dasarnya

- Mempercepat pembacaan dengan DAX
- Bandingkan beberapa nilai dengan satu atribut

.NET

SDK untuk .NET

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
        public static async Task<int> ScanTableAsync(
            AmazonDynamoDBClient client,
            string tableName,
            int startYear,
            int endYear)
        {
            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 client.ScanAsync(request);
                foundCount += response.Items.Count;
                response.Items.ForEach(i => DisplayItem(i));
                request.ExclusiveStartKey = response.LastEvaluatedKey;
            }
            while (response.LastEvaluatedKey.Count > 0);
            return foundCount;
        }
```

- Untuk detail API, lihat Scan di Referensi API AWS SDK untuk .NET .

Bash

AWS CLI dengan skrip Bash

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
###############################################################################
# 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
}
```

Fungsi utilitas yang digunakan dalam contoh ini.

```
###############################################################################
# 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
}
```

- Untuk detail API, lihat Scan in Referensi Perintah AWS CLI .

C++

SDK untuk C++

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
//! 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;
}
```

- Untuk detail API, lihat Scan di Referensi API AWS SDK untuk C++ .

CLI

AWS CLI

Untuk memindai tabel

`scan`Contoh berikut memindai seluruh `MusicCollection` tabel, dan kemudian mempersempit hasilnya menjadi lagu-lagu oleh artis “No One You Know”. Untuk setiap item, hanya judul album dan judul lagu yang dikembalikan.

```
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
```

Isi dari `expression-attribute-names.json`:

```
{
    "#ST": "SongTitle",
    "#AT":"AlbumTitle"
```

```
}
```

Isi dari `expression-attribute-values.json`:

```
{
    ":a": {"S": "No One You Know"}
}
```

Output:

```
{
    "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
}
```

Untuk informasi selengkapnya, lihat Bekerja dengan Pemindaian di DynamoDB di Panduan Pengembang Amazon DynamoDB.

- Untuk detail API, lihat Scan in Referensi Perintah AWS CLI .

Go

SDK untuk Go V2

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
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
}
```

Tentukan struct Movie yang digunakan dalam contoh ini.

```
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"])
}
```

- Untuk detail API, lihat Scan di Referensi API AWS SDK untuk Go .

Java

SDK untuk Java 2.x

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

Memindai tabel Amazon DynamoDbClientDynamoDB menggunakan.

```
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);
        }
    }
}
```

- Untuk detail API, lihat Scan di Referensi API AWS SDK for Java 2.x .

JavaScript

SDK untuk JavaScript (v3)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

Contoh ini menggunakan klien dokumen untuk menyederhanakan penggunaan item di DynamoDB. Untuk detail API, lihat 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;
};
```

- Untuk detail API, lihat Scan di Referensi API AWS SDK untuk JavaScript .

SDK untuk JavaScript (v2)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
// 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 + ")"
      );
    });
  }
});
```

- Untuk informasi selengkapnya, silakan lihat Panduan Developer AWS SDK untuk JavaScript.
- Untuk detail API, lihat Scan di Referensi API AWS SDK untuk JavaScript .

Kotlin

SDK untuk Kotlin

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
suspend fun scanItems(tableNameVal: String) {
    val request =
        ScanRequest {
            tableName = tableNameVal
        }

    DynamoDbClient { 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]}")
            }
        }
    }
}
```

- Untuk detail API, lihat Scan di Referensi API AWS SDK untuk Kotlin.

PHP

SDK untuk PHP

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
        $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);
    }
```

- Untuk detail API, lihat Scan di Referensi API AWS SDK untuk PHP .

PowerShell

Alat untuk PowerShell V4

Contoh 1: Mengembalikan semua item dalam tabel Musik.

```
Invoke-DDBScan -TableName 'Music' | ConvertFrom-DDBItem
```

Output:

```
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
```

Contoh 2: Mengembalikan item dalam tabel Musik dengan CriticRating lebih besar dari atau sama dengan sembilan.

```
$scanFilter = @{
        CriticRating = [Amazon.DynamoDBv2.Model.Condition]@{
            AttributeValueList = @(@{N = '9'})
            ComparisonOperator = 'GE'
        }
    }
```

```
    Invoke-DDBScan -TableName 'Music' -ScanFilter $scanFilter | ConvertFrom-
DDBItem
```

Output:

```
Name                           Value
----                           -----
Genre                          Country
Artist                         No One You Know
Price                          1.94
CriticRating                   9
SongTitle                      Somewhere Down The Road
AlbumTitle                     Somewhat Famous
```

- Untuk detail API, lihat Memindai di Referensi Alat AWS untuk PowerShell Cmdlet (V4).

Python

SDK untuk Python (Boto3)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
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
```

- Untuk detail API, lihat Scan di Referensi API AWS SDK untuk Python (Boto3).

Ruby

SDK untuk Ruby

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
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
```

- Untuk detail API, lihat Scan di Referensi API AWS SDK untuk Ruby .

Rust

SDK untuk Rust

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
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(())
}
```

- Untuk detail API, lihat Scan di Referensi API AWS SDK untuk Rust.

SAP ABAP

SDK untuk SAP ABAP

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
    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.
```

- Untuk detail API, lihat Scan di Referensi API AWS SDK untuk SAP ABAP.

Swift

SDK untuk Swift

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
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
        }
    }
```

- Untuk detail API, lihat Scan di Referensi API AWS SDK untuk Swift.

Untuk daftar lengkap panduan pengembang AWS SDK dan contoh kode, lihatMenggunakan DynamoDB dengan SDK AWS. Topik ini juga mencakup informasi tentang memulai dan detail tentang versi SDK sebelumnya.

Gunakan **`UpdateItem`** dengan AWS SDK atau CLI

Contoh kode berikut menunjukkan cara menggunakan`UpdateItem`.

Contoh tindakan adalah kutipan kode dari program yang lebih besar dan harus dijalankan dalam konteks. Anda dapat melihat tindakan ini dalam konteks dalam contoh kode berikut:

- Pelajari dasar-dasarnya
- Perbarui TTL item secara kondisional
- Hitung operator ekspresi
- Lakukan operasi daftar
- Lakukan operasi peta
- Lakukan operasi set
- Memahami urutan ekspresi pembaruan
- Perbarui TTL item
- Gunakan operasi penghitung atom
- Gunakan operasi bersyarat
- Gunakan nama atribut ekspresi

.NET

SDK untuk .NET

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
        /// <summary>
        /// Updates an existing item in the movies table.
        /// </summary>
        /// <param name="client">An initialized Amazon DynamoDB client object.</
param>
        /// <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 static async Task<bool> UpdateItemAsync(
            AmazonDynamoDBClient client,
            Movie newMovie,
            MovieInfo newInfo,
            string tableName)
        {
            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,
        };

        var response = await client.UpdateItemAsync(request);

        return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
    }
```

- Untuk detail API, lihat UpdateItemdi Referensi AWS SDK untuk .NET API.

Bash

AWS CLI dengan skrip Bash

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

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

}
```

Fungsi utilitas yang digunakan dalam contoh ini.

```
###############################################################################
# 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
}
```

- Untuk detail API, lihat UpdateItemdi Referensi AWS CLI Perintah.

C++

SDK untuk C++

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
//! 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);
}
```

Kode yang menunggu tabel menjadi aktif.

```
//! 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;
}
```

- Untuk detail API, lihat UpdateItemdi Referensi AWS SDK untuk C++ API.

CLI

AWS CLI

Contoh 1: Untuk memperbarui item dalam tabel

Contoh `update-item` berikut memperbarui item dalam tabel `MusicCollection`. Ia menambahkan atribut baru (`Year`) dan memodifikasi `AlbumTitle` atribut. Semua atribut dalam item, seperti yang muncul setelah pembaruan, dikembalikan sebagai respons.

```
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
```

Isi dari `key.json`:

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

Isi dari `expression-attribute-names.json`:

```
{
    "#Y":"Year", "#AT":"AlbumTitle"
}
```

Isi dari `expression-attribute-values.json`:

```
{
    ":y":{"N": "2015"},
    ":t":{"S": "Louder Than Ever"}
}
```

Output:

```
{
    "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
        ]
    }
}
```

Untuk informasi selengkapnya, lihat Menulis Item di Panduan Pengembang Amazon DynamoDB.

Contoh 2: Untuk memperbarui item secara kondisional

Contoh berikut memperbarui item dalam `MusicCollection` tabel, tetapi hanya jika item yang ada belum memiliki `Year` atribut.

```
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)"
```

Isi dari `key.json`:

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

Isi dari `expression-attribute-names.json`:

```
{
    "#Y":"Year",
    "#AT":"AlbumTitle"
}
```

Isi dari `expression-attribute-values.json`:

```
{
    ":y":{"N": "2015"},
    ":t":{"S": "Louder Than Ever"}
}
```

Jika item sudah memiliki `Year` atribut, DynamoDB mengembalikan output berikut.

```
An error occurred (ConditionalCheckFailedException) when calling the UpdateItem
 operation: The conditional request failed
```

Untuk informasi selengkapnya, lihat Menulis Item di Panduan Pengembang Amazon DynamoDB.

- Untuk detail API, lihat UpdateItemdi Referensi AWS CLI Perintah.

Go

SDK untuk Go V2

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
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
}
```

Tentukan struct Movie yang digunakan dalam contoh ini.

```
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"])
}
```

- Untuk detail API, lihat UpdateItemdi Referensi AWS SDK untuk Go API.

Java

SDK untuk Java 2.x

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

Memperbarui item dalam tabel menggunakan 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!");
    }
}
```

- Untuk detail API, lihat UpdateItemdi Referensi AWS SDK for Java 2.x API.

JavaScript

SDK untuk JavaScript (v3)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

Contoh ini menggunakan klien dokumen untuk menyederhanakan penggunaan item di DynamoDB. Untuk detail API, lihat 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;
};
```

- Untuk detail API, lihat UpdateItemdi Referensi AWS SDK untuk JavaScript API.

Kotlin

SDK untuk Kotlin

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
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 { region = "us-east-1" }.use { ddb ->
        ddb.updateItem(request)
        println("Item in $tableNameVal was updated")
    }
}
```

- Untuk detail API, lihat UpdateItemdi AWS SDK untuk referensi API Kotlin.

PHP

SDK untuk PHP

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
        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
                ]
            ],
        ]);
    }
```

- Untuk detail API, lihat UpdateItemdi Referensi AWS SDK untuk PHP API.

PowerShell

Alat untuk PowerShell V4

Contoh 1: Menetapkan atribut genre ke 'Rap' pada item DynamoDB dengan SongTitle kunci partisi dan Artis kunci sortir.

```
$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
```

Output:

```
Name                           Value
----                           -----
Genre                          Rap
```

- Untuk detail API, lihat UpdateItemdi Referensi Alat AWS untuk PowerShell Cmdlet (V4).

Python

SDK untuk Python (Boto3)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

Perbarui item menggunakan ekspresi pembaruan.

```
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"]
```

Perbarui item menggunakan ekspresi pembaruan yang menyertakan operasi aritmatika.

```
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"]
```

Perbarui item hanya jika memenuhi persyaratan tertentu.

```
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"]
```

- Untuk detail API, lihat UpdateItemdi AWS SDK for Python (Boto3) Referensi API.

Ruby

SDK untuk Ruby

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
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
```

- Untuk detail API, lihat UpdateItemdi Referensi AWS SDK untuk Ruby API.

SAP ABAP

SDK untuk SAP ABAP

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
    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.
```

- Untuk detail API, lihat UpdateItemdi AWS SDK untuk referensi SAP ABAP API.

Swift

SDK untuk Swift

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
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
        }
    }
```

- Untuk detail API, lihat referensi UpdateItem AWSSDK untuk Swift API.

Untuk daftar lengkap panduan pengembang AWS SDK dan contoh kode, lihatMenggunakan DynamoDB dengan SDK AWS. Topik ini juga mencakup informasi tentang memulai dan detail tentang versi SDK sebelumnya.

Gunakan **`UpdateTable`** dengan AWS SDK atau CLI

Contoh kode berikut menunjukkan cara menggunakan`UpdateTable`.

Contoh tindakan adalah kutipan kode dari program yang lebih besar dan harus dijalankan dalam konteks. Anda dapat melihat tindakan ini dalam konteks dalam contoh kode berikut:

- Kelola Indeks Sekunder Global
- Perbarui pengaturan throughput hangat tabel
- Bekerja dengan tabel global dan replikasi multi-wilayah
- Bekerja dengan enkripsi tabel

C++

SDK untuk C++

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
//! 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);
}
```

Kode yang menunggu tabel menjadi aktif.

```
//! 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;
}
```

- Untuk detail API, lihat UpdateTabledi Referensi AWS SDK untuk C++ API.

CLI

AWS CLI

Contoh 1: Untuk memodifikasi mode penagihan tabel

`update-table`Contoh berikut meningkatkan kapasitas baca dan tulis yang disediakan di atas meja. `MusicCollection`

```
aws dynamodb update-table \
    --table-name MusicCollection \
    --billing-mode PROVISIONED \
    --provisioned-throughput ReadCapacityUnits=15,WriteCapacityUnits=10
```

Output:

```
{
    "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"
        }
    }
}
```

Untuk informasi selengkapnya, lihat Memperbarui Tabel di Panduan Pengembang Amazon DynamoDB.

Contoh 2: Untuk membuat indeks sekunder global

Contoh berikut menambahkan indeks sekunder global ke `MusicCollection` tabel.

```
aws dynamodb update-table \
    --table-name MusicCollection \
    --attribute-definitions AttributeName=AlbumTitle,AttributeType=S \
```

```
    --global-secondary-index-updates file://gsi-updates.json
```

Isi dari `gsi-updates.json`:

```
[
    {
        "Create": {
            "IndexName": "AlbumTitle-index",
            "KeySchema": [
                {
                    "AttributeName": "AlbumTitle",
                    "KeyType": "HASH"
                }
            ],
            "ProvisionedThroughput": {
                "ReadCapacityUnits": 10,
                "WriteCapacityUnits": 10
            },
            "Projection": {
                "ProjectionType": "ALL"
            }
        }
    }
]
```

Output:

```
{
    "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"
            }
        ]
    }
}
```

Untuk informasi selengkapnya, lihat Memperbarui Tabel di Panduan Pengembang Amazon DynamoDB.

Contoh 3: Untuk mengaktifkan DynamoDB Streams di atas meja

Perintah berikut memungkinkan DynamoDB Streams di atas meja. `MusicCollection`

```
aws dynamodb update-table \
    --table-name MusicCollection \
    --stream-specification StreamEnabled=true,StreamViewType=NEW_IMAGE
```

Output:

```
{
    "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"
    }
}
```

Untuk informasi selengkapnya, lihat Memperbarui Tabel di Panduan Pengembang Amazon DynamoDB.

Contoh 4: Untuk mengaktifkan enkripsi sisi server

Contoh berikut memungkinkan enkripsi sisi server di atas meja. `MusicCollection`

```
aws dynamodb update-table \
    --table-name MusicCollection \
    --sse-specification Enabled=true,SSEType=KMS
```

Output:

```
{
    "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"
        }
    }
}
```

Untuk informasi selengkapnya, lihat Memperbarui Tabel di Panduan Pengembang Amazon DynamoDB.

- Untuk detail API, lihat UpdateTabledi Referensi AWS CLI Perintah.

PowerShell

Alat untuk PowerShell V4

Contoh 1: Memperbarui throughput yang disediakan untuk tabel yang diberikan.

```
Update-DDBTable -TableName "myTable" -ReadCapacity 10 -WriteCapacity 5
```

- Untuk detail API, lihat UpdateTabledi Referensi Alat AWS untuk PowerShell Cmdlet (V4).

Untuk daftar lengkap panduan pengembang AWS SDK dan contoh kode, lihatMenggunakan DynamoDB dengan SDK AWS. Topik ini juga mencakup informasi tentang memulai dan detail tentang versi SDK sebelumnya.

Gunakan **`UpdateTimeToLive`** dengan AWS SDK atau CLI

Contoh kode berikut menunjukkan cara menggunakan`UpdateTimeToLive`.

Contoh tindakan adalah kutipan kode dari program yang lebih besar dan harus dijalankan dalam konteks. Anda dapat melihat tindakan ini dalam konteks dalam contoh kode berikut:

- Bekerja dengan Streams dan Time-to-Live

CLI

AWS CLI

Untuk memperbarui pengaturan Waktu ke Langsung di atas meja

`update-time-to-live`Contoh berikut memungkinkan Time to Live pada tabel yang ditentukan.

```
aws dynamodb update-time-to-live \
    --table-name MusicCollection \
    --time-to-live-specification Enabled=true,AttributeName=ttl
```

Output:

```
{
    "TimeToLiveSpecification": {
        "Enabled": true,
```

```
        "AttributeName": "ttl"
    }
}
```

Untuk informasi selengkapnya, lihat Waktu untuk Hidup di Panduan Pengembang Amazon DynamoDB.

- Untuk detail API, lihat UpdateTimeToLivedi Referensi AWS CLI Perintah.

Java

SDK untuk Java 2.x

Aktifkan TTL pada tabel DynamoDB yang ada menggunakan. 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.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;
        }
    }
```

Nonaktifkan TTL pada tabel DynamoDB yang ada menggunakan. 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.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;
        }
    }
```

- Untuk detail API, lihat UpdateTimeToLivedi Referensi AWS SDK for Java 2.x API.

JavaScript

SDK untuk JavaScript (v3)

Aktifkan TTL pada tabel DynamoDB yang ada.

```
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');
```

Nonaktifkan TTL pada tabel DynamoDB yang ada.

```
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');
```

- Untuk detail API, lihat UpdateTimeToLivedi Referensi AWS SDK untuk JavaScript API.

Python

SDK untuk Python (Boto3)

Aktifkan TTL pada tabel DynamoDB yang ada.

```
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")
```

Nonaktifkan TTL pada tabel DynamoDB yang ada.

```
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")
```

- Untuk detail API, lihat UpdateTimeToLivedi AWS SDK for Python (Boto3) Referensi API.

Untuk daftar lengkap panduan pengembang AWS SDK dan contoh kode, lihatMenggunakan DynamoDB dengan SDK AWS. Topik ini juga mencakup informasi tentang memulai dan detail tentang versi SDK sebelumnya.

Skenario untuk DynamoDB menggunakan AWS SDKs

Contoh kode berikut menunjukkan kepada Anda bagaimana menerapkan skenario umum di DynamoDB dengan. AWS SDKs Skenario ini menunjukkan kepada Anda bagaimana menyelesaikan tugas tertentu dengan memanggil beberapa fungsi dalam DynamoDB atau dikombinasikan dengan yang lain. Layanan AWS Setiap skenario menyertakan tautan ke kode sumber lengkap, di mana Anda dapat menemukan instruksi tentang cara mengatur dan menjalankan kode.

Skenario menargetkan tingkat pengalaman menengah untuk membantu Anda memahami tindakan layanan dalam konteks.

Contoh

- Mempercepat pembacaan DynamoDB dengan DAX menggunakan SDK AWS
- Bekerja dengan skenario Indeks Sekunder Global DynamoDB tingkat lanjut menggunakan v2 AWS Command Line Interface
- Membangun aplikasi untuk mengirimkan data ke tabel DynamoDB
- Bandingkan beberapa nilai dengan satu atribut di DynamoDB dengan SDK AWS
- Perbarui item DynamoDB secara kondisional dengan TTL menggunakan SDK AWS

- Connect ke instance DynamoDB lokal menggunakan SDK AWS
- Hitung operator ekspresi di DynamoDB dengan SDK AWS
- Membuat API REST Gateway API untuk melacak data COVID-19
- Membuat aplikasi messenger dengan Step Functions
- Membuat aplikasi manajemen aset foto yang memungkinkan pengguna mengelola foto menggunakan label
- Membuat tabel DynamoDB dengan Indeks Sekunder Global menggunakan SDK AWS
- Buat tabel DynamoDB dengan pengaturan throughput hangat menggunakan SDK AWS
- Membuat aplikasi web untuk melacak data DynamoDB
- Membuat aplikasi obrolan websocket dengan API Gateway
- Buat item DynamoDB dengan TTL menggunakan SDK AWS
- Hapus data DynamoDB menggunakan pernyataan PartiQL DELETE dengan SDK AWS
- Mendeteksi APD dalam gambar dengan Amazon AWS Rekognition menggunakan SDK
- Masukkan data DynamoDB menggunakan pernyataan PartiQL INSERT dengan SDK AWS
- Menginvokasi fungsi Lambda dari browser
- Kelola Indeks Sekunder Global DynamoDB menggunakan v2 AWS Command Line Interface
- Mengelola kebijakan berbasis sumber daya DynamoDB menggunakan v2 AWS Command Line Interface
- Memantau kinerja Amazon DynamoDB menggunakan SDK AWS
- Melakukan operasi kueri DynamoDB lanjutan menggunakan SDK AWS
- Lakukan operasi daftar di DynamoDB dengan SDK AWS
- Melakukan operasi peta di DynamoDB dengan SDK AWS
- Lakukan operasi set di DynamoDB dengan SDK AWS
- Kueri tabel DynamoDB dengan menggunakan kumpulan pernyataan PartiQL dan SDK AWS
- Kueri tabel DynamoDB menggunakan PartiQL dan SDK AWS
- Kueri tabel DynamoDB menggunakan Global Secondary Index dengan SDK AWS
- Kueri tabel DynamoDB menggunakan kondisi begins_with dengan SDK AWS
- Kueri tabel DynamoDB menggunakan rentang tanggal dalam kunci pengurutan dengan SDK AWS
- Kueri tabel DynamoDB dengan ekspresi filter kompleks dengan SDK AWS
- Kueri tabel DynamoDB dengan ekspresi filter dinamis dengan SDK AWS

- Kueri tabel DynamoDB dengan ekspresi filter dan batas dengan SDK AWS
- Kueri tabel DynamoDB dengan atribut bersarang menggunakan SDK AWS
- Kueri tabel DynamoDB dengan pagination menggunakan SDK AWS
- Kueri tabel DynamoDB dengan pembacaan yang sangat konsisten menggunakan SDK AWS
- Kueri data DynamoDB menggunakan pernyataan PartiQL SELECT dengan SDK AWS
- Kueri tabel DynamoDB untuk item TTL menggunakan SDK AWS
- Kueri tabel DynamoDB menggunakan pola tanggal dan waktu dengan SDK AWS
- Simpan EXIF dan informasi gambar lainnya menggunakan SDK AWS
- Mengatur Kontrol Akses Berbasis Atribut untuk DynamoDB menggunakan v2 AWS Command Line Interface
- Memahami urutan ekspresi pembaruan di DynamoDB dengan SDK AWS
- Memperbarui setelan tabel DynamoDB dengan throughput hangat menggunakan SDK AWS
- Memperbarui item DynamoDB dengan TTL menggunakan SDK AWS
- Perbarui data DynamoDB menggunakan pernyataan PartiQL UPDATE dengan SDK AWS
- Menggunakan API Gateway untuk menginvokasi fungsi Lambda
- Menggunakan Step Functions untuk menginvokasi fungsi Lambda
- Menggunakan model dokumen untuk DynamoDB menggunakan SDK AWS
- Menggunakan model persistensi objek tingkat tinggi untuk DynamoDB menggunakan SDK AWS
- Gunakan operasi penghitung atom di DynamoDB dengan SDK AWS
- Menggunakan operasi bersyarat di DynamoDB dengan SDK AWS
- Gunakan nama atribut ekspresi di DynamoDB dengan SDK AWS
- Menggunakan peristiwa terjadwal untuk menginvokasi fungsi Lambda
- Bekerja dengan DynamoDB Local Secondary Indexes menggunakan v2 AWS Command Line Interface
- Bekerja dengan DynamoDB Streams dan menggunakan v2 Time-to-Live AWS Command Line Interface
- Bekerja dengan tabel global DynamoDB dan replikasi multi-wilayah menggunakan v2 AWS Command Line Interface
- Bekerja dengan penandaan sumber daya DynamoDB menggunakan v2 AWS Command Line Interface
- Bekerja dengan enkripsi tabel DynamoDB menggunakan v2 AWS Command Line Interface

Mempercepat pembacaan DynamoDB dengan DAX menggunakan SDK AWS

Contoh kode berikut ini menunjukkan cara untuk melakukan:

- Buat dan tulis data ke tabel dengan klien DAX dan SDK.
- Dapatkan, kueri, dan pindai tabel dengan kedua klien tersebut dan bandingkan performanya.

Untuk informasi selengkapnya, lihat Melakukan pengembangan dengan Klien DynamoDB Accelerator.

Python

SDK untuk Python (Boto3)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

Buat tabel dengan klien DAX atau Boto3.

```
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.")
```

Tulis data uji ke tabel tersebut.

```
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)
```

Dapatkan item untuk sejumlah iterasi untuk klien DAX maupun klien Boto3 dan laporkan waktu yang dihabiskan untuk setiap klien.

```
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}."
    )
```

Kueri tabel untuk sejumlah iterasi untuk klien DAX maupun klien Boto3 dan laporkan waktu yang dihabiskan untuk setiap klien.

```
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}."
    )
```

Pindai tabel untuk sejumlah iterasi untuk klien DAX maupun klien Boto3 dan laporkan waktu yang dihabiskan untuk setiap klien.

```
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}."
    )
```

Hapus tabel tersebut.

```
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!")
```

- Untuk detail API, lihat topik berikut di Referensi API AWS SDK untuk Python (Boto3).
  - CreateTable
  - DeleteTable
  - GetItem
  - PutItem
  - Kueri
  - Scan

Untuk daftar lengkap panduan pengembang AWS SDK dan contoh kode, lihatMenggunakan DynamoDB dengan SDK AWS. Topik ini juga mencakup informasi tentang memulai dan detail tentang versi SDK sebelumnya.

Bekerja dengan skenario Indeks Sekunder Global DynamoDB tingkat lanjut menggunakan v2 AWS Command Line Interface

Contoh kode berikut menunjukkan cara bekerja dengan konfigurasi Global Secondary Index lanjutan.

- Buat tabel dengan banyak GSIs.
- Buat tabel dengan kapasitas sesuai permintaan dan GSI.
- Masukkan item ke dalam tabel dengan banyak GSIs.
- Kueri beberapa GSIs dengan kondisi berbeda.

Bash

AWS CLI dengan skrip Bash

Buat tabel dengan banyak 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\"]}
            }
        ]"
```

Buat tabel dengan kapasitas sesuai permintaan dan 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\"}
            }
        ]"
```

Masukkan item ke dalam tabel dengan banyak 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"}
    }'
```

Kueri item dari tabel dengan beberapa 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"}}'
```

- Untuk detail API, lihat topik berikut di Referensi Perintah AWS CLI .
  - CreateTable
  - PutItem
  - Kueri

Untuk daftar lengkap panduan pengembang AWS SDK dan contoh kode, lihatMenggunakan DynamoDB dengan SDK AWS. Topik ini juga mencakup informasi tentang memulai dan detail tentang versi SDK sebelumnya.

Membangun aplikasi untuk mengirimkan data ke tabel DynamoDB

Contoh kode berikut menunjukkan cara membangun aplikasi yang mengirimkan data ke tabel Amazon DynamoDB dan memberi tahu Anda saat pengguna memperbarui tabel.

Java

SDK untuk Java 2.x

Menunjukkan cara membuat aplikasi web dinamis yang mengirimkan data menggunakan API Java Amazon DynamoDB dan mengirim pesan teks menggunakan API Java Amazon Simple Notification Service.

Untuk kode sumber lengkap dan instruksi tentang cara mengatur dan menjalankan, lihat contoh lengkapnya di GitHub.

Layanan yang digunakan dalam contoh ini

- DynamoDB
- Amazon SNS

JavaScript

SDK untuk JavaScript (v3)

Contoh ini menunjukkan cara membangun aplikasi yang memungkinkan pengguna mengirimkan data ke tabel Amazon DynamoDB, dan mengirim pesan teks ke administrator menggunakan Amazon Simple Notification Service (Amazon SNS).

Untuk kode sumber lengkap dan instruksi tentang cara mengatur dan menjalankan, lihat contoh lengkapnya di GitHub.

Contoh ini juga tersedia di panduan developer AWS SDK untuk JavaScript v3.

Layanan yang digunakan dalam contoh ini

- DynamoDB
- Amazon SNS

Untuk daftar lengkap panduan pengembang AWS SDK dan contoh kode, lihatMenggunakan DynamoDB dengan SDK AWS. Topik ini juga mencakup informasi tentang memulai dan detail tentang versi SDK sebelumnya.

Bandingkan beberapa nilai dengan satu atribut di DynamoDB dengan SDK AWS

Contoh kode berikut menunjukkan bagaimana membandingkan beberapa nilai dengan atribut tunggal di DynamoDB.

- Gunakan operator IN untuk membandingkan beberapa nilai dengan satu atribut.
- Bandingkan operator IN dengan beberapa kondisi OR.
- Memahami manfaat kompleksitas kinerja dan ekspresi menggunakan IN.

Java

SDK untuk Java 2.x

Bandingkan beberapa nilai dengan atribut tunggal di DynamoDB menggunakan. 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;
    }
```

Contoh penggunaan membandingkan beberapa nilai dengan 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();
        }
    }
```

- Untuk detail API, lihat topik berikut di Referensi API AWS SDK for Java 2.x .
  - Kueri
  - Scan

JavaScript

SDK untuk JavaScript (v3)

Bandingkan beberapa nilai dengan satu atribut menggunakan AWS SDK untuk 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
    };
  }
}
```

Contoh penggunaan membandingkan beberapa nilai dengan AWS SDK untuk 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);
  }
}
```

- Untuk detail API, lihat topik berikut di Referensi API AWS SDK untuk JavaScript .
  - Kueri
  - Scan

Python

SDK untuk Python (Boto3)

Bandingkan beberapa nilai dengan satu atribut menggunakan AWS SDK untuk 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)}
```

Contoh penggunaan membandingkan beberapa nilai dengan AWS SDK untuk 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")
```

- Untuk detail API, lihat topik berikut di Referensi API AWS SDK untuk Python (Boto3).

- Kueri
- Scan

Untuk daftar lengkap panduan pengembang AWS SDK dan contoh kode, lihatMenggunakan DynamoDB dengan SDK AWS. Topik ini juga mencakup informasi tentang memulai dan detail tentang versi SDK sebelumnya.

Perbarui item DynamoDB secara kondisional dengan TTL menggunakan SDK AWS

Contoh kode berikut menunjukkan cara memperbarui TTL item secara kondisional.

Java

SDK untuk Java 2.x

Perbarui TTL pada Item DynamoDB yang ada dalam tabel, dengan kondisi.

```
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;
        }
    }
}
```

- Untuk detail API, lihat UpdateItemdi Referensi AWS SDK for Java 2.x API.

JavaScript

SDK untuk JavaScript (v3)

Perbarui TTL pada Item DynamoDB yang ada dalam tabel, dengan kondisi.

```
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');
```

- Untuk detail API, lihat UpdateItemdi Referensi AWS SDK untuk JavaScript API.

Python

SDK untuk Python (Boto3)

Perbarui TTL pada Item DynamoDB yang ada dalam tabel, dengan kondisi.

```
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",
)
```

- Untuk detail API, lihat UpdateItemdi AWS SDK for Python (Boto3) Referensi API.

Untuk daftar lengkap panduan pengembang AWS SDK dan contoh kode, lihatMenggunakan DynamoDB dengan SDK AWS. Topik ini juga mencakup informasi tentang memulai dan detail tentang versi SDK sebelumnya.

Connect ke instance DynamoDB lokal menggunakan SDK AWS

Contoh kode berikut menunjukkan cara mengganti URL endpoint untuk terhubung ke penyebaran pengembangan lokal DynamoDB dan SDK. AWS

Untuk informasi selengkapnya, lihat DynamoDB Local.

Rust

SDK untuk Rust

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
/// 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:?}"),
    }
}
```

Untuk daftar lengkap panduan pengembang AWS SDK dan contoh kode, lihatMenggunakan DynamoDB dengan SDK AWS. Topik ini juga mencakup informasi tentang memulai dan detail tentang versi SDK sebelumnya.

Hitung operator ekspresi di DynamoDB dengan SDK AWS

Contoh kode berikut menunjukkan cara menghitung operator ekspresi di DynamoDB.

- Memahami batas 300 operator DynamoDB.
- Hitung operator dalam ekspresi kompleks.
- Optimalkan ekspresi agar tetap dalam batas.

Java

SDK untuk Java 2.x

Menunjukkan penghitungan operator ekspresi menggunakan 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;
    }
```

Contoh penggunaan operator ekspresi menghitung dengan 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();
        }
    }
```

- Untuk detail API, lihat UpdateItemdi Referensi AWS SDK for Java 2.x API.

JavaScript

SDK untuk JavaScript (v3)

Menunjukkan penghitungan operator ekspresi menggunakan AWS SDK untuk 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;
}
```

Contoh penggunaan operator ekspresi menghitung dengan AWS SDK untuk 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);
  }
}
```

- Untuk detail API, lihat UpdateItemdi Referensi AWS SDK untuk JavaScript API.

Python

SDK untuk Python (Boto3)

Menunjukkan penghitungan operator ekspresi menggunakan AWS SDK untuk 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
```

Contoh penggunaan operator ekspresi menghitung dengan AWS SDK untuk 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")
```

- Untuk detail API, lihat UpdateItemdi AWS SDK for Python (Boto3) Referensi API.

Untuk daftar lengkap panduan pengembang AWS SDK dan contoh kode, lihatMenggunakan DynamoDB dengan SDK AWS. Topik ini juga mencakup informasi tentang memulai dan detail tentang versi SDK sebelumnya.

Membuat API REST Gateway API untuk melacak data COVID-19

Contoh kode berikut menunjukkan cara membuat API REST yang menyimulasikan sistem untuk melacak kasus COVID-19 harian di Amerika Serikat, menggunakan data fiksi.

Python

SDK untuk Python (Boto3)

Menunjukkan cara menggunakan AWS Chalice dengan membuat REST API tanpa server yang menggunakan Amazon API Gateway,, AWS Lambda dan Amazon DynamoDB. AWS SDK untuk Python (Boto3) API REST menyimulasikan sistem untuk melacak kasus COVID-19 harian di Amerika Serikat, menggunakan data fiksi. Pelajari cara:

- Gunakan AWS Chalice untuk menentukan rute dalam fungsi Lambda yang dipanggil untuk menangani permintaan REST yang datang melalui API Gateway.
- Menggunakan fungsi Lambda untuk mengambil dan menyimpan data dalam tabel DynamoDB untuk melayani permintaan REST.
- Tentukan struktur tabel dan sumber daya peran keamanan dalam AWS CloudFormation template.
- Gunakan AWS Chalice dan CloudFormation untuk mengemas dan menyebarkan semua sumber daya yang diperlukan.
- Gunakan CloudFormation untuk membersihkan semua sumber daya yang dibuat.

Untuk kode sumber lengkap dan instruksi tentang cara mengatur dan menjalankan, lihat contoh lengkapnya di GitHub.

Layanan yang digunakan dalam contoh ini

- API Gateway
- AWS CloudFormation
- DynamoDB
- Lambda

Untuk daftar lengkap panduan pengembang AWS SDK dan contoh kode, lihatMenggunakan DynamoDB dengan SDK AWS. Topik ini juga mencakup informasi tentang memulai dan detail tentang versi SDK sebelumnya.

Membuat aplikasi messenger dengan Step Functions

Contoh kode berikut menunjukkan cara membuat aplikasi AWS Step Functions messenger yang mengambil catatan pesan dari tabel database.

Python

SDK untuk Python (Boto3)

Menunjukkan cara menggunakan AWS SDK untuk Python (Boto3) with AWS Step Functions untuk membuat aplikasi messenger yang mengambil catatan pesan dari tabel Amazon DynamoDB dan mengirimkannya dengan Amazon Simple Queue Service (Amazon SQS). Mesin state terintegrasi dengan AWS Lambda fungsi untuk memindai database untuk pesan yang tidak terkirim.

- Buat mesin status yang mengambil dan memperbarui catatan pesan dari tabel Amazon DynamoDB.
- Perbarui definisi mesin status untuk mengirim pesan ke Amazon Simple Queue Service (Amazon SQS).
- Mulai dan hentikan berjalannya mesin status.
- Terhubung ke Lambda, DynamoDB, dan Amazon SQS dari mesin status menggunakan integrasi layanan.

Untuk kode sumber lengkap dan instruksi tentang cara mengatur dan menjalankan, lihat contoh lengkapnya di GitHub.

Layanan yang digunakan dalam contoh ini

- DynamoDB
- Lambda
- Amazon SQS
- Step Functions

Untuk daftar lengkap panduan pengembang AWS SDK dan contoh kode, lihatMenggunakan DynamoDB dengan SDK AWS. Topik ini juga mencakup informasi tentang memulai dan detail tentang versi SDK sebelumnya.

Membuat aplikasi manajemen aset foto yang memungkinkan pengguna mengelola foto menggunakan label

Contoh kode berikut ini menunjukkan cara membuat aplikasi nirserver yang memungkinkan pengguna mengelola foto menggunakan label.

.NET

SDK untuk .NET

Menunjukkan cara mengembangkan aplikasi manajemen aset foto yang mendeteksi label dalam gambar menggunakan Amazon Rekognition dan menyimpannya untuk pengambilan nanti.

Untuk kode sumber lengkap dan instruksi tentang cara mengatur dan menjalankan, lihat contoh lengkapnya di GitHub.

Untuk mendalami tentang asal usul contoh ini, lihat postingan di Komunitas AWS.

Layanan yang digunakan dalam contoh ini

- API Gateway
- DynamoDB
- Lambda
- Amazon Rekognition
- Amazon S3
- Amazon SNS

C++

SDK untuk C++

Menunjukkan cara mengembangkan aplikasi manajemen aset foto yang mendeteksi label dalam gambar menggunakan Amazon Rekognition dan menyimpannya untuk pengambilan nanti.

Untuk kode sumber lengkap dan instruksi tentang cara mengatur dan menjalankan, lihat contoh lengkapnya di GitHub.

Untuk mendalami tentang asal usul contoh ini, lihat postingan di Komunitas AWS.

Layanan yang digunakan dalam contoh ini

- API Gateway
- DynamoDB
- Lambda

- Amazon Rekognition
- Amazon S3
- Amazon SNS

Java

SDK untuk Java 2.x

Menunjukkan cara mengembangkan aplikasi manajemen aset foto yang mendeteksi label dalam gambar menggunakan Amazon Rekognition dan menyimpannya untuk pengambilan nanti.

Untuk kode sumber lengkap dan instruksi tentang cara mengatur dan menjalankan, lihat contoh lengkapnya di GitHub.

Untuk mendalami tentang asal usul contoh ini, lihat postingan di Komunitas AWS.

Layanan yang digunakan dalam contoh ini

- API Gateway
- DynamoDB
- Lambda
- Amazon Rekognition
- Amazon S3
- Amazon SNS

JavaScript

SDK untuk JavaScript (v3)

Menunjukkan cara mengembangkan aplikasi manajemen aset foto yang mendeteksi label dalam gambar menggunakan Amazon Rekognition dan menyimpannya untuk pengambilan nanti.

Untuk kode sumber lengkap dan instruksi tentang cara mengatur dan menjalankan, lihat contoh lengkapnya di GitHub.

Untuk mendalami tentang asal usul contoh ini, lihat postingan di Komunitas AWS.

Layanan yang digunakan dalam contoh ini

- API Gateway
- DynamoDB
- Lambda
- Amazon Rekognition
- Amazon S3
- Amazon SNS

Kotlin

SDK untuk Kotlin

Menunjukkan cara mengembangkan aplikasi manajemen aset foto yang mendeteksi label dalam gambar menggunakan Amazon Rekognition dan menyimpannya untuk pengambilan nanti.

Untuk kode sumber lengkap dan instruksi tentang cara mengatur dan menjalankan, lihat contoh lengkapnya di GitHub.

Untuk mendalami tentang asal usul contoh ini, lihat postingan di Komunitas AWS.

Layanan yang digunakan dalam contoh ini

- API Gateway
- DynamoDB
- Lambda
- Amazon Rekognition
- Amazon S3
- Amazon SNS

PHP

SDK untuk PHP

Menunjukkan cara mengembangkan aplikasi manajemen aset foto yang mendeteksi label dalam gambar menggunakan Amazon Rekognition dan menyimpannya untuk pengambilan nanti.

Untuk kode sumber lengkap dan instruksi tentang cara mengatur dan menjalankan, lihat contoh lengkapnya di GitHub.

Untuk mendalami tentang asal usul contoh ini, lihat postingan di Komunitas AWS.

Layanan yang digunakan dalam contoh ini

- API Gateway
- DynamoDB
- Lambda
- Amazon Rekognition
- Amazon S3
- Amazon SNS

Rust

SDK untuk Rust

Menunjukkan cara mengembangkan aplikasi manajemen aset foto yang mendeteksi label dalam gambar menggunakan Amazon Rekognition dan menyimpannya untuk pengambilan nanti.

Untuk kode sumber lengkap dan instruksi tentang cara mengatur dan menjalankan, lihat contoh lengkapnya di GitHub.

Untuk mendalami tentang asal usul contoh ini, lihat postingan di Komunitas AWS.

Layanan yang digunakan dalam contoh ini

- API Gateway
- DynamoDB
- Lambda
- Amazon Rekognition
- Amazon S3
- Amazon SNS

Untuk daftar lengkap panduan pengembang AWS SDK dan contoh kode, lihatMenggunakan DynamoDB dengan SDK AWS. Topik ini juga mencakup informasi tentang memulai dan detail tentang versi SDK sebelumnya.

Membuat tabel DynamoDB dengan Indeks Sekunder Global menggunakan SDK AWS

Contoh kode berikut menunjukkan cara membuat tabel dengan indeks sekunder global.

Java

SDK untuk Java 2.x

Buat tabel DynamoDB dengan Global Secondary Index menggunakan. 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();
        }
    }
```

- Untuk detail API, lihat CreateTabledi Referensi AWS SDK for Java 2.x API.

Untuk daftar lengkap panduan pengembang AWS SDK dan contoh kode, lihatMenggunakan DynamoDB dengan SDK AWS. Topik ini juga mencakup informasi tentang memulai dan detail tentang versi SDK sebelumnya.

Buat tabel DynamoDB dengan pengaturan throughput hangat menggunakan SDK AWS

Contoh kode berikut menunjukkan cara membuat tabel dengan throughput hangat diaktifkan.

Java

SDK untuk Java 2.x

Buat tabel DynamoDB dengan pengaturan throughput hangat menggunakan. AWS SDK for Java 2.x

```
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);
    }
```

- Untuk detail API, lihat CreateTabledi Referensi AWS SDK for Java 2.x API.

JavaScript

SDK untuk JavaScript (v3)

Buat tabel DynamoDB dengan pengaturan throughput hangat menggunakan. AWS SDK untuk 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
);
*/
```

- Untuk detail API, lihat CreateTabledi Referensi AWS SDK untuk JavaScript API.

Python

SDK untuk Python (Boto3)

Buat tabel DynamoDB dengan pengaturan throughput hangat menggunakan. AWS SDK untuk 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
```

- Untuk detail API, lihat CreateTabledi AWS SDK for Python (Boto3) Referensi API.

Untuk daftar lengkap panduan pengembang AWS SDK dan contoh kode, lihatMenggunakan DynamoDB dengan SDK AWS. Topik ini juga mencakup informasi tentang memulai dan detail tentang versi SDK sebelumnya.

Membuat aplikasi web untuk melacak data DynamoDB

Contoh-contoh kode berikut menunjukkan cara membuat aplikasi web yang melacak item kerja dalam tabel Amazon DynamoDB dan menggunakan Amazon Simple Email Service (Amazon SES) untuk mengirim laporan.

.NET

SDK untuk .NET

Menunjukkan cara menggunakan Amazon DynamoDB .NET API untuk membuat aplikasi web dinamis yang melacak data kerja DynamoDB.

Untuk kode sumber lengkap dan instruksi tentang cara mengatur dan menjalankan, lihat contoh lengkapnya di GitHub.

Layanan yang digunakan dalam contoh ini

- DynamoDB
- Amazon SES

Java

SDK untuk Java 2.x

Menunjukkan cara menggunakan Amazon DynamoDB API untuk membuat aplikasi web dinamis yang melacak data kerja DynamoDB.

Untuk kode sumber lengkap dan instruksi tentang cara mengatur dan menjalankan, lihat contoh lengkapnya di GitHub.

Layanan yang digunakan dalam contoh ini

- DynamoDB
- Amazon SES

Kotlin

SDK untuk Kotlin

Menunjukkan cara menggunakan Amazon DynamoDB API untuk membuat aplikasi web dinamis yang melacak data kerja DynamoDB.

Untuk kode sumber lengkap dan instruksi tentang cara mengatur dan menjalankan, lihat contoh lengkapnya di GitHub.

Layanan yang digunakan dalam contoh ini

- DynamoDB
- Amazon SES

Python

SDK untuk Python (Boto3)

Menunjukkan cara menggunakan AWS SDK untuk Python (Boto3) untuk membuat layanan REST yang melacak item kerja di Amazon DynamoDB dan laporan email dengan menggunakan Amazon Simple Email Service (Amazon SES). Contoh ini menggunakan rangka kerja web Flask untuk menangani perutean HTTP dan terintegrasi dengan halaman web React untuk menyajikan aplikasi web yang berfungsi penuh.

- Bangun layanan Flask REST yang terintegrasi dengan. Layanan AWS
- Baca, tulis, dan perbarui item kerja yang disimpan dalam tabel DynamoDB.
- Gunakan Amazon SES untuk mengirim laporan email tentang item pekerjaan.

Untuk kode sumber lengkap dan instruksi tentang cara mengatur dan menjalankan, lihat contoh lengkap di Repositori Contoh AWS Kode di. GitHub

Layanan yang digunakan dalam contoh ini

- DynamoDB

- Amazon SES

Untuk daftar lengkap panduan pengembang AWS SDK dan contoh kode, lihatMenggunakan DynamoDB dengan SDK AWS. Topik ini juga mencakup informasi tentang memulai dan detail tentang versi SDK sebelumnya.

Membuat aplikasi obrolan websocket dengan API Gateway

Contoh kode berikut menunjukkan cara membuat aplikasi obrolan yang dilayani oleh API websocket yang dibangun di Amazon API Gateway.

Python

SDK untuk Python (Boto3)

Menunjukkan cara menggunakan AWS SDK untuk Python (Boto3) dengan Amazon API Gateway V2 untuk membuat API websocket yang terintegrasi dengan AWS Lambda dan Amazon DynamoDB.

- Buat API websocket yang dilayani oleh API Gateway.
- Tentukan penangan Lambda yang menyimpan koneksi di DynamoDB dan memposting pesan ke peserta obrolan lainnya.
- Hubungkan ke aplikasi obrolan websocket dan kirim pesan dengan paket Websocket.

Untuk kode sumber lengkap dan instruksi tentang cara mengatur dan menjalankan, lihat contoh lengkapnya di GitHub.

Layanan yang digunakan dalam contoh ini

- API Gateway
- DynamoDB
- Lambda

Untuk daftar lengkap panduan pengembang AWS SDK dan contoh kode, lihatMenggunakan DynamoDB dengan SDK AWS. Topik ini juga mencakup informasi tentang memulai dan detail tentang versi SDK sebelumnya.

Buat item DynamoDB dengan TTL menggunakan SDK AWS

Contoh kode berikut menunjukkan cara membuat item dengan TTL.

Java

SDK untuk 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;
        }
    }
}
```

- Untuk detail API, lihat PutItemdi Referensi AWS SDK for Java 2.x API.

JavaScript

SDK untuk 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');
```

- Untuk detail API, lihat PutItemdi Referensi AWS SDK untuk JavaScript API.

Python

SDK untuk 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"
)
```

- Untuk detail API, lihat PutItemdi AWS SDK for Python (Boto3) Referensi API.

Untuk daftar lengkap panduan pengembang AWS SDK dan contoh kode, lihatMenggunakan DynamoDB dengan SDK AWS. Topik ini juga mencakup informasi tentang memulai dan detail tentang versi SDK sebelumnya.

Hapus data DynamoDB menggunakan pernyataan PartiQL DELETE dengan SDK AWS

Contoh kode berikut menunjukkan cara menghapus data menggunakan pernyataan PartiQL DELETE.

JavaScript

SDK untuk JavaScript (v3)

Hapus item dari tabel DynamoDB menggunakan pernyataan PartiQL DELETE dengan. AWS SDK untuk 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");
};
```

- Untuk detail API, lihat topik berikut di Referensi API AWS SDK untuk JavaScript .
  - BatchExecuteStatement
  - ExecuteStatement

Untuk daftar lengkap panduan pengembang AWS SDK dan contoh kode, lihatMenggunakan DynamoDB dengan SDK AWS. Topik ini juga mencakup informasi tentang memulai dan detail tentang versi SDK sebelumnya.

Mendeteksi APD dalam gambar dengan Amazon AWS Rekognition menggunakan SDK

Contoh kode berikut menunjukkan cara membuat aplikasi yang menggunakan Amazon Rekognition untuk mendeteksi Alat Pelindung Diri (APD) dalam gambar.

Java

SDK untuk Java 2.x

Menunjukkan cara membuat AWS Lambda fungsi yang mendeteksi gambar dengan Alat Pelindung Diri.

Untuk kode sumber lengkap dan instruksi tentang cara mengatur dan menjalankan, lihat contoh lengkapnya di GitHub.

Layanan yang digunakan dalam contoh ini

- DynamoDB
- Amazon Rekognition
- Amazon S3
- Amazon SES

Untuk daftar lengkap panduan pengembang AWS SDK dan contoh kode, lihatMenggunakan DynamoDB dengan SDK AWS. Topik ini juga mencakup informasi tentang memulai dan detail tentang versi SDK sebelumnya.

Masukkan data DynamoDB menggunakan pernyataan PartiQL INSERT dengan SDK AWS

Contoh kode berikut menunjukkan cara menyisipkan data menggunakan pernyataan PartiQL INSERT.

JavaScript

SDK untuk JavaScript (v3)

Masukkan item ke dalam tabel DynamoDB menggunakan pernyataan PartiQL INSERT dengan. AWS SDK untuk 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);
};
```

- Untuk detail API, lihat topik berikut di Referensi API AWS SDK untuk JavaScript .
  - BatchExecuteStatement
  - ExecuteStatement

Untuk daftar lengkap panduan pengembang AWS SDK dan contoh kode, lihatMenggunakan DynamoDB dengan SDK AWS. Topik ini juga mencakup informasi tentang memulai dan detail tentang versi SDK sebelumnya.

Menginvokasi fungsi Lambda dari browser

Contoh kode berikut menunjukkan cara memanggil AWS Lambda fungsi dari browser.

JavaScript

SDK untuk JavaScript (v2)

Anda dapat membuat aplikasi berbasis browser yang menggunakan AWS Lambda fungsi untuk memperbarui tabel Amazon DynamoDB dengan pilihan pengguna.

Untuk kode sumber lengkap dan instruksi tentang cara mengatur dan menjalankan, lihat contoh lengkapnya di GitHub.

Layanan yang digunakan dalam contoh ini

- DynamoDB
- Lambda

SDK untuk JavaScript (v3)

Anda dapat membuat aplikasi berbasis browser yang menggunakan AWS Lambda fungsi untuk memperbarui tabel Amazon DynamoDB dengan pilihan pengguna. Aplikasi ini menggunakan AWS SDK untuk JavaScript v3.

Untuk kode sumber lengkap dan instruksi tentang cara mengatur dan menjalankan, lihat contoh lengkapnya di GitHub.

Layanan yang digunakan dalam contoh ini

- DynamoDB
- Lambda

Untuk daftar lengkap panduan pengembang AWS SDK dan contoh kode, lihatMenggunakan DynamoDB dengan SDK AWS. Topik ini juga mencakup informasi tentang memulai dan detail tentang versi SDK sebelumnya.

Kelola Indeks Sekunder Global DynamoDB menggunakan v2 AWS Command Line Interface

Contoh kode berikut menunjukkan cara mengelola siklus hidup lengkap Global Secondary Indexes.

- Buat tabel dengan Indeks Sekunder Global.
- Tambahkan GSI baru ke tabel yang ada.
- Perbarui (tingkatkan) throughput hangat GSI.
- Data kueri menggunakan GSIs.
- Hapus GSI.

Bash

AWS CLI dengan skrip Bash

Buat tabel dengan Indeks Sekunder Global.

```
# 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}"
```

Tambahkan GSI baru (sesuai permintaan) ke tabel yang ada.

```
# 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\"}}}]"
```

Perbarui (tingkatkan) throughput hangat GSI.

```
# 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}}}]"
```

Data kueri menggunakan 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"}}'
```

Hapus GSI.

```
# Delete a GSI from a table
aws dynamodb update-table \
    --table-name MusicCollection \
    --global-secondary-index-updates \
        "[{\"Delete\":{\"IndexName\":\"GenreIndex\"}}]"
```

- Untuk detail API, lihat topik berikut di Referensi Perintah AWS CLI .
  - CreateTable

- DeleteTable
- Kueri
- UpdateTable

Untuk daftar lengkap panduan pengembang AWS SDK dan contoh kode, lihatMenggunakan DynamoDB dengan SDK AWS. Topik ini juga mencakup informasi tentang memulai dan detail tentang versi SDK sebelumnya.

Mengelola kebijakan berbasis sumber daya DynamoDB menggunakan v2 AWS Command Line Interface

Contoh kode berikut menunjukkan cara mengelola siklus hidup lengkap kebijakan berbasis sumber daya untuk tabel DynamoDB.

- Buat tabel dengan kebijakan sumber daya.
- Dapatkan kebijakan sumber daya.
- Perbarui kebijakan sumber daya.
- Hapus kebijakan sumber daya.

Bash

AWS CLI dengan skrip Bash

Buat tabel dengan kebijakan sumber daya.

```
# 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
```

Dapatkan kebijakan sumber daya.

```
# 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
```

Perbarui kebijakan sumber daya.

```
# 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
```

Hapus kebijakan sumber daya.

```
# Delete the resource-based policy from a table
aws dynamodb delete-resource-policy \
    --resource-arn arn:aws:dynamodb:us-west-2:123456789012:table/MusicCollection
```

- Untuk detail API, lihat topik berikut di Referensi Perintah AWS CLI .
  - CreateTable
  - DeleteResourcePolicy
  - GetResourcePolicy
  - PutResourcePolicy

Untuk daftar lengkap panduan pengembang AWS SDK dan contoh kode, lihatMenggunakan DynamoDB dengan SDK AWS. Topik ini juga mencakup informasi tentang memulai dan detail tentang versi SDK sebelumnya.

Memantau kinerja Amazon DynamoDB menggunakan SDK AWS

Contoh kode berikut menunjukkan cara mengkonfigurasi penggunaan aplikasi DynamoDB untuk memantau kinerja.

Java

SDK untuk Java 2.x

Contoh ini menunjukkan cara mengkonfigurasi aplikasi Java untuk memantau kinerja DynamoDB. Aplikasi mengirimkan data metrik ke CloudWatch tempat Anda dapat memantau kinerja.

Untuk kode sumber lengkap dan instruksi tentang cara mengatur dan menjalankan, lihat contoh lengkapnya di GitHub.

Layanan yang digunakan dalam contoh ini

- CloudWatch
- DynamoDB

Untuk daftar lengkap panduan pengembang AWS SDK dan contoh kode, lihatMenggunakan DynamoDB dengan SDK AWS. Topik ini juga mencakup informasi tentang memulai dan detail tentang versi SDK sebelumnya.

Melakukan operasi kueri DynamoDB lanjutan menggunakan SDK AWS

Contoh kode berikut menunjukkan bagaimana melakukan operasi query lanjutan di DynamoDB.

- Tabel kueri menggunakan berbagai teknik penyaringan dan kondisi.
- Menerapkan pagination untuk set hasil besar.
- Gunakan Global Secondary Indexes untuk pola akses alternatif.
- Menerapkan kontrol konsistensi berdasarkan persyaratan aplikasi.

Java

SDK untuk Java 2.x

Kueri dengan pembacaan yang sangat konsisten menggunakan 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;
        }
    }
```

Kueri menggunakan Indeks Sekunder Global dengan 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);
        }
```

```
    }
```

Kueri dengan pagination menggunakan 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;
        }
    }
```

Kueri dengan filter kompleks menggunakan 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);
    }
```

Kueri dengan ekspresi filter yang dibangun secara dinamis menggunakan 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);
        }
    }
```

Kueri dengan ekspresi filter dan batas penggunaan 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;
        }
    }
```

- Untuk detail API, lihat Kueri di Referensi API AWS SDK for Java 2.x .

JavaScript

SDK untuk JavaScript (v3)

Kueri dengan pembacaan yang sangat konsisten menggunakan AWS SDK untuk 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;
  }
}
```

Kueri menggunakan Indeks Sekunder Global dengan AWS SDK untuk 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;
  }
}
```

Kueri dengan pagination menggunakan AWS SDK untuk 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 };
```

Kueri dengan filter kompleks menggunakan AWS SDK untuk 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;
  }
}
```

Kueri dengan ekspresi filter yang dibangun secara dinamis menggunakan AWS SDK untuk 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;
  }
}
```

- Untuk detail API, lihat Kueri di Referensi API AWS SDK untuk JavaScript .

Python

SDK untuk Python (Boto3)

Kueri dengan pembacaan yang sangat konsisten menggunakan AWS SDK untuk 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
```

Kueri menggunakan Indeks Sekunder Global dengan AWS SDK untuk 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
```

Kueri dengan pagination menggunakan AWS SDK untuk 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
```

Kueri dengan filter kompleks menggunakan AWS SDK untuk 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
```

Kueri dengan ekspresi filter yang dibangun secara dinamis menggunakan AWS SDK untuk 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
```

Kueri dengan ekspresi filter dan batas penggunaan AWS SDK untuk 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
```

- Untuk detail API, lihat Kueri di Referensi API AWS SDK untuk Python (Boto3).

Untuk daftar lengkap panduan pengembang AWS SDK dan contoh kode, lihatMenggunakan DynamoDB dengan SDK AWS. Topik ini juga mencakup informasi tentang memulai dan detail tentang versi SDK sebelumnya.

Lakukan operasi daftar di DynamoDB dengan SDK AWS

Contoh kode berikut menunjukkan bagaimana melakukan operasi daftar di DynamoDB.

- Tambahkan elemen ke atribut daftar.
- Hapus elemen dari atribut daftar.
- Perbarui elemen tertentu dalam daftar berdasarkan indeks.

- Gunakan fungsi daftar append dan daftar indeks.

Java

SDK untuk Java 2.x

Menunjukkan operasi daftar menggunakan 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();
        }
    }
```

Contoh penggunaan operasi daftar dengan 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();
        }
    }
```

- Untuk detail API, lihat UpdateItemdi Referensi AWS SDK for Java 2.x API.

JavaScript

SDK untuk JavaScript (v3)

Menunjukkan operasi daftar menggunakan AWS SDK untuk 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;
}
```

Contoh penggunaan operasi daftar dengan AWS SDK untuk 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;
}
```

- Untuk detail API, lihat UpdateItemdi Referensi AWS SDK untuk JavaScript API.

Python

SDK untuk Python (Boto3)

Menunjukkan operasi daftar menggunakan AWS SDK untuk 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
```

Contoh penggunaan operasi daftar dengan AWS SDK untuk 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")
```

- Untuk detail API, lihat UpdateItemdi AWS SDK for Python (Boto3) Referensi API.

Untuk daftar lengkap panduan pengembang AWS SDK dan contoh kode, lihatMenggunakan DynamoDB dengan SDK AWS. Topik ini juga mencakup informasi tentang memulai dan detail tentang versi SDK sebelumnya.

Melakukan operasi peta di DynamoDB dengan SDK AWS

Contoh kode berikut menunjukkan bagaimana melakukan operasi peta di DynamoDB.

- Tambahkan dan perbarui atribut bersarang dalam struktur peta.
- Hapus bidang tertentu dari peta.
- Bekerja dengan atribut peta yang sangat bersarang.

Java

SDK untuk Java 2.x

Menunjukkan operasi peta menggunakan 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();
        }
    }
```

Contoh penggunaan operasi peta dengan 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();
        }
    }
```

- Untuk detail API, lihat UpdateItemdi Referensi AWS SDK for Java 2.x API.

JavaScript

SDK untuk JavaScript (v3)

Menunjukkan operasi peta menggunakan AWS SDK untuk 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();
}
```

- Untuk detail API, lihat UpdateItemdi Referensi AWS SDK untuk JavaScript API.

Python

SDK untuk Python (Boto3)

Menunjukkan operasi peta menggunakan AWS SDK untuk 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()
```

- Untuk detail API, lihat UpdateItemdi AWS SDK for Python (Boto3) Referensi API.

Untuk daftar lengkap panduan pengembang AWS SDK dan contoh kode, lihatMenggunakan DynamoDB dengan SDK AWS. Topik ini juga mencakup informasi tentang memulai dan detail tentang versi SDK sebelumnya.

Lakukan operasi set di DynamoDB dengan SDK AWS

Contoh kode berikut menunjukkan bagaimana melakukan operasi set di DynamoDB.

- Tambahkan elemen ke atribut set.
- Hapus elemen dari atribut set.
- Gunakan operasi ADD dan DELETE dengan set.

Java

SDK untuk Java 2.x

Menunjukkan operasi set menggunakan 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();
        }
    }
```

Contoh penggunaan operasi set dengan 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();
        }
    }
```

- Untuk detail API, lihat UpdateItemdi Referensi AWS SDK for Java 2.x API.

JavaScript

SDK untuk JavaScript (v3)

Menunjukkan operasi set menggunakan AWS SDK untuk 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;
}
```

Contoh penggunaan operasi set dengan AWS SDK untuk 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);
  }
}
```

- Untuk detail API, lihat UpdateItemdi Referensi AWS SDK untuk JavaScript API.

Python

SDK untuk Python (Boto3)

Menunjukkan operasi set menggunakan AWS SDK untuk 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", {}),
    }
```

Contoh penggunaan operasi set dengan AWS SDK untuk 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")
```

- Untuk detail API, lihat UpdateItemdi AWS SDK for Python (Boto3) Referensi API.

Untuk daftar lengkap panduan pengembang AWS SDK dan contoh kode, lihatMenggunakan DynamoDB dengan SDK AWS. Topik ini juga mencakup informasi tentang memulai dan detail tentang versi SDK sebelumnya.

Kueri tabel DynamoDB dengan menggunakan kumpulan pernyataan PartiQL dan SDK AWS

Contoh-contoh kode berikut menunjukkan cara:

- Dapatkan batch item dengan menjalankan beberapa pernyataan SELECT.

- Tambahkan batch item dengan menjalankan beberapa pernyataan INSERT.
- Perbarui batch item dengan menjalankan beberapa pernyataan UPDATE.
- Hapus batch item dengan menjalankan beberapa pernyataan DELETE.

.NET

SDK untuk .NET

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
// 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;
        }
```

- Untuk detail API, lihat BatchExecuteStatementdi Referensi AWS SDK untuk .NET API.

C++

SDK untuk C++

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
        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;
}
```

- Untuk detail API, lihat BatchExecuteStatementdi Referensi AWS SDK untuk C++ API.

Go

SDK untuk Go V2

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

Jalankan skenario yang membuat tabel dan menjalankan batch kueri PartiQL.

```
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))
}
```

Tentukan struct Movie yang digunakan dalam contoh ini.

```
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"])
}
```

Buat struct dan metode yang menjalankan pernyataan PartiQL.

```
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
}
```

- Untuk detail API, lihat BatchExecuteStatementdi Referensi AWS SDK untuk Go API.

Java

SDK untuk Java 2.x

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
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);
    }
}
```

- Untuk detail API, lihat BatchExecuteStatementdi Referensi AWS SDK for Java 2.x API.

JavaScript

SDK untuk JavaScript (v3)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

Jalankan pernyataan PartiQL batch.

```
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.");
};
```

- Untuk detail API, lihat BatchExecuteStatementdi Referensi AWS SDK untuk JavaScript API.

Kotlin

SDK untuk Kotlin

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
suspend fun main() {
    val ddb = DynamoDbClient { 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")
    }
}
```

- Untuk detail API, lihat BatchExecuteStatementdi AWS SDK untuk referensi API Kotlin.

PHP

SDK untuk PHP

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
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,
                ],
            ],
        ]);
    }
```

- Untuk detail API, lihat BatchExecuteStatementdi Referensi AWS SDK untuk PHP API.

Python

SDK untuk Python (Boto3)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

Buat kelas yang dapat menjalankan batch pernyataan PartiQL.

```
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
```

Jalankan skenario yang membuat tabel dan menjalankan kueri PartiQL dalam batch.

```
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}")
```

- Untuk detail API, lihat BatchExecuteStatementdi AWS SDK for Python (Boto3) Referensi API.

Ruby

SDK untuk Ruby

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

Jalankan skenario yang membuat tabel dan menjalankan batch kueri PartiQL.

```
  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
```

- Untuk detail API, lihat BatchExecuteStatementdi Referensi AWS SDK untuk Ruby API.

Untuk daftar lengkap panduan pengembang AWS SDK dan contoh kode, lihatMenggunakan DynamoDB dengan SDK AWS. Topik ini juga mencakup informasi tentang memulai dan detail tentang versi SDK sebelumnya.

Kueri tabel DynamoDB menggunakan PartiQL dan SDK AWS

Contoh-contoh kode berikut menunjukkan cara:

- Dapatkan item dengan menjalankan pernyataan SELECT.
- Tambahkan item dengan menjalankan pernyataan INSERT.
- Perbarui item dengan menjalankan pernyataan UPDATE.
- Hapus item dengan menjalankan pernyataan DELETE.

.NET

SDK untuk .NET

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
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;
        }
```

- Untuk detail API, lihat ExecuteStatementdi Referensi AWS SDK untuk .NET API.

C++

SDK untuk C++

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
        //  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;
}
```

- Untuk detail API, lihat ExecuteStatementdi Referensi AWS SDK untuk C++ API.

Go

SDK untuk Go V2

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

Jalankan skenario yang membuat tabel dan menjalankan kueri PartiQL.

```
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))
}
```

Tentukan struct Movie yang digunakan dalam contoh ini.

```
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"])
}
```

Buat struct dan metode yang menjalankan pernyataan PartiQL.

```
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
}
```

- Untuk detail API, lihat ExecuteStatementdi Referensi AWS SDK untuk Go API.

Java

SDK untuk Java 2.x

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
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());
    }
}
```

- Untuk detail API, lihat ExecuteStatementdi Referensi AWS SDK for Java 2.x API.

JavaScript

SDK untuk JavaScript (v3)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

Jalankan pernyataan PartiQL tunggal.

```
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.");
};
```

- Untuk detail API, lihat ExecuteStatementdi Referensi AWS SDK untuk JavaScript API.

Kotlin

SDK untuk Kotlin

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
suspend fun main() {
    val ddb = DynamoDbClient { 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)
}
```

- Untuk detail API, lihat ExecuteStatementdi AWS SDK untuk referensi API Kotlin.

PHP

SDK untuk PHP

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
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,
```

```
        ]);
    }
```

- Untuk detail API, lihat ExecuteStatementdi Referensi AWS SDK untuk PHP API.

Python

SDK untuk Python (Boto3)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

Buat kelas yang dapat menjalankan pernyataan PartiQL.

```
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
```

Jalankan skenario yang membuat tabel dan menjalankan kueri PartiQL.

```
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}")
```

- Untuk detail API, lihat ExecuteStatementdi AWS SDK for Python (Boto3) Referensi API.

Ruby

SDK untuk Ruby

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

Jalankan skenario yang membuat tabel dan menjalankan kueri PartiQL.

```
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
```

- Untuk detail API, lihat ExecuteStatementdi Referensi AWS SDK untuk Ruby API.

Rust

SDK untuk Rust

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
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(())
}
```

- Untuk detail API, lihat ExecuteStatementreferensi AWS SDK for Rust API.

Untuk daftar lengkap panduan pengembang AWS SDK dan contoh kode, lihatMenggunakan DynamoDB dengan SDK AWS. Topik ini juga mencakup informasi tentang memulai dan detail tentang versi SDK sebelumnya.

Kueri tabel DynamoDB menggunakan Global Secondary Index dengan SDK AWS

Contoh kode berikut menunjukkan cara query tabel menggunakan Global Secondary Index.

- Kueri tabel DynamoDB menggunakan kunci utamanya.
- Kueri Indeks Sekunder Global (GSI) untuk pola akses alternatif.
- Bandingkan kueri tabel dan kueri GSI.

Java

SDK untuk Java 2.x

Kueri tabel DynamoDB menggunakan kunci utama dan Indeks Sekunder Global (GSI) dengan. 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);
        }
    }
```

Bandingkan kueri tabel secara langsung versus menanyakan GSI dengan. 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);
        }
    }
```

- Untuk detail API, lihat Kueri di Referensi API AWS SDK for Java 2.x .

JavaScript

SDK untuk JavaScript (v3)

Kueri tabel DynamoDB menggunakan kunci utama dengan. AWS SDK untuk 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;
  }
}
```

Kueri Indeks Sekunder Global DynamoDB (GSI) dengan. AWS SDK untuk 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;
  }
}
```

- Untuk detail API, lihat Kueri di Referensi API AWS SDK untuk JavaScript .

Python

SDK untuk Python (Boto3)

Kueri tabel DynamoDB menggunakan kunci utama dan Indeks Sekunder Global (GSI) dengan. AWS SDK untuk 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
```

- Untuk detail API, lihat Kueri di Referensi API AWS SDK untuk Python (Boto3).

Untuk daftar lengkap panduan pengembang AWS SDK dan contoh kode, lihatMenggunakan DynamoDB dengan SDK AWS. Topik ini juga mencakup informasi tentang memulai dan detail tentang versi SDK sebelumnya.

Kueri tabel DynamoDB menggunakan kondisi begins_with dengan SDK AWS

Contoh kode berikut menunjukkan bagaimana untuk query tabel menggunakan kondisi begins_with.

- Gunakan fungsi begins_with dalam ekspresi kondisi kunci.
- Filter item berdasarkan pola awalan di tombol sortir.

Java

SDK untuk Java 2.x

Kueri tabel DynamoDB menggunakan kondisi begins_with pada kunci pengurutan dengan. 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 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;
```

```
        }
    }
```

Peragakan menggunakan begins_with dengan panjang awalan yang berbeda dengan. AWS SDK for Java 2.x

```
    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);
        }
    }
```

- Untuk detail API, lihat Kueri di Referensi API AWS SDK for Java 2.x .

JavaScript

SDK untuk JavaScript (v3)

Kueri tabel DynamoDB menggunakan kondisi begins_with pada kunci pengurutan dengan. AWS SDK untuk JavaScript

```
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;
  }
}
```

- Untuk detail API, lihat Kueri di Referensi API AWS SDK untuk JavaScript .

Python

SDK untuk Python (Boto3)

Kueri tabel DynamoDB menggunakan kondisi begins_with pada kunci pengurutan dengan. AWS SDK untuk Python (Boto3)

```
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
```

- Untuk detail API, lihat Kueri di Referensi API AWS SDK untuk Python (Boto3).

Untuk daftar lengkap panduan pengembang AWS SDK dan contoh kode, lihatMenggunakan DynamoDB dengan SDK AWS. Topik ini juga mencakup informasi tentang memulai dan detail tentang versi SDK sebelumnya.

Kueri tabel DynamoDB menggunakan rentang tanggal dalam kunci pengurutan dengan SDK AWS

Contoh kode berikut menunjukkan bagaimana untuk query tabel menggunakan rentang tanggal dalam kunci sort.

- Item kueri dalam rentang tanggal tertentu.
- Gunakan operator perbandingan pada kunci pengurutan berformat tanggal.

Java

SDK untuk Java 2.x

Kueri tabel DynamoDB untuk item dalam rentang tanggal dengan. 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;
        }
    }
```

Menunjukkan bagaimana untuk query tabel DynamoDB dengan rentang tanggal penyaringan.

```
    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);
        }
    }
```

- Untuk detail API, lihat Kueri di Referensi API AWS SDK for Java 2.x .

JavaScript

SDK untuk JavaScript (v3)

Kueri tabel DynamoDB untuk item dalam rentang tanggal dengan. AWS SDK untuk 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;
  }
}
```

- Untuk detail API, lihat Kueri di Referensi API AWS SDK untuk JavaScript .

Python

SDK untuk Python (Boto3)

Kueri tabel DynamoDB untuk item dalam rentang tanggal dengan. AWS SDK untuk 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
```

- Untuk detail API, lihat Kueri di Referensi API AWS SDK untuk Python (Boto3).

Untuk daftar lengkap panduan pengembang AWS SDK dan contoh kode, lihatMenggunakan DynamoDB dengan SDK AWS. Topik ini juga mencakup informasi tentang memulai dan detail tentang versi SDK sebelumnya.

Kueri tabel DynamoDB dengan ekspresi filter kompleks dengan SDK AWS

Contoh kode berikut menunjukkan bagaimana untuk query tabel dengan ekspresi filter kompleks.

- Terapkan ekspresi filter yang kompleks ke hasil kueri.
- Gabungkan beberapa kondisi menggunakan operator logis.
- Filter item berdasarkan atribut non-kunci.

Java

SDK untuk Java 2.x

Kueri tabel DynamoDB dengan ekspresi filter kompleks menggunakan. 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);
    }
```

- Untuk detail API, lihat Kueri di Referensi API AWS SDK for Java 2.x .

JavaScript

SDK untuk JavaScript (v3)

Kueri tabel DynamoDB dengan ekspresi filter kompleks menggunakan. AWS SDK untuk 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;
  }
}
```

- Untuk detail API, lihat Kueri di Referensi API AWS SDK untuk JavaScript .

Python

SDK untuk Python (Boto3)

Kueri tabel DynamoDB dengan ekspresi filter kompleks menggunakan. AWS SDK untuk 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
```

- Untuk detail API, lihat Kueri di Referensi API AWS SDK untuk Python (Boto3).

Untuk daftar lengkap panduan pengembang AWS SDK dan contoh kode, lihatMenggunakan DynamoDB dengan SDK AWS. Topik ini juga mencakup informasi tentang memulai dan detail tentang versi SDK sebelumnya.

Kueri tabel DynamoDB dengan ekspresi filter dinamis dengan SDK AWS

Contoh kode berikut menunjukkan bagaimana untuk query tabel dengan ekspresi filter dinamis.

- Buat ekspresi filter secara dinamis saat runtime.
- Membangun kondisi filter berdasarkan input pengguna atau status aplikasi.

- Tambahkan atau hapus kriteria filter secara kondisional.

Java

SDK untuk Java 2.x

Kueri tabel DynamoDB dengan ekspresi filter yang dibangun secara dinamis menggunakan. 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();
            }
        }
    }
```

Menunjukkan cara menggunakan ekspresi filter dinamis dengan 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);
        }
    }
```

- Untuk detail API, lihat Kueri di Referensi API AWS SDK for Java 2.x .

JavaScript

SDK untuk JavaScript (v3)

Kueri tabel DynamoDB dengan ekspresi filter yang dibangun secara dinamis menggunakan. AWS SDK untuk 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;
  }
}
```

- Untuk detail API, lihat Kueri di Referensi API AWS SDK untuk JavaScript .

Python

SDK untuk Python (Boto3)

Kueri tabel DynamoDB dengan ekspresi filter yang dibangun secara dinamis menggunakan. AWS SDK untuk 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
```

Menunjukkan cara menggunakan ekspresi filter dinamis dengan AWS SDK untuk 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}")
```

- Untuk detail API, lihat Kueri di Referensi API AWS SDK untuk Python (Boto3).

Untuk daftar lengkap panduan pengembang AWS SDK dan contoh kode, lihatMenggunakan DynamoDB dengan SDK AWS. Topik ini juga mencakup informasi tentang memulai dan detail tentang versi SDK sebelumnya.

Kueri tabel DynamoDB dengan ekspresi filter dan batas dengan SDK AWS

Contoh kode berikut menunjukkan bagaimana untuk query tabel dengan ekspresi filter dan batas.

- Terapkan ekspresi filter ke hasil kueri dengan batas item yang dievaluasi.
- Pahami bagaimana batas memengaruhi hasil kueri yang difilter.
- Kontrol jumlah maksimum item yang diproses dalam kueri.

Java

SDK untuk Java 2.x

Kueri tabel DynamoDB dengan ekspresi filter dan batas penggunaan. 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;
        }
    }
```

- Untuk detail API, lihat Kueri di Referensi API AWS SDK for Java 2.x .

Python

SDK untuk Python (Boto3)

Kueri tabel DynamoDB dengan ekspresi filter dan batas penggunaan. AWS SDK untuk 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
```

Menunjukkan cara menggunakan ekspresi filter dengan batas di AWS SDK untuk 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.")
```

- Untuk detail API, lihat Kueri di Referensi API AWS SDK untuk Python (Boto3).

Untuk daftar lengkap panduan pengembang AWS SDK dan contoh kode, lihatMenggunakan DynamoDB dengan SDK AWS. Topik ini juga mencakup informasi tentang memulai dan detail tentang versi SDK sebelumnya.

Kueri tabel DynamoDB dengan atribut bersarang menggunakan SDK AWS

Contoh kode berikut menunjukkan bagaimana untuk query tabel dengan atribut bersarang.

- Akses dan filter berdasarkan atribut bersarang di item DynamoDB.
- Gunakan ekspresi jalur dokumen untuk mereferensikan elemen bersarang.

Java

SDK untuk Java 2.x

Kueri tabel DynamoDB dengan atribut bersarang menggunakan. 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;
        }
    }
```

Menunjukkan bagaimana untuk query tabel DynamoDB dengan atribut bersarang.

```
    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);
        }
    }
```

- Untuk detail API, lihat Kueri di Referensi API AWS SDK for Java 2.x .

JavaScript

SDK untuk JavaScript (v3)

Kueri tabel DynamoDB dengan atribut bersarang menggunakan. AWS SDK untuk 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;
  }
}
```

- Untuk detail API, lihat Kueri di Referensi API AWS SDK untuk JavaScript .

Python

SDK untuk Python (Boto3)

Kueri tabel DynamoDB dengan atribut bersarang menggunakan. AWS SDK untuk 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
```

- Untuk detail API, lihat Kueri di Referensi API AWS SDK untuk Python (Boto3).

Untuk daftar lengkap panduan pengembang AWS SDK dan contoh kode, lihatMenggunakan DynamoDB dengan SDK AWS. Topik ini juga mencakup informasi tentang memulai dan detail tentang versi SDK sebelumnya.

Kueri tabel DynamoDB dengan pagination menggunakan SDK AWS

Contoh kode berikut menunjukkan bagaimana untuk query tabel dengan pagination.

- Menerapkan pagination untuk hasil query DynamoDB.
- Gunakan LastEvaluatedKey untuk mengambil halaman berikutnya.
- Kontrol jumlah item per halaman dengan parameter Limit.

Java

SDK untuk Java 2.x

Kueri tabel DynamoDB dengan pagination menggunakan. 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;
        }
    }
```

Menunjukkan bagaimana untuk query tabel DynamoDB dengan pagination.

```
    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);
        }
    }
```

- Untuk detail API, lihat Kueri di Referensi API AWS SDK for Java 2.x .

JavaScript

SDK untuk JavaScript (v3)

Kueri tabel DynamoDB dengan pagination menggunakan. AWS SDK untuk 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 };
```

- Untuk detail API, lihat Kueri di Referensi API AWS SDK untuk JavaScript .

Python

SDK untuk Python (Boto3)

Kueri tabel DynamoDB dengan pagination menggunakan. AWS SDK untuk 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
```

- Untuk detail API, lihat Kueri di Referensi API AWS SDK untuk Python (Boto3).

Untuk daftar lengkap panduan pengembang AWS SDK dan contoh kode, lihatMenggunakan DynamoDB dengan SDK AWS. Topik ini juga mencakup informasi tentang memulai dan detail tentang versi SDK sebelumnya.

Kueri tabel DynamoDB dengan pembacaan yang sangat konsisten menggunakan SDK AWS

Contoh kode berikut menunjukkan cara menanyakan tabel dengan pembacaan yang sangat konsisten.

- Konfigurasikan tingkat konsistensi untuk query DynamoDB.
- Gunakan pembacaan yang sangat konsisten untuk mendapatkan up-to-date data terbanyak.
- Memahami pengorbanan antara konsistensi akhirnya dan konsistensi yang kuat.

Java

SDK untuk Java 2.x

Kueri tabel DynamoDB dengan konsistensi baca yang dapat dikonfigurasi menggunakan. 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;
        }
    }
```

- Untuk detail API, lihat Kueri di Referensi API AWS SDK for Java 2.x .

JavaScript

SDK untuk JavaScript (v3)

Kueri tabel DynamoDB dengan konsistensi baca yang dapat dikonfigurasi menggunakan. AWS SDK untuk 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;
  }
```

```
}
```

- Untuk detail API, lihat Kueri di Referensi API AWS SDK untuk JavaScript .

Python

SDK untuk Python (Boto3)

Kueri tabel DynamoDB dengan opsi untuk pembacaan yang sangat konsisten menggunakan. AWS SDK untuk 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
```

- Untuk detail API, lihat Kueri di Referensi API AWS SDK untuk Python (Boto3).

Untuk daftar lengkap panduan pengembang AWS SDK dan contoh kode, lihatMenggunakan DynamoDB dengan SDK AWS. Topik ini juga mencakup informasi tentang memulai dan detail tentang versi SDK sebelumnya.

Kueri data DynamoDB menggunakan pernyataan PartiQL SELECT dengan SDK AWS

Contoh kode berikut menunjukkan bagaimana untuk query data menggunakan pernyataan PartiQL SELECT.

JavaScript

SDK untuk JavaScript (v3)

Item kueri dari tabel DynamoDB menggunakan pernyataan Partiql SELECT dengan. AWS SDK untuk 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");
};
```

- Untuk detail API, lihat topik berikut di Referensi API AWS SDK untuk JavaScript .
  - BatchExecuteStatement
  - ExecuteStatement

Untuk daftar lengkap panduan pengembang AWS SDK dan contoh kode, lihatMenggunakan DynamoDB dengan SDK AWS. Topik ini juga mencakup informasi tentang memulai dan detail tentang versi SDK sebelumnya.

Kueri tabel DynamoDB untuk item TTL menggunakan SDK AWS

Contoh kode berikut menunjukkan cara query untuk item TTL.

Java

SDK untuk Java 2.x

Query Filtered Expression untuk mengumpulkan item TTL dalam tabel DynamoDB menggunakan. 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;
        }
```

- Untuk detail API, lihat Kueri di Referensi API AWS SDK for Java 2.x .

JavaScript

SDK untuk JavaScript (v3)

Query Filtered Expression untuk mengumpulkan item TTL dalam tabel DynamoDB menggunakan. AWS SDK untuk 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');
```

- Untuk detail API, lihat Kueri di Referensi API AWS SDK untuk JavaScript .

Python

SDK untuk Python (Boto3)

Query Filtered Expression untuk mengumpulkan item TTL dalam tabel DynamoDB menggunakan. AWS SDK untuk 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")
```

- Untuk detail API, lihat Kueri di Referensi API AWS SDK untuk Python (Boto3).

Untuk daftar lengkap panduan pengembang AWS SDK dan contoh kode, lihatMenggunakan DynamoDB dengan SDK AWS. Topik ini juga mencakup informasi tentang memulai dan detail tentang versi SDK sebelumnya.

Kueri tabel DynamoDB menggunakan pola tanggal dan waktu dengan SDK AWS

Contoh kode berikut menunjukkan cara menanyakan tabel menggunakan pola tanggal dan waktu.

- Simpan dan kueri date/time nilai di DynamoDB.
- Menerapkan kueri rentang tanggal menggunakan kunci pengurutan.
- Format string tanggal untuk kueri yang efektif.

Java

SDK untuk Java 2.x

Kueri menggunakan rentang tanggal dalam kunci pengurutan dengan 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;
        }
    }
```

Kueri menggunakan variabel tanggal-waktu dengan. 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;
        }
    }
```

Kueri dalam rentang tanggal dalam stempel waktu zaman Unix dengan. 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 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;
        }
    }
```

Kueri dalam rentang tanggal menggunakan LocalDateTime objek dengan 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);
    }
```

- Untuk detail API, lihat Kueri di Referensi API AWS SDK for Java 2.x .

JavaScript

SDK untuk JavaScript (v3)

Kueri menggunakan rentang tanggal dalam kunci pengurutan dengan AWS SDK untuk 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;
  }
}
```

Kueri menggunakan variabel tanggal-waktu dengan. AWS SDK untuk 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;
  }
}
```

- Untuk detail API, lihat Kueri di Referensi API AWS SDK untuk JavaScript .

Python

SDK untuk Python (Boto3)

Kueri menggunakan rentang tanggal dalam kunci pengurutan dengan AWS SDK untuk 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
```

Kueri menggunakan variabel tanggal-waktu dengan. AWS SDK untuk 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}")
```

- Untuk detail API, lihat Kueri di Referensi API AWS SDK untuk Python (Boto3).

Untuk daftar lengkap panduan pengembang AWS SDK dan contoh kode, lihatMenggunakan DynamoDB dengan SDK AWS. Topik ini juga mencakup informasi tentang memulai dan detail tentang versi SDK sebelumnya.

Simpan EXIF dan informasi gambar lainnya menggunakan SDK AWS

Contoh kode berikut ini menunjukkan cara:

- Mendapatkan informasi EXIF dari file JPG, JPEG, atau PNG.
- Mengunggah file gambar ke bucket Amazon S3.
- Menggunakan Amazon Rekognition untuk mengidentifikasi tiga atribut teratas (label) dalam file.
- Menambahkan informasi EXIF dan label ke tabel Amazon DynamoDB di Wilayah.

Rust

SDK untuk Rust

Mendapatkan informasi EXIF dari file JPG, JPEG, atau PNG, mengunggah file gambar ke bucket Amazon S3, menggunakan Amazon Rekognition untuk mengidentifikasi tiga atribut teratas (label di Amazon Rekognition) dalam file, dan menambahkan EXIF dan informasi label ke tabel Amazon DynamoDB di Wilayah.

Untuk kode sumber lengkap dan instruksi tentang cara mengatur dan menjalankan, lihat contoh lengkapnya di GitHub.

Layanan yang digunakan dalam contoh ini

- DynamoDB
- Amazon Rekognition
- Amazon S3

Untuk daftar lengkap panduan pengembang AWS SDK dan contoh kode, lihatMenggunakan DynamoDB dengan SDK AWS. Topik ini juga mencakup informasi tentang memulai dan detail tentang versi SDK sebelumnya.

Mengatur Kontrol Akses Berbasis Atribut untuk DynamoDB menggunakan v2 AWS Command Line Interface

Contoh kode berikut menunjukkan bagaimana menerapkan Attribute-Based Access Control (ABAC) untuk DynamoDB.

- Buat kebijakan IAM untuk ABAC.
- Buat tabel dengan tag untuk departemen yang berbeda.
- Daftar dan filter tabel berdasarkan tag.

Bash

AWS CLI dengan skrip Bash

Buat kebijakan IAM untuk 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
```

Buat tabel dengan tag untuk departemen yang berbeda.

```
# 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
```

Daftar dan filter tabel berdasarkan tag.

```
# 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
```

- Untuk detail API, lihat topik berikut di Referensi Perintah AWS CLI .
  - CreatePolicy
  - CreateTable
  - ListTables

Untuk daftar lengkap panduan pengembang AWS SDK dan contoh kode, lihatMenggunakan DynamoDB dengan SDK AWS. Topik ini juga mencakup informasi tentang memulai dan detail tentang versi SDK sebelumnya.

Memahami urutan ekspresi pembaruan di DynamoDB dengan SDK AWS

Contoh kode berikut menunjukkan cara memahami urutan ekspresi pembaruan.

- Pelajari cara DynamoDB memproses ekspresi pembaruan.
- Memahami urutan operasi dalam ekspresi pembaruan.
- Hindari hasil yang tidak terduga dengan memahami evaluasi ekspresi.

Java

SDK untuk Java 2.x

Menunjukkan urutan ekspresi pembaruan menggunakan 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();
        }
    }
```

Contoh penggunaan urutan ekspresi pembaruan dengan 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();
        }
    }
```

- Untuk detail API, lihat UpdateItemdi Referensi AWS SDK for Java 2.x API.

JavaScript

SDK untuk JavaScript (v3)

Menunjukkan urutan ekspresi pembaruan menggunakan AWS SDK untuk 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
  };
}
```

Contoh penggunaan urutan ekspresi pembaruan dengan AWS SDK untuk 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);
  }
}
```

- Untuk detail API, lihat UpdateItemdi Referensi AWS SDK untuk JavaScript API.

Python

SDK untuk Python (Boto3)

Menunjukkan urutan ekspresi pembaruan menggunakan AWS SDK untuk 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
```

Contoh penggunaan urutan ekspresi pembaruan dengan AWS SDK untuk 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")
```

- Untuk detail API, lihat UpdateItemdi AWS SDK for Python (Boto3) Referensi API.

Untuk daftar lengkap panduan pengembang AWS SDK dan contoh kode, lihatMenggunakan DynamoDB dengan SDK AWS. Topik ini juga mencakup informasi tentang memulai dan detail tentang versi SDK sebelumnya.

Memperbarui setelan tabel DynamoDB dengan throughput hangat menggunakan SDK AWS

Contoh kode berikut menunjukkan cara memperbarui pengaturan throughput hangat tabel.

Java

SDK untuk Java 2.x

Perbarui pengaturan throughput hangat pada tabel DynamoDB yang ada menggunakan. AWS SDK for Java 2.x

```
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);
    }
```

- Untuk detail API, lihat UpdateTabledi Referensi AWS SDK for Java 2.x API.

JavaScript

SDK untuk JavaScript (v3)

Perbarui pengaturan throughput hangat pada tabel DynamoDB yang ada menggunakan. AWS SDK untuk JavaScript

```
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
);
*/
```

- Untuk detail API, lihat UpdateTabledi Referensi AWS SDK untuk JavaScript API.

Python

SDK untuk Python (Boto3)

Perbarui pengaturan throughput hangat pada tabel DynamoDB yang ada menggunakan. AWS SDK untuk Python (Boto3)

```
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
```

- Untuk detail API, lihat UpdateTabledi AWS SDK for Python (Boto3) Referensi API.

Untuk daftar lengkap panduan pengembang AWS SDK dan contoh kode, lihatMenggunakan DynamoDB dengan SDK AWS. Topik ini juga mencakup informasi tentang memulai dan detail tentang versi SDK sebelumnya.

Memperbarui item DynamoDB dengan TTL menggunakan SDK AWS

Contoh kode berikut menunjukkan cara memperbarui TTL item.

Java

SDK untuk Java 2.x

Perbarui TTL pada item DynamoDB yang ada dalam tabel.

```
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;
        }
    }
```

- Untuk detail API, lihat UpdateItemdi Referensi AWS SDK for Java 2.x API.

JavaScript

SDK untuk 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');
```

- Untuk detail API, lihat UpdateItemdi Referensi AWS SDK untuk JavaScript API.

Python

SDK untuk 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"
)
```

- Untuk detail API, lihat UpdateItemdi AWS SDK for Python (Boto3) Referensi API.

Untuk daftar lengkap panduan pengembang AWS SDK dan contoh kode, lihatMenggunakan DynamoDB dengan SDK AWS. Topik ini juga mencakup informasi tentang memulai dan detail tentang versi SDK sebelumnya.

Perbarui data DynamoDB menggunakan pernyataan PartiQL UPDATE dengan SDK AWS

Contoh kode berikut menunjukkan cara memperbarui data menggunakan pernyataan PartiQL UPDATE.

JavaScript

SDK untuk JavaScript (v3)

Perbarui item dalam tabel DynamoDB menggunakan pernyataan PartiQL UPDATE dengan. AWS SDK untuk 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(),
    },
  ]);
};
```

- Untuk detail API, lihat topik berikut di Referensi API AWS SDK untuk JavaScript .
  - BatchExecuteStatement
  - ExecuteStatement

Untuk daftar lengkap panduan pengembang AWS SDK dan contoh kode, lihatMenggunakan DynamoDB dengan SDK AWS. Topik ini juga mencakup informasi tentang memulai dan detail tentang versi SDK sebelumnya.

Menggunakan API Gateway untuk menginvokasi fungsi Lambda

Contoh kode berikut menunjukkan cara membuat AWS Lambda fungsi yang dipanggil oleh Amazon API Gateway.

Java

SDK untuk Java 2.x

Menunjukkan cara membuat AWS Lambda fungsi dengan menggunakan Lambda Java runtime API. Contoh ini memanggil AWS layanan yang berbeda untuk melakukan kasus penggunaan tertentu. Contoh ini menunjukkan cara membuat fungsi Lambda yang diinvokasi oleh Amazon API Gateway yang memindai peringatan hari jadi kerja di tabel Amazon DynamoDB dan menggunakan Amazon Simple Notification Service (Amazon SNS) untuk mengirim pesan teks berisi ucapan selamat kepada karyawan Anda pada tanggal hari jadi kerja satu tahun mereka.

Untuk kode sumber lengkap dan instruksi tentang cara mengatur dan menjalankan, lihat contoh lengkapnya di GitHub.

Layanan yang digunakan dalam contoh ini

- API Gateway
- DynamoDB
- Lambda
- Amazon SNS

JavaScript

SDK untuk JavaScript (v3)

Menunjukkan cara membuat AWS Lambda fungsi dengan menggunakan API JavaScript runtime Lambda. Contoh ini memanggil AWS layanan yang berbeda untuk melakukan kasus penggunaan tertentu. Contoh ini menunjukkan cara membuat fungsi Lambda yang diinvokasi oleh Amazon API Gateway yang memindai peringatan hari jadi kerja di tabel Amazon DynamoDB dan menggunakan Amazon Simple Notification Service (Amazon SNS)

untuk mengirim pesan teks berisi ucapan selamat kepada karyawan Anda pada tanggal hari jadi kerja satu tahun mereka.

Untuk kode sumber lengkap dan instruksi tentang cara mengatur dan menjalankan, lihat contoh lengkapnya di GitHub.

Contoh ini juga tersedia di panduan developer AWS SDK untuk JavaScript v3.

Layanan yang digunakan dalam contoh ini

- API Gateway
- DynamoDB
- Lambda
- Amazon SNS

Python

SDK untuk Python (Boto3)

Contoh ini menunjukkan cara membuat dan menggunakan Amazon API Gateway REST API yang menargetkan suatu AWS Lambda fungsi. Handler Lambda menunjukkan cara merutekan berdasarkan metode HTTP; cara mendapatkan data dari string kueri, header, dan badan; dan cara mengembalikan respons JSON.

- Menyebarkan fungsi Lambda.
- Buat API REST API Gateway.
- Buat sumber daya REST yang menargetkan fungsi Lambda.
- Berikan izin untuk mengizinkan API Gateway menjalankan fungsi Lambda.
- Gunakan paket Requests untuk mengirim permintaan ke REST API.
- Bersihkan semua sumber daya yang dibuat selama demo.

Contoh ini paling baik dilihat di GitHub. Untuk kode sumber lengkap dan instruksi tentang cara mengatur dan menjalankan, lihat contoh lengkapnya di GitHub.

Layanan yang digunakan dalam contoh ini

- API Gateway
- DynamoDB
- Lambda

- Amazon SNS

Untuk daftar lengkap panduan pengembang AWS SDK dan contoh kode, lihatMenggunakan DynamoDB dengan SDK AWS. Topik ini juga mencakup informasi tentang memulai dan detail tentang versi SDK sebelumnya.

Menggunakan Step Functions untuk menginvokasi fungsi Lambda

Contoh kode berikut menunjukkan cara membuat mesin AWS Step Functions status yang memanggil AWS Lambda fungsi secara berurutan.

Java

SDK untuk Java 2.x

Menunjukkan cara membuat alur kerja AWS tanpa server dengan menggunakan AWS Step Functions dan. AWS SDK for Java 2.x Setiap langkah alur kerja diimplementasikan menggunakan AWS Lambda fungsi.

Untuk kode sumber lengkap dan instruksi tentang cara mengatur dan menjalankan, lihat contoh lengkapnya di GitHub.

Layanan yang digunakan dalam contoh ini

- DynamoDB
- Lambda
- Amazon SES
- Step Functions

Untuk daftar lengkap panduan pengembang AWS SDK dan contoh kode, lihatMenggunakan DynamoDB dengan SDK AWS. Topik ini juga mencakup informasi tentang memulai dan detail tentang versi SDK sebelumnya.

Menggunakan model dokumen untuk DynamoDB menggunakan SDK AWS

Contoh kode berikut menunjukkan cara melakukan operasi Create, Read, Update, and Delete (CRUD) dan batch menggunakan model dokumen untuk DynamoDB dan SDK. AWS

Untuk informasi lebih lanjut, lihat Model dokumen.

.NET

SDK untuk .NET

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

Lakukan operasi CRUD menggunakan model dokumen.

```
/// <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);
            }
        }
    }
```

Lakukan operasi tulis batch menggunakan model dokumen.

```
    /// <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();
        }
    }
```

Pindai tabel menggunakan model dokumen.

```
   /// <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}");
            }
        }
    }
```

Kueri dan pindai tabel menggunakan model dokumen.

```
    /// <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}");
            }
        }
    }
```

Untuk daftar lengkap panduan pengembang AWS SDK dan contoh kode, lihatMenggunakan DynamoDB dengan SDK AWS. Topik ini juga mencakup informasi tentang memulai dan detail tentang versi SDK sebelumnya.

Menggunakan model persistensi objek tingkat tinggi untuk DynamoDB menggunakan SDK AWS

Contoh kode berikut menunjukkan cara melakukan operasi Create, Read, Update, and Delete (CRUD) dan batch menggunakan model persistensi objek untuk DynamoDB dan SDK. AWS

Untuk informasi lebih lanjut, lihat Model persistensi objek.

.NET

SDK untuk .NET

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

Lakukan operasi CRUD menggunakan model persistensi objek tingkat tinggi.

```
/// <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");
            }
        }
    }
```

Lakukan operasi tulis batch menggunakan model persistensi objek tingkat tinggi.

```
    /// <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);
        }
    }
```

Memetakan data arbitrer ke tabel menggunakan model persistensi objek tingkat tinggi.

```
    /// <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);
        }
    }
```

Kueri dan pindai tabel menggunakan model persistensi objek tingkat tinggi.

```
    /// <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}");
            }
        }
    }
```

Untuk daftar lengkap panduan pengembang AWS SDK dan contoh kode, lihatMenggunakan DynamoDB dengan SDK AWS. Topik ini juga mencakup informasi tentang memulai dan detail tentang versi SDK sebelumnya.

Gunakan operasi penghitung atom di DynamoDB dengan SDK AWS

Contoh kode berikut menunjukkan cara menggunakan operasi penghitung atom di DynamoDB.

- Penghitung kenaikan secara atom menggunakan operasi ADD dan SET.
- Penghitung kenaikan aman yang mungkin tidak ada.
- Menerapkan penguncian optimis untuk operasi counter.

Java

SDK untuk Java 2.x

Menunjukkan operasi penghitung atom menggunakan 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;
    }
```

Contoh penggunaan operasi penghitung atom dengan 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();
        }
    }
```

- Untuk detail API, lihat UpdateItemdi Referensi AWS SDK for Java 2.x API.

JavaScript

SDK untuk JavaScript (v3)

Menunjukkan operasi penghitung atom menggunakan AWS SDK untuk 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;
}
```

Contoh penggunaan operasi penghitung atom dengan AWS SDK untuk 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);
  }
}
```

- Untuk detail API, lihat UpdateItemdi Referensi AWS SDK untuk JavaScript API.

Python

SDK untuk Python (Boto3)

Menunjukkan operasi penghitung atom menggunakan AWS SDK untuk 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
```

Contoh penggunaan operasi penghitung atom dengan AWS SDK untuk 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")
```

- Untuk detail API, lihat UpdateItemdi AWS SDK for Python (Boto3) Referensi API.

Untuk daftar lengkap panduan pengembang AWS SDK dan contoh kode, lihatMenggunakan DynamoDB dengan SDK AWS. Topik ini juga mencakup informasi tentang memulai dan detail tentang versi SDK sebelumnya.

Menggunakan operasi bersyarat di DynamoDB dengan SDK AWS

Contoh kode berikut menunjukkan bagaimana menggunakan operasi bersyarat di DynamoDB.

- Menerapkan penulisan bersyarat untuk mencegah penimpaan data.
- Gunakan ekspresi kondisi untuk menegakkan aturan bisnis.
- Tangani kegagalan pemeriksaan bersyarat dengan anggun.

Java

SDK untuk Java 2.x

Menunjukkan operasi bersyarat menggunakan 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;
    }
```

Contoh penggunaan operasi bersyarat dengan 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();
        }
    }
```

- Untuk detail API, lihat topik berikut di Referensi API AWS SDK for Java 2.x .
  - DeleteItem
  - PutItem
  - UpdateItem

JavaScript

SDK untuk JavaScript (v3)

Menunjukkan operasi bersyarat menggunakan AWS SDK untuk 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;
}
```

Contoh penggunaan operasi bersyarat dengan AWS SDK untuk 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);
  }
}
```

- Untuk detail API, lihat topik berikut di Referensi API AWS SDK untuk JavaScript .
  - DeleteItem
  - PutItem
  - UpdateItem

Python

SDK untuk Python (Boto3)

Menunjukkan operasi bersyarat menggunakan AWS SDK untuk 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
```

Contoh penggunaan operasi bersyarat dengan AWS SDK untuk 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"
    )
```

- Untuk detail API, lihat topik berikut di Referensi API AWS SDK untuk Python (Boto3).
  - DeleteItem
  - PutItem
  - UpdateItem

Untuk daftar lengkap panduan pengembang AWS SDK dan contoh kode, lihatMenggunakan DynamoDB dengan SDK AWS. Topik ini juga mencakup informasi tentang memulai dan detail tentang versi SDK sebelumnya.

Gunakan nama atribut ekspresi di DynamoDB dengan SDK AWS

Contoh kode berikut menunjukkan bagaimana menggunakan nama atribut ekspresi di DynamoDB.

- Bekerja dengan kata-kata yang dicadangkan dalam ekspresi DynamoDB.
- Gunakan placeholder nama atribut ekspresi.
- Menangani karakter khusus dalam nama atribut.

Java

SDK untuk Java 2.x

Menunjukkan nama atribut ekspresi menggunakan 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);
    }
```

Contoh penggunaan nama atribut ekspresi dengan 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();
        }
    }
```

- Untuk detail API, lihat topik berikut di Referensi API AWS SDK for Java 2.x .
  - Kueri
  - UpdateItem

JavaScript

SDK untuk JavaScript (v3)

Menunjukkan nama atribut ekspresi menggunakan AWS SDK untuk 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;
}
```

Contoh penggunaan nama atribut ekspresi dengan AWS SDK untuk 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);
  }
}
```

- Untuk detail API, lihat topik berikut di Referensi API AWS SDK untuk JavaScript .
  - Kueri
  - UpdateItem

Python

SDK untuk Python (Boto3)

Menunjukkan nama atribut ekspresi menggunakan AWS SDK untuk 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
```

Contoh penggunaan nama atribut ekspresi dengan AWS SDK untuk 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")
```

- Untuk detail API, lihat topik berikut di Referensi API AWS SDK untuk Python (Boto3).
  - Kueri
  - UpdateItem

Untuk daftar lengkap panduan pengembang AWS SDK dan contoh kode, lihatMenggunakan DynamoDB dengan SDK AWS. Topik ini juga mencakup informasi tentang memulai dan detail tentang versi SDK sebelumnya.

Menggunakan peristiwa terjadwal untuk menginvokasi fungsi Lambda

Contoh kode berikut menunjukkan cara membuat AWS Lambda fungsi yang dipanggil oleh acara EventBridge terjadwal Amazon.

Java

SDK untuk Java 2.x

Menunjukkan cara membuat acara EventBridge terjadwal Amazon yang memanggil AWS Lambda fungsi. Konfigurasikan EventBridge untuk menggunakan ekspresi cron untuk menjadwalkan saat fungsi Lambda dipanggil. Dalam contoh ini, Anda membuat fungsi Lambda menggunakan API runtime Java Lambda. Contoh ini memanggil AWS layanan yang berbeda untuk melakukan kasus penggunaan tertentu. Contoh ini menunjukkan cara membuat aplikasi yang mengirimkan pesan teks seluler kepada karyawan Anda berisi ucapan selamat pada hari jadi setahun kerja mereka.

Untuk kode sumber lengkap dan instruksi tentang cara mengatur dan menjalankan, lihat contoh lengkapnya di GitHub.

Layanan yang digunakan dalam contoh ini

- CloudWatch Log
- DynamoDB
- EventBridge
- Lambda
- Amazon SNS

JavaScript

SDK untuk JavaScript (v3)

Menunjukkan cara membuat acara EventBridge terjadwal Amazon yang memanggil AWS Lambda fungsi. Konfigurasikan EventBridge untuk menggunakan ekspresi cron untuk menjadwalkan saat fungsi Lambda dipanggil. Dalam contoh ini, Anda membuat fungsi Lambda menggunakan API runtime JavaScript Lambda. Contoh ini memanggil AWS layanan yang berbeda untuk melakukan kasus penggunaan tertentu. Contoh ini menunjukkan cara membuat aplikasi yang mengirimkan pesan teks seluler kepada karyawan Anda berisi ucapan selamat pada hari jadi setahun kerja mereka.

Untuk kode sumber lengkap dan instruksi tentang cara mengatur dan menjalankan, lihat contoh lengkapnya di GitHub.

Contoh ini juga tersedia di panduan developer AWS SDK untuk JavaScript v3.

Layanan yang digunakan dalam contoh ini

- CloudWatch Log
- DynamoDB
- EventBridge
- Lambda
- Amazon SNS

Python

SDK untuk Python (Boto3)

Contoh ini menunjukkan cara mendaftarkan AWS Lambda fungsi sebagai target EventBridge acara Amazon terjadwal. Penangan Lambda menulis pesan ramah dan data peristiwa lengkap ke Amazon CloudWatch Logs untuk pengambilan nanti.

- Menyebarkan fungsi Lambda.
- Membuat acara EventBridge terjadwal dan menjadikan fungsi Lambda sebagai target.
- Memberikan izin untuk membiarkan EventBridge menjalankan fungsi Lambda.
- Mencetak data terbaru dari CloudWatch Log untuk menampilkan hasil pemanggilan terjadwal.
- Membersihkan semua sumber daya yang dibuat selama demo.

Contoh ini paling baik dilihat di GitHub. Untuk kode sumber lengkap dan instruksi tentang cara mengatur dan menjalankan, lihat contoh lengkapnya di GitHub.

Layanan yang digunakan dalam contoh ini

- CloudWatch Log
- DynamoDB
- EventBridge
- Lambda
- Amazon SNS

Untuk daftar lengkap panduan pengembang AWS SDK dan contoh kode, lihatMenggunakan DynamoDB dengan SDK AWS. Topik ini juga mencakup informasi tentang memulai dan detail tentang versi SDK sebelumnya.

Bekerja dengan DynamoDB Local Secondary Indexes menggunakan v2 AWS Command Line Interface

Contoh kode berikut menunjukkan cara membuat dan query tabel dengan Local Secondary Indexes.

- Buat tabel dengan Local Secondary Index (LSI).
- Buat tabel dengan beberapa LSIs dengan jenis proyeksi yang berbeda.
- Data kueri menggunakan LSIs.

Bash

AWS CLI dengan skrip Bash

Buat tabel dengan Indeks Sekunder Lokal.

```
# 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
```

Buat tabel dengan banyak 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
```

Data kueri menggunakan 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"}
    }'
```

- Untuk detail API, lihat topik berikut di Referensi Perintah AWS CLI .
  - CreateTable
  - Kueri

Untuk daftar lengkap panduan pengembang AWS SDK dan contoh kode, lihatMenggunakan DynamoDB dengan SDK AWS. Topik ini juga mencakup informasi tentang memulai dan detail tentang versi SDK sebelumnya.

Bekerja dengan DynamoDB Streams dan menggunakan v2 Time-to-Live AWS Command Line Interface

Contoh kode berikut menunjukkan cara mengelola DynamoDB Streams dan fitur. Time-to-Live

- Buat tabel dengan Streams diaktifkan.
- Jelaskan Aliran.
- Buat fungsi Lambda untuk memproses Streams.
- Aktifkan TTL di atas meja.
- Tambahkan item dengan atribut TTL.
- Jelaskan pengaturan TTL.

Bash

AWS CLI dengan skrip Bash

Buat tabel dengan Streams diaktifkan.

```
# 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
```

Jelaskan Aliran.

```
# 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
```

Buat fungsi Lambda untuk 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"
```

Aktifkan TTL di atas meja.

```
# 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"
```

Tambahkan item dengan atribut TTL.

```
# 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'"}
    }'
```

Jelaskan pengaturan TTL.

```
# Describe TTL settings for a table
aws dynamodb describe-time-to-live \
    --table-name TTLDemo
```

- Untuk detail API, lihat topik berikut di Referensi Perintah AWS CLI .
  - AttachRolePolicy
  - CreateRole
  - CreateTable
  - DescribeTable
  - DescribeTimeToLive
  - PutItem
  - UpdateTimeToLive

Untuk daftar lengkap panduan pengembang AWS SDK dan contoh kode, lihatMenggunakan DynamoDB dengan SDK AWS. Topik ini juga mencakup informasi tentang memulai dan detail tentang versi SDK sebelumnya.

Bekerja dengan tabel global DynamoDB dan replikasi multi-wilayah menggunakan v2 AWS Command Line Interface

Contoh kode berikut menunjukkan cara mengelola tabel global DynamoDB dengan replikasi multi-region.

- Buat tabel dengan replikasi multi-wilayah.
- Masukkan dan dapatkan item dari tabel replika.

- Hapus replika.

Bash

AWS CLI dengan skrip Bash

Buat tabel dengan replikasi multi-wilayah.

```
# Step 1: Create a new table in us-west-2
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-west-2

# Step 2: Create replicas in us-east-1 and us-east-2
aws dynamodb update-table \
    --table-name MusicTable \
    --replica-updates '[{"Create": {"RegionName": "us-east-1"}}, {"Create":
 {"RegionName": "us-east-2"}}]' \
    --multi-region-consistency STRONG \
    --region us-west-2
```

Jelaskan tabel multi-wilayah.

```
# Describe the base table
aws dynamodb describe-table --table-name MusicTable --region us-west-2
```

Letakkan item dalam tabel replika.

```
# Write a single item to one of the replica tables.
aws dynamodb put-item \
    --table-name MusicTable \
    --item '{"Artist": {"S":"item_1"},"SongTitle": {"S":"Song Value 1"}}' \
```

```
    --region us-east-2
```

Dapatkan item dari tabel replika.

```
# Get item from the other two replicas
aws dynamodb get-item \
    --table-name MusicTable \
    --key '{"Artist": {"S":"item_1"},"SongTitle": {"S":"Song Value 1"}}' \
    --consistent-read \
    --region us-east-1

aws dynamodb get-item \
    --table-name MusicTable \
    --key '{"Artist": {"S":"item_1"},"SongTitle": {"S":"Song Value 1"}}' \
    --consistent-read \
    --region us-west-2
```

Hapus replika.

```
# Remove the replica tables.
aws dynamodb update-table \
    --table-name MusicTable \
    --replica-updates '[{"Delete": {"RegionName": "us-east-2"}}, {"Delete":
 {"RegionName": "us-east-1"}}]' \
    --region us-west-2
```

- Untuk detail API, lihat topik berikut di Referensi Perintah AWS CLI .
  - CreateTable
  - GetItem
  - PutItem
  - UpdateTable

Untuk daftar lengkap panduan pengembang AWS SDK dan contoh kode, lihatMenggunakan DynamoDB dengan SDK AWS. Topik ini juga mencakup informasi tentang memulai dan detail tentang versi SDK sebelumnya.

Bekerja dengan penandaan sumber daya DynamoDB menggunakan v2 AWS Command Line Interface

Contoh kode berikut menunjukkan cara mengelola tag untuk sumber daya DynamoDB.

- Buat tabel dengan tag.
- Daftar tag untuk sumber daya.
- Tambahkan tag ke sumber daya.
- Hapus tag dari sumber daya.
- Filter tabel berdasarkan tag.

Bash

AWS CLI dengan skrip Bash

Buat tabel dengan tag.

```
# 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
```

Daftar tag untuk sumber daya.

```
# 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
```

Tambahkan tag ke sumber daya.

```
# Add tags to an existing table
aws dynamodb tag-resource \
    --resource-arn $TABLE_ARN \
    --tags \
        Key=CostCenter,Value=12345 \
        Key=BackupSchedule,Value=Daily
```

Hapus tag dari sumber daya.

```
# Remove tags from a table
aws dynamodb untag-resource \
    --resource-arn $TABLE_ARN \
    --tag-keys Owner BackupSchedule
```

Filter tabel berdasarkan tag.

```
# 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
```

- Untuk detail API, lihat topik berikut di Referensi Perintah AWS CLI .
  - CreateTable
  - ListTagsOfResource
  - TagResource
  - UntagResource

Untuk daftar lengkap panduan pengembang AWS SDK dan contoh kode, lihatMenggunakan DynamoDB dengan SDK AWS. Topik ini juga mencakup informasi tentang memulai dan detail tentang versi SDK sebelumnya.

Bekerja dengan enkripsi tabel DynamoDB menggunakan v2 AWS Command Line Interface

Contoh kode berikut menunjukkan cara mengelola opsi enkripsi untuk tabel DynamoDB.

- Buat tabel dengan enkripsi default.
- Buat tabel dengan CMK yang dikelola pelanggan.
- Perbarui pengaturan enkripsi tabel.
- Jelaskan enkripsi tabel.

Bash

AWS CLI dengan skrip Bash

Buat tabel dengan enkripsi default.

```
# 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
```

Buat tabel dengan CMK yang dikelola pelanggan.

```
# 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
```

Perbarui enkripsi tabel.

```
# 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
```

Jelaskan enkripsi tabel.

```
# Describe the table to see encryption settings
aws dynamodb describe-table \
    --table-name CustomerData \
    --query "Table.SSEDescription"
```

- Untuk detail API, lihat topik berikut di Referensi Perintah AWS CLI .
  - CreateKey
  - CreateTable
  - DescribeTable
  - UpdateTable

Untuk daftar lengkap panduan pengembang AWS SDK dan contoh kode, lihatMenggunakan DynamoDB dengan SDK AWS. Topik ini juga mencakup informasi tentang memulai dan detail tentang versi SDK sebelumnya.

Contoh tanpa server untuk DynamoDB

Contoh kode berikut menunjukkan bagaimana menggunakan DynamoDB dengan. AWS SDKs

Contoh

- Memanggil fungsi Lambda dari pemicu DynamoDB
- Melaporkan kegagalan item batch untuk fungsi Lambda dengan pemicu DynamoDB

Memanggil fungsi Lambda dari pemicu DynamoDB

Contoh kode berikut menunjukkan bagaimana menerapkan fungsi Lambda yang menerima peristiwa yang dipicu dengan menerima catatan dari aliran DynamoDB. Fungsi mengambil payload DynamoDB dan mencatat isi catatan.

.NET

SDK untuk .NET

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di repositori contoh Nirserver.

Mengkonsumsi acara DynamoDB dengan Lambda menggunakan.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 untuk Go V2

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di repositori contoh Nirserver.

Mengkonsumsi acara DynamoDB dengan Lambda menggunakan 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 untuk Java 2.x

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di repositori contoh Nirserver.

Mengkonsumsi acara DynamoDB dengan Lambda menggunakan 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 untuk JavaScript (v3)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di repositori contoh Nirserver.

Mengkonsumsi acara DynamoDB dengan Lambda menggunakan. 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)}`);
};
```

Mengkonsumsi acara DynamoDB dengan Lambda menggunakan. 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 untuk PHP

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di repositori contoh Nirserver.

Mengkonsumsi acara DynamoDB dengan Lambda menggunakan 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 untuk Python (Boto3)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di repositori contoh Nirserver.

Mengkonsumsi acara DynamoDB dengan Lambda menggunakan 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 untuk Ruby

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di repositori contoh Nirserver.

Mengkonsumsi acara DynamoDB dengan Lambda menggunakan 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 untuk Rust

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di repositori contoh Nirserver.

Mengkonsumsi acara DynamoDB dengan Lambda menggunakan 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(())

}
```

Untuk daftar lengkap panduan pengembang AWS SDK dan contoh kode, lihatMenggunakan DynamoDB dengan SDK AWS. Topik ini juga mencakup informasi tentang memulai dan detail tentang versi SDK sebelumnya.

Melaporkan kegagalan item batch untuk fungsi Lambda dengan pemicu DynamoDB

Contoh kode berikut menunjukkan cara mengimplementasikan respons batch sebagian untuk fungsi Lambda yang menerima peristiwa dari aliran DynamoDB. Fungsi melaporkan kegagalan item batch dalam respons, memberi sinyal ke Lambda untuk mencoba lagi pesan tersebut nanti.

.NET

SDK untuk .NET

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di repositori contoh Nirserver.

Melaporkan kegagalan item batch DynamoDB dengan Lambda menggunakan.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 untuk Go V2

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di repositori contoh Nirserver.

Melaporkan kegagalan item batch DynamoDB dengan Lambda menggunakan 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 untuk Java 2.x

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di repositori contoh Nirserver.

Melaporkan kegagalan item batch DynamoDB dengan Lambda menggunakan 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 untuk JavaScript (v3)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di repositori contoh Nirserver.

Melaporkan kegagalan item batch DynamoDB dengan penggunaan Lambda. 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: [] };
};
```

Melaporkan kegagalan item batch DynamoDB dengan penggunaan Lambda. 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 untuk PHP

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di repositori contoh Nirserver.

Melaporkan kegagalan item batch DynamoDB dengan Lambda menggunakan 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 untuk Python (Boto3)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di repositori contoh Nirserver.

Melaporkan kegagalan item batch DynamoDB dengan Lambda menggunakan 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 untuk Ruby

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di repositori contoh Nirserver.

Melaporkan kegagalan item batch DynamoDB dengan Lambda menggunakan 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 untuk Rust

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di repositori contoh Nirserver.

Melaporkan kegagalan item batch DynamoDB dengan Lambda menggunakan 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
}
```

Untuk daftar lengkap panduan pengembang AWS SDK dan contoh kode, lihatMenggunakan DynamoDB dengan SDK AWS. Topik ini juga mencakup informasi tentang memulai dan detail tentang versi SDK sebelumnya.

AWS kontribusi komunitas untuk DynamoDB

AWS kontribusi komunitas adalah contoh yang dibuat dan dikelola oleh banyak tim AWS. Untuk memberikan umpan balik, gunakan mekanisme yang disediakan di repositori terkait.

Contoh

- Membangun dan menguji aplikasi tanpa server

Membangun dan menguji aplikasi tanpa server

Contoh kode berikut menunjukkan cara membangun dan menguji aplikasi tanpa server menggunakan API Gateway dengan Lambda dan DynamoDB

.NET

SDK untuk .NET

Menunjukkan cara membangun dan menguji aplikasi tanpa server yang terdiri dari API Gateway dengan Lambda dan DynamoDB menggunakan .NET SDK.

Untuk kode sumber lengkap dan instruksi tentang cara mengatur dan menjalankan, lihat contoh lengkapnya di GitHub.

Layanan yang digunakan dalam contoh ini

- API Gateway
- DynamoDB
- Lambda

Go

SDK untuk Go V2

Menunjukkan cara membuat dan menguji aplikasi tanpa server yang terdiri dari API Gateway dengan Lambda dan DynamoDB menggunakan Go SDK.

Untuk kode sumber lengkap dan instruksi tentang cara mengatur dan menjalankan, lihat contoh lengkapnya di GitHub.

Layanan yang digunakan dalam contoh ini

- API Gateway
- DynamoDB
- Lambda

Java

SDK untuk Java 2.x

Menunjukkan cara membangun dan menguji aplikasi tanpa server yang terdiri dari API Gateway dengan Lambda dan DynamoDB menggunakan Java SDK.

Untuk kode sumber lengkap dan instruksi tentang cara mengatur dan menjalankan, lihat contoh lengkapnya di GitHub.

Layanan yang digunakan dalam contoh ini

- API Gateway
- DynamoDB
- Lambda

Rust

SDK untuk Rust

Menunjukkan cara membuat dan menguji aplikasi tanpa server yang terdiri dari API Gateway dengan Lambda dan DynamoDB menggunakan Rust SDK.

Untuk kode sumber lengkap dan instruksi tentang cara mengatur dan menjalankan, lihat contoh lengkapnya di GitHub.

Layanan yang digunakan dalam contoh ini

- API Gateway
- DynamoDB
- Lambda

Untuk daftar lengkap panduan pengembang AWS SDK dan contoh kode, lihatMenggunakan DynamoDB dengan SDK AWS. Topik ini juga mencakup informasi tentang memulai dan detail tentang versi SDK sebelumnya.

Keamanan dan kepatuhan di Amazon DynamoDB

Keamanan cloud di AWS adalah prioritas tertinggi. Sebagai AWS pelanggan, Anda mendapat manfaat dari pusat data dan arsitektur jaringan yang dibangun untuk memenuhi persyaratan organisasi yang paling sensitif terhadap keamanan.

Keamanan adalah tanggung jawab bersama antara Anda AWS dan Anda. Model tanggung jawab bersama menjelaskan hal ini sebagai keamanan cloud dan keamanan dalam cloud:

- Keamanan cloud — AWS bertanggung jawab untuk melindungi infrastruktur yang menjalankan AWS layanan di AWS Cloud. AWS juga memberi Anda layanan yang dapat Anda gunakan dengan aman. Efektivitas keamanan kami diuji dan diverifikasi secara rutin oleh auditor pihak ketiga sebagai bagian dari program kepatuhan AWS. Untuk mempelajari tentang program kepatuhan yang berlaku untuk DynamoDB, lihat Layanan AWS dalam Cakupan Program Kepatuhan.
- Keamanan di cloud — Tanggung jawab Anda ditentukan oleh AWS layanan yang Anda gunakan. Anda juga bertanggung jawab atas faktor-faktor lain termasuk sensitivitas data Anda, persyaratan organisasi Anda, serta undang-undang dan peraturan yang berlaku.

Dokumentasi ini akan membantu Anda memahami cara menerapkan model tanggung jawab bersama saat menggunakan DynamoDB. Topik berikut menunjukkan cara mengonfigurasi DynamoDB untuk memenuhi tujuan keamanan dan kepatuhan Anda. Anda juga akan belajar cara menggunakan AWS layanan lain yang dapat membantu Anda memantau dan mengamankan sumber daya DynamoDB Anda.

Topik

- AWS kebijakan terkelola untuk Amazon DynamoDB
- Menggunakan kebijakan berbasis sumber daya untuk DynamoDB
- Menggunakan kontrol akses berbasis atribut dengan DynamoDB
- Perlindungan data di DynamoDB
- AWS Identity and Access Management (IAM) dan DynamoDB
- Validasi kepatuhan oleh industri untuk DynamoDB
- Ketahanan dan pemulihan bencana di Amazon DynamoDB
- Keamanan infrastruktur di Amazon DynamoDB
- AWS PrivateLink untuk DynamoDB

- Analisis konfigurasi dan kerentanan di Amazon DynamoDB
- Praktik Terbaik Keamanan untuk Amazon DynamoDB

AWS kebijakan terkelola untuk Amazon DynamoDB

DynamoDB AWS menggunakan kebijakan terkelola untuk menentukan sekumpulan izin yang dibutuhkan layanan untuk melakukan tindakan tertentu. DynamoDB memelihara dan memperbarui AWS kebijakan yang dikelola. Anda tidak dapat mengubah izin dalam kebijakan AWS terkelola. Untuk informasi selengkapnya tentang kebijakan AWS AWS terkelola, lihat kebijakan terkelola di Panduan Pengguna IAM.

DynamoDB terkadang dapat menambahkan izin tambahan ke kebijakan terkelola untuk AWS mendukung fitur baru. Jenis pembaruan ini akan memengaruhi semua identitas (pengguna, grup, dan peran) di mana kebijakan tersebut dilampirkan. Kebijakan AWS terkelola kemungkinan besar akan diperbarui saat fitur baru diluncurkan atau saat operasi baru tersedia. DynamoDB tidak akan menghapus izin dari AWS kebijakan terkelola, sehingga pembaruan kebijakan tidak akan merusak izin yang ada. Untuk daftar lengkap kebijakan AWS terkelola, lihat kebijakan AWS terkelola.

AWS kebijakan terkelola: Dynamo DBReplication ServiceRolePolicy

Anda tidak dapat melampirkan kebijakan `DynamoDBReplicationServiceRolePolicy` ke entitas IAM Anda. Kebijakan ini dilampirkan ke peran tertaut layanan yang memungkinkan DynamoDB melakukan tindakan atas nama Anda. Untuk informasi selengkapnya, lihat Menggunakan IAM dengan tabel global.

Kebijakan ini memberikan izin yang memungkinkan peran tertaut layanan untuk melakukan replikasi data antara replika tabel global. Kebijakan ini juga memberikan izin administratif untuk mengelola replika tabel global atas nama Anda.

Detail izin

Kebijakan ini memberikan izin untuk melakukan hal berikut:

- `dynamodb` – Melakukan replikasi data dan mengelola replika tabel.
- `application-autoscaling`— Mengambil dan mengelola pengaturan Auto Scaling tabel
- `account` – Mengambil status wilayah untuk mengevaluasi aksesibilitas replika.
- `iam`— Untuk membuat peran terkait layanan untuk Auto Scaling aplikasi jika peran terkait layanan belum ada.

Definisi kebijakan yang dikelola ini dapat ditemukan di sini.

AWS kebijakan terkelola: AmazonDynamo DBFull Access_v2

`AmazonDynamoDBFullAccess_v2`Kebijakan cakupan bawah memberikan hak akses khusus kepada pengguna. Anda dapat melampirkan kebijakan `AmazonDynamoDBFullAccess_v2` ke identitas IAM Anda. Kebijakan ini memberikan akses administratif ke sumber daya Amazon DynamoDB dan memberikan identitas IAM (seperti pengguna, grup, atau peran) akses ke DynamoDB yang terintegrasi dengannya untuk menggunakan Layanan AWS semua fitur DynamoDB. Menggunakan kebijakan ini memungkinkan akses ke semua fitur DynamoDB yang tersedia di. AWS Management Console

Detail izin

Kebijakan ini mencakup izin berikut:

- `Amazon DynamoDB`
- `DynamoDB Accelerator`
- `AWS KMS`
- `AWS Resource Groups Tagging`
- `Lambda`
- `Application Auto Scaling`
- `CloudWatch`
- `Amazon Kinesis`
- `Amazon EC2`
- `IAM`

Untuk meninjau kebijakan dalam `JSON` format, lihat AmazonDynamoDBFullAccess_v2.

AWS kebijakan terkelola: AmazonDynamo DBRead OnlyAccess

Anda dapat melampirkan kebijakan `AmazonDynamoDBReadOnlyAccess` ke identitas IAM Anda.

Kebijakan ini memberikan akses hanya-baca ke Amazon DynamoDB.

Detail izin

Kebijakan ini mencakup izin berikut:

- `Amazon DynamoDB`— Menyediakan akses read-only ke Amazon DynamoDB.
- `Amazon DynamoDB Accelerator (DAX)`— Menyediakan akses read-only ke Amazon DynamoDB Accelerator (DAX).
- `Application Auto Scaling`— Memungkinkan prinsipal untuk melihat konfigurasi dari Application Auto Scaling. Ini diperlukan agar pengguna dapat melihat kebijakan penskalaan otomatis yang dilampirkan ke tabel.
- `CloudWatch`— Memungkinkan kepala sekolah untuk melihat data metrik dan alarm yang dikonfigurasi. CloudWatch Ini diperlukan agar pengguna dapat melihat ukuran tabel yang dapat ditagih dan CloudWatch alarm yang telah dikonfigurasi untuk tabel.
- `AWS Data Pipeline`— Memungkinkan prinsipal untuk melihat AWS Data Pipeline dan objek terkait.
- `Amazon EC2`— Memungkinkan kepala sekolah untuk melihat Amazon EC2 VPCs, subnet, dan grup keamanan.
- `IAM`— Memungkinkan kepala sekolah untuk melihat peran IAM.
- `AWS KMS`— Memungkinkan kepala sekolah untuk melihat kunci yang dikonfigurasi di. AWS KMS Ini diperlukan agar pengguna dapat melihat AWS KMS keys bahwa mereka membuat dan mengelola di akun mereka.
- `Amazon SNS`— Memungkinkan kepala sekolah untuk mencantumkan topik dan langganan Amazon SNS berdasarkan topik.
- `AWS Resource Groups`— Memungkinkan kepala sekolah untuk melihat grup sumber daya dan kueri mereka.
- `AWS Resource Groups Tagging`— Memungkinkan prinsipal untuk mencantumkan semua sumber daya yang ditandai atau ditandai sebelumnya di suatu Wilayah.
- `Kinesis`— Memungkinkan kepala sekolah untuk melihat deskripsi aliran data Kinesis.
- `Amazon CloudWatch Contributor Insights`— Izinkan kepala sekolah untuk melihat data deret waktu yang dikumpulkan oleh aturan Contributor Insights.

Untuk meninjau kebijakan dalam `JSON` format, lihat AmazonDynamoDBReadOnlyAccess.

DynamoDB memperbarui kebijakan terkelola AWS

Tabel ini menunjukkan pembaruan kebijakan manajemen AWS akses untuk DynamoDB.

Perubahan	Deskripsi	Tanggal Diubah
`AmazonDynamoDBFullAccess` — Usang	Kebijakan ini telah digantikan oleh kebijakan tercakup bawah bernama. `AmazonDynamoDBFullAccess_v2` Setelah April 2025, Anda tidak dapat melampirkan `AmazonDynamoDBFullAccess` kebijakan ke pengguna, grup, atau peran baru mana pun. Untuk informasi selengkapnya, lihat AWS kebijakan terkelola: AmazonDynamo DBFull Access_v2.	April 28, 2025
Pembaruan `AmazonDynamoDBReadOnlyAccess` pada kebijakan yang sudah ada	`AmazonDynamoDBReadOnlyAccess` menambahkan izin: `dynamodb:GetAbacStatus` dan`dynamodb:UpdateAbacStatus` . Izin ini memungkinkan Anda untuk melihat status ABAC dan mengaktifkan ABAC untuk Wilayah Anda Akun AWS saat ini.	November 18, 2024
Pembaruan `AmazonDynamoDBReadOnlyAccess` pada kebijakan yang sudah ada	`AmazonDynamoDBReadOnlyAccess` menambahkan izin `dynamodb:GetResourcePolicy` . Izin ini menyediakan akses ke kebijakan berbasis sumber daya baca yang dilampirkan ke sumber daya DynamoDB.	Maret 20, 2024
Pembaruan `DynamoDBReplicationServiceR`	`DynamoDBReplicationServiceRolePolicy` menambahkan izin `dynamodb:GetResourcePolicy` . Izin ini	15 Desember 2023

Perubahan	Deskripsi	Tanggal Diubah
olePolicy pada kebijakan yang sudah ada	memungkinkan peran terkait layanan untuk membaca kebijakan berbasis sumber daya yang dilampirkan ke sumber daya DynamoDB.	
Pembaruan DynamoDBReplicationServiceRolePolicy pada kebijakan yang sudah ada	DynamoDBReplicationServiceRolePolicy menambahkan izin account:ListRegions . Izin ini memungkinkan peran tertaut layanan dapat mengevaluasi aksesibilitas replika	10 Mei 2023
DynamoDBReplicationServiceRolePolicy ditambahkan ke daftar kebijakan yang dikelola	Menambahkan informasi tentang kebijakan yang dikelola DynamoDBReplicationServiceRolePolicy , yang digunakan oleh peran tertaut layanan tabel global DynamoDB.	10 Mei 2023
Tabel global DynamoDB mulai melacak perubahan	Tabel global DynamoDB mulai melacak perubahan untuk AWS kebijakan terkelolanya.	10 Mei 2023

Menggunakan kebijakan berbasis sumber daya untuk DynamoDB

DynamoDB mendukung kebijakan berbasis sumber daya untuk tabel, indeks, dan aliran. Kebijakan berbasis sumber daya memungkinkan Anda menentukan izin akses dengan menentukan siapa yang memiliki akses ke setiap sumber daya, dan tindakan yang diizinkan untuk dilakukan pada setiap sumber daya.

Anda dapat melampirkan kebijakan berbasis sumber daya ke sumber daya DynamoDB, seperti tabel atau aliran. Dalam kebijakan ini, Anda menentukan izin untuk prinsip Identity and Access

Management (IAM) yang dapat melakukan tindakan tertentu pada sumber daya DynamoDB ini. Misalnya, kebijakan yang dilampirkan pada tabel akan berisi izin untuk mengakses tabel dan indeksnya. Akibatnya, kebijakan berbasis sumber daya dapat membantu Anda menyederhanakan kontrol akses untuk tabel, indeks, dan aliran DynamoDB, dengan menentukan izin di tingkat sumber daya. Ukuran maksimum kebijakan yang dapat Anda lampirkan ke sumber daya DynamoDB adalah 20 KB.

Manfaat signifikan menggunakan kebijakan berbasis sumber daya adalah menyederhanakan kontrol akses lintas akun untuk menyediakan akses lintas akun ke prinsip-prinsip IAM yang berbeda. Akun AWS Untuk informasi selengkapnya, lihat Kebijakan berbasis sumber daya untuk akses lintas akun.

Kebijakan berbasis sumber daya juga mendukung integrasi dengan IAM Access Analyzer external access analyzer dan kemampuan Block Public Access (BPA). IAM Access Analyzer melaporkan akses lintas akun ke entitas eksternal yang ditentukan dalam kebijakan berbasis sumber daya. Ini juga memberikan visibilitas untuk membantu Anda menyempurnakan izin dan menyesuaikan diri dengan prinsip hak istimewa paling sedikit. BPA membantu Anda mencegah akses publik ke tabel, indeks, dan aliran DynamoDB Anda, dan secara otomatis diaktifkan dalam alur kerja pembuatan dan modifikasi kebijakan berbasis sumber daya.

Topik

- Membuat tabel dengan kebijakan berbasis sumber daya
- Melampirkan kebijakan ke tabel DynamoDB yang ada
- Melampirkan kebijakan berbasis sumber daya ke aliran DynamoDB
- Menghapus kebijakan berbasis sumber daya dari tabel DynamoDB
- Akses lintas akun dengan kebijakan berbasis sumber daya di DynamoDB
- Memblokir akses publik dengan kebijakan berbasis sumber daya di DynamoDB
- Operasi API DynamoDB didukung oleh kebijakan berbasis sumber daya
- Otorisasi dengan kebijakan berbasis identitas IAM dan kebijakan berbasis sumber daya DynamoDB
- Contoh kebijakan berbasis sumber daya DynamoDB
- Pertimbangan kebijakan berbasis sumber daya DynamoDB
- Praktik terbaik kebijakan berbasis sumber daya DynamoDB

Membuat tabel dengan kebijakan berbasis sumber daya

Anda dapat menambahkan kebijakan berbasis sumber daya saat membuat tabel menggunakan konsol DynamoDB, CreateTableAPI, SDK, atau template. AWS CLIAWS AWS CloudFormation

AWS CLI

Contoh berikut membuat tabel bernama *MusicCollection* menggunakan `create-table` AWS CLI perintah. Perintah ini juga menyertakan `resource-policy` parameter yang menambahkan kebijakan berbasis sumber daya ke tabel. Kebijakan ini memungkinkan pengguna *John* untuk melakukan tindakan RestoreTableToPointInTimeGetItem,, dan PutItemAPI pada tabel.

Ingatlah untuk mengganti *italicized* teks dengan informasi spesifik sumber daya Anda.

```
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 Management Console

1. Masuk ke AWS Management Console dan buka konsol DynamoDB di. https://console.aws.amazon.com/dynamodb/
2. Di dasbor, pilih Buat tabel.
3. Dalam Rincian tabel, masukkan nama tabel, kunci partisi, dan urutkan detail kunci.
4. Di Pengaturan tabel, pilih Sesuaikan pengaturan.
5. (Opsional) Tentukan opsi Anda untuk kelas Tabel, Kalkulator kapasitas, Pengaturan kapasitas baca/tulis, Indeks sekunder, Enkripsi saat istirahat, dan Perlindungan penghapusan.
6. Dalam kebijakan berbasis sumber daya, tambahkan kebijakan untuk menentukan izin akses untuk tabel dan indeksnya. Dalam kebijakan ini, Anda menentukan siapa yang memiliki akses ke sumber daya ini, dan tindakan yang diizinkan untuk dilakukan pada setiap sumber daya. Untuk menambahkan kebijakan, lakukan salah satu hal berikut:

   - Ketik atau tempel dokumen kebijakan JSON. Untuk detail tentang bahasa kebijakan IAM, lihat Membuat kebijakan menggunakan editor JSON di Panduan Pengguna IAM.

     > **Tip**
     >
     > Untuk melihat contoh kebijakan berbasis sumber daya di Panduan Pengembang Amazon DynamoDB, pilih Contoh kebijakan.

   - Pilih Tambahkan pernyataan baru untuk menambahkan pernyataan baru dan masukkan informasi di bidang yang disediakan. Ulangi langkah ini sebanyak jumlah pernyataan yang ingin Anda tambahkan.

   > **Important**
   >
   > Pastikan Anda menyelesaikan peringatan, kesalahan, atau saran keamanan apa pun sebelum menyimpan kebijakan Anda.

   Contoh kebijakan IAM berikut memungkinkan pengguna *`John`* untuk melakukan tindakan RestoreTableToPointInTime, GetItem, dan PutItemAPI pada tabel*`MusicCollection`*.

   Ingatlah untuk mengganti *`italicized`* teks dengan informasi spesifik sumber daya Anda.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "AWS": "arn:aws:iam::123456789012:user/John"
      },
      "Action": [
        "dynamodb:RestoreTableToPointInTime",
        "dynamodb:GetItem",
        "dynamodb:PutItem"
      ],
      "Resource": "arn:aws:dynamodb:us-east-1:123456789012:table/MusicCollection"
    }
  ]
}
```

7. (Opsional) Pilih Pratinjau akses eksternal di sudut kanan bawah untuk melihat bagaimana kebijakan baru memengaruhi publik dan akses lintas akun ke sumber daya Anda. Sebelum Anda menyimpan kebijakan Anda, Anda dapat memeriksa apakah itu memperkenalkan temuan Penganalisa Akses IAM baru atau menyelesaikan temuan yang ada. Jika Anda tidak melihat penganalisis aktif, pilih Buka Penganalisis Akses untuk membuat penganalisis akun di Penganalisis Akses IAM. Untuk informasi selengkapnya, lihat Pratinjau akses.
8. Pilih Buat tabel.

AWS CloudFormation Template

Using the AWS::DynamoDB::Table resource

CloudFormation Template berikut membuat tabel dengan aliran menggunakan sumber daya AWS: :DynamoDB: :Table. Template ini juga menyertakan kebijakan berbasis sumber daya yang dilampirkan ke tabel dan aliran.

```
{
    "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

CloudFormation Template berikut membuat tabel dengan sumber daya AWS: :DynamoDB:: dan melampirkan kebijakan berbasis GlobalTable sumber daya ke tabel dan alirannya.

```
{
    "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": "*"
                                    }]
                                }
                            }
                        }
                    }
                ]
            }
        }
    }
```

```
}
```

Melampirkan kebijakan ke tabel DynamoDB yang ada

Anda dapat melampirkan kebijakan berbasis sumber daya ke tabel yang ada atau mengubah kebijakan yang ada menggunakan konsol DynamoDB, PutResourcePolicyAPI, SDK, atau templat. AWS CLIAWSAWS CloudFormation

AWS CLI contoh untuk melampirkan kebijakan baru

Contoh kebijakan IAM berikut menggunakan `put-resource-policy` AWS CLI perintah untuk melampirkan kebijakan berbasis sumber daya ke tabel yang ada. Contoh ini memungkinkan pengguna *John* untuk melakukan tindakan GetItem, PutItem, UpdateItem, dan UpdateTableAPI pada tabel yang ada bernama*MusicCollection*.

Ingatlah untuk mengganti *italicized* teks dengan informasi spesifik sumber daya Anda.

```
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 contoh untuk memperbarui kebijakan yang ada secara kondisional

Untuk memperbarui kebijakan tabel berbasis sumber daya yang ada secara kondisional, Anda dapat menggunakan parameter opsional. `expected-revision-id` Contoh berikut hanya akan memperbarui kebijakan jika ada di DynamoDB dan ID revisi saat ini cocok dengan parameter yang disediakan. `expected-revision-id`

```
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 Management Console

1. Masuk ke AWS Management Console dan buka konsol DynamoDB di. https://console.aws.amazon.com/dynamodb/
2. Dari dasbor, pilih tabel yang ada.
3. Arahkan ke tab Izin, dan pilih Buat kebijakan tabel.
4. Di editor kebijakan berbasis sumber daya, tambahkan kebijakan yang ingin dilampirkan dan pilih Buat kebijakan.

Contoh kebijakan IAM berikut memungkinkan pengguna *John* untuk melakukan tindakan GetItem,, PutItemUpdateItem, dan UpdateTableAPI pada tabel yang ada bernama*MusicCollection*.

Ingatlah untuk mengganti *italicized* teks dengan informasi spesifik sumber daya Anda.

```
{
  "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 SDK for Java 2.x

Contoh kebijakan IAM berikut menggunakan `putResourcePolicy` metode untuk melampirkan kebijakan berbasis sumber daya ke tabel yang ada. Kebijakan ini memungkinkan pengguna untuk melakukan tindakan GetItemAPI pada tabel yang ada.

```
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" +
                "}";
    }
}
```

Melampirkan kebijakan berbasis sumber daya ke aliran DynamoDB

Anda dapat melampirkan kebijakan berbasis sumber daya ke aliran tabel yang ada atau mengubah kebijakan yang ada menggunakan konsol DynamoDB, PutResourcePolicyAPI, SDK, atau templat. AWS CLIAWSAWS CloudFormation

> **Note**
>
> Anda tidak dapat melampirkan kebijakan ke aliran saat membuatnya menggunakan CreateTableatau UpdateTable APIs. Namun, Anda dapat mengubah atau menghapus

kebijakan setelah tabel dihapus. Anda juga dapat mengubah atau menghapus kebijakan aliran yang dinonaktifkan.

AWS CLI

Contoh kebijakan IAM berikut menggunakan `put-resource-policy` AWS CLI perintah untuk melampirkan kebijakan berbasis sumber daya ke aliran tabel bernama. *`MusicCollection`* Contoh ini memungkinkan pengguna *`John`* untuk melakukan tindakan GetRecordsGetShardIterator,, dan DescribeStreamAPI di stream.

Ingatlah untuk mengganti *`italicized`* teks dengan informasi spesifik sumber daya Anda.

```
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 Management Console

1. Masuk ke AWS Management Console dan buka konsol DynamoDB di. https://console.aws.amazon.com/dynamodb/

2. Pada dasbor konsol DynamoDB, pilih Tabel dan kemudian pilih tabel yang ada.

   Pastikan tabel yang Anda pilih memiliki aliran yang diaktifkan. Untuk informasi tentang mengaktifkan aliran untuk tabel, lihatMengaktifkan aliran.

3. Pilih tab Izin.

4. Dalam kebijakan berbasis sumber daya untuk aliran aktif, pilih Buat kebijakan aliran.

5. Di editor kebijakan berbasis Sumber Daya, tambahkan kebijakan untuk menentukan izin akses untuk aliran. Dalam kebijakan ini, Anda menentukan siapa yang memiliki akses ke aliran dan tindakan yang diizinkan untuk dilakukan di streaming. Untuk menambahkan kebijakan, lakukan salah satu hal berikut:

   - Ketik atau tempel dokumen kebijakan JSON. Untuk detail tentang bahasa kebijakan IAM, lihat Membuat kebijakan menggunakan editor JSON di Panduan Pengguna IAM.

     > **Tip**
     >
     > Untuk melihat contoh kebijakan berbasis sumber daya di Panduan Pengembang Amazon DynamoDB, pilih Contoh kebijakan.

   - Pilih Tambahkan pernyataan baru untuk menambahkan pernyataan baru dan masukkan informasi di bidang yang disediakan. Ulangi langkah ini sebanyak jumlah pernyataan yang ingin Anda tambahkan.

   > **Important**
   >
   > Pastikan Anda menyelesaikan peringatan, kesalahan, atau saran keamanan apa pun sebelum menyimpan kebijakan Anda.

6. (Opsional) Pilih Pratinjau akses eksternal di sudut kanan bawah untuk melihat bagaimana kebijakan baru memengaruhi publik dan akses lintas akun ke sumber daya Anda. Sebelum Anda menyimpan kebijakan Anda, Anda dapat memeriksa apakah itu memperkenalkan temuan Penganalisa Akses IAM baru atau menyelesaikan temuan yang ada. Jika Anda tidak melihat penganalisis aktif, pilih Buka Penganalisis Akses untuk membuat penganalisis akun di Penganalisis Akses IAM. Untuk informasi selengkapnya, lihat Pratinjau akses.

7. Pilih Buat kebijakan.

Contoh kebijakan IAM berikut melampirkan kebijakan berbasis sumber daya ke aliran tabel bernama. *MusicCollection* Contoh ini memungkinkan pengguna *John* untuk melakukan tindakan GetRecordsGetShardIterator,, dan DescribeStreamAPI di stream.

Ingatlah untuk mengganti *italicized* teks dengan informasi spesifik sumber daya Anda.

```
{
  "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"
      ]
    }
  ]
}
```

Menghapus kebijakan berbasis sumber daya dari tabel DynamoDB

Anda dapat menghapus kebijakan berbasis sumber daya dari tabel yang ada dengan menggunakan konsol DynamoDB, DeleteResourcePolicyAPI, SDK, atau templat. AWS CLI AWS AWS CloudFormation

AWS CLI

Contoh berikut menggunakan `delete-resource-policy` AWS CLI perintah untuk menghapus kebijakan berbasis sumber daya dari tabel bernama. *MusicCollection*

Ingatlah untuk mengganti *italicized* teks dengan informasi spesifik sumber daya Anda.

```
aws dynamodb delete-resource-policy \
```

```
    --resource-arn arn:aws:dynamodb:us-west-2:123456789012:table/MusicCollection
```

AWS Management Console

1. Masuk ke AWS Management Console dan buka konsol DynamoDB di. https://console.aws.amazon.com/dynamodb/
2. Pada dasbor konsol DynamoDB, pilih Tabel dan kemudian pilih tabel yang ada.
3. Pilih Izin.
4. Dari menu tarik-turun Kelola kebijakan, pilih Hapus kebijakan.
5. Di kotak dialog Hapus kebijakan berbasis sumber daya untuk tabel, ketik **`confirm`** untuk mengonfirmasi tindakan penghapusan.
6. Pilih Hapus.

Akses lintas akun dengan kebijakan berbasis sumber daya di DynamoDB

Dengan menggunakan kebijakan berbasis sumber daya, Anda dapat memberikan akses lintas akun ke sumber daya yang tersedia di berbagai sumber daya. Akun AWS Semua akses lintas akun yang diizinkan oleh kebijakan berbasis sumber daya akan dilaporkan melalui temuan akses eksternal IAM Access Analyzer jika Anda memiliki penganalisis yang sama dengan sumber daya. Wilayah AWS Penganalisis Akses IAM menjalankan pemeriksaan kebijakan untuk memvalidasi kebijakan Anda terhadap tata bahasa kebijakan IAM dan praktik terbaik. Pemeriksaan ini menghasilkan temuan dan memberikan rekomendasi yang dapat ditindaklanjuti untuk membantu Anda membuat kebijakan yang fungsional dan sesuai dengan praktik terbaik keamanan. Anda dapat melihat temuan aktif dari IAM Access Analyzer di tab Permissions konsol DynamoDB.

Untuk informasi tentang memvalidasi kebijakan menggunakan IAM Access Analyzer, lihat validasi kebijakan IAM Access Analyzer di Panduan Pengguna IAM. Untuk melihat daftar peringatan, kesalahan, dan saran yang ditampilkan oleh Penganalisis Akses IAM, lihat referensi pemeriksaan kebijakan Penganalisis Akses IAM.

Untuk memberikan GetItemizin kepada pengguna A di akun A untuk mengakses tabel B di akun B, lakukan langkah-langkah berikut:

1. Lampirkan kebijakan berbasis sumber daya ke tabel B yang memberikan izin kepada pengguna A untuk melakukan tindakan. `GetItem`
2. Lampirkan kebijakan berbasis identitas ke pengguna A yang memberinya izin untuk melakukan tindakan pada `GetItem` tabel B.

Menggunakan opsi Pratinjau akses eksternal yang tersedia di konsol DynamoDB, Anda dapat melihat pratinjau bagaimana kebijakan baru memengaruhi akses publik dan lintas akun ke sumber daya Anda. Sebelum Anda menyimpan kebijakan Anda, Anda dapat memeriksa apakah itu memperkenalkan temuan Penganalisa Akses IAM baru atau menyelesaikan temuan yang ada. Jika Anda tidak melihat penganalisis aktif, pilih Buka Penganalisis Akses untuk membuat penganalisis akun di Penganalisis Akses IAM. Untuk informasi selengkapnya, lihat Pratinjau akses.

Parameter nama tabel di bidang data DynamoDB dan bidang kontrol APIs menerima Nama Sumber Daya Amazon (ARN) lengkap dari tabel untuk mendukung operasi lintas akun. Jika Anda hanya memberikan parameter nama tabel alih-alih ARN lengkap, operasi API akan dilakukan pada tabel di akun tempat pemohon berada. Untuk contoh kebijakan yang menggunakan akses lintas akun, lihatKebijakan berbasis sumber daya untuk akses lintas akun.

Akun pemilik sumber daya akan dikenakan biaya bahkan ketika kepala sekolah dari akun lain membaca dari atau menulis ke tabel DynamoDB di akun pemilik. Jika tabel telah menyediakan throughput, jumlah semua permintaan dari akun pemilik dan pemohon di akun lain akan menentukan apakah permintaan akan dibatasi (jika penskalaan otomatis dinonaktifkan) atau diperkecil jika penskalaan otomatis diaktifkan.

Permintaan akan dicatat di CloudTrail log pemilik dan akun pemohon sehingga masing-masing dari dua akun dapat melacak akun mana yang mengakses data apa.

> **Note**
>
> Akses lintas akun pesawat kontrol APIs memiliki batas transaksi per detik (TPS) yang lebih rendah dari 500 permintaan.

Memblokir akses publik dengan kebijakan berbasis sumber daya di DynamoDB

Blokir Akses Publik (BPA) adalah fitur yang mengidentifikasi dan mencegah melampirkan kebijakan berbasis sumber daya yang memberikan akses publik ke tabel, indeks, atau aliran DynamoDB Anda di seluruh akun Amazon Web Services () Anda.AWS Dengan BPA, Anda dapat mencegah akses publik ke sumber daya DynamoDB Anda. BPA melakukan pemeriksaan selama pembuatan atau modifikasi kebijakan berbasis sumber daya dan membantu meningkatkan postur keamanan Anda dengan DynamoDB.

BPA menggunakan penalaran otomatis untuk menganalisis akses yang diberikan oleh kebijakan berbasis sumber daya Anda dan memberi tahu Anda jika izin tersebut ditemukan pada saat mengelola kebijakan berbasis sumber daya. Analisis memverifikasi akses di semua pernyataan kebijakan berbasis sumber daya, tindakan, dan kumpulan kunci kondisi yang digunakan dalam kebijakan Anda.

> **Important**
>
> BPA membantu melindungi sumber daya Anda dengan mencegah akses publik diberikan melalui kebijakan berbasis sumber daya yang langsung dilampirkan ke sumber daya DynamoDB Anda, seperti tabel, indeks, dan aliran. Selain menggunakan BPA, hati-hati memeriksa kebijakan berikut untuk mengonfirmasi bahwa mereka tidak memberikan akses publik:
>
> - Kebijakan berbasis identitas yang dilampirkan pada AWS prinsipal terkait (misalnya, peran IAM)
> - Kebijakan berbasis sumber daya yang dilampirkan pada AWS sumber daya terkait (misalnya, kunci (KMS AWS Key Management Service))

Anda harus memastikan bahwa prinsipal tidak menyertakan * entri atau bahwa salah satu kunci kondisi yang ditentukan membatasi akses dari prinsipal ke sumber daya. Jika kebijakan berbasis sumber daya memberikan akses publik ke tabel, indeks, atau streaming Anda, Akun AWS DynamoDB akan memblokir Anda dari membuat atau memodifikasi kebijakan hingga spesifikasi dalam kebijakan diperbaiki dan dianggap non-publik.

Anda dapat membuat kebijakan non-publik dengan menentukan satu atau beberapa prinsip di dalam blok. `Principal` Contoh kebijakan berbasis sumber daya berikut memblokir akses publik dengan menentukan dua prinsip.

```
{
  "Effect": "Allow",
  "Principal": {
    "AWS": [
      "123456789012",
      "111122223333"
    ]
  },
  "Action": "dynamodb:*",
```

```
  "Resource": "*"
}
```

Kebijakan yang membatasi akses dengan menentukan kunci kondisi tertentu juga tidak dianggap publik. Seiring dengan evaluasi prinsipal yang ditentukan dalam kebijakan berbasis sumber daya, kunci kondisi tepercaya berikut digunakan untuk menyelesaikan evaluasi kebijakan berbasis sumber daya untuk akses non-publik:

- `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`

Selain itu, agar kebijakan berbasis sumber daya bersifat non-publik, nilai untuk Amazon Resource Name (ARN) dan kunci string tidak boleh berisi wildcard atau variabel. Jika kebijakan berbasis sumber daya menggunakan `aws:PrincipalIsAWSService` kunci, Anda harus memastikan bahwa Anda telah menetapkan nilai kunci ke true.

Kebijakan berikut membatasi akses ke pengguna `John` di akun yang ditentukan. Kondisi tersebut membuat `Principal` terkendala dan tidak dianggap publik.

```
{
```

```
  "Effect": "Allow",
  "Principal": {
    "AWS": "*"
  },
  "Action": "dynamodb:*",
  "Resource": "*",
  "Condition": {
    "StringEquals": {
      "aws:PrincipalArn": "arn:aws:iam::123456789012:user/John"
    }
  }
}
```

Contoh berikut dari kebijakan berbasis sumber daya non-publik kendala menggunakan operator. `sourceVPC` `StringEquals`

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "AWS": "*"
      },
      "Action": "dynamodb:*",
      "Resource": "arn:aws:dynamodb:us-west-2:123456789012:table/MusicCollection",
      "Condition": {
        "StringEquals": {
          "aws:SourceVpc": [
            "vpc-91237329"
          ]
        }
      }
    }
  ]
}
```

Operasi API DynamoDB didukung oleh kebijakan berbasis sumber daya

Topik ini mencantumkan operasi API yang didukung oleh kebijakan berbasis sumber daya. Namun, untuk akses lintas akun, Anda hanya dapat menggunakan serangkaian APIs DynamoDB tertentu melalui kebijakan berbasis sumber daya. Anda tidak dapat melampirkan kebijakan berbasis sumber

daya ke jenis sumber daya, seperti pencadangan dan impor. Tindakan IAM, yang sesuai dengan APIs operasi pada jenis sumber daya ini, dikecualikan dari tindakan IAM yang didukung dalam kebijakan berbasis sumber daya. Karena administrator tabel mengonfigurasi setelan tabel internal dalam akun yang sama APIs, seperti UpdateTimeToLivedan DisableKinesisStreamingDestination, tidak mendukung akses lintas akun melalui kebijakan berbasis sumber daya.

Bidang data DynamoDB dan APIs bidang kontrol yang mendukung akses lintas akun juga mendukung overloading nama tabel, yang memungkinkan Anda menentukan ARN tabel alih-alih nama tabel. Anda dapat menentukan tabel ARN dalam `TableName` parameter ini. APIs Namun, tidak semua ini APIs mendukung akses lintas akun.

Topik

- Operasi API bidang data
- Operasi API PartiQL
- Kontrol operasi API pesawat
- Versi 2019.11.21 (Saat ini) operasi API tabel global
- Versi 2017.11.29 (Legacy) tabel global operasi API
- Tag operasi API
- Operasi Backup dan Restore API
- Operasi API Pencadangan/Pemulihan Berkelanjutan (PITR)
- Operasi API Contributor Insights
- Ekspor operasi API
- Impor operasi API
- Operasi API Amazon Kinesis Data Streams
- Operasi API kebijakan berbasis sumber daya
- Time-to-Live Operasi API
- Operasi API lainnya
- Operasi API DynamoDB Streams

Operasi API bidang data

Tabel berikut mencantumkan dukungan API-level yang disediakan oleh operasi API bidang data untuk kebijakan berbasis sumber daya dan akses lintas akun.

Data Plane - Tabel/indeks APIs	Dukungan kebijakan berbasis sumber daya	Dukungan lintas akun
DeleteItem	Ya	Ya
GetItem	Ya	Ya
PutItem	Ya	Ya
Kueri	Ya	Ya
Scan	Ya	Ya
UpdateItem	Ya	Ya
TransactGetItems	Ya	Ya
TransactWriteItems	Ya	Ya
BatchGetItem	Ya	Ya
BatchWriteItem	Ya	Ya

Operasi API PartiQL

Tabel berikut mencantumkan dukungan API-level yang disediakan oleh operasi API PartiQL untuk kebijakan berbasis sumber daya dan akses lintas akun.

PartiQL APIs	Dukungan kebijakan berbasis sumber daya	Dukungan lintas akun
BatchExecuteStatement	Ya	Tidak
ExecuteStatement	Ya	Tidak
ExecuteTransaction	Ya	Tidak

Kontrol operasi API pesawat

Tabel berikut mencantumkan dukungan API-level yang disediakan oleh operasi API bidang kontrol untuk kebijakan berbasis sumber daya dan akses lintas akun.

Control Plane - Tabel APIs	Dukungan kebijakan berbasis sumber daya	Dukungan lintas akun
CreateTable	Tidak	Tidak
DeleteTable	Ya	Ya
DescribeTable	Ya	Ya
UpdateTable	Ya	Ya

Versi 2019.11.21 (Saat ini) operasi API tabel global

Tabel berikut mencantumkan dukungan API-level yang disediakan oleh Versi 2019.11.21 (Saat ini) operasi API tabel global untuk kebijakan berbasis sumber daya dan akses lintas akun.

Versi 2019.11.21 (Saat ini) tabel global APIs	Dukungan kebijakan berbasis sumber daya	Dukungan lintas akun
DescribeTableReplicaAutoScaling	Ya	Tidak
UpdateTableReplicaAutoScaling	Ya	Tidak

Versi 2017.11.29 (Legacy) tabel global operasi API

Tabel berikut mencantumkan dukungan API-level yang disediakan oleh Version 2017.11.29 (Legacy) operasi API tabel global untuk kebijakan berbasis sumber daya dan akses lintas akun.

Versi 2017.11.29 (Legacy) tabel global APIs	Dukungan kebijakan berbasis sumber daya	Dukungan lintas akun
CreateGlobalTable	Tidak	Tidak
DescribeGlobalTable	Tidak	Tidak
DescribeGlobalTableSettings	Tidak	Tidak
ListGlobalTables	Tidak	Tidak
UpdateGlobalTable	Tidak	Tidak
UpdateGlobalTableSettings	Tidak	Tidak

Tag operasi API

Tabel berikut mencantumkan dukungan API-level yang disediakan oleh operasi API yang terkait dengan tag untuk kebijakan berbasis sumber daya dan akses lintas akun.

Tag APIs	Dukungan kebijakan berbasis sumber daya	Dukungan lintas akun
ListTagsOfResource	Ya	Ya
TagResource	Ya	Ya
UntagResource	Ya	Ya

Operasi Backup dan Restore API

Tabel berikut mencantumkan dukungan API-level yang disediakan oleh operasi API yang terkait dengan pencadangan dan pemulihan untuk kebijakan berbasis sumber daya dan akses lintas akun.

Backup dan Restore APIs	Dukungan kebijakan berbasis sumber daya	Dukungan lintas akun
CreateBackup	Ya	Tidak

Backup dan Restore APIs	Dukungan kebijakan berbasis sumber daya	Dukungan lintas akun
DescribeBackup	Tidak	Tidak
DeleteBackup	Tidak	Tidak
RestoreTableFromBackup	Tidak	Tidak

Operasi API Pencadangan/Pemulihan Berkelanjutan (PITR)

Tabel berikut mencantumkan dukungan API-level yang disediakan oleh operasi API yang terkait dengan Continuous Backup/Restore (PITR) untuk kebijakan berbasis sumber daya dan akses lintas akun.

Pencadangan/Pemulihan Berkelanjutan (PITR) APIs	Dukungan kebijakan berbasis sumber daya	Dukungan lintas akun
DescribeContinuousBackups	Ya	Tidak
RestoreTableToPointInTime	Ya	Tidak
UpdateContinuousBackups	Ya	Tidak

Operasi API Contributor Insights

Tabel berikut mencantumkan dukungan API-level yang disediakan oleh operasi API yang terkait dengan Continuous Backup/Restore (PITR) untuk kebijakan berbasis sumber daya dan akses lintas akun.

Wawasan Kontributor APIs	Dukungan kebijakan berbasis sumber daya	Dukungan lintas akun
DescribeContributorInsights	Ya	Tidak
ListContributorInsights	Tidak	Tidak

Wawasan Kontributor APIs	Dukungan kebijakan berbasis sumber daya	Dukungan lintas akun
UpdateContributorInsights	Ya	Tidak

Ekspor operasi API

Tabel berikut mencantumkan dukungan API-level yang disediakan oleh operasi Export API untuk kebijakan berbasis sumber daya dan akses lintas akun.

Ekspor APIs	Dukungan kebijakan berbasis sumber daya	Dukungan lintas akun
DescribeExport	Tidak	Tidak
ExportTableToPointInTime	Ya	Tidak
ListExports	Tidak	Tidak

Impor operasi API

Tabel berikut mencantumkan dukungan API-level yang disediakan oleh operasi Impor API untuk kebijakan berbasis sumber daya dan akses lintas akun.

Impor APIs	Dukungan kebijakan berbasis sumber daya	Dukungan lintas akun
DescribeImport	Tidak	Tidak
ImportTable	Tidak	Tidak
ListImports	Tidak	Tidak

Operasi API Amazon Kinesis Data Streams

Tabel berikut mencantumkan dukungan API-level yang disediakan oleh operasi Kinesis Data Streams API untuk kebijakan berbasis sumber daya dan akses lintas akun.

Kinesis APIs	Dukungan kebijakan berbasis sumber daya	Dukungan lintas akun
DescribeKinesisStreamingDestination	Ya	Tidak
DisableKinesisStreamingDestination	Ya	Tidak
EnableKinesisStreamingDestination	Ya	Tidak
UpdateKinesisStreamingDestination	Ya	Tidak

Operasi API kebijakan berbasis sumber daya

Tabel berikut mencantumkan dukungan API-level yang disediakan oleh operasi API kebijakan berbasis sumber daya untuk kebijakan berbasis sumber daya dan akses lintas akun.

Kebijakan berbasis sumber daya APIs	Dukungan kebijakan berbasis sumber daya	Dukungan lintas akun
GetResourcePolicy	Ya	Tidak
PutResourcePolicy	Ya	Tidak
DeleteResourcePolicy	Ya	Tidak

Time-to-Live Operasi API

Tabel berikut mencantumkan dukungan API-level yang disediakan oleh operasi API time to live (TTL) untuk kebijakan berbasis sumber daya dan akses lintas akun.

TTL APIs	Dukungan kebijakan berbasis sumber daya	Dukungan lintas akun
DescribeTimeToLive	Ya	Tidak
UpdateTimeToLive	Ya	Tidak

Operasi API lainnya

Tabel berikut mencantumkan dukungan API-level yang disediakan oleh operasi API lainnya untuk kebijakan berbasis sumber daya dan akses lintas akun.

Lainnya APIs	Dukungan kebijakan berbasis sumber daya	Dukungan lintas akun
DescribeLimits	Tidak	Tidak
DescribeEndpoints	Tidak	Tidak
ListBackups	Tidak	Tidak
ListTables	Tidak	Tidak

Operasi API DynamoDB Streams

Tabel berikut mencantumkan dukungan tingkat API dari APIs DynamoDB Streams untuk kebijakan berbasis sumber daya dan akses lintas akun.

DynamoDB Streams APIs	Dukungan kebijakan berbasis sumber daya	Dukungan lintas akun
DescribeStream	Ya	Ya
GetRecords	Ya	Ya
GetShardIterator	Ya	Ya
ListStreams	Tidak	Tidak

Otorisasi dengan kebijakan berbasis identitas IAM dan kebijakan berbasis sumber daya DynamoDB

Kebijakan berbasis identitas dilampirkan pada identitas, seperti pengguna IAM, grup pengguna, dan peran. Ini adalah dokumen kebijakan IAM yang mengontrol tindakan apa yang dapat dilakukan identitas, pada sumber daya mana, dan dalam kondisi apa. Kebijakan berbasis identitas dapat dikelola atau kebijakan inline.

Kebijakan berbasis sumber daya adalah dokumen kebijakan IAM yang Anda lampirkan ke sumber daya, seperti tabel DynamoDB. Kebijakan ini memberikan izin prinsipal yang ditentukan untuk melakukan tindakan tertentu pada sumber daya tersebut dan menetapkan di bawah ketentuan yang berlaku. Misalnya, kebijakan berbasis sumber daya untuk tabel DynamoDB juga menyertakan indeks yang terkait dengan tabel. Kebijakan berbasis sumber daya merupakan kebijakan inline. Tidak ada kebijakan berbasis sumber daya terkelola.

Untuk informasi selengkapnya tentang kebijakan ini, lihat Kebijakan berbasis identitas dan kebijakan berbasis sumber daya di Panduan Pengguna IAM.

Jika prinsipal IAM berasal dari akun yang sama dengan pemilik sumber daya, kebijakan berbasis sumber daya cukup untuk menentukan izin akses ke sumber daya. Anda masih dapat memilih untuk memiliki kebijakan berbasis identitas IAM bersama dengan kebijakan berbasis sumber daya. Untuk akses lintas akun, Anda harus secara eksplisit mengizinkan akses dalam kebijakan identitas dan sumber daya sebagaimana ditentukan dalam. Akses lintas akun dengan kebijakan berbasis sumber daya di DynamoDB Saat Anda menggunakan kedua jenis kebijakan tersebut, kebijakan dievaluasi seperti yang dijelaskan dalam Menentukan apakah permintaan diizinkan atau ditolak dalam akun.

Contoh kebijakan berbasis sumber daya DynamoDB

Saat Anda menentukan ARN di `Resource` bidang kebijakan berbasis sumber daya, kebijakan akan berlaku hanya jika ARN yang ditentukan cocok dengan ARN sumber daya DynamoDB yang dilampirkan.

> **Note**
>
> Ingatlah untuk mengganti *`italicized`* teks dengan informasi spesifik sumber daya Anda.

Kebijakan berbasis sumber daya untuk tabel

Kebijakan berbasis sumber daya berikut yang dilampirkan ke tabel DynamoDB bernama*MusicCollection*, memberikan pengguna IAM dan izin untuk melakukan *John* dan *Jane* tindakan pada sumber daya. GetItemBatchGetItem*MusicCollection*

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "1111",
        "Effect": "Allow",
        "Principal": {
          "AWS": [
            "arn:aws:iam::111122223333:user/John",
            "arn:aws:iam::111122223333:user/Jane"
          ]
        },
        "Action": [
          "dynamodb:GetItem",
          "dynamodb:BatchGetItem"
        ],
        "Resource": [
          "arn:aws:dynamodb:us-west-2:123456789012:table/MusicCollection"
        ]
    }
  ]
}
```

Kebijakan berbasis sumber daya untuk aliran

Kebijakan berbasis sumber daya berikut yang dilampirkan ke aliran DynamoDB bernama `2024-02-12T18:57:26.492` memberikan pengguna IAM *John* dan *Jane* izin untuk melakukan GetRecords, serta tindakan API pada sumber daya. GetShardIteratorDescribeStream`2024-02-12T18:57:26.492`

```
{
  "Version": "2012-10-17",
```

```
  "Statement": [
    {
      "Sid": "1111",
      "Effect": "Allow",
      "Principal": {
        "AWS": [
          "arn:aws:iam::111122223333:user/John",
          "arn:aws:iam::111122223333:user/Jane"
        ]
      },
      "Action": [
        "dynamodb:DescribeStream",
        "dynamodb:GetRecords",
        "dynamodb:GetShardIterator"
      ],
      "Resource": [
        "arn:aws:dynamodb:us-west-2:123456789012:table/MusicCollection/
stream/2024-02-12T18:57:26.492"
      ]
    }
  ]
}
```

Kebijakan berbasis sumber daya untuk akses untuk melakukan semua tindakan pada sumber daya tertentu

Untuk memungkinkan pengguna melakukan semua tindakan pada tabel dan semua indeks terkait dengan tabel, Anda dapat menggunakan wildcard (*) untuk mewakili tindakan dan sumber daya yang terkait dengan tabel. Menggunakan karakter wild card untuk sumber daya, akan memungkinkan pengguna mengakses tabel DynamoDB dan semua indeks terkait, termasuk yang belum dibuat. Misalnya, kebijakan berikut akan memberikan *John* izin kepada pengguna untuk melakukan tindakan apa pun di atas *MusicCollection* meja dan semua indeksnya, termasuk indeks apa pun yang akan dibuat di masa mendatang.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "1111",
      "Effect": "Allow",
```

```
      "Principal":"arn:aws:iam::111122223333:user/John",
      "Action": "dynamodb:*",
      "Resource": [
        "arn:aws:dynamodb:us-west-2:123456789012:table/MusicCollection",
        "arn:aws:dynamodb:us-west-2:123456789012:table/MusicCollection/index/*"
      ]
    }
  ]
}
```

Kebijakan berbasis sumber daya untuk akses lintas akun

Anda dapat menentukan izin untuk identitas IAM lintas akun untuk mengakses sumber daya DynamoDB. Misalnya, Anda mungkin memerlukan pengguna dari akun tepercaya untuk mendapatkan akses membaca konten tabel Anda, dengan syarat bahwa mereka hanya mengakses item tertentu dan atribut tertentu dalam item tersebut. Kebijakan berikut memungkinkan akses ke pengguna *John* dari Akun AWS ID tepercaya *111111111111* untuk mengakses data dari tabel di akun *123456789012* menggunakan GetItemAPI. Kebijakan ini memastikan bahwa pengguna hanya dapat mengakses item dengan kunci utama *Jane* dan pengguna hanya dapat mengambil atribut `Artist` dan`SongTitle`, tetapi tidak ada atribut lainnya.

> **Important**
>
> Jika Anda tidak menentukan `SPECIFIC_ATTRIBUTES` kondisi, Anda akan melihat semua atribut untuk item yang dikembalikan.

```
{
    "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"
                }
            }
        }
    ]
}
```

Selain kebijakan berbasis sumber daya sebelumnya, kebijakan berbasis identitas yang dilampirkan pada pengguna *John* juga perlu mengizinkan tindakan API agar akses lintas akun berfungsi. `GetItem` Berikut ini adalah contoh kebijakan berbasis identitas yang harus Anda lampirkan ke pengguna. *John*

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "CrossAccountIdentityBasedPolicy",
            "Effect": "Allow",
            "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"
                }
            }
        }
    ]
}
```

Pengguna John dapat membuat `GetItem` permintaan dengan menentukan tabel ARN dalam `table-name` parameter untuk mengakses *`MusicCollection`* tabel di akun. *`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
```

Kebijakan berbasis sumber daya dengan kondisi alamat IP

Anda dapat menerapkan kondisi untuk membatasi alamat IP sumber, virtual private cloud (VPCs), dan titik akhir VPC (VPCE). Anda dapat menentukan izin berdasarkan alamat sumber permintaan asal. Misalnya, Anda mungkin ingin mengizinkan pengguna mengakses sumber daya DynamoDB hanya jika mereka diakses dari sumber IP tertentu, seperti titik akhir VPN perusahaan. Tentukan alamat IP ini dalam `Condition` pernyataan.

Contoh berikut memungkinkan pengguna *`John`* mengakses sumber daya DynamoDB apa pun saat sumbernya IPs dan. `54.240.143.0/24` `2001:DB8:1234:5678::/64`

```
{
  "Id":"PolicyId2",
  "Version":"2012-10-17",
  "Statement":[
    {
      "Sid":"AllowIPmix",
      "Effect":"Allow",
      "Principal":"arn:aws:iam::111111111111:user/John",
      "Action":"dynamodb:*",
      "Resource":"*",
      "Condition": {
        "IpAddress": {
          "aws:SourceIp": [
            "54.240.143.0/24",
```

```
            "2001:DB8:1234:5678::/64"
          ]
        }
      }
    }
  ]
}
```

Anda juga dapat menolak semua akses ke sumber daya DynamoDB kecuali jika sumbernya adalah titik akhir VPC tertentu, misalnya. *vpce-1a2b3c4d*

```
{
  "Id":"PolicyId",
  "Version":"2012-10-17",
  "Statement": [
    {
      "Sid": "AccessToSpecificVPCEOnly",
      "Principal": "*",
      "Action": "dynamodb:*",
      "Effect": "Deny",
      "Resource": "*",
      "Condition": {
        "StringNotEquals":{
          "aws:sourceVpce":"vpce-1a2b3c4d"
        }
      }
    }
  ]
}
```

Kebijakan berbasis sumber daya menggunakan peran IAM

Anda juga dapat menentukan peran layanan IAM dalam kebijakan berbasis sumber daya. Entitas IAM yang mengambil peran ini dibatasi oleh tindakan yang diizinkan yang ditentukan untuk peran dan kumpulan sumber daya tertentu dalam kebijakan berbasis sumber daya.

Contoh berikut memungkinkan entitas IAM untuk melakukan semua tindakan DynamoDB pada sumber daya dan DynamoDB. *MusicCollection* *MusicCollection*

```
{
  "Version": "2012-10-17",
```

```
  "Statement": [
    {
      "Sid": "1111",
      "Effect": "Allow",
      "Principal": { "AWS": "arn:aws:iam::111122223333:role/John" },
      "Action": "dynamodb:*",
      "Resource": [
        "arn:aws:dynamodb:us-west-2:123456789012:table/MusicCollection",
        "arn:aws:dynamodb:us-west-2:123456789012:table/MusicCollection/*"
      ]
    }
  ]
}
```

Pertimbangan kebijakan berbasis sumber daya DynamoDB

Saat Anda menentukan kebijakan berbasis sumber daya untuk sumber daya DynamoDB Anda, pertimbangan berikut berlaku:

Pertimbangan umum

- Ukuran maksimum yang didukung untuk dokumen kebijakan berbasis sumber daya adalah 20 KB. DynamoDB menghitung spasi putih saat menghitung ukuran kebijakan terhadap batas ini.
- Pembaruan kebijakan berikutnya untuk sumber daya tertentu diblokir selama 15 detik setelah pembaruan kebijakan berhasil untuk sumber daya yang sama.
- Saat ini, Anda hanya dapat melampirkan kebijakan berbasis sumber daya ke aliran yang ada. Anda tidak dapat melampirkan kebijakan ke aliran saat membuatnya.

Pertimbangan tabel global

- Kebijakan berbasis sumber daya tidak didukung untuk replika versi tabel Global 2017.11.29 (Legacy).
- Dalam kebijakan berbasis sumber daya, jika tindakan untuk DynamoDB service-linked role (SLR) untuk mereplikasi data untuk tabel global ditolak, menambahkan atau menghapus replika akan gagal dengan kesalahan.
- GlobalTableSumber daya AWS: :DynamoDB:: tidak mendukung pembuatan replika dan menambahkan kebijakan berbasis sumber daya ke replika itu dalam pembaruan tumpukan yang sama di Wilayah selain Wilayah tempat Anda menerapkan pembaruan tumpukan.

Pertimbangan lintas akun

- Akses lintas akun menggunakan kebijakan berbasis sumber daya tidak mendukung tabel terenkripsi dengan kunci AWS terkelola karena Anda tidak dapat memberikan akses lintas akun ke kebijakan KMS terkelola. AWS

AWS CloudFormation pertimbangan

- Kebijakan berbasis sumber daya tidak mendukung deteksi drift. Jika Anda memperbarui kebijakan berbasis sumber daya di luar template AWS CloudFormation tumpukan, Anda harus memperbarui CloudFormation tumpukan dengan perubahan.
- Kebijakan berbasis sumber daya tidak mendukung perubahan di luar band. Jika Anda menambahkan, memperbarui, atau menghapus kebijakan di luar CloudFormation templat, perubahan tidak akan ditimpa jika tidak ada perubahan pada kebijakan di dalam templat.

  Misalnya, katakan bahwa template Anda berisi kebijakan berbasis sumber daya yang kemudian Anda perbarui di luar templat. Jika Anda tidak membuat perubahan apa pun pada kebijakan di template, kebijakan yang diperbarui di DynamoDB tidak akan disinkronkan dengan kebijakan di templat.

  Sebaliknya, katakan bahwa template Anda tidak berisi kebijakan berbasis sumber daya, tetapi Anda menambahkan kebijakan di luar templat. Kebijakan ini tidak akan dihapus dari DynamoDB selama Anda tidak menambahkannya ke template. Saat Anda menambahkan kebijakan ke template dan memperbarui tumpukan, kebijakan yang ada di DynamoDB akan diperbarui agar sesuai dengan yang ditentukan dalam templat.

Praktik terbaik kebijakan berbasis sumber daya DynamoDB

Topik ini menjelaskan praktik terbaik untuk menentukan izin akses untuk sumber daya DynamoDB Anda dan tindakan yang diizinkan pada sumber daya ini.

Sederhanakan kontrol akses ke sumber daya DynamoDB

Jika AWS Identity and Access Management prinsipal yang memerlukan akses ke sumber daya DynamoDB adalah bagian yang Akun AWS sama dengan pemilik sumber daya, kebijakan berbasis identitas IAM tidak diperlukan untuk setiap prinsipal. Kebijakan berbasis sumber daya yang melekat pada sumber daya yang diberikan sudah cukup. Jenis konfigurasi ini menyederhanakan kontrol akses.

Lindungi sumber daya DynamoDB Anda dengan kebijakan berbasis sumber daya

Untuk semua tabel dan aliran DynamoDB, buat kebijakan berbasis sumber daya untuk menerapkan kontrol akses untuk sumber daya ini. Kebijakan berbasis sumber daya memungkinkan Anda memusatkan izin di tingkat sumber daya, menyederhanakan kontrol akses ke tabel DynamoDB, indeks, dan aliran, serta mengurangi overhead administrasi. Jika tidak ada kebijakan berbasis sumber daya yang ditentukan untuk tabel atau aliran, akses ke tabel atau aliran akan ditolak secara implisit, kecuali kebijakan berbasis identitas yang terkait dengan prinsip IAM mengizinkan akses.

Terapkan izin hak istimewa paling sedikit

Saat Anda menetapkan izin dengan kebijakan berbasis sumber daya untuk sumber daya DynamoDB, berikan hanya izin yang diperlukan untuk melakukan tindakan. Anda melakukannya dengan mendefinisikan tindakan yang dapat diambil pada sumber daya tertentu dalam kondisi tertentu, yang juga dikenal sebagai izin dengan hak akses paling rendah. Anda dapat memulai dengan izin luas saat menjelajahi izin yang diperlukan untuk beban kerja atau kasus penggunaan Anda. Saat kasus penggunaan Anda matang, Anda dapat bekerja untuk mengurangi izin yang Anda berikan untuk bekerja menuju hak istimewa yang paling sedikit.

Menganalisis aktivitas akses lintas akun untuk menghasilkan kebijakan hak istimewa paling sedikit

IAM Access Analyzer melaporkan akses lintas akun ke entitas eksternal yang ditentukan dalam kebijakan berbasis sumber daya, dan memberikan visibilitas untuk membantu Anda menyempurnakan izin dan menyesuaikan diri dengan hak istimewa yang paling sedikit. Untuk informasi selengkapnya tentang pembuatan kebijakan, lihat pembuatan kebijakan IAM Access Analyzer.

Gunakan IAM Access Analyzer untuk menghasilkan kebijakan hak istimewa paling sedikit

Untuk hanya memberikan izin yang diperlukan untuk melakukan tugas, Anda dapat membuat kebijakan berdasarkan aktivitas akses yang masuk AWS CloudTrail. IAM Access Analyzer menganalisis layanan dan tindakan yang digunakan kebijakan Anda.

Menggunakan kontrol akses berbasis atribut dengan DynamoDB

Kontrol akses berbasis atribut (ABAC) adalah strategi otorisasi yang menentukan izin akses berdasarkan kondisi tag dalam kebijakan berbasis identitas Anda atau kebijakan lain, seperti

kebijakan berbasis sumber daya dan kebijakan IAM AWS organisasi. Anda dapat melampirkan tag ke tabel DynamoDB, yang kemudian dievaluasi terhadap kondisi berbasis tag. Indeks yang terkait dengan tabel mewarisi tag yang Anda tambahkan ke tabel. Anda dapat menambahkan hingga 50 tag untuk setiap tabel DynamoDB. Ukuran maksimum yang didukung untuk semua tag dalam tabel adalah 10 KB. Untuk informasi selengkapnya tentang menandai sumber daya DynamoDB dan pembatasan penandaan, lihat Menandai sumber daya di DynamoDB dan. the section called "Pembatasan penandaan di DynamoDB"

Untuk informasi selengkapnya tentang penggunaan tag untuk mengontrol akses ke AWS sumber daya, lihat topik berikut di Panduan Pengguna IAM:

- Untuk apa ABAC AWS
- Mengontrol akses ke AWS sumber daya menggunakan tag

Menggunakan ABAC, Anda dapat menerapkan tingkat akses yang berbeda untuk tim dan aplikasi Anda untuk melakukan tindakan pada tabel DynamoDB menggunakan kebijakan yang lebih sedikit. Anda dapat menentukan tag dalam elemen kondisi kebijakan IAM untuk mengontrol akses ke tabel atau indeks DynamoDB Anda. Kondisi ini menentukan tingkat akses yang dimiliki oleh prinsipal IAM, pengguna, atau peran, ke tabel dan indeks DynamoDB. Ketika prinsipal IAM membuat permintaan akses ke DynamoDB, tag sumber daya dan identitas dievaluasi terhadap kondisi tag dalam kebijakan IAM. Setelah itu, kebijakan menjadi efektif hanya jika kondisi tag terpenuhi. Ini memungkinkan Anda untuk membuat kebijakan IAM yang secara efektif mengatakan salah satu dari berikut ini:

- Izinkan pengguna untuk mengelola hanya sumber daya yang memiliki tag dengan kunci *X* dan nilai *Y*.
- Tolak akses ke semua pengguna ke sumber daya yang ditandai dengan kunci *X*.

Misalnya, Anda dapat membuat kebijakan yang memungkinkan pengguna memperbarui tabel hanya jika memiliki pasangan nilai kunci tag:. `"environment": "staging"` Anda dapat menggunakan kunci ResourceTag kondisi aws: untuk mengizinkan atau menolak akses ke tabel berdasarkan tag yang dilampirkan ke tabel tersebut.

Anda dapat menyertakan kondisi berbasis atribut saat membuat kebijakan atau yang lebih baru menggunakan AWS Management Console, AWS API, AWS Command Line Interface (AWS CLI), AWS SDK, atau. AWS CloudFormation

Contoh berikut memungkinkan UpdateItemtindakan pada tabel bernama `MusicTable` jika menyertakan kunci tag dengan nama `environment` dan nilai`production`.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "dynamodb:UpdateItem"
      ],
      "Resource": "arn:aws:dynamodb:*:*:table/MusicTable",
      "Condition": {
        "StringEquals": {
          "aws:ResourceTag/environment": "production"
        }
      }
    }
  ]
}
```

Topik

- Mengapa saya harus menggunakan ABAC?
- Kunci kondisi untuk mengimplementasikan ABAC dengan DynamoDB
- Pertimbangan untuk menggunakan ABAC dengan DynamoDB
- Mengaktifkan ABAC di DynamoDB
- Menggunakan ABAC dengan tabel dan indeks DynamoDB
- Contoh untuk menggunakan ABAC dengan tabel dan indeks DynamoDB
- Memecahkan masalah kesalahan ABAC umum untuk tabel dan indeks DynamoDB

Mengapa saya harus menggunakan ABAC?

- Manajemen kebijakan yang lebih sederhana: Anda menggunakan lebih sedikit kebijakan karena Anda tidak perlu membuat kebijakan yang berbeda untuk menentukan tingkat akses untuk setiap prinsipal IAM.
- Kontrol akses yang dapat diskalakan: Menskalakan kontrol akses lebih mudah dengan ABAC karena Anda tidak perlu memperbarui kebijakan saat membuat sumber daya DynamoDB baru.

Anda dapat menggunakan tag untuk mengotorisasi akses ke prinsipal IAM yang berisi tag yang cocok dengan tag sumber daya. Anda dapat menggunakan prinsipal IAM baru atau sumber daya DynamoDB dan menerapkan tag yang sesuai untuk secara otomatis memberikan izin yang diperlukan tanpa harus membuat perubahan kebijakan apa pun.

- Manajemen izin berbutir halus: Merupakan praktik terbaik untuk memberikan hak istimewa paling sedikit saat Anda membuat kebijakan. Menggunakan ABAC, Anda dapat membuat tag untuk prinsipal IAM, dan menggunakannya untuk memberikan akses ke tindakan dan sumber daya tertentu yang cocok dengan tag pada prinsipal IAM.
- Penyelarasan dengan direktori perusahaan: Anda dapat memetakan tag dengan atribut karyawan yang ada dari direktori perusahaan Anda untuk menyelaraskan kebijakan kontrol akses Anda dengan struktur organisasi Anda.

Kunci kondisi untuk mengimplementasikan ABAC dengan DynamoDB

Anda dapat menggunakan kunci kondisi berikut dalam AWS kebijakan Anda untuk mengontrol tingkat akses ke tabel dan indeks DynamoDB Anda:

- aws: ResourceTag /tag-key: Mengontrol akses berdasarkan apakah pasangan nilai kunci tag pada tabel atau indeks DynamoDB cocok dengan kunci tag dan nilai dalam kebijakan. Kunci kondisi ini relevan dengan semua APIs yang beroperasi pada tabel atau indeks yang ada.

  `dynamodb:ResourceTag`Kondisi dievaluasi seolah-olah Anda tidak melampirkan tag apa pun ke sumber daya.
- aws: RequestTag /tag-key: Memungkinkan membandingkan pasangan nilai kunci tag yang diteruskan dalam permintaan dengan pasangan tag yang Anda tentukan dalam kebijakan. Kunci kondisi ini relevan dengan APIs yang berisi tag sebagai bagian dari payload permintaan. Ini APIs termasuk CreateTabledan TagResource.
- aws: TagKeys: Membandingkan kunci tag dalam permintaan dengan kunci yang Anda tentukan dalam kebijakan. Kunci kondisi ini relevan dengan APIs yang berisi tag sebagai bagian dari payload permintaan. Ini APIs termasuk`CreateTable`,`TagResource`, dan`UntagResource`.

Pertimbangan untuk menggunakan ABAC dengan DynamoDB

Bila Anda menggunakan ABAC dengan tabel atau indeks DynamoDB, pertimbangan berikut berlaku:

- Penandaan dan ABAC tidak didukung untuk DynamoDB Streams.

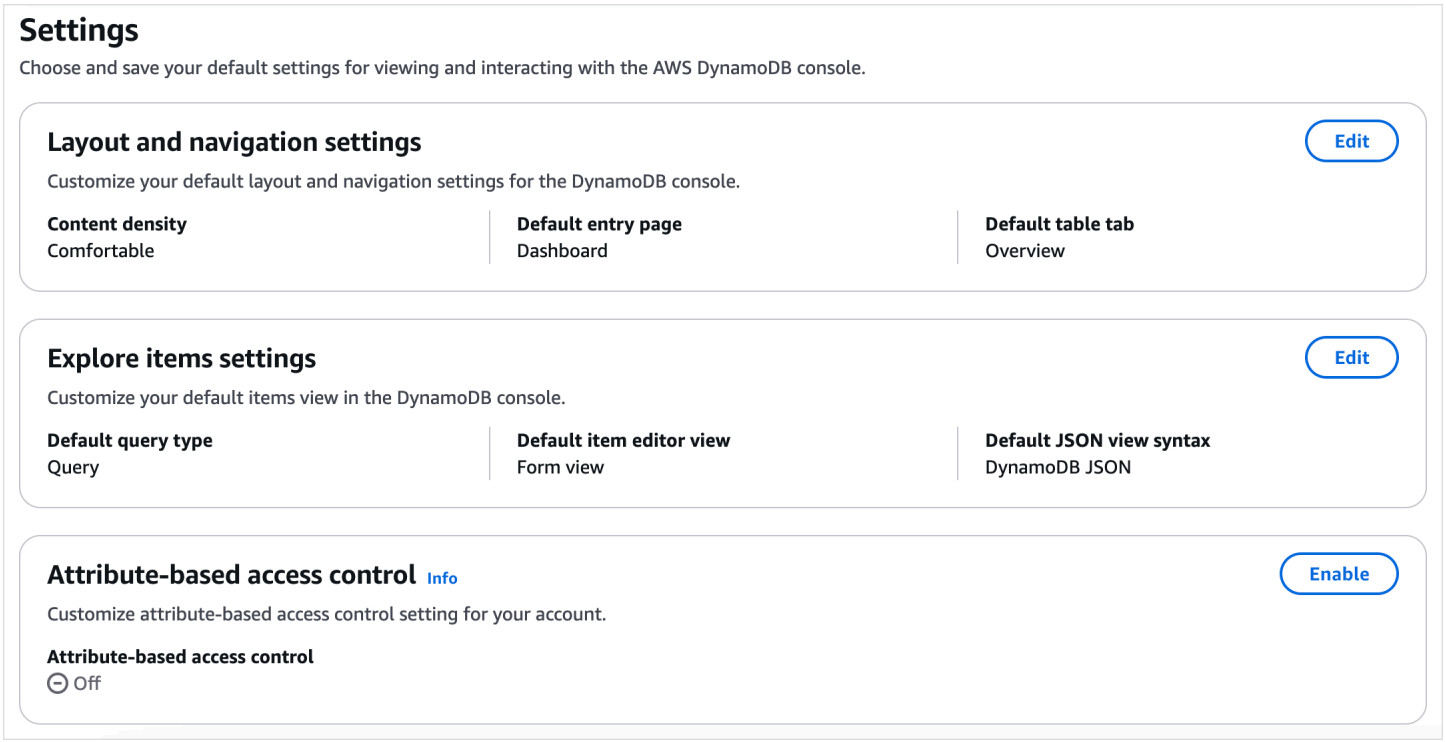
- Penandaan dan ABAC tidak didukung untuk backup DynamoDB. Untuk menggunakan ABAC dengan cadangan, kami sarankan Anda menggunakannya. AWS Backup
- Tag tidak disimpan dalam tabel yang dipulihkan. Anda perlu menambahkan tag ke tabel yang dipulihkan sebelum dapat menggunakan kondisi berbasis tag dalam kebijakan Anda.

Mengaktifkan ABAC di DynamoDB

Untuk sebagian besar Akun AWS, ABAC diaktifkan secara default. Menggunakan konsol DynamoDB, Anda dapat mengonfirmasi apakah ABAC diaktifkan untuk akun Anda. Untuk melakukan ini, pastikan Anda membuka konsol DynamoDB dengan peran yang memiliki izin dynamodb:. GetAbacStatus Kemudian, buka halaman Pengaturan konsol DynamoDB.

Jika Anda tidak melihat kartu kontrol akses berbasis Atribut atau jika kartu menampilkan status Aktif, itu berarti ABAC diaktifkan untuk akun Anda. Namun, jika Anda melihat kartu kontrol akses berbasis Atribut dengan status Mati, seperti yang ditunjukkan pada gambar berikut, ABAC tidak diaktifkan untuk akun Anda.

Kontrol akses berbasis atribut — tidak diaktifkan



ABAC tidak diaktifkan Akun AWS untuk kondisi berbasis tag yang ditentukan dalam kebijakan berbasis identitas atau kebijakan lainnya masih perlu diaudit. Jika ABAC tidak diaktifkan untuk akun

Anda, kondisi berbasis tag dalam kebijakan Anda yang dimaksudkan untuk bertindak pada tabel atau indeks DynamoDB akan dievaluasi seolah-olah tidak ada tag yang ada untuk sumber daya atau permintaan API Anda. Saat ABAC diaktifkan untuk akun Anda, kondisi berbasis tag dalam kebijakan akun Anda akan dievaluasi dengan mempertimbangkan tag yang dilampirkan pada tabel atau permintaan API Anda.

Untuk mengaktifkan ABAC untuk akun Anda, kami sarankan Anda terlebih dahulu mengaudit kebijakan Anda seperti yang dijelaskan di Audit kebijakan bagian ini. Kemudian, sertakan izin yang diperlukan untuk ABAC dalam kebijakan IAM Anda. Terakhir, lakukan langkah-langkah yang dijelaskan Mengaktifkan ABAC di konsol untuk mengaktifkan ABAC untuk akun Anda di Wilayah saat ini. Setelah mengaktifkan ABAC, Anda dapat memilih keluar dalam tujuh hari kalender berikutnya setelah memilih.

Topik

- Mengaudit kebijakan Anda sebelum mengaktifkan ABAC
- Izin IAM diperlukan untuk mengaktifkan ABAC
- Mengaktifkan ABAC di konsol

Mengaudit kebijakan Anda sebelum mengaktifkan ABAC

Sebelum Anda mengaktifkan ABAC untuk akun Anda, audit kebijakan Anda untuk mengonfirmasi bahwa kondisi berbasis tag yang mungkin ada dalam kebijakan dalam akun Anda diatur sebagaimana dimaksud. Mengaudit kebijakan Anda akan membantu menghindari kejutan dari perubahan otorisasi dengan alur kerja DynamoDB Anda setelah ABAC diaktifkan. Untuk melihat contoh penggunaan kondisi berbasis atribut dengan tag, dan perilaku sebelum dan sesudah implementasi ABAC, lihat. Contoh untuk menggunakan ABAC dengan tabel dan indeks DynamoDB

Izin IAM diperlukan untuk mengaktifkan ABAC

Anda memerlukan `dynamodb:UpdateAbacStatus` izin untuk mengaktifkan ABAC untuk akun Anda di Wilayah saat ini. Untuk mengonfirmasi apakah ABAC diaktifkan untuk akun Anda, Anda juga harus memiliki `dynamodb:GetAbacStatus` izin. Dengan izin ini, Anda dapat melihat status ABAC untuk akun di Wilayah mana pun. Anda memerlukan izin ini selain izin yang diperlukan untuk mengakses konsol DynamoDB.

Kebijakan IAM berikut memberikan izin untuk mengaktifkan ABAC dan melihat statusnya untuk akun di Wilayah saat ini.

```
{
"version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "dynamodb:UpdateAbacStatus",
                "dynamodb:GetAbacStatus"
             ],
            "Resource": "*"
        }
    ]
}
```

Mengaktifkan ABAC di konsol

1. Masuk ke AWS Management Console dan buka konsol DynamoDB di. https://console.aws.amazon.com/dynamodb/
2. Dari panel navigasi atas, pilih Wilayah yang ingin Anda aktifkan ABAC.
3. Di panel navigasi kiri, pilih Pengaturan.
4. Pada halaman Pengaturan, lakukan hal berikut:

    a. Di kartu kontrol akses berbasis atribut, pilih Aktifkan.

    b. Di kotak pengaturan kontrol akses berbasis atribut Konfirmasi, pilih Aktifkan untuk mengonfirmasi pilihan Anda.

    Ini memungkinkan ABAC untuk Wilayah saat ini dan kartu kontrol akses berbasis Atribut menunjukkan status Aktif.

    Jika Anda ingin memilih keluar setelah mengaktifkan ABAC di konsol, Anda dapat melakukannya dalam tujuh hari kalender berikutnya setelah memilih. Untuk memilih keluar, pilih Nonaktifkan di kartu kontrol akses berbasis Atribut di halaman Pengaturan.

    > **Note**
    >
    > Memperbarui status ABAC adalah operasi asinkron. Jika tag dalam kebijakan Anda tidak segera dievaluasi, Anda mungkin perlu menunggu beberapa saat karena penerapan perubahan pada akhirnya konsisten.

Menggunakan ABAC dengan tabel dan indeks DynamoDB

Langkah-langkah berikut menunjukkan cara mengatur izin menggunakan ABAC. Dalam skenario contoh ini, Anda akan menambahkan tag ke tabel DynamoDB dan membuat peran IAM dengan kebijakan yang menyertakan kondisi berbasis tag. Kemudian, Anda akan menguji izin yang diizinkan pada tabel DynamoDB dengan mencocokkan kondisi tag.

Topik

- Langkah 1: Tambahkan tag ke tabel DynamoDB
- Langkah 2: Buat peran IAM dengan kebijakan termasuk kondisi berbasis tag
- Langkah 3: Uji izin yang diizinkan

Langkah 1: Tambahkan tag ke tabel DynamoDB

Anda dapat menambahkan tag ke tabel DynamoDB baru atau yang sudah ada menggunakan AWS Management Console AWS , API,AWS CLI() AWS Command Line Interface , SDK AWS , atau. AWS CloudFormation Misalnya, perintah CLI tag-resource berikut menambahkan tag ke tabel bernama. `MusicTable`

```
aws dynamodb tag-resource –resource-arn arn:aws:dynamodb:us-east-1:123456789012:table/
MusicTable –tags Key=environment,Value=staging
```

Langkah 2: Buat peran IAM dengan kebijakan termasuk kondisi berbasis tag

Buat kebijakan IAM menggunakan ResourceTagkunci kondisi aws: /tag-key untuk membandingkan pasangan nilai kunci tag yang ditentukan dalam kebijakan IAM dengan pasangan kunci-nilai yang dilampirkan ke tabel. Contoh kebijakan berikut memungkinkan pengguna untuk menempatkan atau memperbarui item dalam tabel jika tabel ini berisi pasangan nilai kunci tag:. `"environment": "staging"` Jika tabel tidak memiliki pasangan nilai kunci tag yang ditentukan, tindakan ini ditolak.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "dynamodb:PutItem",
```

```
                "dynamodb:UpdateItem"
            ],
            "Resource": "arn:aws:dynamodb:*:*:table/*",
            "Condition": {
                "StringEquals": {
                    "aws:ResourceTag/environment": "staging"
                }
            }
        }
    ]
}
```

Langkah 3: Uji izin yang diizinkan

1. Lampirkan kebijakan IAM ke pengguna pengujian atau peran di Anda Akun AWS. Pastikan bahwa prinsipal IAM yang Anda gunakan belum memiliki akses ke tabel DynamoDB melalui kebijakan yang berbeda.
2. Pastikan tabel DynamoDB Anda berisi `"environment"` kunci tag dengan nilai. `"staging"`
3. Lakukan `dynamodb:PutItem` dan `dynamodb:UpdateItem` tindakan pada tabel yang ditandai. Tindakan ini akan berhasil jika pasangan nilai kunci `"environment": "staging"` tag hadir.

   Jika Anda melakukan tindakan ini pada tabel yang tidak memiliki pasangan nilai kunci `"environment": "staging"` tag, permintaan Anda akan gagal dengan. `AccessDeniedException`

Anda juga dapat meninjau kasus penggunaan sampel lainnya yang dijelaskan di bagian berikut untuk menerapkan ABAC dan melakukan lebih banyak pengujian.

Contoh untuk menggunakan ABAC dengan tabel dan indeks DynamoDB

Contoh berikut menggambarkan beberapa kasus penggunaan untuk mengimplementasikan kondisi berbasis atribut menggunakan tag.

Topik

- Contoh 1: Izinkan tindakan menggunakan aws: ResourceTag
- Contoh 2: Izinkan tindakan menggunakan aws: RequestTag
- Contoh 3: Tolak tindakan menggunakan aws: TagKeys

Contoh 1: Izinkan tindakan menggunakan aws: ResourceTag

Dengan menggunakan kunci `aws:ResourceTag/tag-key` kondisi, Anda dapat membandingkan pasangan nilai kunci tag yang ditentukan dalam kebijakan IAM dengan pasangan nilai kunci yang dilampirkan dalam tabel DynamoDB. Misalnya, Anda dapat mengizinkan tindakan tertentu, seperti PutItem, jika kondisi tag cocok dengan kebijakan IAM dan tabel. Untuk melakukan ini, lakukan langkah-langkah berikut:

Using the AWS CLI

1. Buat tabel. Contoh berikut menggunakan AWS CLI perintah create-table untuk membuat tabel bernama. `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. Tambahkan tag ke tabel ini. Contoh AWS CLI perintah tag-resource berikut menambahkan pasangan nilai kunci tag ke. `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. Buat kebijakan inline dan tambahkan ke peran yang memiliki kebijakan AmazonDynamoDBReadOnlyAccess AWS terkelola yang dilampirkan padanya, seperti yang ditunjukkan pada contoh berikut.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "dynamodb:PutItem",
      "Resource": "arn:aws:dynamodb:*:*:table/*",
      "Condition": {
        "StringEquals": {
          "aws:ResourceTag/Title": "ProductManager"
        }
```

```
      }
    }
  ]
}
```

Kebijakan ini memungkinkan `PutItem` tindakan pada tabel jika kunci tag dan nilai yang dilampirkan ke tabel cocok dengan tag yang ditentukan dalam kebijakan.

4. Asumsikan peran dengan kebijakan yang dijelaskan dalam Langkah 3.
5. Gunakan AWS CLI perintah put-item untuk menempatkan item ke. `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. Pindai tabel untuk memverifikasi apakah item telah ditambahkan ke tabel.

```
aws dynamodb scan --table-name myMusicTable  --region us-east-1
```

Using the AWS SDK for Java 2.x

1. Buat tabel. Contoh berikut menggunakan CreateTableAPI untuk membuat tabel bernama`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. Tambahkan tag ke tabel ini. TagResourceAPI dalam contoh berikut menambahkan pasangan nilai kunci tag `Title: ProductManager` ke. `myMusicTable`

```
TagResourceRequest tagResourceRequest = TagResourceRequest.builder()
    .resourceArn(tableArn)
    .tags(
        Arrays.asList(
            Tag.builder()
            .key("Title")
            .value("ProductManager")
            .build()
        )
    )
    .build();
dynamoDB.tagResource(tagResourceRequest);
```

3. Buat kebijakan inline dan tambahkan ke peran yang memiliki kebijakan AmazonDynamoDBReadOnlyAccess AWS terkelola yang dilampirkan padanya, seperti yang ditunjukkan pada contoh berikut.

```
{
```

```
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "dynamodb:PutItem",
      "Resource": "arn:aws:dynamodb:*:*:table/*",
      "Condition": {
        "StringEquals": {
          "aws:ResourceTag/Title": "ProductManager"
        }
      }
    }
  ]
}
```

Kebijakan ini memungkinkan `PutItem` tindakan pada tabel jika kunci tag dan nilai yang dilampirkan ke tabel cocok dengan tag yang ditentukan dalam kebijakan.

4. Asumsikan peran dengan kebijakan yang dijelaskan dalam Langkah 3.
5. Gunakan PutItemAPI untuk menempatkan item ke file`myMusicTable`.

```
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. Pindai tabel untuk memverifikasi apakah item telah ditambahkan ke tabel.

```
ScanRequest scanRequest = ScanRequest.builder()
                .tableName(tableName)
```

```
                .build();

ScanResponse scanResponse = dynamoDB.scan(scanRequest);
```

Using the AWS SDK untuk Python (Boto3)

1. Buat tabel. Contoh berikut menggunakan CreateTableAPI untuk membuat tabel bernamamyMusicTable.

```
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. Tambahkan tag ke tabel ini. TagResourceAPI dalam contoh berikut menambahkan pasangan nilai kunci tag `Title: ProductManager` ke. myMusicTable

```
tag_resouce_response = ddb_client.tag_resource(
    ResourceArn=table_arn,
    Tags=[
        {
            'Key': 'Title',
            'Value': 'ProductManager'
        },
    ]
```

```
)
```

3. Buat kebijakan inline dan tambahkan ke peran yang memiliki kebijakan AmazonDynamoDBReadOnlyAccess AWS terkelola yang dilampirkan padanya, seperti yang ditunjukkan pada contoh berikut.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
        "Effect": "Allow",
        "Action": "dynamodb:PutItem",
        "Resource": "arn:aws:dynamodb:*:*:table/*",
        "Condition": {
            "StringEquals": {
            "aws:ResourceTag/Title": "ProductManager"
            }
        }
        }
    ]
    }
```

Kebijakan ini memungkinkan `PutItem` tindakan pada tabel jika kunci tag dan nilai yang dilampirkan ke tabel cocok dengan tag yang ditentukan dalam kebijakan.

4. Asumsikan peran dengan kebijakan yang dijelaskan dalam Langkah 3.
5. Gunakan PutItemAPI untuk menempatkan item ke file`myMusicTable`.

```
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. Pindai tabel untuk memverifikasi apakah item telah ditambahkan ke tabel.

```
scan_response = client.scan(
    TableName='myMusicTable'
)
```

Tanpa ABAC

Jika ABAC tidak diaktifkan untuk Anda Akun AWS, kondisi tag dalam kebijakan IAM dan tabel DynamoDB tidak cocok. Akibatnya, `PutItem` tindakan mengembalikan `AccessDeniedException` karena efek `AmazonDynamoDBReadOnlyAccess` kebijakan.

```
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.
```

Dengan ABAC

Jika ABAC diaktifkan untuk Anda Akun AWS, `put-item` tindakan berhasil diselesaikan dan menambahkan item baru ke tabel Anda. Hal ini karena kebijakan inline pada tabel memungkinkan `PutItem` tindakan jika kondisi tag dalam kebijakan IAM dan tabel cocok.

Contoh 2: Izinkan tindakan menggunakan aws: RequestTag

Dengan menggunakan RequestTagkunci kondisi aws: /tag-key, Anda dapat membandingkan pasangan nilai kunci tag yang diteruskan dalam permintaan Anda dengan pasangan tag yang ditentukan dalam kebijakan IAM. Misalnya, Anda dapat mengizinkan tindakan tertentu, seperti`CreateTable`, menggunakan `aws:RequestTag` jika kondisi tag tidak cocok. Untuk melakukan ini, lakukan langkah-langkah berikut:

Using the AWS CLI

1. Buat kebijakan inline dan tambahkan ke peran yang memiliki kebijakan AmazonDynamoDBReadOnlyAccess AWS terkelola yang dilampirkan padanya, seperti yang ditunjukkan pada contoh berikut.

   ```
   {
       "Version": "2012-10-17",
       "Statement": [
   ```

```
        {
            "Effect": "Allow",
            "Action": [
                "dynamodb:CreateTable",
                "dynamodb:TagResource"
            ],
            "Resource": "arn:aws:dynamodb:*:*:table/*",
            "Condition": {
                "StringEquals": {
                    "aws:RequestTag/Owner": "John"
                }
            }
        }
    ]
}
```

2. Buat tabel yang berisi pasangan nilai kunci tag dari. `"Owner": "John"`

```
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 untuk Python (Boto3)

1. Buat kebijakan inline dan tambahkan ke peran yang memiliki kebijakan AmazonDynamoDBReadOnlyAccess AWS terkelola yang dilampirkan padanya, seperti yang ditunjukkan pada contoh berikut.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "dynamodb:CreateTable",
                "dynamodb:TagResource"
            ],
            "Resource": "arn:aws:dynamodb:*:*:table/*",
```

```
            "Condition": {
                "StringEquals": {
                    "aws:RequestTag/Owner": "John"
                }
            }
        }
    ]
}
```

2. Buat tabel yang berisi pasangan nilai kunci tag dari. `"Owner": "John"`

```
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'
        },
    ],
)
```

Tanpa ABAC

Jika ABAC tidak diaktifkan untuk Anda Akun AWS, kondisi tag dalam kebijakan sebaris dan tabel DynamoDB tidak cocok. Akibatnya, `CreateTable` permintaan gagal dan tabel Anda tidak dibuat.

```
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.
```

Dengan ABAC

Jika ABAC diaktifkan untuk Anda Akun AWS, permintaan pembuatan tabel Anda berhasil diselesaikan. Karena pasangan nilai kunci tag `"Owner": "John"` hadir dalam `CreateTable` permintaan, kebijakan inline memungkinkan pengguna `John` untuk melakukan tindakan. `CreateTable`

Contoh 3: Tolak tindakan menggunakan aws: TagKeys

Menggunakan aws: TagKeys condition key, Anda dapat membandingkan kunci tag dalam permintaan dengan kunci yang ditentukan dalam kebijakan IAM. Misalnya, Anda dapat menolak tindakan tertentu, sepertiCreateTable, menggunakan `aws:TagKeys` jika kunci tag tertentu tidak ada dalam permintaan. Untuk melakukan ini, lakukan langkah-langkah berikut:

Using the AWS CLI

1. Tambahkan kebijakan terkelola pelanggan ke peran yang memiliki kebijakan AWS terkelola AmazonDynamoDBFullAccess yang dilampirkan padanya, seperti yang ditunjukkan pada contoh berikut.

```
{
    "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. Asumsikan peran yang dilampirkan kebijakan, dan buat tabel dengan kunci tag`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 untuk Python (Boto3)

1. Tambahkan kebijakan terkelola pelanggan ke peran yang memiliki kebijakan AWS terkelola AmazonDynamoDBFullAccess yang dilampirkan padanya, seperti yang ditunjukkan pada contoh berikut.

```
{
    "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. Asumsikan peran yang dilampirkan kebijakan, dan buat tabel dengan kunci tag`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'
        },
    ],
)
```

Tanpa ABAC

Jika ABAC tidak diaktifkan untuk Anda Akun AWS, DynamoDB tidak mengirim kunci tag dalam `create-table` perintah ke IAM. `Null`Kondisi memastikan bahwa kondisi mengevaluasi `false` jika

tidak ada kunci tag dalam permintaan. Karena `Deny` kebijakan tidak cocok, `create-table` perintah berhasil diselesaikan.

Dengan ABAC

Jika ABAC diaktifkan untuk Anda Akun AWS, kunci tag yang diteruskan dalam `create-table` perintah diteruskan ke IAM. Kunci tag `Title` dievaluasi terhadap kunci tag berbasis kondisi`CostCenter`, yang ada dalam kebijakan. `Deny` Kunci tag `Title` tidak cocok dengan kunci tag yang ada dalam `Deny` kebijakan karena `StringNotEquals` operator. Oleh karena itu, `CreateTable` tindakan gagal dan tabel Anda tidak dibuat. Menjalankan `create-table` perintah mengembalikan file`AccessDeniedException`.

```
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.
```

Memecahkan masalah kesalahan ABAC umum untuk tabel dan indeks DynamoDB

Topik ini memberikan saran pemecahan masalah untuk kesalahan umum dan masalah yang mungkin Anda temui saat menerapkan ABAC di tabel atau indeks DynamoDB.

Kunci kondisi khusus layanan dalam kebijakan menghasilkan kesalahan

Kunci kondisi khusus layanan tidak dianggap sebagai kunci kondisi yang valid. Jika Anda telah menggunakan kunci tersebut dalam kebijakan Anda, ini akan mengakibatkan kesalahan. Untuk memperbaiki masalah ini, Anda harus mengganti kunci kondisi khusus layanan dengan kunci kondisi yang sesuai untuk mengimplementasikan ABAC di DynamoDB.

Misalnya, Anda telah menggunakan kunci `dynamodb:ResourceTag` kondisi dalam kebijakan sebaris yang menjalankan PutItempermintaan. Bayangkan bahwa permintaan gagal dengan file`AccessDeniedException`. Contoh berikut menunjukkan kebijakan inline yang salah dengan kunci kondisi. `dynamodb:ResourceTag`

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
```

```
            "Effect": "Allow",
            "Action": [
                "dynamodb:PutItem"
            ],
            "Resource": "arn:aws:dynamodb:*:*:table/*",
            "Condition": {
                "StringEquals": {
                    "dynamodb:ResourceTag/Owner": "John"
                }
            }
        }
    ]
}
```

Untuk memperbaiki masalah ini, ganti kunci `dynamodb:ResourceTag` kondisi dengan`aws:ResourceTag`, seperti yang ditunjukkan pada contoh berikut.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "dynamodb:PutItem"
            ],
            "Resource": "arn:aws:dynamodb:*:*:table/*",
            "Condition": {
                "StringEquals": {
                    "aws:ResourceTag/Owner": "John"
                }
            }
        }
    ]
}
```

Tidak dapat memilih keluar dari ABAC

Jika ABAC diaktifkan untuk akun Anda Dukungan, Anda tidak akan dapat memilih keluar dari ABAC melalui konsol DynamoDB. Untuk memilih keluar, hubungi Dukungan.

Anda dapat memilih keluar dari ABAC sendiri hanya jika yang berikut ini benar:

- Anda menggunakan cara swalayan untuk memilih melalui konsol DynamoDB.

- Anda memilih keluar dalam tujuh hari kalender setelah memilih.

Perlindungan data di DynamoDB

Amazon DynamoDB menyediakan infrastruktur penyimpanan yang sangat tahan lama yang dirancang untuk penyimpanan data penting dan primer. Objek disimpan secara redundan pada sejumlah perangkat di beberapa fasilitas di Wilayah Amazon DynamoDB.

DynamoDB melindungi data pengguna yang disimpan saat istirahat dan juga data dalam transit antara klien lokal dan DynamoDB, dan antara DynamoDB dan sumber daya lain dalam Wilayah yang sama. AWS AWS

Topik

- Enkripsi DynamoDB saat diam
- Mengamankan koneksi DynamoDB menggunakan titik akhir VPC dan kebijakan IAM”

Enkripsi DynamoDB saat diam

Semua data pengguna yang disimpan di Amazon DynamoDB dienkripsi sepenuhnya saat diam. Enkripsi DynamoDB saat diam memberikan keamanan yang ditingkatkan dengan mengenkripsi semua data diam Anda menggunakan kunci enkripsi yang disimpan di AWS Key Management Service (AWS KMS). Fungsi ini membantu mengurangi beban operasional dan kompleksitas yang terlibat dalam melindungi data sensitif. Dengan enkripsi saat diam, Anda dapat membuat aplikasi yang sensitif terhadap keamanan yang memenuhi persyaratan kepatuhan dan peraturan enkripsi yang ketat.

Enkripsi DynamoDB saat diam menyediakan lapisan tambahan perlindungan data dengan selalu mengamankan data Anda dalam tabel terenkripsi, termasuk kunci primer, indeks sekunder lokal dan global, aliran, tabel global, cadangan, dan klaster DynamoDB Accelerator (DAX) setiap kali data disimpan dalam media yang tahan lama. Kebijakan organisasi, peraturan industri atau pemerintah, dan persyaratan kepatuhan kerap mewajibkan penggunaan enkripsi saat diam untuk meningkatkan keamanan data aplikasi Anda. Untuk informasi selengkapnya tentang enkripsi untuk aplikasi database, lihat SDK Enkripsi AWS Database.

Enkripsi saat istirahat terintegrasi dengan AWS KMS untuk mengelola kunci enkripsi yang digunakan untuk mengenkripsi tabel Anda. Untuk informasi selengkapnya tentang tipe dan status kunci, lihat AWS Key Management Service konsep di Panduan AWS Key Management Service Pengembang.

Saat membuat tabel baru, Anda dapat memilih salah satu AWS KMS key jenis berikut untuk mengenkripsi tabel Anda. Anda dapat beralih di antara jenis-jenis kunci ini kapan saja.

- Kunci milik AWS — Jenis enkripsi default. Kuncinya dimiliki oleh DynamoDB (tanpa biaya tambahan).
- Kunci yang dikelola AWS — Kunci disimpan di akun Anda dan dikelola oleh AWS KMS (AWS KMS dikenakan biaya).
- Kunci yang dikelola pelanggan – Kunci disimpan di akun Anda serta dibuat, dimiliki, dan dikelola oleh Anda. Anda memiliki kontrol penuh atas kunci KMS (AWS KMS dikenakan biaya).

Untuk informasi selengkapnya tentang jenis kunci, lihat Kunci dan AWS kunci pelanggan.

> **Note**
>
> - Saat membuat klaster DAX baru dengan enkripsi saat diam diaktifkan, Kunci yang dikelola AWS akan digunakan untuk mengenkripsi data diam di klaster.
> - Jika tabel Anda memiliki kunci urutan, beberapa kunci urutan yang menandai batas kisaran disimpan dalam plaintext di metadata tabel.

Ketika Anda mengakses tabel terenkripsi, DynamoDB mendekripsi data tabel secara transparan. Anda tidak perlu mengubah kode atau aplikasi apa pun untuk menggunakan atau mengelola tabel terenkripsi. DynamoDB terus memberikan latensi milidetik satu digit yang sama yang Anda harapkan, dan semua kueri DynamoDB bekerja dengan lancar pada data terenkripsi Anda.

Anda dapat menentukan kunci enkripsi saat membuat tabel baru atau mengganti kunci enkripsi pada tabel yang ada dengan menggunakan AWS Management Console, AWS Command Line Interface (AWS CLI), atau Amazon DynamoDB API. Untuk mempelajari caranya, lihat Mengelola tabel yang dienkripsi di DynamoDB.

Enkripsi saat istirahat menggunakan Kunci milik AWS ditawarkan tanpa biaya tambahan. Namun, AWS KMS biaya berlaku untuk Kunci yang dikelola AWS dan untuk kunci yang dikelola pelanggan. Untuk informasi selengkapnya tentang harga, silakan lihat harga AWS KMS.

Enkripsi DynamoDB saat istirahat tersedia di AWS semua Wilayah, termasuk Wilayah AWS Tiongkok (Beijing) dan AWS China (Ningxia) dan Wilayah (AS). AWS GovCloud Untuk informasi selengkapnya,

lihat Enkripsi DynamoDB saat istirahat: Cara kerjanya dan Catatan penggunaan enkripsi DynamoDB saat diam.

Enkripsi DynamoDB saat istirahat: Cara kerjanya

Enkripsi Amazon DynamoDB saat diam mengenkripsi data Anda menggunakan Advanced Encryption Standard 256-bit (AES-256), yang membantu mengamankan data Anda dari akses tidak sah ke penyimpanan dasar.

Enkripsi saat istirahat terintegrasi dengan AWS Key Management Service (AWS KMS) untuk mengelola kunci enkripsi yang digunakan untuk mengenkripsi tabel Anda.

> **Note**
>
> Pada Mei 2022, AWS KMS mengubah jadwal rotasi Kunci yang dikelola AWS dari setiap tiga tahun (sekitar 1.095 hari) menjadi setiap tahun (sekitar 365 hari).
> Baru Kunci yang dikelola AWS secara otomatis diputar satu tahun setelah dibuat, dan kira-kira setiap tahun setelahnya.
> Yang Kunci yang dikelola AWS ada secara otomatis diputar satu tahun setelah rotasi terbaru mereka, dan setiap tahun setelahnya.

Kunci milik AWS

Kunci milik AWS tidak disimpan di AWS akun Anda. Mereka adalah bagian dari kumpulan kunci KMS yang AWS memiliki dan mengelola untuk digunakan di beberapa AWS akun. AWS Layanan dapat digunakan Kunci milik AWS untuk melindungi data Anda. Kunci milik AWS digunakan oleh DynamoDB diputar setiap tahun (sekitar 365 hari).

Anda tidak dapat melihat, mengelola, atau menggunakan Kunci milik AWS, atau mengaudit penggunaannya. Namun, Anda tidak perlu melakukan hal apa pun atau mengubah program apa pun untuk melindungi kunci yang mengenkripsi data Anda.

Anda tidak dikenakan biaya bulanan atau biaya penggunaan untuk penggunaan Kunci milik AWS, dan mereka tidak dihitung terhadap AWS KMS kuota untuk akun Anda.

Kunci yang dikelola AWS

Kunci yang dikelola AWS adalah kunci KMS di akun Anda yang dibuat, dikelola, dan digunakan atas nama Anda oleh AWS layanan yang terintegrasi dengannya AWS KMS. Anda dapat melihat Kunci

yang dikelola AWS di akun Anda, melihat kebijakan kuncinya, dan mengaudit penggunaannya di log AWS CloudTrail . Namun, Anda tidak dapat mengelola kunci KMS ini atau mengubah izinnya.

Enkripsi saat istirahat secara otomatis terintegrasi dengan AWS KMS untuk mengelola untuk DynamoDB `aws/dynamodb` () yang digunakan Kunci yang dikelola AWS untuk mengenkripsi tabel Anda. Jika Kunci yang dikelola AWS tidak ada saat Anda membuat tabel DynamoDB terenkripsi, AWS KMS secara otomatis membuat kunci baru untuk Anda. Kunci ini digunakan dengan tabel terenkripsi yang dibuat di masa depan. AWS KMS menggabungkan perangkat keras dan perangkat lunak yang aman dan sangat tersedia untuk menyediakan sistem manajemen kunci yang diskalakan untuk cloud.

Untuk informasi selengkapnya tentang mengelola izin Kunci yang dikelola AWS, lihat Mengotorisasi penggunaan Kunci yang dikelola AWS dalam Panduan AWS Key Management Service Pengembang.

Kunci yang dikelola pelanggan

Kunci yang dikelola pelanggan adalah kunci KMS di AWS akun Anda yang Anda buat, miliki, dan kelola. Anda memiliki kontrol penuh atas KMS ini, termasuk membangun dan memelihara kebijakan kuncinya, kebijakan IAM, dan pemberian izin; mengaktifkan dan menonaktifkannya; merotasi materi kriptografinya; menambahkan tag; membuat alias yang merujuk kepadanya; dan menjadwalkan penghapusan KMS tersebut. Untuk informasi selengkapnya tentang mengelola izin kunci terkelola pelanggan, lihat Kunci terkelola pelanggan.

Saat Anda menentukan kunci yang dikelola pelanggan sebagai kunci enkripsi tingkat tabel, tabel DynamoDB, indeks sekunder lokal dan global, dan aliran dienkripsi dengan kunci yang sama dengan yang dikelola pelanggan. Cadangan sesuai permintaan dienkripsi dengan kunci enkripsi tingkat tabel yang ditentukan pada saat cadangan dibuat. Memperbarui kunci enkripsi tingkat tabel tidak mengubah kunci enkripsi yang terkait dengan cadangan sesuai permintaan yang ada.

Mengatur status kunci yang dikelola pelanggan ke nonaktif atau menjadwalkannya untuk dihapus akan membuat semua pengguna dan layanan DynamoDB tidak dapat mengenkripsi atau mendekripsi data serta melakukan operasi baca dan tulis pada tabel. DynamoDB harus memiliki akses ke kunci enkripsi Anda untuk memastikan bahwa Anda dapat terus mengakses tabel Anda dan untuk mencegah kehilangan data.

Jika Anda menonaktifkan kunci yang dikelola pelanggan atau menjadwalkan penghapusannya, status tabel Anda menjadi Tidak dapat diakses. Untuk memastikan bahwa Anda dapat terus bekerja dengan tabel, Anda harus memberikan akses DynamoDB ke kunci enkripsi yang ditentukan dalam

waktu tujuh hari. Segera setelah layanan mendeteksi bahwa kunci enkripsi Anda tidak dapat diakses, DynamoDB mengirimkan pemberitahuan email untuk mengingatkan Anda.

> **Note**
>
> - Jika kunci yang dikelola pelanggan tetap tidak dapat diakses untuk layanan DynamoDB selama lebih dari tujuh hari, tabel diarsipkan dan tidak dapat diakses lagi. DynamoDB menciptakan cadangan sesuai permintaan atas tabel Anda, dan Anda menerima tagihan terkait cadangan tersebut. Anda dapat menggunakan cadangan sesuai permintaan ini untuk memulihkan data Anda ke tabel baru. Untuk memulai pemulihan, kunci yang dikelola pelanggan terakhir pada tabel harus diaktifkan, dan DynamoDB harus dapat mengaksesnya.
> - Jika kunci yang dikelola pelanggan yang digunakan untuk mengenkripsi replika tabel global tidak dapat diakses, DynamoDB akan menghapus replika ini dari grup replikasi. Replika ini tidak akan dihapus dan replikasi dari dan ke wilayah ini akan dihentikan, 20 jam setelah mendeteksi bahwa kunci yang dikelola pelanggan tidak dapat diakses.

Untuk informasi selengkapnya, lihat mengaktifkan kunci dan menghapus kunci.

Catatan tentang penggunaan Kunci yang dikelola AWS

Amazon DynamoDB tidak dapat membaca data tabel Anda kecuali memiliki akses ke kunci KMS yang disimpan di akun Anda. AWS KMS DynamoDB menggunakan enkripsi amplop dan hierarki kunci untuk mengenkripsi data. Kunci AWS KMS enkripsi Anda digunakan untuk mengenkripsi kunci root hierarki kunci ini. Untuk informasi lebih lanjut, lihat Enkripsi amplop di Panduan Developer AWS Key Management Service .

DynamoDB tidak AWS KMS memanggil setiap operasi DynamoDB. Kuncinya disegarkan setiap 5 menit sekali per penelepon dengan lalu lintas aktif.

Pastikan bahwa Anda telah mengonfigurasi SDK untuk menggunakan kembali koneksi. Jika tidak, Anda akan mengalami latensi dari DynamoDB harus membangun kembali entri AWS KMS cache baru untuk setiap operasi DynamoDB. Selain itu, Anda mungkin harus menghadapi CloudTrail biaya AWS KMS dan biaya yang lebih tinggi. Misalnya, untuk melakukannya menggunakan SDK Node.js, Anda dapat membuat agen HTTPS baru dengan `keepAlive` diaktifkan. Untuk informasi selengkapnya, lihat Mengonfigurasi keepAlive di Node.js di Panduan Developer AWS SDK untuk JavaScript .

Catatan penggunaan enkripsi DynamoDB saat diam

Pertimbangkan hal berikut saat Anda menggunakan enkripsi saat diam di Amazon DynamoDB.

Semua data tabel dienkripsi

Enkripsi di sisi server saat diam diaktifkan pada semua data tabel DynamoDB dan tidak dapat dinonaktifkan. Anda tidak dapat mengenkripsi subset item saja dalam tabel.

Enkripsi saat diam hanya mengenkripsi data saat statis (saat diam) pada media penyimpanan persisten. Jika keamanan data menjadi perhatian atas data bergerak atau data dalam penggunaan, Anda mungkin perlu melakukan tindakan tambahan:

- Data bergerak: Semua data Anda di DynamoDB dienkripsi saat bergerak. Secara default, komunikasi ke dan dari DynamoDB menggunakan protokol HTTPS, yang melindungi lalu lintas jaringan menggunakan enkripsi Secure Sockets Layer (SSL)/Keamanan Lapisan Pengangkutan (TLS).
- Data dalam penggunaan: Melindungi data Anda sebelum mengirimkannya ke DynamoDB menggunakan enkripsi di sisi klien. Untuk informasi selengkapnya, lihat Enkripsi di sisi klien dan di sisi server dalam Panduan Developer Amazon DynamoDB Encryption Client.

Anda dapat menggunakan aliran dengan tabel terenkripsi. Aliran DynamoDB selalu dienkripsi dengan kunci enkripsi tingkat tabel. Untuk informasi selengkapnya, lihat Tangkapan data perubahan DynamoDB Streams.

Cadangan DynamoDB dienkripsi, dan tabel yang dipulihkan dari cadangan juga diaktifkan enkripsinya. Anda dapat menggunakan Kunci milik AWS, Kunci yang dikelola AWS, atau kunci yang dikelola pelanggan untuk mengenkripsi data cadangan Anda. Untuk informasi selengkapnya, lihat Backup dan restore untuk DynamoDB.

Indeks sekunder lokal dan indeks sekunder global dienkripsi menggunakan kunci yang sama seperti tabel dasar.

Jenis enkripsi

> **Note**
>
> Kunci yang dikelola pelanggan tidak didukung di Tabel Global Versi 2017. Jika ingin menggunakan kunci yang dikelola pelanggan di Tabel Global DynamoDB, Anda perlu meningkatkan tabel ke Tabel Global Versi 2019, lalu mengaktifkannya.

Pada AWS Management Console, jenis enkripsi adalah `KMS` ketika Anda menggunakan Kunci yang dikelola AWS atau kunci yang dikelola pelanggan untuk mengenkripsi data Anda. Jenis enkripsi adalah `DEFAULT` ketika Anda menggunakan Kunci milik AWS. Di Amazon DynamoDB API, jenis enkripsi `KMS` adalah saat Anda menggunakan atau kunci Kunci yang dikelola AWS yang dikelola pelanggan. Apabila tidak ada jenis enkripsi, data Anda dienkripsi menggunakan Kunci milik AWS. Anda dapat beralih antara Kunci milik AWS, Kunci yang dikelola AWS, dan kunci yang dikelola pelanggan pada waktu tertentu. Anda dapat menggunakan konsol, AWS Command Line Interface (AWS CLI), atau Amazon DynamoDB API untuk mengganti kunci enkripsi.

Perhatikan batasan berikut saat menggunakan kunci yang dikelola pelanggan:

- Anda tidak dapat menggunakan kunci yang dikelola pelanggan dengan klaster DynamoDB Accelerator (DAX). Untuk informasi selengkapnya, lihat Enkripsi DAX saat diam.
- Anda dapat menggunakan kunci yang dikelola pelanggan untuk mengenkripsi tabel yang menggunakan transaksi. Namun, guna memastikan ketahanan untuk propagasi transaksi, salinan permintaan transaksi disimpan sementara oleh layanan dan dienkripsi menggunakan Kunci milik AWS. Data yang ditempatkan di tabel dan indeks sekunder Anda selalu dienkripsi saat diam menggunakan kunci yang dikelola pelanggan.
- Anda dapat menggunakan kunci yang dikelola pelanggan untuk mengenkripsi tabel yang menggunakan Contributor Insights. Namun, data yang dikirimkan ke Amazon CloudWatch dienkripsi dengan file. Kunci milik AWS
- Saat Anda beralih ke kunci terkelola pelanggan baru, pastikan untuk tetap mengaktifkan kunci asli hingga proses selesai. AWS masih akan membutuhkan kunci asli untuk mendekripsi data sebelum mengenkripsi dengan kunci baru. Proses akan selesai ketika SSEDescription Status tabel DIAKTIFKAN dan kunci KMSMaster KeyArn yang dikelola pelanggan baru ditampilkan. Pada titik ini, kunci asli dapat dinonaktifkan atau dijadwalkan untuk dihapus.
- Setelah kunci yang dikelola pelanggan baru ditampilkan, tabel dan cadangan sesuai permintaan baru dienkripsi dengan kunci baru.

- Setiap cadangan sesuai permintaan yang ada tetap dienkripsi dengan kunci yang dikelola pelanggan yang digunakan saat cadangan tersebut dibuat. Anda akan membutuhkan kunci yang sama untuk memulihkan cadangan tersebut. Anda dapat mengidentifikasi kunci untuk periode ketika setiap cadangan dibuat dengan menggunakan DescribeBackup API untuk melihat cadangan tersebut SSEDescription.
- Jika Anda menonaktifkan kunci yang dikelola pelanggan atau menjadwalkannya untuk dihapus, data apa pun di DynamoDB Streams masih dikenakan masa aktif 24 jam. Data aktivitas yang tidak diambil memenuhi syarat untuk dipangkas saat berusia lebih dari 24 jam.
- Jika Anda menonaktifkan kunci yang dikelola pelanggan atau menjadwalkannya untuk dihapus, penghapusan Waktu untuk Tayang (TTL) berlanjut selama 30 menit. Penghapusan TTL ini terus dipancarkan ke DynamoDB Streams dan tunduk pada interval pemangkasan/retensi standar.

  Untuk informasi selengkapnya, lihat mengaktifkan kunci dan menghapus kunci.

Menggunakan kunci KMS dan kunci data

Fitur enkripsi DynamoDB saat istirahat menggunakan AWS KMS key dan hierarki kunci data untuk melindungi data tabel Anda. DynamoDB menggunakan hierarki kunci yang sama untuk melindungi aliran, tabel global, dan cadangan DynamoDB ketika ditulis ke media tahan lama.

Kami menyarankan Anda merencanakan strategi enkripsi Anda sebelum menerapkan tabel Anda di DynamoDB. Jika Anda menyimpan data sensitif atau rahasia di DynamoDB, pertimbangkan untuk menyertakan enkripsi di sisi klien dalam paket Anda. Dengan cara ini, Anda dapat mengenkripsi data sedekat mungkin dengan asalnya, dan memastikan perlindungannya sepanjang siklus hidupnya. Untuk informasi selengkapnya, lihat dokumentasi klien enkripsi DynamoDB.

AWS KMS key

Enkripsi saat diam melindungi tabel DynamoDB Anda di bawah AWS KMS key. Secara default, DynamoDB menggunakan Kunci milik AWS, kunci enkripsi multi-penghuni yang dibuat dan dikelola di akun layanan DynamoDB. Namun Anda dapat mengenkripsi tabel DynamoDB Anda di bawah kunci yang dikelola pelanggan untuk DynamoDB (`aws/dynamodb`) di Akun AWS Anda. Anda dapat memilih kunci KMS yang berbeda untuk setiap tabel. Kunci KMS yang Anda pilih untuk tabel juga digunakan untuk mengenkripsi indeks sekunder lokal dan global, aliran, dan cadangan.

Anda memilih kunci KMS untuk tabel saat membuat atau memperbarui tabel. Anda dapat mengubah kunci KMS untuk tabel kapan saja, baik di konsol DynamoDB atau dengan

menggunakan operasi. UpdateTable Proses pengalihan kunci berjalan tanpa hambatan dan tidak memerlukan waktu henti atau menurunkan layanan.

> **Important**
>
> DynamoDB hanya mendukung kunci KMS simetris. Anda tidak dapat menggunakan kunci KMS asimetris untuk mengenkripsi tabel DynamoDB Anda.

Gunakan kunci yang dikelola pelanggan untuk mendapatkan fitur berikut:

- Anda membuat dan mengelola kunci KMS, termasuk menetapkan kebijakan kunci, kebijakan IAM, dan pemberian izin untuk mengontrol akses ke kunci KMS. Anda dapat mengaktifkan dan menonaktifkan kunci KMS, mengaktifkan dan menonaktifkan rotasi kunci otomatis, dan menghapus kunci KMS ketika sudah tidak digunakan.
- Anda dapat menggunakan kunci yang dikelola pelanggan dengan material kunci yang diimpor atau kunci yang dikelola pelanggan di penyimpanan kunci kustom yang Anda miliki dan kelola.
- Anda dapat mengaudit enkripsi dan dekripsi tabel DynamoDB Anda dengan memeriksa panggilan API DynamoDB ke dalam log. AWS KMSAWS CloudTrail

Gunakan Kunci yang dikelola AWS jika Anda memerlukan salah satu fitur berikut:

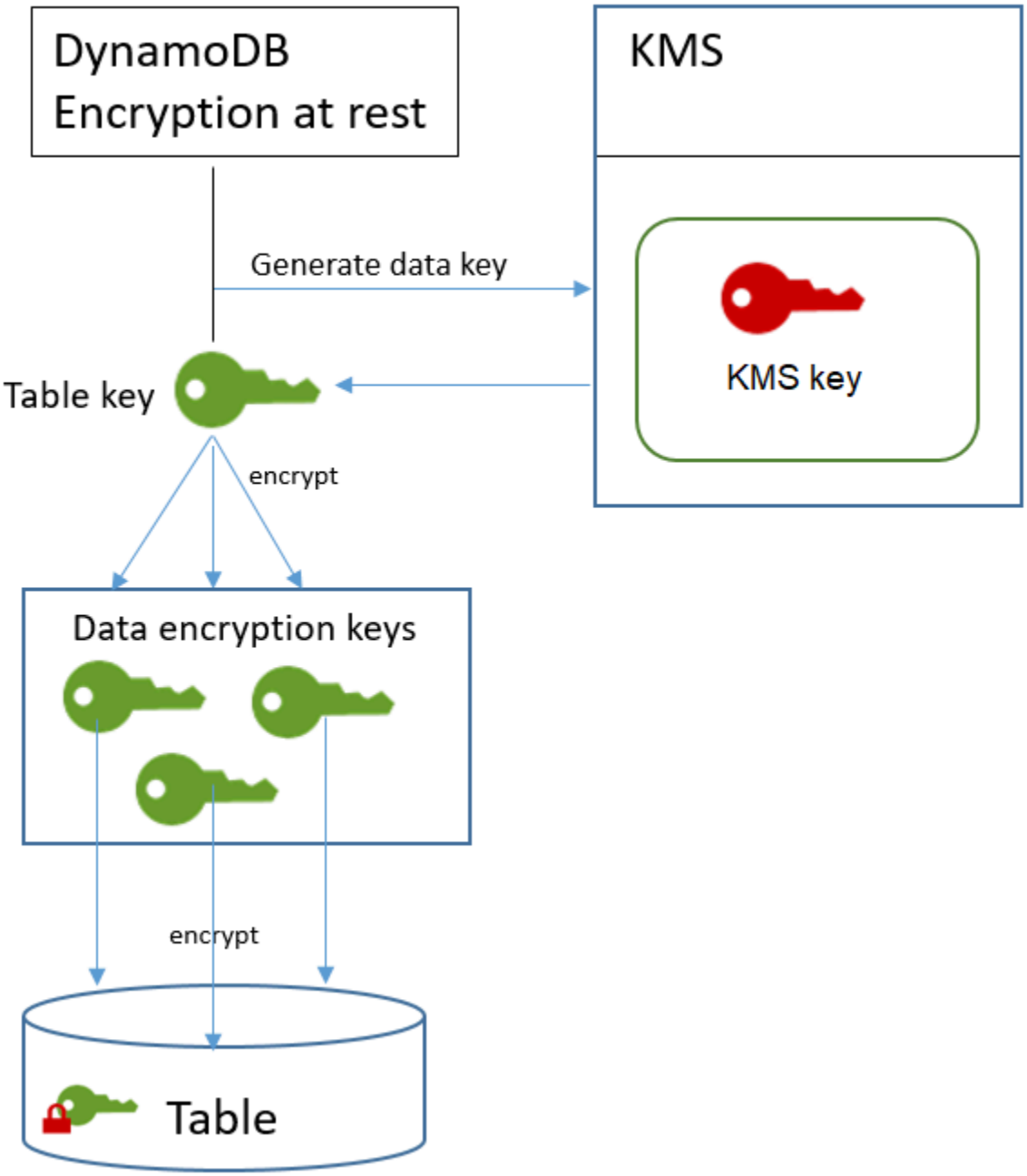
- Anda dapat melihat kunci KMS dan melihat kebijakan kuncinya. (Anda tidak dapat mengubah kebijakan kunci.)
- Anda dapat mengaudit enkripsi dan dekripsi tabel DynamoDB Anda dengan memeriksa panggilan API DynamoDB ke dalam log. AWS KMSAWS CloudTrail

Namun, Kunci milik AWS ini gratis dan penggunaannya tidak dihitung terhadap AWS KMS sumber daya atau kuota permintaan. Kunci yang dikelola pelanggan dan Kunci yang dikelola AWS dikenakan biaya untuk setiap panggilan API dan AWS KMS kuota berlaku untuk kunci KMS ini.

Kunci tabel

DynamoDB menggunakan kunci KMS untuk tabel guna menghasilkan dan mengenkripsi kunci data unik untuk tabel tersebut, yang dikenal sebagai kunci tabel. Kunci tabel bertahan sepanjang masa hidup tabel terenkripsi.

Kunci tabel digunakan sebagai kunci enkripsi kunci. DynamoDB menggunakan kunci tabel ini untuk melindungi kunci enkripsi data yang digunakan untuk mengenkripsi data tabel. DynamoDB menghasilkan kunci enkripsi data yang unik untuk setiap struktur dasar dalam tabel, tetapi beberapa item tabel mungkin dilindungi oleh kunci enkripsi data yang sama.



Saat pertama kali mengakses tabel terenkripsi, DynamoDB mengirimkan permintaan untuk menggunakan kunci KMS AWS KMS untuk mendekripsi kunci tabel. Kemudian, layanan ini menggunakan kunci tabel plaintext untuk mendekripsi kunci enkripsi data, dan menggunakan kunci enkripsi data plaintext untuk mendekripsi data tabel.

DynamoDB menyimpan dan menggunakan kunci tabel dan kunci enkripsi data di luar. AWS KMS Layanan ini melindungi semua kunci dengan enkripsi Advanced Encryption Standard (AES) dan

kunci enkripsi 256-bit. Kemudian, layanan ini menyimpan kunci terenkripsi dengan data terenkripsi agar kunci tersebut tersedia untuk mendekripsi data tabel sesuai permintaan.

Jika Anda mengubah kunci KMS untuk tabel Anda, DynamoDB menghasilkan kunci tabel baru. Kemudian, layanan ini menggunakan kunci tabel baru untuk melakukan enkripsi ulang kunci enkripsi data.

Melakukan cache pada kunci tabel

Untuk menghindari panggilan AWS KMS untuk setiap operasi DynamoDB, DynamoDB menyimpan tombol tabel plaintext untuk setiap pemanggil dalam memori. Jika DynamoDB mendapat permintaan untuk kunci tabel cache setelah lima menit tidak aktif, ia akan mengirimkan permintaan baru AWS KMS untuk mendekripsi kunci tabel. Panggilan ini akan menangkap setiap perubahan yang dibuat pada kebijakan akses kunci KMS di AWS KMS atau AWS Identity and Access Management (IAM) sejak permintaan terakhir untuk mendekripsi kunci tabel.

Mengotorisasi penggunaan kunci KMS Anda

Jika Anda menggunakan kunci yang dikelola pelanggan atau Kunci yang dikelola AWS di akun Anda untuk melindungi tabel DynamoDB Anda, kebijakan pada kunci KMS tersebut harus memberikan izin kepada DynamoDB untuk menggunakannya atas nama Anda. Konteks otorisasi pada Kunci yang dikelola AWS for DynamoDB menyertakan kebijakan utamanya dan hibah yang mendelegasikan izin untuk menggunakannya.

Anda memiliki kontrol penuh atas kebijakan dan pemberian izin pada kunci yang dikelola pelanggan. Karena Kunci yang dikelola AWS ada di akun Anda, Anda dapat melihat kebijakan dan pemberian izinnya. Tetapi, karena dikelola oleh AWS, Anda tidak dapat mengubah kebijakan.

DynamoDB tidak memerlukan otorisasi tambahan untuk menggunakan default untuk melindungi tabel DynamoDB di tabel Kunci milik AWSAnda. Akun AWS

Topik

- Kebijakan utama untuk Kunci yang dikelola AWS
- Kebijakan kunci untuk kunci yang dikelola pelanggan
- Menggunakan pemberian izin untuk mengotorisasi DynamoDB

Kebijakan utama untuk Kunci yang dikelola AWS

Ketika DynamoDB menggunakan Kunci yang dikelola AWS untuk DynamoDB (`aws/dynamodb`) dalam operasi kriptografi, layanan ini melakukannya atas nama pengguna yang mengakses sumber daya DynamoDB. Kebijakan utama pada Kunci yang dikelola AWS memberikan semua pengguna di akun izin untuk menggunakan Kunci yang dikelola AWS untuk operasi tertentu. Namun izin hanya diberikan ketika DynamoDB membuat permintaan atas nama pengguna. ViaService Kondisi dalam kebijakan kunci tidak mengizinkan pengguna untuk menggunakan Kunci yang dikelola AWS kecuali permintaan berasal dari layanan DynamoDB.

Kebijakan utama ini, seperti kebijakan semua Kunci yang dikelola AWS, ditetapkan oleh AWS. Anda tidak dapat mengubahnya, tetapi Anda dapat melihatnya kapan saja. Untuk detailnya, lihat Melihat kebijakan kunci.

Pernyataan kebijakan dalam kebijakan kunci memiliki efek sebagai berikut:

- Izinkan pengguna di akun untuk menggunakan Kunci yang dikelola AWS for DynamoDB dalam operasi kriptografi ketika permintaan berasal dari DynamoDB atas nama mereka. Kebijakan ini juga memungkinkan pengguna untuk membuat pemberian izin untuk kunci KMS.
- Memungkinkan identitas IAM yang diotorisasi di akun untuk melihat properti Kunci yang dikelola AWS untuk DynamoDB dan mencabut pemberian izin yang memungkinkan DynamoDB menggunakan kunci KMS. DynamoDB menggunakan pemberian izin untuk operasi pemeliharaan berkelanjutan.
- Memungkinkan DynamoDB melakukan operasi read-only untuk menemukan DynamoDB untuk akun Kunci yang dikelola AWS Anda.

```
{
  "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" : "*"
  } ]
}
```

Kebijakan kunci untuk kunci yang dikelola pelanggan

Saat Anda memilih kunci yang dikelola pelanggan untuk melindungi tabel DynamoDB, DynamoDB mendapat izin untuk menggunakan kunci KMS atas nama pengguna utama yang membuat pilihan. Pengguna utama tersebut, pengguna atau peran, harus memiliki izin pada kunci KMS yang dibutuhkan DynamoDB. Anda dapat memberikan izin ini dalam kebijakan kunci, kebijakan IAM, atau pemberian izin.

Minimal, DynamoDB memerlukan izin berikut pada kunci yang dikelola pelanggan:

- kms:Encrypt
- kms:Decrypt
- kms: ReEncrypt * (untuk kms: ReEncryptFrom dan kms:ReEncryptTo)
- kms: GenerateDataKey * (untuk kms: GenerateDataKey dan kms:) GenerateDataKeyWithoutPlaintext

- km: DescribeKey
- km: CreateGrant

Sebagai contoh, kebijakan kunci berikut hanya menyediakan izin yang diperlukan. Kebijakan ini memiliki efek sebagai berikut:

- Memungkinkan DynamoDB untuk menggunakan kunci KMS dalam operasi kriptografi dan membuat pemberian izin, tetapi hanya ketika bertindak atas nama pengguna utama di akun yang memiliki izin untuk menggunakan DynamoDB. Jika pengguna utama yang disebutkan dalam pernyataan kebijakan tidak memiliki izin untuk menggunakan DynamoDB, panggilan gagal, bahkan ketika panggilan berasal dari layanan DynamoDB.
- Kunci ViaService kondisi kms: mengizinkan izin hanya jika permintaan berasal dari DynamoDB atas nama prinsipal yang tercantum dalam pernyataan kebijakan. Pengguna utama ini tidak dapat memanggil operasi ini secara langsung. Perhatikan bahwa nilai `kms:ViaService`, `dynamodb.*.amazonaws.com`, memiliki tanda bintang (*) di posisi Wilayah. DynamoDB memerlukan izin untuk independen dari apa pun sehingga dapat membuat panggilan lintas Wilayah AWS wilayah untuk mendukung tabel global DynamoDB.
- Memberikan administrator kunci KMS (pengguna yang dapat mengasumsikan peran `db-team`) akses hanya-baca ke kunci KMS dan izin untuk mencabut pemberian izin, termasuk pemberian izin yang diperlukan DynamoDB untuk melindungi tabel.

Sebelum menggunakan kebijakan kunci contoh, ganti prinsip contoh dengan prinsip aktual dari Anda. Akun AWS

```
{
  "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": "*"
    }
  ]
}
```

Menggunakan pemberian izin untuk mengotorisasi DynamoDB

Selain kebijakan kunci, DynamoDB menggunakan pemberian izin untuk mengatur izin pada kunci yang dikelola pelanggan atau Kunci yang dikelola AWS untuk DynamoDB (`aws/dynamodb`). Untuk melihat hibah pada kunci KMS di akun Anda, gunakan operasi. ListGrants DynamoDB tidak memerlukan pemberian izin, atau izin tambahan apa pun, untuk menggunakan Kunci milik AWS untuk melindungi tabel Anda.

DynamoDB menggunakan izin yang diberikan ketika melakukan pemeliharaan sistem latar belakang dan tugas perlindungan data berkelanjutan. Layanan ini juga menggunakan pemberian izin untuk menghasilkan kunci tabel.

Setiap pemberian izin berlaku spesifik pada sebuah tabel. Jika akun mencakup beberapa tabel yang dienkripsi di bawah kunci KMS yang sama, ada pemberian izin dari setiap jenis untuk setiap tabel.

Hibah dibatasi oleh konteks enkripsi DynamoDB, yang mencakup nama tabel dan Akun AWS ID, dan itu termasuk izin untuk pensiun hibah jika tidak lagi diperlukan.

Untuk membuat pemberian izin, DynamoDB harus memiliki izin untuk memanggil `CreateGrant` atas nama pengguna yang membuat tabel terenkripsi. Untuk Kunci yang dikelola AWS, DynamoDB `kms:CreateGrant` mendapat izin dari kebijakan kunci, yang memungkinkan pengguna akun untuk CreateGrantmemanggil kunci KMS hanya ketika DynamoDB membuat permintaan atas nama pengguna yang berwenang.

Kebijakan kunci juga dapat memungkinkan akun untuk mencabut pemberian izin pada kunci KMS. Namun, jika Anda mencabut pemberian izin pada tabel dienkripsi yang aktif, DynamoDB tidak akan mampu melindungi dan menjaga tabel tersebut.

Konteks enkripsi DynamoDB

Konteks enkripsi adalah seperangkat pasangan kunci-nilai yang berisi data non-rahasia yang arbitrer. Ketika Anda menyertakan konteks enkripsi dalam permintaan untuk mengenkripsi data, secara AWS KMS kriptografis mengikat konteks enkripsi ke data terenkripsi. Untuk mendekripsi data, Anda harus meneruskan konteks enkripsi yang sama.

DynamoDB menggunakan konteks enkripsi yang sama di semua operasi kriptografi. AWS KMS Jika Anda menggunakan kunci yang dikelola pelanggan atau Kunci yang dikelola AWS untuk melindungi tabel DynamoDB Anda, Anda dapat menggunakan konteks enkripsi untuk mengidentifikasi penggunaan kunci KMS dalam catatan dan log audit. Itu juga muncul dalam plaintext di log, seperti dan AWS CloudTrailAmazon CloudWatch Logs.

Konteks enkripsi juga dapat digunakan sebagai syarat untuk otorisasi dalam kebijakan dan pemberian izin. DynamoDB menggunakan konteks enkripsi untuk membatasi hibah yang memungkinkan akses ke kunci yang dikelola pelanggan Kunci yang dikelola AWS atau di akun dan wilayah Anda.

Dalam permintaannya AWS KMS, DynamoDB menggunakan konteks enkripsi dengan dua pasangan kunci-nilai.

```
"encryptionContextSubset": {
    "aws:dynamodb:tableName": "Books"
    "aws:dynamodb:subscriberId": "111122223333"
}
```

- Tabel – Pasangan kunci–nilai pertama mengidentifikasi tabel yang dienkripsi DynamoDB. Kuncinya adalah `aws:dynamodb:tableName`. Nilainya adalah nama tabel.

  ```
  "aws:dynamodb:tableName": "<table-name>"
  ```

  Misalnya:

  ```
  "aws:dynamodb:tableName": "Books"
  ```

- Akun – Pasangan kunci–nilai kedua mengidentifikasi Akun AWS. Kuncinya adalah `aws:dynamodb:subscriberId`. Nilainya adalah ID akun.

  ```
  "aws:dynamodb:subscriberId": "<account-id>"
  ```

  Misalnya:

  ```
  "aws:dynamodb:subscriberId": "111122223333"
  ```

Memantau interaksi DynamoDB dengan AWS KMS

Jika Anda menggunakan kunci yang dikelola pelanggan atau Kunci yang dikelola AWSuntuk melindungi tabel DynamoDB, Anda dapat AWS CloudTrail menggunakan log untuk melacak permintaan yang dikirimkan DynamoDB atas nama Anda. AWS KMS

Permintaan `GenerateDataKey`, `Decrypt`, dan `CreateGrant` dibahas dalam bagian ini. Selain itu, DynamoDB menggunakan DescribeKeyoperasi untuk menentukan apakah kunci KMS yang Anda pilih ada di akun dan wilayah. Ini juga menggunakan RetireGrantoperasi untuk menghapus hibah saat Anda menghapus tabel.

GenerateDataKey

Ketika Anda mengaktifkan enkripsi saat diam pada tabel, DynamoDB menciptakan kunci tabel unik. Ini mengirimkan GenerateDataKeypermintaan untuk AWS KMS yang menentukan kunci KMS untuk tabel.

Peristiwa yang mencatat operasi `GenerateDataKey` serupa dengan peristiwa contoh berikut. Pengguna adalah akun layanan DynamoDB. Parameternya mencakup Amazon Resource Name (ARN) dari kunci KMS, penentu kunci yang memerlukan kunci 256-bit, dan konteks enkripsi yang mengidentifikasi tabel dan Akun AWS.

```
{
    "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"
}
```


Dekripsi

Ketika Anda mengakses tabel DynamoDB yang dienkripsi, DynamoDB perlu mendekripsi kunci tabel agar dapat mendekripsi kunci di bawahnya dalam hierarki. Layanan ini kemudian mendekripsi data dalam tabel. Untuk mendekripsi kunci tabel. DynamoDB mengirimkan permintaan Dekripsi yang menentukan kunci AWS KMS KMS untuk tabel.

Peristiwa yang mencatat operasi `Decrypt` serupa dengan peristiwa contoh berikut. Pengguna adalah kepala sekolah Anda Akun AWS yang mengakses tabel. Parameter termasuk kunci tabel terenkripsi (sebagai gumpalan ciphertext) dan konteks enkripsi yang mengidentifikasi tabel dan file. Akun AWS AWS KMS memperoleh ID kunci KMS dari ciphertext.

```
{
    "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

Saat Anda menggunakan kunci yang dikelola pelanggan atau Kunci yang dikelola AWS untuk melindungi tabel DynamoDB Anda, DynamoDB menggunakan pemberian izin untuk mengizinkan layanan guna melakukan perlindungan dan pemeliharaan data berkelanjutan serta tugas ketahanan. Pemberian izin ini tidak diperlukan pada Kunci milik AWS.

Pemberian izin yang dibuat DynamoDB spesifik pada sebuah tabel. Prinsipal dalam CreateGrantpermintaan adalah pengguna yang membuat tabel.

Peristiwa yang mencatat operasi `CreateGrant` serupa dengan peristiwa contoh berikut. Parameternya mencakup Amazon Resource Name (ARN) dari kunci KMS untuk tabel tersebut, pengguna utama yang izin dan pengguna utama yang dipensiunkan (layanan DynamoDB), serta operasi yang dicakup oleh pemberian izin. Hal ini juga mencakup kendala yang mengharuskan semua operasi enkripsi menggunakan konteks enkripsi yang ditentukan.

```
{
    "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"
}
```

Mengelola tabel yang dienkripsi di DynamoDB

Anda dapat menggunakan AWS Management Console atau AWS Command Line Interface (AWS CLI) untuk menentukan kunci enkripsi pada tabel baru dan memperbarui kunci enkripsi pada tabel yang ada di Amazon DynamoDB.

Topik

- Menentukan kunci enkripsi untuk tabel baru
- Memperbarui kunci enkripsi

Menentukan kunci enkripsi untuk tabel baru

Ikuti langkah berikut untuk menentukan kunci enkripsi pada tabel baru menggunakan konsol Amazon DynamoDB atau AWS CLI.

Membuat tabel yang dienkripsi (konsol)

1. Masuk ke AWS Management Console dan buka konsol DynamoDB di. https://console.aws.amazon.com/dynamodb/
2. Di panel navigasi di sisi kiri konsol, pilih Tabel.
3. Pilih Buat Tabel. Untuk Nama tabel, masukkan **`Music`**. Untuk kunci primer, masukkan **`Artist`**, dan untuk kunci urutan, masukkan **`SongTitle`**, keduanya sebagai string.
4. Di Pengaturan, pastikan bahwa Sesuaikan pengaturan dipilih.

> **Note**
>
> Jika Gunakan pengaturan default dipilih, tabel dienkripsi saat istirahat dengan tanpa Kunci milik AWS biaya tambahan.

5. Di bawah Enkripsi saat istirahat, pilih jenis enkripsi - Kunci milik AWS Kunci yang dikelola AWS, atau kunci yang dikelola pelanggan.

   - Dimiliki oleh Amazon DynamoDB. AWS kunci yang dimiliki, khusus dimiliki dan dikelola oleh DynamoDB. Anda tidak dikenakan biaya tambahan untuk menggunakan kunci ini.
   - AWS kunci terkelola. Alias kunci: `aws/dynamodb`. Kunci disimpan di akun Anda dan dikelola oleh AWS Key Management Service (AWS KMS). AWS KMS dikenakan biaya.
   - Disimpan di akun Anda, serta dimiliki dan dikelola oleh Anda. Kunci yang dikelola pelanggan. Kunci disimpan di akun Anda dan dikelola oleh AWS Key Management Service (AWS KMS). AWS KMS dikenakan biaya.

   > **Note**
   >
   > Jika Anda memilih untuk memiliki dan mengelola kunci Anda sendiri, pastikan Kebijakan Kunci KMS diatur dengan tepat. Untuk informasi selengkapnya termasuk contoh, lihat Kebijakan kunci untuk kunci yang dikelola pelanggan.

6. Pilih Buat tabel untuk membuat tabel terenkripsi. Untuk mengonfirmasi jenis enkripsi, pilih detail tabel pada tab Gambaran umum dan tinjau bagian Detail tambahan.

Membuat tabel yang dienkripsi (AWS CLI)

Gunakan AWS CLI untuk membuat tabel dengan default Kunci milik AWS, kunci yang dikelola pelanggan Kunci yang dikelola AWS, atau kunci yang dikelola pelanggan untuk Amazon DynamoDB.

Untuk membuat tabel terenkripsi dengan default Kunci milik AWS

- Buat tabel `Music` yang dienkripsi seperti berikut.

```
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**
>
> Tabel ini sekarang dienkripsi menggunakan default Kunci milik AWS di akun layanan DynamoDB.

Untuk membuat tabel terenkripsi dengan untuk Kunci yang dikelola AWS DynamoDB

- Buat tabel `Music` yang dienkripsi seperti berikut.

```
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
```

Status `SSEDescription` tabel diatur menjadi `ENABLED` dan `SSEType` adalah `KMS`.

```
"SSEDescription": {
  "SSEType": "KMS",
  "Status": "ENABLED",
  "KMSMasterKeyArn": "arn:aws:kms:us-east-1:123456789012:key/abcd1234-abcd-1234-
a123-ab1234a1b234",
}
```

Untuk membuat tabel yang dienkripsi dengan kunci yang dikelola pelanggan untuk DynamoDB

- Buat tabel `Music` yang dienkripsi seperti berikut.

```
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
```

  Status `SSEDescription` tabel diatur menjadi `ENABLED` dan `SSEType` adalah `KMS`.

```
"SSEDescription": {
  "SSEType": "KMS",
  "Status": "ENABLED",
  "KMSMasterKeyArn": "arn:aws:kms:us-east-1:123456789012:key/abcd1234-abcd-1234-
a123-ab1234a1b234",
}
```

Memperbarui kunci enkripsi

Anda juga dapat menggunakan konsol DynamoDB atau AWS CLI untuk memperbarui kunci enkripsi tabel yang ada antara Kunci yang dikelola AWS,, dan kunci Kunci milik AWS yang dikelola pelanggan kapan saja.

Memperbarui kunci enkripsi (konsol)

1. Masuk ke AWS Management Console dan buka konsol DynamoDB di. https://console.aws.amazon.com/dynamodb/
2. Di panel navigasi di sisi kiri konsol, pilih Tabel.
3. Pilih tabel yang akan Anda perbarui.
4. Pilih dropdown Tindakan, lalu pilih opsi Perbarui pengaturan.

5. Buka tab Pengaturan tambahan.
6. Di bawah Enkripsi, pilih Kelola enkripsi.
7. Pilih jenis enkripsi:

   - Dimiliki oleh Amazon DynamoDB. AWS KMS Kuncinya dimiliki dan dikelola oleh DynamoDB. Anda tidak dikenakan biaya tambahan untuk menggunakan kunci ini.
   - AWS kunci terkelola Alias kunci:`aws/dynamodb`. Kuncinya disimpan di akun Anda dan dikelola oleh AWS Key Management Service. (AWS KMS). AWS KMS dikenakan biaya.
   - Disimpan di akun Anda, serta dimiliki dan dikelola oleh Anda. Kuncinya disimpan di akun Anda dan dikelola oleh AWS Key Management Service. (AWS KMS). AWS KMS dikenakan biaya.

   > **Note**
   >
   > Jika Anda memilih untuk memiliki dan mengelola kunci Anda sendiri, pastikan Kebijakan Kunci KMS diatur dengan tepat. Untuk informasi selengkapnya, lihat Kebijakan kunci untuk kunci yang dikelola pelanggan.

   Lalu pilih Simpan untuk memperbarui tabel yang dienkripsi. Untuk mengonfirmasi jenis enkripsi, periksa detail tabel di bawah tab Gambaran Umum.

Memperbarui kunci enkripsi (AWS CLI)

Contoh-contoh berikut menunjukkan cara memperbarui tabel yang dienkripsi menggunakan AWS CLI.

Untuk memperbarui tabel terenkripsi dengan default Kunci milik AWS

- Perbarui tabel `Music` yang dienkripsi, seperti pada contoh berikut.

```
aws dynamodb update-table \
  --table-name Music \
  --sse-specification Enabled=false
```

> **Note**
>
> Tabel ini sekarang dienkripsi menggunakan default Kunci milik AWS di akun layanan DynamoDB.

Untuk memperbarui tabel terenkripsi dengan untuk Kunci yang dikelola AWS DynamoDB

- Perbarui tabel `Music` yang dienkripsi, seperti pada contoh berikut.

```
aws dynamodb update-table \
  --table-name Music \
  --sse-specification Enabled=true
```

Status `SSEDescription` dari deskripsi tabel diatur menjadi `ENABLED` dan `SSEType` adalah `KMS`.

```
"SSEDescription": {
  "SSEType": "KMS",
  "Status": "ENABLED",
  "KMSMasterKeyArn": "arn:aws:kms:us-east-1:123456789012:key/abcd1234-abcd-1234-
a123-ab1234a1b234",
}
```

Untuk memperbarui tabel yang dienkripsi dengan kunci yang dikelola pelanggan untuk DynamoDB

- Perbarui tabel `Music` yang dienkripsi, seperti pada contoh berikut.

```
aws dynamodb update-table \
  --table-name Music \
  --sse-specification Enabled=true,SSEType=KMS,KMSMasterKeyId=abcd1234-abcd-1234-
a123-ab1234a1b234
```

Status `SSEDescription` dari deskripsi tabel diatur menjadi `ENABLED` dan `SSEType` adalah `KMS`.

```
"SSEDescription": {
  "SSEType": "KMS",
```

```
  "Status": "ENABLED",
  "KMSMasterKeyArn": "arn:aws:kms:us-east-1:123456789012:key/abcd1234-abcd-1234-
a123-ab1234a1b234",
}
```

Mengamankan koneksi DynamoDB menggunakan titik akhir VPC dan kebijakan IAM”

Sambungan dilindungi baik antara Amazon DynamoDB dan aplikasi lokal dan antara DynamoDB dan sumber daya lain dalam Wilayah yang sama. AWS AWS

Kebijakan yang diperlukan untuk titik akhir

Amazon DynamoDB menyediakan API DescribeEndpoints yang memungkinkan Anda menghitung informasi titik akhir regional. Untuk permintaan ke titik akhir DynamoDB publik, API merespons terlepas dari kebijakan DynamoDB IAM yang dikonfigurasi, meskipun ada penolakan eksplisit atau implisit dalam kebijakan titik akhir IAM atau VPC. Ini karena DynamoDB sengaja melewatkan otorisasi untuk API. `DescribeEndpoints`

Untuk permintaan dari titik akhir VPC, kebijakan titik akhir IAM dan Cloud Privat Virtual (VPC) harus mengotorisasi panggilan API `DescribeEndpoints` untuk pengguna utama Identity and Access Management (IAM) yang meminta menggunakan tindakan `dynamodb:DescribeEndpoints` IAM. Jika tidak, akses ke API `DescribeEndpoints` akan ditolak.

Berikut adalah sebuah contoh kebijakan titik akhir.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": "(Include IAM Principals)",
            "Action": "dynamodb:DescribeEndpoints",
            "Resource": "*"
        }
    ]
}
```

Lalu lintas antara layanan dan aplikasi serta klien on-premise

Anda memiliki dua opsi konektivitas antara jaringan pribadi Anda dan AWS:

- AWS Site-to-Site VPN Koneksi. Untuk informasi selengkapnya, lihat Apa itu AWS Site-to-Site VPN? dalam AWS Site-to-Site VPN Panduan Pengguna.
- AWS Direct Connect Koneksi. Untuk informasi selengkapnya, lihat Apa itu AWS Direct Connect? dalam AWS Direct Connect Panduan Pengguna.

Akses ke DynamoDB melalui jaringan melalui dipublikasikan. AWS APIs Klien harus mendukung Keamanan Lapisan Pengangkutan (TLS) 1.2. Kami merekomendasikan TLS 1.3. Klien juga harus mendukung suite cipher dengan perfect forward secrecy (PFS) seperti Ephemeral Diffie-Hellman (DHE) atau Elliptic Curve Ephemeral Diffie-Hellman (ECDHE). Sebagian besar sistem modern seperti Java 7 dan versi yang lebih baru support mode ini. Selain itu, Anda harus menandatangani permintaan menggunakan kunci akses ID dan kunci akses rahasia yang terkait dengan IAM pengguna utama, atau Anda dapat menggunakan AWS Security Token Service (STS) untuk membuat kredensial keamanan sementara guna menandatangani permintaan.

Lalu lintas antara sumber daya AWS di Wilayah yang sama

Titik akhir Amazon Virtual Private Cloud (Amazon VPC) untuk DynamoDB adalah entitas logis dalam VPC yang memungkinkan konektivitas hanya ke DynamoDB. Amazon VPC merutekan permintaan ke DynamoDB dan merutekan respons kembali ke VPC. Untuk informasi selengkapnya, lihat Titik akhir VPC di Panduan Pengguna Amazon VPC. Misalnya, kebijakan yang dapat Anda gunakan untuk mengontrol akses dari titik akhir VPC, lihat Menggunakan Kebijakan IAM untuk mengontrol akses ke DynamoDB.

> **Note**
>
> Titik akhir Amazon VPC tidak dapat diakses melalui atau. AWS Site-to-Site VPN AWS Direct Connect

AWS Identity and Access Management (IAM) dan DynamoDB

AWS Identity and Access Management adalah AWS layanan yang membantu administrator mengontrol akses ke AWS sumber daya dengan aman. Administrator mengontrol siapa yang dapat diautentikasi (masuk) dan diotorisasi (memiliki izin) untuk menggunakan sumber daya Amazon

DynamoDB dan DynamoDB Accelerator. Anda dapat menggunakan IAM untuk mengelola izin akses dan menerapkan kebijakan keamanan untuk Amazon DynamoDB dan DynamoDB Accelerator. IAM adalah AWS layanan yang dapat Anda gunakan tanpa biaya tambahan.

Topik

- Manajemen Identitas dan Akses untuk Amazon DynamoDB
- Menggunakan ketentuan kebijakan IAM untuk kontrol akses terperinci

Manajemen Identitas dan Akses untuk Amazon DynamoDB

AWS Identity and Access Management (IAM) adalah Layanan AWS yang membantu administrator mengontrol akses ke AWS sumber daya dengan aman. Administrator IAM mengontrol siapa yang dapat diautentikasi (masuk) dan diotorisasi (memiliki izin) untuk menggunakan sumber daya DynamoDB. IAM adalah Layanan AWS yang dapat Anda gunakan tanpa biaya tambahan.

Topik

- Audiens
- Mengautentikasi dengan identitas
- Mengelola akses menggunakan kebijakan
- Cara kerja Amazon DynamoDB dengan IAM
- Contoh kebijakan berbasis identitas untuk Amazon DynamoDB
- Pemecahan masalah identitas dan akses Amazon DynamoDB
- Kebijakan IAM untuk mencegah pembelian kapasitas terpesan DynamoDB

Audiens

Bagaimana Anda menggunakan AWS Identity and Access Management (IAM) berbeda, tergantung pada pekerjaan yang Anda lakukan di DynamoDB.

Pengguna layanan – Jika Anda menggunakan layanan DynamoDB untuk melakukan tugas Anda, administrator Anda akan memberi Anda kredensial dan izin yang diperlukan. Saat Anda menggunakan lebih banyak fitur DynamoDB untuk melakukan pekerjaan Anda, izin tambahan mungkin diperlukan. Memahami cara akses dikelola dapat membantu Anda meminta izin yang tepat

dari administrator Anda. Jika Anda tidak dapat mengakses fitur di DynamoDB, lihat Pemecahan masalah identitas dan akses Amazon DynamoDB.

Administrator layanan – Jika Anda bertanggung jawab atas sumber daya DynamoDB di perusahaan Anda, Anda mungkin memiliki akses penuh ke DynamoDB. Tugas Anda adalah menentukan fitur dan sumber daya DynamoDB mana yang harus diakses oleh pengguna layanan Anda. Kemudian, Anda harus mengirimkan permintaan kepada administrator IAM Anda untuk mengubah izin pengguna layanan Anda. Tinjau informasi di halaman ini untuk memahami konsep dasar IAM. Untuk mempelajari selengkapnya tentang bagaimana perusahaan Anda dapat menggunakan IAM dengan DynamoDB, lihat Cara kerja Amazon DynamoDB dengan IAM.

Administrator IAM – Jika Anda adalah administrator IAM, Anda mungkin ingin mempelajari detail tentang cara menulis kebijakan untuk mengelola akses ke DynamoDB. Untuk melihat contoh kebijakan berbasis identitas DynamoDB yang dapat Anda gunakan di IAM, lihat Contoh kebijakan berbasis identitas untuk Amazon DynamoDB.

Mengautentikasi dengan identitas

Otentikasi adalah cara Anda masuk AWS menggunakan kredensi identitas Anda. Anda harus diautentikasi (masuk ke AWS) sebagai Pengguna root akun AWS, sebagai pengguna IAM, atau dengan mengasumsikan peran IAM.

Anda dapat masuk AWS sebagai identitas federasi dengan menggunakan kredensi yang disediakan melalui sumber identitas. AWS IAM Identity Center Pengguna (IAM Identity Center), autentikasi masuk tunggal perusahaan Anda, dan kredensi Google atau Facebook Anda adalah contoh identitas federasi. Saat Anda masuk sebagai identitas terfederasi, administrator Anda sebelumnya menyiapkan federasi identitas menggunakan peran IAM. Ketika Anda mengakses AWS dengan menggunakan federasi, Anda secara tidak langsung mengambil peran.

Bergantung pada jenis pengguna Anda, Anda dapat masuk ke AWS Management Console atau portal AWS akses. Untuk informasi selengkapnya tentang masuk AWS, lihat Cara masuk ke Panduan AWS Sign-In Pengguna Anda Akun AWS.

Jika Anda mengakses AWS secara terprogram, AWS sediakan kit pengembangan perangkat lunak (SDK) dan antarmuka baris perintah (CLI) untuk menandatangani permintaan Anda secara kriptografis dengan menggunakan kredensil Anda. Jika Anda tidak menggunakan AWS alat, Anda harus menandatangani permintaan sendiri. Guna mengetahui informasi selengkapnya tentang penggunaan metode yang disarankan untuk menandatangani permintaan sendiri, lihat AWS Signature Version 4 untuk permintaan API dalam Panduan Pengguna IAM.

Apa pun metode autentikasi yang digunakan, Anda mungkin diminta untuk menyediakan informasi keamanan tambahan. Misalnya, AWS merekomendasikan agar Anda menggunakan otentikasi multi-faktor (MFA) untuk meningkatkan keamanan akun Anda. Untuk mempelajari selengkapnya, lihat Autentikasi multi-faktor dalam Panduan Pengguna AWS IAM Identity Center dan Autentikasi multi-faktor AWS di IAM dalam Panduan Pengguna IAM.

Akun AWS pengguna root

Saat Anda membuat Akun AWS, Anda mulai dengan satu identitas masuk yang memiliki akses lengkap ke semua Layanan AWS dan sumber daya di akun. Identitas ini disebut pengguna Akun AWS root dan diakses dengan masuk dengan alamat email dan kata sandi yang Anda gunakan untuk membuat akun. Kami sangat menyarankan agar Anda tidak menggunakan pengguna root untuk tugas sehari-hari. Lindungi kredensial pengguna root Anda dan gunakan kredensial tersebut untuk melakukan tugas yang hanya dapat dilakukan pengguna root. Untuk daftar lengkap tugas yang mengharuskan Anda masuk sebagai pengguna root, lihat Tugas yang memerlukan kredensial pengguna root dalam Panduan Pengguna IAM.

Identitas gabungan

Sebagai praktik terbaik, mewajibkan pengguna manusia, termasuk pengguna yang memerlukan akses administrator, untuk menggunakan federasi dengan penyedia identitas untuk mengakses Layanan AWS dengan menggunakan kredensi sementara.

Identitas federasi adalah pengguna dari direktori pengguna perusahaan Anda, penyedia identitas web, direktori Pusat Identitas AWS Directory Service, atau pengguna mana pun yang mengakses Layanan AWS dengan menggunakan kredensil yang disediakan melalui sumber identitas. Ketika identitas federasi mengakses Akun AWS, mereka mengambil peran, dan peran memberikan kredensi sementara.

Untuk manajemen akses terpusat, kami sarankan Anda menggunakan AWS IAM Identity Center. Anda dapat membuat pengguna dan grup di Pusat Identitas IAM, atau Anda dapat menghubungkan dan menyinkronkan ke sekumpulan pengguna dan grup di sumber identitas Anda sendiri untuk digunakan di semua aplikasi Akun AWS dan aplikasi Anda. Untuk informasi tentang Pusat Identitas IAM, lihat Apakah itu Pusat Identitas IAM? dalam Panduan Pengguna AWS IAM Identity Center .

Pengguna dan grup IAM

Pengguna IAM adalah identitas dalam diri Anda Akun AWS yang memiliki izin khusus untuk satu orang atau aplikasi. Jika memungkinkan, kami merekomendasikan untuk mengandalkan kredensial

sementara, bukan membuat pengguna IAM yang memiliki kredensial jangka panjang seperti kata sandi dan kunci akses. Namun, jika Anda memiliki kasus penggunaan tertentu yang memerlukan kredensial jangka panjang dengan pengguna IAM, kami merekomendasikan Anda merotasi kunci akses. Untuk informasi selengkapnya, lihat Merotasi kunci akses secara teratur untuk kasus penggunaan yang memerlukan kredensial jangka panjang dalam Panduan Pengguna IAM.

Grup IAM adalah identitas yang menentukan sekumpulan pengguna IAM. Anda tidak dapat masuk sebagai grup. Anda dapat menggunakan grup untuk menentukan izin bagi beberapa pengguna sekaligus. Grup mempermudah manajemen izin untuk sejumlah besar pengguna sekaligus. Misalnya, Anda dapat meminta kelompok untuk menyebutkan IAMAdmins dan memberikan izin kepada grup tersebut untuk mengelola sumber daya IAM.

Pengguna berbeda dari peran. Pengguna secara unik terkait dengan satu orang atau aplikasi, tetapi peran dimaksudkan untuk dapat digunakan oleh siapa pun yang membutuhkannya. Pengguna memiliki kredensial jangka panjang permanen, tetapi peran memberikan kredensial sementara. Untuk mempelajari selengkapnya, lihat Kasus penggunaan untuk pengguna IAM dalam Panduan Pengguna IAM.

Peran IAM

Peran IAM adalah identitas dalam diri Anda Akun AWS yang memiliki izin khusus. Peran ini mirip dengan pengguna IAM, tetapi tidak terkait dengan orang tertentu. Untuk mengambil peran IAM sementara AWS Management Console, Anda dapat beralih dari pengguna ke peran IAM (konsol). Anda dapat mengambil peran dengan memanggil operasi AWS CLI atau AWS API atau dengan menggunakan URL kustom. Untuk informasi selengkapnya tentang cara menggunakan peran, lihat Metode untuk mengambil peran dalam Panduan Pengguna IAM.

Peran IAM dengan kredensial sementara berguna dalam situasi berikut:

- Akses pengguna terfederasi – Untuk menetapkan izin ke identitas terfederasi, Anda membuat peran dan menentukan izin untuk peran tersebut. Ketika identitas terfederasi mengautentikasi, identitas tersebut terhubung dengan peran dan diberi izin yang ditentukan oleh peran. Untuk informasi tentang peran untuk federasi, lihat Buat peran untuk penyedia identitas pihak ketiga dalam Panduan Pengguna IAM. Jika menggunakan Pusat Identitas IAM, Anda harus mengonfigurasi set izin. Untuk mengontrol apa yang dapat diakses identitas Anda setelah identitas tersebut diautentikasi, Pusat Identitas IAM akan mengorelasikan set izin ke peran dalam IAM. Untuk informasi tentang set izin, lihat Set izin dalam Panduan Pengguna AWS IAM Identity Center .
- Izin pengguna IAM sementara – Pengguna atau peran IAM dapat mengambil peran IAM guna mendapatkan berbagai izin secara sementara untuk tugas tertentu.

- Akses lintas akun – Anda dapat menggunakan peran IAM untuk mengizinkan seseorang (prinsipal tepercaya) di akun lain untuk mengakses sumber daya di akun Anda. Peran adalah cara utama untuk memberikan akses lintas akun. Namun, dengan beberapa Layanan AWS, Anda dapat melampirkan kebijakan secara langsung ke sumber daya (alih-alih menggunakan peran sebagai proxy). Untuk mempelajari perbedaan antara peran dan kebijakan berbasis sumber daya untuk akses lintas akun, lihat Akses sumber daya lintas akun di IAM dalam Panduan Pengguna IAM.
- Akses lintas layanan — Beberapa Layanan AWS menggunakan fitur lain Layanan AWS. Misalnya, saat Anda melakukan panggilan dalam suatu layanan, biasanya layanan tersebut menjalankan aplikasi di Amazon EC2 atau menyimpan objek di Amazon S3. Sebuah layanan mungkin melakukannya menggunakan izin prinsipal yang memanggil, menggunakan peran layanan, atau peran terkait layanan.
  - Sesi akses teruskan (FAS) — Saat Anda menggunakan pengguna atau peran IAM untuk melakukan tindakan AWS, Anda dianggap sebagai prinsipal. Ketika Anda menggunakan beberapa layanan, Anda mungkin melakukan sebuah tindakan yang kemudian menginisiasi tindakan lain di layanan yang berbeda. FAS menggunakan izin dari pemanggilan utama Layanan AWS, dikombinasikan dengan permintaan Layanan AWS untuk membuat permintaan ke layanan hilir. Permintaan FAS hanya dibuat ketika layanan menerima permintaan yang memerlukan interaksi dengan orang lain Layanan AWS atau sumber daya untuk menyelesaikannya. Dalam hal ini, Anda harus memiliki izin untuk melakukan kedua tindakan tersebut. Untuk detail kebijakan ketika mengajukan permintaan FAS, lihat Sesi akses maju.
  - Peran layanan – Peran layanan adalah peran IAM yang dijalankan oleh layanan untuk melakukan tindakan atas nama Anda. Administrator IAM dapat membuat, mengubah, dan menghapus peran layanan dari dalam IAM. Untuk informasi selengkapnya, lihat Buat sebuah peran untuk mendelegasikan izin ke Layanan AWS dalam Panduan pengguna IAM.
  - Peran terkait layanan — Peran terkait layanan adalah jenis peran layanan yang ditautkan ke peran layanan. Layanan AWS Layanan tersebut dapat menjalankan peran untuk melakukan tindakan atas nama Anda. Peran terkait layanan muncul di Anda Akun AWS dan dimiliki oleh layanan. Administrator IAM dapat melihat, tetapi tidak dapat mengedit izin untuk peran terkait layanan.
- Aplikasi yang berjalan di Amazon EC2 — Anda dapat menggunakan peran IAM untuk mengelola kredensi sementara untuk aplikasi yang berjalan pada EC2 instance dan membuat AWS CLI atau AWS permintaan API. Ini lebih baik untuk menyimpan kunci akses dalam EC2 instance. Untuk menetapkan AWS peran ke EC2 instance dan membuatnya tersedia untuk semua aplikasinya, Anda membuat profil instance yang dilampirkan ke instance. Profil instance berisi peran dan memungkinkan program yang berjalan pada EC2 instance untuk mendapatkan kredensi

sementara. Untuk informasi selengkapnya, lihat Menggunakan peran IAM untuk memberikan izin ke aplikasi yang berjalan di EC2 instans Amazon di Panduan Pengguna IAM.

Mengelola akses menggunakan kebijakan

Anda mengontrol akses AWS dengan membuat kebijakan dan melampirkannya ke AWS identitas atau sumber daya. Kebijakan adalah objek AWS yang, ketika dikaitkan dengan identitas atau sumber daya, menentukan izinnya. AWS mengevaluasi kebijakan ini ketika prinsipal (pengguna, pengguna root, atau sesi peran) membuat permintaan. Izin dalam kebijakan menentukan apakah permintaan diizinkan atau ditolak. Sebagian besar kebijakan disimpan AWS sebagai dokumen JSON. Untuk informasi selengkapnya tentang struktur dan isi dokumen kebijakan JSON, lihat Gambaran umum kebijakan JSON dalam Panduan Pengguna IAM.

Administrator dapat menggunakan kebijakan AWS JSON untuk menentukan siapa yang memiliki akses ke apa. Yaitu, principal dapat melakukan tindakan pada suatu sumber daya, dan dalam suatu syarat.

Secara default, pengguna dan peran tidak memiliki izin. Untuk memberikan izin kepada pengguna untuk melakukan tindakan di sumber daya yang mereka perlukan, administrator IAM dapat membuat kebijakan IAM. Administrator kemudian dapat menambahkan kebijakan IAM ke peran, dan pengguna dapat mengambil peran.

Kebijakan IAM mendefinisikan izin untuk suatu tindakan terlepas dari metode yang Anda gunakan untuk melakukan operasinya. Misalnya, anggaplah Anda memiliki kebijakan yang mengizinkan tindakan `iam:GetRole`. Pengguna dengan kebijakan tersebut bisa mendapatkan informasi peran dari AWS Management Console, API AWS CLI, atau AWS API.

Kebijakan berbasis identitas

Kebijakan berbasis identitas adalah dokumen kebijakan izin JSON yang dapat Anda lampirkan ke sebuah identitas, seperti pengguna IAM, grup pengguna IAM, atau peran IAM. Kebijakan ini mengontrol jenis tindakan yang dapat dilakukan oleh pengguna dan peran, di sumber daya mana, dan berdasarkan kondisi seperti apa. Untuk mempelajari cara membuat kebijakan berbasis identitas, lihat Tentukan izin IAM kustom dengan kebijakan terkelola pelanggan dalam Panduan Pengguna IAM.

Kebijakan berbasis identitas dapat dikategorikan lebih lanjut sebagai kebijakan inline atau kebijakan yang dikelola. Kebijakan inline disematkan langsung ke satu pengguna, grup, atau peran. Kebijakan

terkelola adalah kebijakan mandiri yang dapat Anda lampirkan ke beberapa pengguna, grup, dan peran dalam. Akun AWS Kebijakan AWS terkelola mencakup kebijakan terkelola dan kebijakan yang dikelola pelanggan. Untuk mempelajari cara memilih antara kebijakan yang dikelola atau kebijakan inline, lihat Pilih antara kebijakan yang dikelola dan kebijakan inline dalam Panduan Pengguna IAM.

Kebijakan berbasis sumber daya

Kebijakan berbasis sumber daya adalah dokumen kebijakan JSON yang Anda lampirkan ke sumber daya. Contoh kebijakan berbasis sumber daya adalah kebijakan kepercayaan peran IAM dan kebijakan bucket Amazon S3. Dalam layanan yang mendukung kebijakan berbasis sumber daya, administrator layanan dapat menggunakannya untuk mengontrol akses ke sumber daya tertentu. Untuk sumber daya tempat kebijakan dilampirkan, kebijakan menentukan tindakan apa yang dapat dilakukan oleh prinsipal tertentu pada sumber daya tersebut dan dalam kondisi apa. Anda harus menentukan prinsipal dalam kebijakan berbasis sumber daya. Prinsipal dapat mencakup akun, pengguna, peran, pengguna federasi, atau. Layanan AWS

Kebijakan berbasis sumber daya merupakan kebijakan inline yang terletak di layanan tersebut. Anda tidak dapat menggunakan kebijakan AWS terkelola dari IAM dalam kebijakan berbasis sumber daya.

Daftar kontrol akses (ACLs)

Access control lists (ACLs) mengontrol prinsipal mana (anggota akun, pengguna, atau peran) yang memiliki izin untuk mengakses sumber daya. ACLs mirip dengan kebijakan berbasis sumber daya, meskipun mereka tidak menggunakan format dokumen kebijakan JSON.

Amazon S3, AWS WAF, dan Amazon VPC adalah contoh layanan yang mendukung. ACLs Untuk mempelajari selengkapnya ACLs, lihat Ringkasan daftar kontrol akses (ACL) di Panduan Pengembang Layanan Penyimpanan Sederhana Amazon.

Jenis-jenis kebijakan lain

AWS mendukung jenis kebijakan tambahan yang kurang umum. Jenis-jenis kebijakan ini dapat mengatur izin maksimum yang diberikan kepada Anda oleh jenis kebijakan yang lebih umum.

- Batasan izin – Batasan izin adalah fitur lanjutan tempat Anda mengatur izin maksimum yang dapat diberikan oleh kebijakan berbasis identitas ke entitas IAM (pengguna IAM atau peran IAM). Anda dapat menetapkan batasan izin untuk suatu entitas. Izin yang dihasilkan adalah perpotongan antara kebijakan berbasis identitas milik entitas dan batasan izinnya. Kebijakan berbasis sumber daya yang menentukan pengguna atau peran dalam bidang `Principal` tidak dibatasi oleh batasan izin. Penolakan eksplisit dalam salah satu kebijakan ini akan menggantikan pemberian

izin. Untuk informasi selengkapnya tentang batasan izin, lihat Batasan izin untuk entitas IAM dalam Panduan Pengguna IAM.

- Kebijakan kontrol layanan (SCPs) — SCPs adalah kebijakan JSON yang menentukan izin maksimum untuk organisasi atau unit organisasi (OU) di. AWS Organizations AWS Organizations adalah layanan untuk mengelompokkan dan mengelola secara terpusat beberapa Akun AWS yang dimiliki bisnis Anda. Jika Anda mengaktifkan semua fitur dalam suatu organisasi, maka Anda dapat menerapkan kebijakan kontrol layanan (SCPs) ke salah satu atau semua akun Anda. SCP membatasi izin untuk entitas di akun anggota, termasuk masing-masing. Pengguna root akun AWS Untuk informasi selengkapnya tentang Organizations dan SCPs, lihat Kebijakan kontrol layanan di Panduan AWS Organizations Pengguna.
- Kebijakan kontrol sumber daya (RCPs) — RCPs adalah kebijakan JSON yang dapat Anda gunakan untuk menetapkan izin maksimum yang tersedia untuk sumber daya di akun Anda tanpa memperbarui kebijakan IAM yang dilampirkan ke setiap sumber daya yang Anda miliki. RCP membatasi izin untuk sumber daya di akun anggota dan dapat memengaruhi izin efektif untuk identitas, termasuk Pengguna root akun AWS, terlepas dari apakah itu milik organisasi Anda. Untuk informasi selengkapnya tentang Organizations dan RCPs, termasuk daftar dukungan Layanan AWS tersebut RCPs, lihat Kebijakan kontrol sumber daya (RCPs) di Panduan AWS Organizations Pengguna.
- Kebijakan sesi – Kebijakan sesi adalah kebijakan lanjutan yang Anda berikan sebagai parameter ketika Anda membuat sesi sementara secara programatis untuk peran atau pengguna terfederasi. Izin sesi yang dihasilkan adalah perpotongan antara kebijakan berbasis identitas pengguna atau peran dan kebijakan sesi. Izin juga bisa datang dari kebijakan berbasis sumber daya. Penolakan eksplisit dalam salah satu kebijakan ini akan menggantikan pemberian izin. Untuk informasi selengkapnya, lihat Kebijakan sesi dalam Panduan Pengguna IAM.

Berbagai jenis kebijakan

Ketika beberapa jenis kebijakan berlaku pada suatu permintaan, izin yang dihasilkan lebih rumit untuk dipahami. Untuk mempelajari cara AWS menentukan apakah akan mengizinkan permintaan saat beberapa jenis kebijakan terlibat, lihat Logika evaluasi kebijakan di Panduan Pengguna IAM.

Cara kerja Amazon DynamoDB dengan IAM

Sebelum Anda menggunakan IAM untuk mengelola akses ke DynamoDB, pelajari fitur IAM apakah yang terdapat untuk digunakan pada DynamoDB.

Fitur IAM	Dukungan DynamoDB
Kebijakan berbasis identitas	Ya
Kebijakan berbasis sumber daya	Ya
Tindakan kebijakan	Ya
Sumber daya kebijakan	Ya
Kunci kondisi kebijakan	Ya
ACLs	Tidak
ABAC (tanda dalam kebijakan)	Ya
Kredensial sementara	Ya
Izin principal	Ya
Peran layanan	Ya
Peran terkait layanan	Ya

Untuk mendapatkan tampilan tingkat tinggi tentang cara kerja DynamoDB dan layanan AWS lainnya dengan sebagian besar fitur IAM, AWS lihat layanan yang bekerja dengan IAM di Panduan Pengguna IAM.

Kebijakan berbasis identitas untuk DynamoDB

Mendukung kebijakan berbasis identitas: Ya

Kebijakan berbasis identitas adalah dokumen kebijakan izin JSON yang dapat Anda lampirkan ke sebuah identitas, seperti pengguna IAM, grup pengguna IAM, atau peran IAM. Kebijakan ini mengontrol jenis tindakan yang dapat dilakukan oleh pengguna dan peran, di sumber daya mana, dan berdasarkan kondisi seperti apa. Untuk mempelajari cara membuat kebijakan berbasis identitas, lihat Tentukan izin IAM kustom dengan kebijakan terkelola pelanggan dalam Panduan Pengguna IAM.

Dengan kebijakan berbasis identitas IAM, Anda dapat menentukan secara spesifik apakah tindakan dan sumber daya diizinkan atau ditolak, serta kondisi yang menjadi dasar dikabulkan atau ditolaknya tindakan tersebut. Anda tidak dapat menentukan secara spesifik prinsipal dalam sebuah kebijakan berbasis identitas karena prinsipal berlaku bagi pengguna atau peran yang melekat kepadanya. Untuk mempelajari semua elemen yang dapat Anda gunakan dalam kebijakan JSON, lihat Referensi elemen kebijakan JSON IAM dalam Panduan Pengguna IAM.

Contoh kebijakan berbasis identitas untuk DynamoDB

Untuk melihat contoh kebijakan berbasis identitas DynamoDB, silakan lihat Contoh kebijakan berbasis identitas untuk Amazon DynamoDB.

Kebijakan berbasis sumber daya dalam DynamoDB

Mendukung kebijakan berbasis sumber daya: Ya

Kebijakan berbasis sumber daya adalah dokumen kebijakan JSON yang Anda lampirkan ke sumber daya. Contoh kebijakan berbasis sumber daya adalah kebijakan kepercayaan peran IAM dan kebijakan bucket Amazon S3. Dalam layanan yang mendukung kebijakan berbasis sumber daya, administrator layanan dapat menggunakannya untuk mengontrol akses ke sumber daya tertentu. Untuk sumber daya tempat kebijakan dilampirkan, kebijakan menentukan tindakan apa yang dapat dilakukan oleh prinsipal tertentu pada sumber daya tersebut dan dalam kondisi apa. Anda harus menentukan prinsipal dalam kebijakan berbasis sumber daya. Prinsipal dapat mencakup akun, pengguna, peran, pengguna federasi, atau. Layanan AWS

Untuk mengaktifkan akses lintas akun, Anda dapat menentukan secara spesifik seluruh akun atau entitas IAM di akun lain sebagai prinsipal dalam kebijakan berbasis sumber daya. Menambahkan prinsipal akun silang ke kebijakan berbasis sumber daya hanya setengah dari membangun hubungan kepercayaan. Ketika prinsipal dan sumber daya berbeda Akun AWS, administrator IAM di akun tepercaya juga harus memberikan izin entitas utama (pengguna atau peran) untuk mengakses sumber daya. Mereka memberikan izin dengan melampirkan kebijakan berbasis identitas kepada entitas. Namun, jika kebijakan berbasis sumber daya memberikan akses ke principal dalam akun yang sama, tidak diperlukan kebijakan berbasis identitas tambahan. Untuk informasi selengkapnya, lihat Akses sumber daya lintas akun di IAM dalam Panduan Pengguna IAM.

Tindakan kebijakan untuk DynamoDB

Mendukung tindakan kebijakan: Ya

Administrator dapat menggunakan kebijakan AWS JSON untuk menentukan siapa yang memiliki akses ke apa. Yaitu, di mana utama dapat melakukan tindakan pada sumber daya, dan dalam kondisi apa.

Elemen `Action` dari kebijakan JSON menjelaskan tindakan yang dapat Anda gunakan untuk mengizinkan atau menolak akses dalam sebuah kebijakan. Tindakan kebijakan biasanya memiliki nama yang sama dengan operasi AWS API terkait. Ada beberapa pengecualian, misalnya tindakan hanya izin yang tidak memiliki operasi API yang cocok. Ada juga beberapa operasi yang memerlukan beberapa tindakan dalam suatu kebijakan. Tindakan tambahan ini disebut tindakan dependen.

Menyertakan tindakan dalam kebijakan untuk memberikan izin untuk melakukan operasi terkait.

Untuk melihat daftar tindakan DynamoDB, lihat Tindakan yang ditentukan oleh Amazon DynamoDB di Referensi Otorisasi Layanan.

Tindakan kebijakan di DynamoDB menggunakan prefiks berikut sebelum tindakan:

```
aws
```

Untuk menetapkan secara spesifik beberapa tindakan dalam satu pernyataan, pisahkan tindakan tersebut dengan koma.

```
"Action": [
      "aws:action1",
      "aws:action2"
         ]
```

Untuk melihat contoh kebijakan berbasis identitas DynamoDB, silakan lihat Contoh kebijakan berbasis identitas untuk Amazon DynamoDB.

Sumber daya kebijakan untuk DynamoDB

Mendukung sumber daya kebijakan: Ya

Administrator dapat menggunakan kebijakan AWS JSON untuk menentukan siapa yang memiliki akses ke apa. Yaitu, di mana utama dapat melakukan tindakan pada sumber daya, dan dalam kondisi apa.

Elemen kebijakan JSON `Resource` menentukan objek yang menjadi target penerapan tindakan. Pernyataan harus menyertakan elemen `Resource` atau `NotResource`. Praktik terbaiknya, tentukan

sumber daya menggunakan Amazon Resource Name (ARN). Anda dapat melakukan ini untuk tindakan yang mendukung jenis sumber daya tertentu, yang dikenal sebagai izin tingkat sumber daya.

Untuk tindakan yang tidak mendukung izin di tingkat sumber daya, misalnya operasi pencantuman, gunakan wildcard (*) untuk menunjukkan bahwa pernyataan tersebut berlaku untuk semua sumber daya.

```
"Resource": "*"
```

Untuk melihat daftar jenis sumber daya DynamoDB dan ARNs jenisnya, lihat Sumber daya yang ditentukan oleh Amazon DynamoDB di Referensi Otorisasi Layanan. Untuk mempelajari tindakan mana yang dapat Anda tentukan ARN setiap sumber daya, lihat Tindakan yang ditentukan oleh Amazon DynamoDB.

Untuk melihat contoh kebijakan berbasis identitas DynamoDB, silakan lihat Contoh kebijakan berbasis identitas untuk Amazon DynamoDB.

Kunci persyaratan kebijakan untuk DynamoDB

Mendukung kunci kondisi kebijakan khusus layanan: Yes

Administrator dapat menggunakan kebijakan AWS JSON untuk menentukan siapa yang memiliki akses ke apa. Yaitu, di mana utama dapat melakukan tindakan pada sumber daya, dan dalam kondisi apa.

Elemen `Condition` (atau blok `Condition`) akan memungkinkan Anda menentukan kondisi yang menjadi dasar suatu pernyataan berlaku. Elemen `Condition` bersifat opsional. Anda dapat membuat ekspresi bersyarat yang menggunakan operator kondisi, misalnya sama dengan atau kurang dari, untuk mencocokkan kondisi dalam kebijakan dengan nilai-nilai yang diminta.

Jika Anda menentukan beberapa elemen `Condition` dalam sebuah pernyataan, atau beberapa kunci dalam elemen `Condition` tunggal, maka AWS akan mengevaluasinya menggunakan operasi `AND` logis. Jika Anda menentukan beberapa nilai untuk satu kunci kondisi, AWS mengevaluasi kondisi menggunakan `OR` operasi logis. Semua kondisi harus dipenuhi sebelum izin pernyataan diberikan.

Anda juga dapat menggunakan variabel placeholder saat menentukan kondisi. Sebagai contoh, Anda dapat memberikan izin kepada pengguna IAM untuk mengakses sumber daya hanya jika

izin tersebut mempunyai tanda yang sesuai dengan nama pengguna IAM mereka. Untuk informasi selengkapnya, lihat Elemen kebijakan IAM: variabel dan tanda dalam Panduan Pengguna IAM.

AWS mendukung kunci kondisi global dan kunci kondisi khusus layanan. Untuk melihat semua kunci kondisi AWS global, lihat kunci konteks kondisi AWS global di Panduan Pengguna IAM.

Untuk melihat daftar kunci persyaratan DynamoDB, lihat Kunci persyaratan untuk Amazon DynamoDB dalam Referensi Otorisasi Layanan. Untuk mempelajari tindakan dan sumber daya yang dapat Anda gunakan dengan kunci kondisi, lihat Tindakan yang ditentukan oleh Amazon DynamoDB.

Untuk melihat contoh kebijakan berbasis identitas DynamoDB, silakan lihat Contoh kebijakan berbasis identitas untuk Amazon DynamoDB.

Daftar kontrol akses (ACLs) di DynamoDB

Mendukung ACLs: Tidak

Access control lists (ACLs) mengontrol prinsipal mana (anggota akun, pengguna, atau peran) yang memiliki izin untuk mengakses sumber daya. ACLs mirip dengan kebijakan berbasis sumber daya, meskipun mereka tidak menggunakan format dokumen kebijakan JSON.

Kontrol akses berbasis atribut (ABAC) dengan DynamoDB

Mendukung ABAC (tanda dalam kebijakan): Ya

Kontrol akses berbasis atribut (ABAC) adalah strategi otorisasi yang menentukan izin berdasarkan atribut. Dalam AWS, atribut ini disebut tag. Anda dapat melampirkan tag ke entitas IAM (pengguna atau peran) dan ke banyak AWS sumber daya. Penandaan ke entitas dan sumber daya adalah langkah pertama dari ABAC. Kemudian rancanglah kebijakan ABAC untuk mengizinkan operasi ketika tanda milik prinsipal cocok dengan tanda yang ada di sumber daya yang ingin diakses.

ABAC sangat berguna di lingkungan yang berkembang dengan cepat dan berguna di situasi saat manajemen kebijakan menjadi rumit.

Untuk mengendalikan akses berdasarkan tanda, berikan informasi tentang tanda di elemen kondisi dari kebijakan menggunakan kunci kondisi `aws:ResourceTag/`*`key-name`*, `aws:RequestTag/`*`key-name`*, atau `aws:TagKeys`.

Jika sebuah layanan mendukung ketiga kunci kondisi untuk setiap jenis sumber daya, nilainya adalah Ya untuk layanan tersebut. Jika suatu layanan mendukung ketiga kunci kondisi untuk hanya beberapa jenis sumber daya, nilainya adalah Parsial.

Untuk informasi selengkapnya tentang ABAC, lihat Tentukan izin dengan otorisasi ABAC dalam Panduan Pengguna IAM. Untuk melihat tutorial yang menguraikan langkah-langkah pengaturan ABAC, lihat Menggunakan kontrol akses berbasis atribut (ABAC) dalam Panduan Pengguna IAM.

Menggunakan Kredensial sementara dengan DynamoDB

Mendukung kredensial sementara: Ya

Beberapa Layanan AWS tidak berfungsi saat Anda masuk menggunakan kredensil sementara. Untuk informasi tambahan, termasuk yang Layanan AWS bekerja dengan kredensi sementara, lihat Layanan AWS yang bekerja dengan IAM di Panduan Pengguna IAM.

Anda menggunakan kredensi sementara jika Anda masuk AWS Management Console menggunakan metode apa pun kecuali nama pengguna dan kata sandi. Misalnya, ketika Anda mengakses AWS menggunakan tautan masuk tunggal (SSO) perusahaan Anda, proses tersebut secara otomatis membuat kredensil sementara. Anda juga akan secara otomatis membuat kredensial sementara ketika Anda masuk ke konsol sebagai seorang pengguna lalu beralih peran. Untuk informasi selengkapnya tentang peralihan peran, lihat Beralih dari pengguna ke peran IAM (konsol) dalam Panduan Pengguna IAM.

Anda dapat membuat kredenal sementara secara manual menggunakan API AWS CLI atau AWS . Anda kemudian dapat menggunakan kredensil sementara tersebut untuk mengakses. AWS AWS merekomendasikan agar Anda secara dinamis menghasilkan kredensi sementara alih-alih menggunakan kunci akses jangka panjang. Untuk informasi selengkapnya, lihat Kredensial keamanan sementara di IAM.

Izin pengguna utama lintas layanan untuk DynamoDB

Mendukung sesi akses maju (FAS): Ya

Saat Anda menggunakan pengguna atau peran IAM untuk melakukan tindakan AWS, Anda dianggap sebagai prinsipal. Ketika Anda menggunakan beberapa layanan, Anda mungkin melakukan sebuah tindakan yang kemudian menginisiasi tindakan lain di layanan yang berbeda. FAS menggunakan izin dari pemanggilan utama Layanan AWS, dikombinasikan dengan permintaan Layanan AWS untuk membuat permintaan ke layanan hilir. Permintaan FAS hanya dibuat ketika layanan menerima permintaan yang memerlukan interaksi dengan orang lain Layanan AWS atau sumber daya untuk menyelesaikannya. Dalam hal ini, Anda harus memiliki izin untuk melakukan kedua tindakan tersebut. Untuk detail kebijakan ketika mengajukan permintaan FAS, lihat Sesi akses maju.

Peran layanan untuk DynamoDB

Mendukung peran layanan: Ya

Peran layanan adalah peran IAM yang diambil oleh sebuah layanan untuk melakukan tindakan atas nama Anda. Administrator IAM dapat membuat, mengubah, dan menghapus peran layanan dari dalam IAM. Untuk informasi selengkapnya, lihat Buat sebuah peran untuk mendelegasikan izin ke Layanan AWS dalam Panduan pengguna IAM.

> **Warning**
>
> Mengubah izin untuk sebuah peran layanan dapat merusak fungsionalitas DynamoDB. Edit peran layanan hanya jika DynamoDB memberikan panduan untuk melakukannya.

Peran terkait layanan untuk DynamoDB

Mendukung peran terkait layanan: Ya

Peran terkait layanan adalah jenis peran layanan yang ditautkan ke. Layanan AWS Layanan tersebut dapat menjalankan peran untuk melakukan tindakan atas nama Anda. Peran terkait layanan muncul di Anda Akun AWS dan dimiliki oleh layanan. Administrator IAM dapat melihat, tetapi tidak dapat mengedit izin untuk peran terkait layanan.

Untuk detail tentang pembuatan atau manajemen peran terkait layanan, lihat Layanan AWS yang berfungsi dengan IAM. Cari layanan dalam tabel yang memiliki `Yes` di kolom Peran terkait layanan. Pilih tautan Ya untuk melihat dokumentasi peran terkait layanan untuk layanan tersebut.

Peran terkait layanan yang didukung di DynamoDB

Peran terkait layanan berikut didukung di DynamoDB.

- DynamoDB menggunakan AWSServiceRoleForDynamoDBReplicationperan terkait layanan untuk replikasi tabel global. Wilayah AWS Lihat the section called “Menggunakan IAM dengan tabel global DynamoDB” untuk informasi selengkapnya tentang AWSServiceRoleForDynamoDBReplicationperan terkait layanan.
- DynamoDB Accelerator (DAX) menggunakan AWSServiceRoleForperan terkait layanan DAX untuk mengonfigurasi dan memelihara kluster DAX. Lihat the section called “Menggunakan peran tertaut layanan untuk DAX” untuk informasi selengkapnya tentang peran AWSServiceRoleForterkait layanan DAX.

Selain peran terkait layanan DynamoDB ini, DynamoDB menggunakan layanan Application Auto Scaling untuk mengelola pengaturan throughput secara otomatis pada tabel mode kapasitas yang disediakan. Layanan Application Auto Scaling menggunakan peran terkait layanan AWSServiceRoleForApplicationAutoScaling_DynamoDBTableuntuk mengelola pengaturan throughput pada tabel DynamoDB yang mengaktifkan penskalaan otomatis. Lihat Peran terkait layanan untuk Application Auto Scaling untuk informasi selengkapnya.

Contoh kebijakan berbasis identitas untuk Amazon DynamoDB

Secara default, pengguna dan peran tidak memiliki izin untuk membuat atau melakukan modifikasi atas sumber daya DynamoDB. Mereka juga tidak dapat melakukan tugas dengan menggunakan AWS Management Console, AWS Command Line Interface (AWS CLI), atau AWS API. Untuk memberikan izin kepada pengguna untuk melakukan tindakan di sumber daya yang mereka perlukan, administrator IAM dapat membuat kebijakan IAM. Administrator kemudian dapat menambahkan kebijakan IAM ke peran, dan pengguna dapat mengambil peran.

Untuk mempelajari cara membuat kebijakan berbasis identitas IAM dengan menggunakan contoh dokumen kebijakan JSON ini, lihat Membuat kebijakan IAM (konsol) di Panduan Pengguna IAM.

Untuk detail tentang tindakan dan jenis sumber daya yang ditentukan oleh DynamoDB, termasuk format untuk setiap jenis sumber daya, lihat Kunci tindakan, sumber daya, dan kondisi untuk Amazon DynamoDB dalam Referensi Otorisasi Layanan. ARNs

Topik

- Praktik terbaik kebijakan
- Menggunakan konsol DynamoDB
- Mengizinkan pengguna melihat izin mereka sendiri
- Menggunakan kebijakan berbasis identitas dengan Amazon DynamoDB

Praktik terbaik kebijakan

Kebijakan berbasis identitas menentukan apakah seseorang dapat membuat, mengakses, atau menghapus sumber daya DynamoDB yang ada di akun Anda. Tindakan ini mengenakan biaya kepada Akun AWS Anda. Ketika Anda membuat atau mengedit kebijakan berbasis identitas, ikuti panduan dan rekomendasi ini:

- Mulailah dengan kebijakan AWS terkelola dan beralih ke izin hak istimewa paling sedikit — Untuk mulai memberikan izin kepada pengguna dan beban kerja Anda, gunakan kebijakan AWS terkelola

yang memberikan izin untuk banyak kasus penggunaan umum. Mereka tersedia di Anda Akun AWS. Kami menyarankan Anda mengurangi izin lebih lanjut dengan menentukan kebijakan yang dikelola AWS pelanggan yang khusus untuk kasus penggunaan Anda. Untuk informasi selengkapnya, lihat Kebijakan yang dikelola AWS atau Kebijakan yang dikelola AWS untuk fungsi tugas dalam Panduan Pengguna IAM.

- Menerapkan izin dengan hak akses paling rendah – Ketika Anda menetapkan izin dengan kebijakan IAM, hanya berikan izin yang diperlukan untuk melakukan tugas. Anda melakukannya dengan mendefinisikan tindakan yang dapat diambil pada sumber daya tertentu dalam kondisi tertentu, yang juga dikenal sebagai izin dengan hak akses paling rendah. Untuk informasi selengkapnya tentang cara menggunakan IAM untuk mengajukan izin, lihat Kebijakan dan izin dalam IAM dalam Panduan Pengguna IAM.
- Gunakan kondisi dalam kebijakan IAM untuk membatasi akses lebih lanjut – Anda dapat menambahkan suatu kondisi ke kebijakan Anda untuk membatasi akses ke tindakan dan sumber daya. Sebagai contoh, Anda dapat menulis kondisi kebijakan untuk menentukan bahwa semua permintaan harus dikirim menggunakan SSL. Anda juga dapat menggunakan ketentuan untuk memberikan akses ke tindakan layanan jika digunakan melalui yang spesifik Layanan AWS, seperti AWS CloudFormation. Untuk informasi selengkapnya, lihat Elemen kebijakan JSON IAM: Kondisi dalam Panduan Pengguna IAM.
- Gunakan IAM Access Analyzer untuk memvalidasi kebijakan IAM Anda untuk memastikan izin yang aman dan fungsional – IAM Access Analyzer memvalidasi kebijakan baru dan yang sudah ada sehingga kebijakan tersebut mematuhi bahasa kebijakan IAM (JSON) dan praktik terbaik IAM. IAM Access Analyzer menyediakan lebih dari 100 pemeriksaan kebijakan dan rekomendasi yang dapat ditindaklanjuti untuk membantu Anda membuat kebijakan yang aman dan fungsional. Untuk informasi selengkapnya, lihat Validasi kebijakan dengan IAM Access Analyzer dalam Panduan Pengguna IAM.
- Memerlukan otentikasi multi-faktor (MFA) - Jika Anda memiliki skenario yang mengharuskan pengguna IAM atau pengguna root di Anda, Akun AWS aktifkan MFA untuk keamanan tambahan. Untuk meminta MFA ketika operasi API dipanggil, tambahkan kondisi MFA pada kebijakan Anda. Untuk informasi selengkapnya, lihat Amankan akses API dengan MFA dalam Panduan Pengguna IAM.

Untuk informasi selengkapnya tentang praktik terbaik dalam IAM, lihat Praktik terbaik keamanan di IAM dalam Panduan Pengguna IAM.

Menggunakan konsol DynamoDB

Untuk mengakses konsol Amazon DynamoDB, Anda harus memiliki serangkaian izin minimum. Izin ini harus memungkinkan Anda untuk membuat daftar dan melihat detail tentang sumber daya DynamoDB di sumber daya Anda. Akun AWS Jika Anda membuat kebijakan berbasis identitas yang lebih ketat daripada izin minimum yang diperlukan, konsol tidak akan berfungsi sebagaimana mestinya untuk entitas (pengguna atau peran) dengan kebijakan tersebut.

Anda tidak perlu mengizinkan izin konsol minimum untuk pengguna yang melakukan panggilan hanya ke AWS CLI atau AWS API. Sebaliknya, izinkan akses hanya ke tindakan yang cocok dengan operasi API yang coba dilakukan.

Untuk memastikan bahwa pengguna dan peran masih dapat menggunakan konsol DynamoDB, lampirkan juga DynamoDB atau kebijakan terkelola ke `ConsoleAccess` entitas `ReadOnly` AWS . Untuk informasi selengkapnya, lihat Menambahkan izin ke pengguna dalam Panduan Pengguna IAM.

Mengizinkan pengguna melihat izin mereka sendiri

Contoh ini menunjukkan cara membuat kebijakan yang mengizinkan pengguna IAM melihat kebijakan inline dan terkelola yang dilampirkan ke identitas pengguna mereka. Kebijakan ini mencakup izin untuk menyelesaikan tindakan ini di konsol atau menggunakan API atau secara terprogram. 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": "*"
        }
    ]
}
```

Menggunakan kebijakan berbasis identitas dengan Amazon DynamoDB

Topik ini mencakup penggunaan kebijakan berbasis identitas AWS Identity and Access Management (IAM) dengan Amazon DynamoDB dan memberikan contoh. Contoh ini menunjukkan bagaimana administrator akun dapat melampirkan kebijakan izin ke identitas IAM (pengguna, grup, dan peran) dan dengan demikian memberikan izin untuk melakukan operasi pada sumber daya Amazon DynamoDB.

Bagian dalam topik ini mencakup hal berikut:

- Izin IAM diperlukan untuk menggunakan konsol Amazon DynamoDB
- AWS kebijakan IAM terkelola (telah ditentukan) untuk Amazon DynamoDB
- Contoh kebijakan yang dikelola pelanggan

Berikut adalah contoh kebijakan izin.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "DescribeQueryScanBooksTable",
            "Effect": "Allow",
            "Action": [
                "dynamodb:DescribeTable",
                "dynamodb:Query",
                "dynamodb:Scan"
```

```
            ],
            "Resource": "arn:aws:dynamodb:us-west-2:account-id:table/Books"
        }
    ]
}
```

Kebijakan sebelumnya memiliki satu pernyataan yang memberikan izin untuk tiga tindakan DynamoDB (`dynamodb:DescribeTable`,, dan`dynamodb:Scan`) pada tabel di `us-west-2` AWS Wilayah`dynamodb:Query`, yang dimiliki oleh akun yang ditentukan oleh. AWS *`account-id`* Amazon Resource Name (ARN) dalam nilai `Resource`menentukan tabel tempat izin berlaku.

Izin IAM diperlukan untuk menggunakan konsol Amazon DynamoDB

Untuk bekerja dengan konsol DynamoDB, pengguna harus memiliki set izin minimum yang memungkinkan pengguna untuk bekerja dengan sumber daya DynamoDB AWS akun mereka. Selain izin DynamoDB ini, konsol memerlukan izin:

- CloudWatch Izin Amazon untuk menampilkan metrik dan grafik.
- AWS Data Pipeline izin untuk mengekspor dan mengimpor data DynamoDB.
- AWS Identity and Access Management izin untuk mengakses peran yang diperlukan untuk ekspor dan impor.
- Izin Layanan Pemberitahuan Sederhana Amazon untuk memberi tahu Anda setiap kali CloudWatch alarm dipicu.
- AWS Lambda izin untuk memproses catatan DynamoDB Streams.

Jika Anda membuat kebijakan IAM yang lebih ketat daripada izin minimum yang diperlukan, konsol tidak akan berfungsi sebagaimana mestinya bagi pengguna dengan kebijakan IAM tersebut. Untuk memastikan bahwa pengguna tersebut masih dapat menggunakan konsol DynamoDB, lampirkan juga kebijakan terkelola `AmazonDynamoDBReadOnlyAccess` AWS ke pengguna, seperti yang dijelaskan dalam. AWS kebijakan IAM terkelola (telah ditentukan) untuk Amazon DynamoDB

Anda tidak perlu mengizinkan izin konsol minimum untuk pengguna yang melakukan panggilan hanya ke AWS CLI atau Amazon DynamoDB API.

> **Note**
>
> Jika Anda merujuk ke titik akhir VPC, Anda juga perlu mengotorisasi panggilan DescribeEndpoints API untuk prinsipal IAM yang meminta dengan tindakan IAM (dynamodb:).

DescribeEndpoints Untuk informasi selengkapnya, lihat Kebijakan yang diperlukan untuk titik akhir.

AWS kebijakan IAM terkelola (telah ditentukan) untuk Amazon DynamoDB

AWS mengatasi beberapa kasus penggunaan umum dengan menyediakan kebijakan IAM mandiri yang dibuat dan dikelola oleh. AWS Kebijakan AWS terkelola ini memberikan izin yang diperlukan untuk kasus penggunaan umum sehingga Anda dapat menghindari keharusan menyelidiki izin mana yang diperlukan. Untuk informasi selengkapnya, lihat Kebijakan Terkelola AWS dalam Panduan Pengguna IAM.

Kebijakan AWS terkelola berikut, yang dapat Anda lampirkan ke pengguna di akun Anda, khusus untuk DynamoDB dan dikelompokkan berdasarkan skenario kasus penggunaan:

- AmazonDynamoDBReadOnlyAccess— Memberikan akses hanya-baca ke sumber daya DynamoDB melalui file. AWS Management Console
- AmazonDynamoDBFullAkses - Memberikan akses penuh ke sumber daya DynamoDB melalui file. AWS Management Console

Anda dapat meninjau kebijakan izin AWS terkelola ini dengan masuk ke konsol IAM dan mencari kebijakan tertentu di sana.

Important

Praktik terbaiknya adalah membuat kebijakan IAM khusus yang memberikan hak akses paling rendah kepada pengguna, peran, atau grup yang memerlukannya.

Contoh kebijakan yang dikelola pelanggan

Di bagian ini, Anda dapat menemukan contoh kebijakan yang memberikan izin untuk berbagai tindakan DynamoDB. Kebijakan ini berfungsi saat Anda menggunakan AWS SDKs atau AWS CLI. Saat menggunakan konsol, Anda perlu memberikan izin tambahan yang khusus untuk konsol tersebut. Untuk informasi selengkapnya, lihat Izin IAM diperlukan untuk menggunakan konsol Amazon DynamoDB.

> **Note**
>
> Semua contoh kebijakan berikut menggunakan salah satu AWS Wilayah dan berisi nama akun IDs dan tabel fiktif.

Contoh:

- Kebijakan IAM untuk memberikan izin untuk semua tindakan DynamoDB pada tabel
- Kebijakan IAM untuk memberikan izin hanya-baca pada item di tabel DynamoDB
- Kebijakan IAM untuk memberikan akses ke tabel DynamoDB tertentu dan indeksnya
- Kebijakan IAM untuk membaca, menulis, memperbarui, dan menghapus akses pada tabel DynamoDB
- Kebijakan IAM untuk memisahkan lingkungan DynamoDB di akun yang sama AWS
- Kebijakan IAM untuk mencegah pembelian kapasitas terpesan DynamoDB
- Kebijakan IAM untuk memberikan akses baca hanya untuk DynamoDB stream (bukan untuk tabel)
- Kebijakan IAM untuk mengizinkan AWS Lambda fungsi mengakses catatan aliran DynamoDB
- Kebijakan IAM untuk akses baca dan tulis ke klaster DynamoDB Accelerator (DAX)

Panduan Pengguna IAM, mencakup tiga contoh DynamoDB tambahan:

- Amazon DynamoDB: Mengizinkan Akses ke Tabel Tertentu
- Amazon DynamoDB: Mengizinkan Akses ke Kolom Tertentu
- Amazon DynamoDB: Mengizinkan Akses Tingkat Baris ke DynamoDB Berdasarkan ID Amazon Cognito

Kebijakan IAM untuk memberikan izin untuk semua tindakan DynamoDB pada tabel

Kebijakan berikut memberikan izin untuk semua tindakan DynamoDB pada tabel yang disebut `Books`. Sumber daya ARN yang ditentukan dalam `Resource` mengidentifikasi tabel di Wilayah tertentu. AWS Jika Anda mengganti nama tabel `Books` dalam ARN `Resource` dengan karakter wildcard (*), semua tindakan DynamoDB diizinkan pada semua tabel di akun. Pertimbangkan dengan cermat kemungkinan implikasi keamanan sebelum menggunakan karakter wildcard pada kebijakan ini atau kebijakan IAM apa pun.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "AllAPIActionsOnBooks",
            "Effect": "Allow",
            "Action": "dynamodb:*",
            "Resource": "arn:aws:dynamodb:us-west-2:123456789012:table/Books"
        }
    ]
}
```

> **Note**
>
> Ini adalah contoh penggunaan karakter wildcard (*) untuk mengizinkan semua tindakan, termasuk administrasi, operasi data, pemantauan, dan pembelian kapasitas terpesan DynamoDB. Sebaliknya, praktik terbaiknya adalah menentukan secara eksplisit setiap tindakan yang akan diberikan dan hanya tindakan yang dibutuhkan pengguna, peran, atau grup tersebut.

Kebijakan IAM untuk memberikan izin hanya-baca pada item di tabel DynamoDB

Kebijakan izin berikut hanya memberikan izin untuk tindakan `GetItem`, `BatchGetItem`, `Scan`, `Query`, dan `ConditionCheckItem` DynamoDB, dan sebagai hasilnya, menetapkan akses hanya-baca pada tabel `Books`.

```
{
    "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"
```

```
        }
    ]
}
```

Kebijakan IAM untuk memberikan akses ke tabel DynamoDB tertentu dan indeksnya

Kebijakan berikut ini memberikan izin untuk tindakan modifikasi data pada tabel DynamoDB yang disebut `Books` dan semua indeks tabel tersebut. Untuk informasi selengkapnya tentang cara kerja indeks, lihat Meningkatkan akses data dengan indeks sekunder di DynamoDB.

```
{
    "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/*"
            ]
        }
    ]
}
```

Kebijakan IAM untuk membaca, menulis, memperbarui, dan menghapus akses pada tabel DynamoDB

Gunakan kebijakan ini jika Anda perlu mengizinkan aplikasi Anda membuat, membaca, memperbarui, dan menghapus data di tabel, indeks, dan aliran Amazon DynamoDB. Gantikan nama AWS Region, ID akun Anda, dan nama tabel atau karakter wildcard (*) jika sesuai.

```
{
    "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/*"
```

```
            ]
        }
    ]
}
```

Untuk memperluas kebijakan ini untuk mencakup semua tabel DynamoDB di AWS semua Wilayah untuk akun ini, gunakan wildcard (*) untuk Region dan nama tabel. Sebagai contoh:

```
"Resource":[
                "arn:aws:dynamodb:*:123456789012:table/*",
                "arn:aws:dynamodb:*:123456789012:table/*/index/*"
                ]
```

Kebijakan IAM untuk memisahkan lingkungan DynamoDB di akun yang sama AWS

Misalkan Anda memiliki lingkungan terpisah di mana setiap lingkungan memelihara versi tabelnya sendiri yang bernama `ProductCatalog`. Jika Anda membuat dua `ProductCatalog` tabel di AWS akun yang sama, bekerja di satu lingkungan dapat memengaruhi lingkungan lain karena cara pengaturan izin. Misalnya, kuota pada jumlah operasi bidang kontrol bersamaan (seperti`CreateTable`) ditetapkan pada tingkat AWS akun.

Sebagai hasilnya, setiap tindakan di satu lingkungan mengurangi jumlah operasi yang tersedia di lingkungan lain. Ada juga risiko bahwa kode di satu lingkungan mungkin secara tidak sengaja mengakses tabel di lingkungan lain.

> **Note**
>
> Jika Anda ingin memisahkan beban kerja produksi dan pengujian untuk membantu mengontrol potensi "radius ledakan" suatu peristiwa, praktik terbaiknya adalah membuat akun AWS terpisah untuk beban kerja pengujian dan produksi. Untuk informasi selengkapnya, lihat Manajemen dan Pemisahan Akun AWS.

Misalkan Anda memiliki dua pengembang, Amit dan Alice, yang sedang menguji tabel `ProductCatalog`. Alih-alih setiap pengembang memerlukan AWS akun terpisah, pengembang Anda dapat berbagi AWS akun pengujian yang sama. Di akun pengujian ini, Anda dapat membuat salinan tabel yang sama untuk dikerjakan oleh setiap pengembang, seperti `Alice_ProductCatalog` dan `Amit_ProductCatalog`. Dalam hal ini, Anda dapat membuat pengguna Alice dan Amit di AWS akun yang Anda buat untuk lingkungan pengujian. Anda kemudian

dapat memberikan izin kepada pengguna ini untuk melakukan tindakan DynamoDB pada tabel yang mereka miliki.

Untuk memberikan izin pengguna IAM ini, Anda dapat melakukan salah satu hal berikut:

- Buat kebijakan terpisah untuk setiap pengguna, lalu lampirkan setiap kebijakan ke penggunanya secara terpisah. Misalnya, Anda dapat melampirkan kebijakan berikut kepada pengguna Alice untuk mengizinkannya mengakses tindakan DynamoDB pada tabel `Alice_ProductCatalog`:

```
{
    "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/*"
        }
    ]
}
```

Kemudian, Anda dapat membuat kebijakan serupa dengan sumber daya berbeda (tabel `Amit_ProductCatalog`) untuk pengguna Amit.

- Daripada melampirkan kebijakan ke pengguna individual, Anda dapat menggunakan variabel kebijakan IAM untuk menulis kebijakan tunggal dan melampirkannya ke grup. Anda perlu membuat grup dan, untuk contoh ini, menambahkan pengguna Alice dan Amit ke grup. Contoh berikut memberikan izin untuk melakukan semua tindakan DynamoDB pada tabel `${aws:username}_ProductCatalog`. Variabel kebijakan `${aws:username}` diganti dengan nama pengguna pemohon ketika kebijakan dievaluasi. Misalnya, jika Alice mengirimkan permintaan untuk menambahkan item, tindakan tersebut hanya diperbolehkan jika Alice menambahkan item ke tabel `Alice_ProductCatalog`.

```
{
    "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**
>
> Saat menggunakan variabel kebijakan IAM, Anda harus secara eksplisit menentukan bahasa kebijakan IAM versi `2012-10-17` dalam kebijakan. Versi default bahasa kebijakan IAM (`2008-10-17`) tidak mendukung variabel kebijakan.

Alih-alih mengidentifikasi tabel tertentu sebagai sumber daya seperti yang biasa Anda lakukan, Anda bisa menggunakan karakter wildcard (*) untuk memberikan izin pada semua tabel yang nama tabelnya diawali dengan pengguna yang membuat permintaan, seperti yang ditunjukkan dalam contoh berikut.

```
"Resource":"arn:aws:dynamodb:us-west-2:123456789012:table/${aws:username}_*"
```

Kebijakan IAM untuk mencegah pembelian kapasitas terpesan DynamoDB

Dengan kapasitas terpesan Amazon DynamoDB, Anda membayar biaya satu kali di muka dan berkomitmen untuk membayar tingkat penggunaan minimum dengan penghematan yang signifikan selama jangka waktu tertentu. Anda dapat menggunakan AWS Management Console untuk melihat dan membeli kapasitas cadangan. Namun, Anda mungkin tidak ingin semua pengguna di organisasi Anda dapat membeli kapasitas terpesan. Untuk informasi selengkapnya tentang kapasitas terpesan, lihat harga Amazon DynamoDB.

DynamoDB menyediakan operasi API berikut untuk mengendalikan akses ke manajemen kapasitas terpesan:

- `dynamodb:DescribeReservedCapacity` – Mengembalikan pembelian kapasitas terpesan yang saat ini berlaku.
- `dynamodb:DescribeReservedCapacityOfferings` – Mengembalikan detail tentang paket kapasitas terpesan yang saat ini ditawarkan oleh AWS.
- `dynamodb:PurchaseReservedCapacityOfferings` – Melakukan pembelian aktual kapasitas terpesan.

AWS Management Console Menggunakan tindakan API ini untuk menampilkan informasi kapasitas cadangan dan melakukan pembelian. Anda tidak dapat memanggil operasi ini dari program aplikasi karena hanya dapat diakses dari konsol. Namun, Anda dapat mengizinkan atau menolak akses ke operasi ini dalam kebijakan izin IAM.

Kebijakan berikut memungkinkan pengguna untuk melihat pembelian dan penawaran kapasitas cadangan dengan menggunakan AWS Management Console — tetapi pembelian baru ditolak.

```
{
    "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:*"
        }
    ]
}
```

Perhatikan bahwa kebijakan ini menggunakan karakter wildcard (*) untuk mengizinkan penjelasan izin untuk semua, dan untuk menolak pembelian kapasitas tersimpan DynamoDB untuk semua.

Kebijakan IAM untuk memberikan akses baca hanya untuk DynamoDB stream (bukan untuk tabel)

Saat Anda mengaktifkan DynamoDB Streams pada tabel, informasi diambil tentang setiap modifikasi pada item dalam tabel. Untuk informasi selengkapnya, lihat Tangkapan data perubahan DynamoDB Streams.

Dalam beberapa kasus, Anda mungkin ingin mencegah aplikasi membaca data dari tabel DynamoDB, tetapi tetap mengizinkan akses ke aliran tabel tersebut. Misalnya, Anda dapat mengonfigurasi AWS Lambda untuk melakukan polling aliran dan menjalankan fungsi Lambda saat pembaruan item terdeteksi, lalu melakukan pemrosesan tambahan.

Tindakan berikut tersedia untuk mengontrol akses ke DynamoDB stream:

- `dynamodb:DescribeStream`
- `dynamodb:GetRecords`
- `dynamodb:GetShardIterator`
- `dynamodb:ListStreams`

Contoh kebijakan berikut memberikan izin pengguna untuk mengakses aliran tabel bernama `GameScores`. Karakter wildcard (*) di ARN cocok dengan aliran yang terkait dengan tabel tersebut.

```
{
    "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/*"
        }
    ]
```

```
}
```

Perhatikan bahwa kebijakan ini memberikan akses ke aliran tabel `GameScores`, tetapi tidak ke tabel itu sendiri.

Kebijakan IAM untuk mengizinkan AWS Lambda fungsi mengakses catatan aliran DynamoDB

Jika Anda ingin tindakan tertentu dilakukan berdasarkan peristiwa dalam aliran DynamoDB, Anda dapat menulis fungsi AWS Lambda yang dipicu oleh peristiwa ini. Suatu fungsi Lambda seperti ini memerlukan izin untuk membaca data dari aliran DynamoDB. Untuk informasi selengkapnya tentang menggunakan Lambda dengan DynamoDB Streams, lihat DynamoDB Streams dan pemicu AWS Lambda.

Untuk memberikan izin kepada Lambda, gunakan kebijakan izin yang terkait dengan peran IAM fungsi Lambda (juga dikenal sebagai peran eksekusi). Tentukan kebijakan ini saat Anda membuat fungsi Lambda.

Misalnya, Anda dapat mengaitkan kebijakan izin berikut dengan peran eksekusi untuk memberikan izin Lambda untuk melakukan tindakan DynamoDB Streams terdaftar.

```
{
    "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/*"
        }
    ]
}
```

Untuk informasi selengkapnya, lihat izin AWS Lambda dalam Panduan Pengembang AWS Lambda .

Kebijakan IAM untuk akses baca dan tulis ke klaster DynamoDB Accelerator (DAX)

Kebijakan berikut mengizinkan akses baca, tulis, pembaruan, dan hapus ke klaster DynamoDB Accelerator (DAX), tetapi tidak untuk tabel DynamoDB terkait. Untuk menggunakan kebijakan ini, ganti nama AWS Wilayah, ID akun, dan nama klaster DAX Anda.

> **Note**
>
> Kebijakan ini memberikan akses ke klaster DAX, tetapi tidak ke tabel DynamoDB yang terkait. Pastikan bahwa klaster DAX Anda memiliki kebijakan yang benar untuk melakukan operasi yang sama ini pada tabel DynamoDB atas nama Anda.

```
{
    "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"
        }
    ]
}
```

Untuk memperluas kebijakan ini guna mencakup akses DAX untuk semua AWS Wilayah untuk akun, gunakan karakter wildcard (*) untuk nama Wilayah.

```
"Resource": "arn:aws:dax:*:123456789012:cache/MyDAXCluster"
```

Pemecahan masalah identitas dan akses Amazon DynamoDB

Gunakan informasi berikut untuk membantu Anda mendiagnosis dan memperbaiki masalah umum yang mungkin Anda hadapi ketika bekerja dengan DynamoDB dan IAM.

Topik

- Saya tidak berwenang melakukan tindakan di DynamoDB
- Saya tidak berwenang untuk melakukan iam: PassRole
- Saya ingin mengizinkan orang di luar saya Akun AWS untuk mengakses sumber daya DynamoDB saya

Saya tidak berwenang melakukan tindakan di DynamoDB

Jika AWS Management Console memberitahu Anda bahwa Anda tidak berwenang untuk melakukan tindakan, maka Anda harus menghubungi administrator Anda untuk bantuan. Administrator Anda adalah orang yang memberikan nama pengguna dan kata sandi Anda.

Contoh kesalahan berikut terjadi ketika pengguna `mateojackson` mencoba menggunakan konsol untuk melihat detail tentang suatu sumber daya *`my-example-widget`* fiktif, tetapi tidak memiliki izin `aws:`*`GetWidget`* fiktif.

```
User: arn:aws:iam::123456789012:user/mateojackson is not authorized to perform:
 aws:GetWidget on resource: my-example-widget
```

Dalam hal ini, Mateo meminta administratornya untuk memperbarui kebijakannya untuk mengizinkan dia mengakses sumber daya *`my-example-widget`* menggunakan tindakan `aws:`*`GetWidget`*.

Saya tidak berwenang untuk melakukan iam: PassRole

Jika Anda menerima kesalahan yang menyatakan bahwa Anda tidak berwenang untuk melakukan tindakan `iam:PassRole`, kebijakan Anda harus diperbarui agar Anda dapat meneruskan peran ke DynamoDB.

Beberapa Layanan AWS memungkinkan Anda untuk meneruskan peran yang ada ke layanan tersebut alih-alih membuat peran layanan baru atau peran terkait layanan. Untuk melakukannya, Anda harus memiliki izin untuk meneruskan peran ke layanan.

Contoh galat berikut terjadi ketika pengguna IAM bernama `marymajor` mencoba menggunakan konsol tersebut untuk melakukan tindakan di DynamoDB. Namun, tindakan tersebut memerlukan layanan untuk mendapatkan izin yang diberikan oleh peran layanan. Mary tidak memiliki izin untuk meneruskan peran tersebut pada layanan.

```
User: arn:aws:iam::123456789012:user/marymajor is not authorized to perform:
 iam:PassRole
```

Dalam kasus ini, kebijakan Mary harus diperbarui agar dia mendapatkan izin untuk melakukan tindakan `iam:PassRole` tersebut.

Jika Anda memerlukan bantuan, hubungi AWS administrator Anda. Administrator Anda adalah orang yang memberi Anda kredensial masuk.

Saya ingin mengizinkan orang di luar saya Akun AWS untuk mengakses sumber daya DynamoDB saya

Anda dapat membuat peran yang dapat digunakan pengguna di akun lain atau orang-orang di luar organisasi Anda untuk mengakses sumber daya Anda. Anda dapat menentukan siapa saja yang dipercaya untuk mengambil peran tersebut. Untuk layanan yang mendukung kebijakan berbasis sumber daya atau daftar kontrol akses (ACLs), Anda dapat menggunakan kebijakan tersebut untuk memberi orang akses ke sumber daya Anda.

Untuk mempelajari selengkapnya, periksa referensi berikut:

- Untuk mempelajari apakah DynamoDB mendukung fitur-fitur ini, lihat Cara kerja Amazon DynamoDB dengan IAM.
- Untuk mempelajari cara menyediakan akses ke sumber daya Anda di seluruh sumber daya Akun AWS yang Anda miliki, lihat Menyediakan akses ke pengguna IAM di pengguna lain Akun AWS yang Anda miliki di Panduan Pengguna IAM.
- Untuk mempelajari cara menyediakan akses ke sumber daya Anda kepada pihak ketiga Akun AWS, lihat Menyediakan akses yang Akun AWS dimiliki oleh pihak ketiga dalam Panduan Pengguna IAM.
- Untuk mempelajari cara memberikan akses melalui federasi identitas, lihat Menyediakan akses ke pengguna terautentikasi eksternal (federasi identitas) dalam Panduan Pengguna IAM.
- Untuk mempelajari perbedaan antara menggunakan peran dan kebijakan berbasis sumber daya untuk akses lintas akun, lihat Akses sumber daya lintas akun di IAM di Panduan Pengguna IAM.

Kebijakan IAM untuk mencegah pembelian kapasitas terpesan DynamoDB

Dengan kapasitas terpesan Amazon DynamoDB, Anda membayar biaya satu kali di muka dan berkomitmen untuk membayar tingkat penggunaan minimum dengan penghematan yang signifikan selama jangka waktu tertentu. Anda dapat menggunakan AWS Management Console untuk melihat dan membeli kapasitas cadangan. Namun, Anda mungkin tidak ingin semua pengguna di organisasi Anda dapat membeli kapasitas terpesan. Untuk informasi selengkapnya tentang kapasitas terpesan, lihat harga Amazon DynamoDB.

DynamoDB menyediakan operasi API berikut untuk mengendalikan akses ke manajemen kapasitas terpesan:

- `dynamodb:DescribeReservedCapacity` – Mengembalikan pembelian kapasitas terpesan yang saat ini berlaku.
- `dynamodb:DescribeReservedCapacityOfferings` – Mengembalikan detail tentang paket kapasitas terpesan yang saat ini ditawarkan oleh AWS.
- `dynamodb:PurchaseReservedCapacityOfferings` – Melakukan pembelian aktual kapasitas terpesan.

AWS Management Console Menggunakan tindakan API ini untuk menampilkan informasi kapasitas cadangan dan melakukan pembelian. Anda tidak dapat memanggil operasi ini dari program aplikasi karena hanya dapat diakses dari konsol. Namun, Anda dapat mengizinkan atau menolak akses ke operasi ini dalam kebijakan izin IAM.

Kebijakan berikut memungkinkan pengguna untuk melihat pembelian dan penawaran kapasitas cadangan dengan menggunakan AWS Management Console — tetapi pembelian baru ditolak.

```
{
    "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:*"
        }
    ]
}
```

Perhatikan bahwa kebijakan ini menggunakan karakter wildcard (*) untuk mengizinkan penjelasan izin untuk semua, dan untuk menolak pembelian kapasitas terpesan DynamoDB untuk semua.

Menggunakan ketentuan kebijakan IAM untuk kontrol akses terperinci

Ketika memberikan izin di DynamoDB, Anda dapat menetapkan syarat yang menentukan bagaimana kebijakan izin berlaku.

Gambaran Umum

Di DynamoDB, Anda memiliki pilihan untuk menentukan syarat saat memberikan izin menggunakan kebijakan IAM (lihat Manajemen Identitas dan Akses untuk Amazon DynamoDB). Sebagai contoh, Anda dapat:

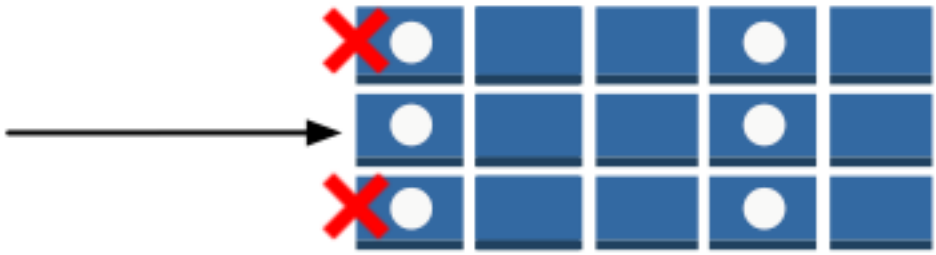
- Memberi izin kepada pengguna akan akses hanya-baca pada item dan atribut tertentu dalam tabel atau indeks sekunder.
- Memberi izin kepada pengguna akan akses hanya-tulis pada atribut tertentu dalam tabel, berdasarkan identitas pengguna tersebut.

Di DynamoDB, Anda dapat menentukan ketentuan dalam kebijakan IAM menggunakan kunci ketentuan, seperti yang diilustrasikan dalam kasus penggunaan di bagian berikut.

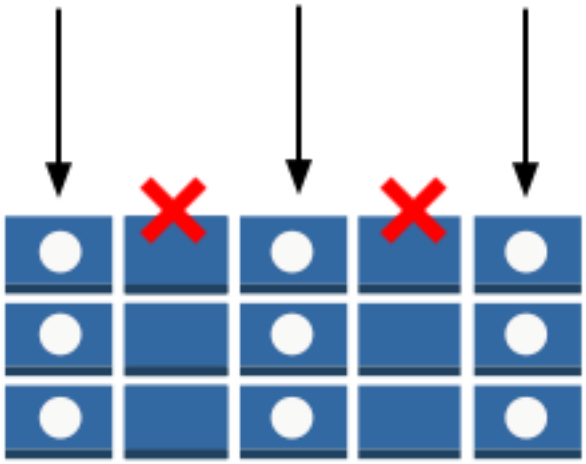
Kasus penggunaan izin

Selain mengontrol akses ke tindakan DynamoDB API, Anda juga dapat mengontrol akses ke item data dan atribut individual. Misalnya, Anda dapat melakukan hal berikut:

- Memberikan izin pada tabel, namun batasi akses ke item tertentu dalam tabel tersebut berdasarkan nilai kunci primer tertentu. Misalnya aplikasi jejaring sosial untuk permainan, yang mana semua data permainan yang disimpan pengguna disimpan dalam satu tabel, tetapi tidak ada pengguna yang dapat mengakses item data yang bukan miliknya, seperti yang diperlihatkan dalam ilustrasi berikut:



- Menyembunyikan informasi sehingga hanya subset dari atribut yang terlihat oleh pengguna. Misalnya aplikasi yang menampilkan data penerbangan untuk bandara terdekat, berdasarkan lokasi pengguna. Nama maskapai penerbangan, waktu kedatangan dan keberangkatan, serta nomor penerbangan semuanya ditampilkan. Namun atribut seperti nama pilot atau jumlah penumpang disembunyikan, seperti terlihat pada ilustrasi berikut:



Untuk menerapkan kontrol akses terperinci semacam ini, Anda menulis kebijakan izin IAM yang menentukan kondisi untuk mengakses kredensial keamanan dan izin terkait. Anda kemudian menerapkan kebijakan tersebut kepada pengguna, grup, atau peran yang Anda buat menggunakan konsol IAM. Kebijakan IAM Anda dapat membatasi akses ke masing-masing item dalam tabel, akses ke atribut dalam item tersebut, atau keduanya secara bersamaan.

Anda juga dapat menggunakan federasi identitas web untuk mengontrol akses oleh pengguna yang diautentikasi dengan Login dengan Amazon, Facebook, atau Google. Untuk informasi selengkapnya, lihat Menggunakan federasi identitas web.

Anda menggunakan elemen `Condition` IAM untuk menerapkan kebijakan kontrol akses yang terperinci. Dengan menambahkan elemen `Condition` ke kebijakan izin, Anda dapat mengizinkan atau menolak akses ke item dan atribut dalam tabel dan indeks DynamoDB, berdasarkan kebutuhan bisnis khusus Anda.

Sebagai contoh, pertimbangkan aplikasi game seluler yang memungkinkan pemain memilih dan memainkan berbagai permainan berbeda. Aplikasi ini menggunakan tabel DynamoDB bernama `GameScores` untuk melacak skor tertinggi dan data pengguna lainnya. Setiap item dalam tabel diidentifikasi secara unik berdasarkan ID pengguna dan nama permainan yang dimainkan pengguna. Tabel `GameScores` memiliki kunci primer yang terdiri dari kunci partisi (`UserId`) dan kunci urutan (`GameTitle`). Pengguna hanya memiliki akses ke data game yang terkait dengan ID pengguna mereka. Pengguna yang ingin memainkan game harus memiliki peran IAM bernama `GameRole`, yang memiliki kebijakan keamanan yang melekat padanya.

Untuk mengelola izin pengguna di aplikasi ini, Anda dapat menulis kebijakan izin seperti berikut:

```
{
   "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"
         }
      }
   }
]
}
```

Selain memberikan izin untuk tindakan DynamoDB tertentu (elemen `Action`) pada tabel `GameScores` (elemen `Resource`), elemen `Condition` menggunakan kunci kondisi khusus untuk DynamoDB berikut yang membatasi izin sebagai berikut:

- `dynamodb:LeadingKeys` – Kunci ketentuan ini memungkinkan pengguna untuk mengakses hanya item yang nilai kunci partisinya cocok dengan ID pengguna mereka. ID ini, `${www.amazon.com:user_id}`, adalah variabel substitusi. Untuk informasi selengkapnya tentang variabel substitusi, lihat Menggunakan federasi identitas web.
- `dynamodb:Attributes` – Kunci ketentuan ini membatasi akses ke atribut tertentu sehingga hanya tindakan yang tercantum dalam kebijakan izin yang dapat mengembalikan nilai untuk atribut ini. Selain itu, klausul `StringEqualsIfExists` memastikan bahwa aplikasi harus selalu menyediakan daftar atribut tertentu untuk ditindaklanjuti dan aplikasi tidak dapat meminta semua atribut.

Ketika kebijakan IAM dievaluasi, hasilnya selalu true (akses diizinkan) atau false (akses ditolak). Jika ada bagian dari elemen `Condition` yang false, seluruh kebijakan mengevaluasi ke false dan akses ditolak.

> **Important**
>
> Jika Anda menggunakan `dynamodb:Attributes`, Anda harus menentukan nama semua atribut kunci primer dan kunci indeks untuk tabel dan indeks sekunder apa pun yang tercantum dalam kebijakan. Jika tidak, DynamoDB tidak dapat menggunakan atribut kunci ini untuk melakukan tindakan yang diminta.

Dokumen kebijakan IAM hanya dapat berisi karakter Unicode berikut: tab horizontal (U+0009), umpan baris (U+000A), pengembalian pengangkutan (U+000D), dan karakter dalam rentang U+0020 hingga U+00FF.

Menentukan ketentuan: Menggunakan kunci syarat

AWS menyediakan satu set kunci kondisi yang telah ditentukan (AWS-wide condition keys) untuk semua AWS layanan yang mendukung IAM untuk kontrol akses. Misalnya, Anda dapat menggunakan kunci kondisi `aws:SourceIp` untuk memeriksa alamat IP pemohon sebelum mengizinkan tindakan yang akan dilakukan. Untuk informasi selengkapnya dan daftar tombol AWS-wide, lihat Kunci Tersedia untuk Ketentuan di Panduan Pengguna IAM.

Tabel berikut menunjukkan kunci kondisi layanan khusus DynamoDB yang berlaku untuk DynamoDB.

Kunci Kondisi DynamoDB	Deskripsi
`dynamodb:LeadingKeys`	Mewakili atribut kunci pertama dari tabel—dengan kata lain, kunci partisi. Nama kunci `LeadingKeys` bersifat jamak, meskipun kuncinya digunakan dengan tindakan satu item. Selain itu, Anda harus menggunakan pengubah `ForAllValues` ketika menggunakan `LeadingKeys` dalam suatu kondisi.
`dynamodb:Select`	Mewakili parameter `Select` dari permintaan `Query` atau `Scan`. `Select` dapat berupa nilai berikut: • `ALL_ATTRIBUTES` • `ALL_PROJECTED_ATTRIBUTES` • `SPECIFIC_ATTRIBUTES` • `COUNT`
`dynamodb:Attributes`	Mewakili daftar nama atribut dalam suatu permintaan, atau atribut yang dikembalikan dari permintaan. Nilai `Attributes` diberi nama dengan cara yang sama dan memiliki arti yang sama sebagai parameter untuk tindakan DynamoDB API tertentu, seperti yang ditunjukkan berikut: • `AttributesToGet`

Kunci Kondisi DynamoDB	Deskripsi
	Digunakan oleh: `BatchGetItem, GetItem, Query, Scan` • `AttributeUpdates` Digunakan oleh: `UpdateItem` • `Expected` Digunakan oleh: `DeleteItem, PutItem, UpdateItem` • `Item` Digunakan oleh: `PutItem` • `ScanFilter` Digunakan oleh: `Scan`
`dynamodb:ReturnValues`	Mewakili parameter `ReturnValues` dari suatu permintaan. `ReturnValues` dapat berupa nilai berikut: • `ALL_OLD` • `UPDATED_OLD` • `ALL_NEW` • `UPDATED_NEW` • `NONE`
`dynamodb:ReturnConsumedCapacity`	Mewakili parameter `ReturnConsumedCapacity` dari suatu permintaan. `ReturnConsumedCapacity` dapat berupa salah satu nilai berikut: • `TOTAL` • `NONE`

Membatasi akses pengguna

Banyak kebijakan izin IAM memungkinkan pengguna mengakses hanya item dalam tabel yang nilai kunci partisinya cocok dengan pengidentifikasi pengguna. Misalnya, aplikasi game sebelumnya

membatasi akses dengan cara ini sehingga pengguna hanya dapat mengakses data game yang dikaitkan dengan ID pengguna mereka. Variabel substitusi IAM `${www.amazon.com:user_id}`, `${graph.facebook.com:id}`, dan `${accounts.google.com:sub}` berisi pengenal pengguna untuk Login with Amazon, Facebook, dan Google. Untuk mempelajari cara aplikasi masuk ke salah satu penyedia identitas ini dan memperoleh pengidentifikasi ini, lihat Menggunakan federasi identitas web.

> **Important**
>
> Kontrol akses berbutir halus tidak didukung untuk membatasi replikasi tabel global. Menerapkan kondisi kebijakan untuk kontrol akses berbutir halus ke prinsipal layanan DynamoDB atau peran terkait layanan yang digunakan untuk replikasi tabel global dapat mengganggu replikasi dalam tabel global.

> **Note**
>
> Masing-masing contoh pada bagian berikut menetapkan klausul `Effect` untuk `Allow` dan menentukan hanya tindakan, sumber daya, dan parameter yang diizinkan. Akses hanya diizinkan untuk apa yang tercantum secara eksplisit dalam kebijakan IAM.
> Dalam beberapa kasus, dimungkinkan untuk menulis ulang kebijakan ini sehingga kebijakan berbasis penolakan (yaitu, pengaturan klausul `Effect` untuk `Deny` dan membalikkan semua logika dalam kebijakan). Namun, kami menyarankan agar Anda menghindari penggunaan kebijakan berbasis penolakan dengan DynamoDB karena kebijakan tersebut sulit ditulis dengan benar, dibandingkan dengan kebijakan berbasis izin. Selain itu, perubahan di masa mendatang pada DynamoDB API (atau perubahan input API yang ada) dapat membuat kebijakan berbasis penolakan tidak efektif.

Kebijakan Contoh: Menggunakan ketentuan untuk kontrol parameter terperinci

Bagian ini menunjukkan beberapa kebijakan untuk mengimplementasikan kontrol akses terperinci pada tabel dan indeks DynamoDB.

> **Note**
>
> Semua contoh menggunakan Wilayah us-west-2 dan berisi akun fiktif. IDs

Video di bawah ini menjelaskan kontrol akses berbutir halus di DynamoDB menggunakan kondisi kebijakan IAM.

1: Memberikan izin yang membatasi akses ke item dengan nilai kunci partisi tertentu

Kebijakan izin berikut memberikan izin yang memungkinkan seperangkat tindakan DynamoDB pada tabel `GamesScore`. Kebijakan tersebut menggunakan kunci ketentuan `dynamodb:LeadingKeys` untuk membatasi tindakan pengguna hanya pada item yang nilai kunci partisi `UserID`-nya cocok dengan ID pengguna unik Login with Amazon untuk aplikasi ini.

> **Important**
>
> Daftar tindakan tidak termasuk izin untuk `Scan` karena `Scan` mengembalikan semua item apa pun kunci awalnya.

```
{
   "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**
>
> Saat menggunakan variabel kebijakan, Anda harus secara jelas menyebutkan versi `2012-10-17` dalam kebijakan tersebut. Versi default bahasa kebijakan akses, `2008-10-17`, tidak mendukung variabel kebijakan.

Untuk menerapkan akses hanya baca, Anda dapat menghapus tindakan apa pun yang dapat mengubah data. Dalam kebijakan berikut, hanya tindakan yang menyediakan akses baca saja yang disertakan dalam ketentuan.

```
{
   "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**
>
> Jika Anda menggunakan `dynamodb:Attributes`, Anda harus menentukan nama semua atribut kunci primer dan kunci indeks, untuk tabel dan indeks sekunder apa pun yang tercantum dalam kebijakan. Jika tidak, DynamoDB tidak dapat menggunakan atribut kunci ini untuk melakukan tindakan yang diminta.

2: Memberikan izin yang membatasi akses ke atribut tertentu dalam tabel

Kebijakan izin berikut mengizinkan akses ke hanya dua atribut tertentu dalam tabel dengan menambahkan kunci ketentuan `dynamodb:Attributes`. Atribut ini dapat dibaca, ditulis, atau dievaluasi dalam penulisan bersyarat atau filter pemindaian.

```
{
   "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**
>
> Kebijakan ini mengambil pendekatan daftar izin, yang mengizinkan akses ke set atribut bernama. Anda dapat menulis kebijakan serupa yang menolak akses ke atribut lain. Kami tidak merekomendasikan pendekatan daftar penolakan ini. Pengguna dapat menentukan nama-nama atribut yang ditolak ini dengan mengikuti prinsip hak istimewa terkecil, seperti yang dijelaskan di Wikipedia di http://en.wikipedia. org/wiki/Principle_of_least_privilege, dan gunakan pendekatan daftar izinkan untuk menghitung semua nilai yang diizinkan, daripada menentukan atribut yang ditolak.

Kebijakan ini tidak mengizinkan `PutItem`, `DeleteItem`, atau `BatchWriteItem`. Tindakan ini selalu menggantikan seluruh item sebelumnya, yang memungkinkan pengguna menghapus nilai sebelumnya untuk atribut yang tidak boleh mereka akses.

Klausa `StringEqualsIfExists` dalam kebijakan izin memastikan hal berikut:

- Jika pengguna menentukan parameter `Select`, nilainya harus `SPECIFIC_ATTRIBUTES`. Persyaratan ini mencegah tindakan API mengembalikan atribut apa pun yang tidak diizinkan, seperti dari proyeksi indeks.
- Jika pengguna menentukan parameter `ReturnValues`, nilainya harus `NONE`, `UPDATED_OLD`, atau `UPDATED_NEW`. Hal ini diperlukan karena tindakan `UpdateItem` tersebut juga melakukan operasi baca implisit untuk memeriksa apakah suatu item ada sebelum menggantinya, dan agar nilai atribut sebelumnya dapat dikembalikan jika diminta. Membatasi `ReturnValues` dengan cara ini memastikan bahwa pengguna hanya dapat membaca atau menulis atribut yang diizinkan.
- Klausa `StringEqualsIfExists` memastikan bahwa hanya satu dari parameter ini — `Select` atau `ReturnValues` — yang dapat digunakan per permintaan, dalam konteks tindakan yang diizinkan.

Berikut ini adalah beberapa variasi pada kebijakan ini:

- Untuk hanya mengizinkan tindakan baca, Anda dapat menghapus `UpdateItem` dari daftar tindakan yang diizinkan. Karena tidak ada tindakan yang tersisa menerima `ReturnValues`, Anda dapat menghapus `ReturnValues` dari kondisi. Anda juga dapat mengubah `StringEqualsIfExists` menjadi `StringEquals` karena parameter `Select` selalu memiliki nilai (`ALL_ATTRIBUTES`, kecuali ditentukan lain).
- Untuk hanya mengizinkan tindakan tulis, Anda dapat menghapus semuanya kecuali `UpdateItem` dari daftar tindakan yang diizinkan. Karena `UpdateItem` tidak menggunakan parameter `Select`, Anda dapat menghapus `Select` dari kondisi. Anda juga harus mengubah `StringEqualsIfExists` menjadi `StringEquals` karena parameter `ReturnValues` selalu memiliki nilai (`NONE`, kecuali ditentukan lain).
- Untuk mengizinkan semua atribut yang namanya cocok dengan pola, gunakan `StringLike`, bukan `StringEquals`, dan gunakan pola multi-karakter yang cocok dengan karakter wildcard (*).

3: Memberikan izin untuk mencegah pembaruan pada atribut tertentu

Kebijakan izin berikut membatasi akses pengguna untuk memperbarui hanya atribut tertentu yang diidentifikasi oleh kunci kondisi `dynamodb:Attributes`. Kondisi `StringNotLike` mencegah aplikasi memperbarui atribut yang ditentukan menggunakan kunci kondisi `dynamodb:Attributes`.

```
{
   "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"
               ]
            },
            "StringEquals":{
               "dynamodb:ReturnValues":[
                  "NONE",
                  "UPDATED_OLD",
```

```
                    "UPDATED_NEW"
                ]
            }
        }
    }
  ]
}
```

Perhatikan hal-hal berikut:

- Tindakan `UpdateItem`, seperti tindakan tulis lainnya, memerlukan akses baca ke item sehingga dapat mengembalikan nilai sebelum dan sesudah pembaruan. Dalam kebijakan tersebut, Anda membatasi tindakan untuk mengakses hanya atribut yang diizinkan yang akan diperbarui dengan menentukan kunci ketentuan `dynamodb:ReturnValues`. Kunci kondisi membatasi `ReturnValues` dalam permintaan untuk menentukan hanya `NONE`, `UPDATED_OLD`, atau `UPDATED_NEW` dan tidak mencakup `ALL_OLD` atau `ALL_NEW`.
- Tindakan `PutItem` dan `DeleteItem` mengganti seluruh item, dengan demikian memungkinkan aplikasi memodifikasi atribut apa pun. Jadi ketika membatasi aplikasi untuk memperbarui hanya atribut tertentu, Anda tidak boleh memberikan izin untuk ini APIs.

4: Berikan izin untuk mengkueri hanya atribut yang diproyeksikan dalam indeks

Kebijakan izin berikut memungkinkan kueri pada indeks sekunder (`TopScoreDateTimeIndex`) dengan menggunakan kunci ketentuan `dynamodb:Attributes`. Kebijakan ini juga membatasi kueri untuk meminta atribut tertentu saja yang telah diproyeksikan ke dalam indeks.

Untuk mengharuskan aplikasi menentukan daftar atribut dalam kueri, kebijakan juga menentukan kunci ketentuan `dynamodb:Select` untuk mengharuskan parameter `Select` dari tindakan `Query` DynamoDB adalah `SPECIFIC_ATTRIBUTES`. Daftar atribut terbatas pada daftar tertentu yang disediakan menggunakan kunci ketentuan `dynamodb:Attributes`.

```
{
   "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"
            }
         }
      }
   ]
}
```

Kebijakan izin berikut ini serupa, tetapi kueri harus meminta semua atribut yang telah diproyeksikan ke dalam indeks.

```
{
   "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"
            }
         }
```

```
      }
   ]
}
```

5: Memberikan izin untuk membatasi akses ke atribut tertentu dan nilai-nilai kunci partisi

Kebijakan izin berikut memungkinkan tindakan DynamoDB tertentu (ditentukan dalam elemen `Action`) pada tabel dan indeks tabel (ditentukan dalam elemen `Resource`). Kebijakan menggunakan kunci ketentuan `dynamodb:LeadingKeys` untuk membatasi izin hanya untuk item yang nilai kunci partisinya cocok dengan ID Facebook pengguna.

```
{
   "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"
                ]
            }
        }
    }
  ]
}
```

Perhatikan hal-hal berikut:

- Tulis tindakan yang diizinkan oleh kebijakan (`UpdateItem`) hanya dapat memodifikasi `attribute-A` atau `attribute-B`.
- Karena kebijakan mengizinkan `UpdateItem`, aplikasi dapat menyisipkan item baru, dan atribut yang tersembunyi akan null dalam item baru. Jika atribut ini diproyeksikan ke dalam `TopScoreDateTimeIndex`, kebijakan memiliki manfaat tambahan untuk mencegah kueri yang menyebabkan pengambilan dari tabel.
- Aplikasi tidak dapat membaca atribut apa pun selain yang tercantum dalam `dynamodb:Attributes`. Dengan adanya kebijakan ini, aplikasi harus menetapkan parameter `Select` ke `SPECIFIC_ATTRIBUTES` dalam permintaan baca, dan hanya atribut dalam daftar izin yang dapat diminta. Untuk permintaan tulis, aplikasi tidak dapat mengatur `ReturnValues` ke `ALL_OLD` atau `ALL_NEW` dan tidak dapat melakukan operasi tulis bersyarat berdasarkan atribut lainnya.

Topik terkait

- Manajemen Identitas dan Akses untuk Amazon DynamoDB
- Izin DynamoDB API: Referensi tindakan, sumber daya, dan kondisi

Menggunakan federasi identitas web

Jika Anda menulis aplikasi yang ditargetkan untuk sejumlah besar pengguna, Anda dapat menggunakan federasi identitas web untuk autentikasi dan otorisasi. Federasi identitas web menghilangkan kebutuhan untuk membuat pengguna individual. Sebagai gantinya, pengguna dapat masuk ke penyedia identitas dan kemudian mendapatkan kredensi keamanan sementara dari AWS Security Token Service ()AWS STS. Aplikasi kemudian dapat menggunakan kredensi ini untuk mengakses AWS layanan.

Federasi identitas web mendukung penyedia identitas berikut:

- Login with Amazon
- Facebook
- Google

Sumber daya tambahan untuk federasi identitas web

Sumber daya berikut ini dapat membantu Anda mempelajari lebih jauh lagi tentang federasi identitas web:

- Postingan Web Identity Federation using the AWS SDK untuk .NET pada blog AWS Developer memberi panduan tentang cara menggunakan federasi identitas web dengan Facebook. Ini termasuk cuplikan kode di C # yang menunjukkan bagaimana mengasumsikan peran IAM dengan identitas web dan cara menggunakan kredenal keamanan sementara untuk mengakses sumber daya. AWS
- AWS Mobile SDK for iOS dan AWS Mobile SDK for Android berisi aplikasi sampel. Aplikasi ini mencakup kode yang menunjukkan cara menginvokasi penyedia identitas, kemudian cara menggunakan informasi dari penyedia tersebut untuk mendapatkan dan menggunakan kredensial keamanan sementara.
- Artikel Federasi Identitas Web dengan Aplikasi Seluler membahas federasi identitas web dan menunjukkan contoh bagaimana menggunakan federasi identitas web untuk mengakses AWS sumber daya.

Contoh kebijakan untuk federasi identitas web

Untuk menunjukkan bagaimana Anda dapat menggunakan federasi identitas web dengan DynamoDB, kunjungi kembali tabel GameScoresyang diperkenalkan di. Menggunakan ketentuan kebijakan IAM untuk kontrol akses terperinci Berikut adalah kunci utama untuk GameScores.

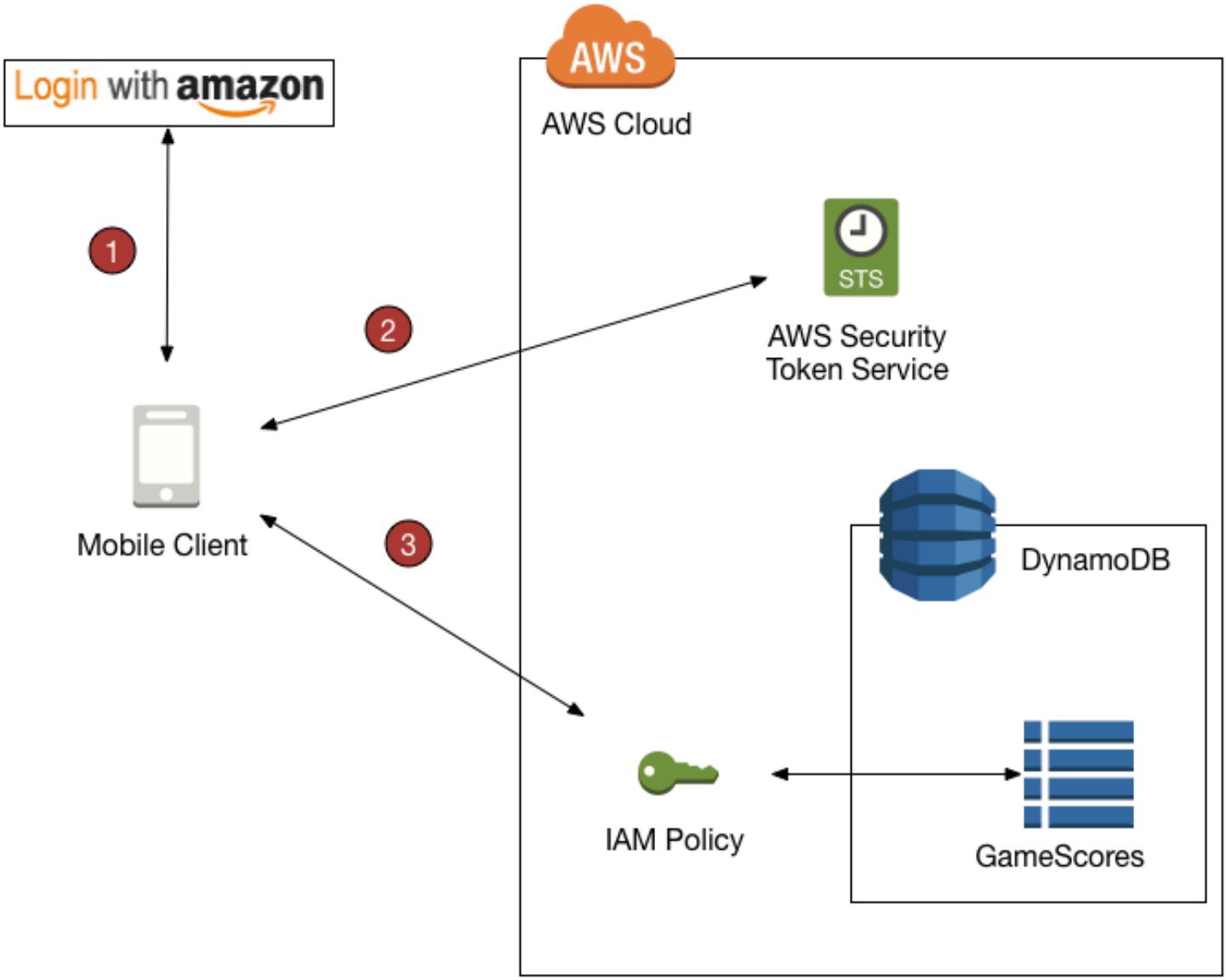
Nama Tabel	Jenis Kunci Primer	Nama dan Jenis Kunci Partisi	Nama dan Jenis Kunci Urutan
GameScores (UserId, GameTitle, ...)	Komposit	Nama Atribut: UserId Tipe: String	Nama Atribut: GameTitle Tipe: String

Sekarang anggaplah bahwa aplikasi game seluler menggunakan tabel ini, dan aplikasi perlu mendukung ribuan, atau bahkan jutaan, pengguna. Pada skala ini, menjadi sangat sulit untuk mengelola pengguna aplikasi individu, dan untuk menjamin bahwa setiap pengguna hanya dapat mengakses data mereka sendiri dalam GameScorestabel. Untungnya, banyak pengguna sudah memiliki akun dengan penyedia identitas pihak ketiga, seperti Facebook, Google, atau Login with Amazon. Jadi masuk akal untuk menggunakan salah satu penyedia ini untuk tugas autentikasi.

Untuk melakukannya menggunakan federasi identitas web, developer aplikasi harus mendaftarkan aplikasi dengan penyedia identitas (seperti Login with Amazon) dan memperoleh ID aplikasi unik. Selanjutnya, developer perlu membuat peran IAM. (Untuk contoh ini, peran ini diberi nama GameRole.) Peran harus memiliki dokumen kebijakan IAM yang dilampirkan padanya, menentukan kondisi di mana aplikasi dapat mengakses GameScorestabel.

Ketika ingin bermain game, pengguna masuk ke akun Login with Amazon mereka dari dalam aplikasi game. Aplikasi kemudian memanggil AWS Security Token Service (AWS STS), memberikan Login with Amazon app ID dan meminta keanggotaan GameRole. AWS STS mengembalikan AWS kredensi sementara ke aplikasi dan memungkinkannya mengakses GameScorestabel, tunduk pada dokumen GameRolekebijakan.

Diagram berikut menunjukkan bagaimana potongan-potongan ini cocok satu sama lain.



Gambaran umum federasi identitas web

1. Aplikasi memanggil penyedia identitas pihak ketiga untuk mengautentikasi pengguna dan aplikasi. Penyedia identitas mengembalikan token identitas web ke aplikasi.
2. Aplikasi memanggil AWS STS dan meneruskan token identitas web sebagai input. AWS STS mengotorisasi aplikasi dan memberinya kredensi AWS akses sementara. Aplikasi ini diizinkan untuk mengambil peran IAM (GameRole) dan mengakses AWS sumber daya sesuai dengan kebijakan keamanan peran.

3. Aplikasi memanggil DynamoDB untuk mengakses tabel. GameScores Karena telah diasumsikan GameRole, aplikasi tunduk pada kebijakan keamanan yang terkait dengan peran itu. Dokumen kebijakan mencegah aplikasi mengakses data yang bukan milik pengguna.

Sekali lagi, berikut adalah kebijakan keamanan untuk GameRoleitu ditunjukkan diMenggunakan ketentuan kebijakan IAM untuk kontrol akses terperinci:

```
{
   "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"
            }
```

```
            }
        }
    ]
}
```

`Condition`Klausa menentukan item mana GameScoresyang terlihat oleh aplikasi. Hal ini dilakukan dengan membandingkan ID Login with Amazon untuk nilai kunci partisi `UserId` dalam `GameScores`. Hanya item milik pengguna saat ini yang dapat diproses menggunakan salah satu tindakan DynamoDB yang tercantum dalam kebijakan ini. Item lain dalam tabel tidak dapat diakses. Selain itu, hanya atribut tertentu yang tercantum dalam kebijakan yang dapat diakses.

Mempersiapkan penggunaan federasi identitas web

Jika Anda adalah developer aplikasi dan ingin menggunakan federasi identitas web untuk aplikasi Anda, ikuti langkah-langkah berikut:

1. Daftar sebagai developer dengan penyedia identitas pihak ketiga. Tautan eksternal berikut memberikan informasi tentang mendaftar dengan penyedia identitas yang didukung:
   - Login dengan Amazon Developer Center
   - Pendaftaran di situs Facebook
   - Menggunakan OAuth 2.0 untuk Mengakses Google APIs di situs Google
2. Daftarkan aplikasi Anda dengan penyedia identitas. Saat Anda melakukannya, penyedia memberikan ID yang unik ke aplikasi Anda. Jika Anda ingin aplikasi bekerja dengan beberapa penyedia identitas, Anda perlu memperoleh ID aplikasi dari masing-masing penyedia.
3. Buat satu atau beberapa peran IAM. Anda memerlukan satu peran untuk setiap penyedia identitas untuk setiap aplikasi. Misalnya, Anda dapat membuat peran yang dapat diasumsikan oleh aplikasi saat pengguna masuk menggunakan Login with Amazon, peran kedua untuk aplikasi yang sama di mana pengguna masuk menggunakan Facebook, dan peran ketiga untuk aplikasi di mana pengguna masuk menggunakan Google.

   Sebagai bagian dari proses pembuatan peran, Anda harus melampirkan kebijakan IAM ke peran tersebut. Dokumen kebijakan Anda harus menentukan sumber daya DynamoDB yang diperlukan oleh aplikasi Anda, dan izin untuk mengakses sumber daya tersebut.

Untuk informasi selengkapnya, lihat Tentang Federasi Identitas Web dalam Panduan Pengguna IAM.

> **Note**
>
> Sebagai alternatif AWS Security Token Service, Anda dapat menggunakan Amazon Cognito. Amazon Cognito adalah layanan pilihan untuk mengelola kredensial sementara untuk aplikasi seluler. Untuk informasi selengkapnya, lihat Mendapatkan kredensial dalam Panduan Developer Amazon Cognito.

Membuat kebijakan IAM menggunakan konsol DynamoDB

Konsol DynamoDB dapat membantu Anda membuat kebijakan IAM untuk digunakan dengan federasi identitas web. Untuk melakukannya, Anda memilih tabel DynamoDB dan menentukan penyedia identitas, tindakan, serta atribut untuk disertakan dalam kebijakan. Konsol DynamoDB kemudian menghasilkan kebijakan yang dapat Anda lampirkan ke peran IAM.

1. Masuk ke AWS Management Console dan buka konsol DynamoDB di. https://console.aws.amazon.com/dynamodb/
2. Di panel navigasi, pilih Tabel.
3. Dalam daftar tabel, pilih tabel yang ingin Anda buatkan kebijakan IAM.
4. Pilih tombol Tindakan, dan pilih Buat Kebijakan Kontrol Akses.
5. Pilih penyedia identitas, tindakan, dan atribut untuk kebijakan.

   Jika pengaturan sudah sesuai keinginan Anda, pilih Buat Kebijakan. Kebijakan yang dihasilkan muncul.
6. Pilih Lihat Dokumentasi, lalu ikuti langkah-langkah yang diperlukan untuk melampirkan kebijakan yang dihasilkan ke peran IAM.

Menulis aplikasi Anda untuk menggunakan federasi identitas web

Untuk menggunakan federasi identitas web, aplikasi Anda harus mengasumsikan peran IAM yang Anda buat. Sejak saat itu, aplikasi menghormati kebijakan akses yang Anda lampirkan pada peran tersebut.

Saat runtime, jika aplikasi Anda menggunakan federasi identitas web, aplikasi harus mengikuti langkah-langkah berikut:

1. Autentikasi dengan penyedia identitas pihak ketiga. Aplikasi Anda harus memanggil penyedia identitas menggunakan antarmuka yang mereka sediakan. Cara yang tepat di mana Anda

mengautentikasi pengguna bergantung pada penyedia dan pada platform apa aplikasi Anda berjalan. Biasanya, jika pengguna belum masuk, penyedia identitas berperan untuk menampilkan halaman masuk untuk penyedia tersebut.

Setelah penyedia identitas mengautentikasi pengguna, penyedia mengembalikan token identitas web ke aplikasi Anda. Format token ini bergantung pada penyedia, tetapi biasanya berupa string karakter yang sangat panjang.

2. Dapatkan kredensi AWS keamanan sementara. Untuk melakukan ini, aplikasi Anda mengirimkan permintaan `AssumeRoleWithWebIdentity` untuk AWS Security Token Service (AWS STS). Permintaan ini berisi hal berikut ini:
   - Token identitas web dari langkah sebelumnya
   - ID aplikasi dari penyedia identitas
   - Amazon Resource Name (ARN) dari peran IAM yang Anda buat untuk penyedia identitas ini untuk aplikasi ini

   AWS STS mengembalikan satu set kredensi AWS keamanan yang kedaluwarsa setelah jangka waktu tertentu (3.600 detik, secara default).

   Berikut ini adalah sampel permintaan dan respons dari tindakan `AssumeRoleWithWebIdentity` dalam AWS STS. Token identitas web diperoleh dari penyedia identitas Login with Amazon.

```
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. Akses AWS sumber daya. Respons dari AWS STS berisi informasi yang dibutuhkan aplikasi Anda untuk mengakses sumber daya DynamoDB:

   - Bidang `AccessKeyID`, `SecretAccessKey`, dan `SessionToken` berisi kredensial keamanan yang valid hanya untuk pengguna dan aplikasi ini.
   - Bidang `Expiration` menandakan batas waktu untuk kredensial ini, setelah itu mereka tidak lagi valid.
   - Bidang `AssumedRoleId` berisi nama peran IAM khusus sesi yang telah diasumsikan oleh aplikasi. Aplikasi ini menghormati kontrol akses dalam dokumen kebijakan IAM selama sesi ini.
   - Bidang `SubjectFromWebIdentityToken` berisi ID unik yang muncul dalam variabel kebijakan IAM untuk penyedia identitas khusus ini. Berikut ini adalah variabel kebijakan IAM untuk penyedia yang didukung, dan beberapa contoh nilai untuk mereka:

Variabel Kebijakan	Nilai Contoh
`${www.amazon.com:user_id}`	`amzn1.account.AGJZDKHJKAUUS W6C44CHPEXAMPLE`
`${graph.facebook.com:id}`	`123456789`
`${accounts.google.com:sub}`	`123456789012345678901`

Untuk kebijakan IAM contoh di mana variabel kebijakan ini digunakan, lihat Kebijakan Contoh: Menggunakan ketentuan untuk kontrol parameter terperinci.

Untuk informasi selengkapnya tentang cara AWS STS menghasilkan kredenal akses sementara, lihat Meminta Kredensial Keamanan Sementara di Panduan Pengguna IAM.

Izin DynamoDB API: Referensi tindakan, sumber daya, dan kondisi

Saat Anda menyiapkan Manajemen Identitas dan Akses untuk Amazon DynamoDB dan menulis kebijakan izin yang dapat Anda lampirkan ke identitas IAM (kebijakan berbasis identitas), Anda dapat menggunakan daftar Kunci tindakan, sumber daya, dan kondisi untuk Amazon DynamoDB dalam Panduan Pengguna IAM sebagai referensi. Halaman ini mencantumkan setiap operasi DynamoDB API, tindakan terkait yang dapat Anda berikan izin untuk melakukan tindakan, dan AWS sumber daya yang dapat Anda berikan izin. Anda menentukan tindakan dalam bidang `Action` kebijakan, dan Anda menentukan nilai sumber daya pada bidang `Resource` kebijakan.

Anda dapat menggunakan kunci kondisi AWS-wide dalam kebijakan DynamoDB Anda untuk menyatakan kondisi. Untuk daftar lengkap kunci AWS-wide, lihat referensi elemen kebijakan IAM JSON di Panduan Pengguna IAM.

Selain tombol kondisi AWS-wide, DynamoDB memiliki kunci spesifiknya sendiri yang dapat Anda gunakan dalam kondisi. Untuk informasi selengkapnya, lihat Menggunakan ketentuan kebijakan IAM untuk kontrol akses terperinci.

Topik terkait

- Manajemen Identitas dan Akses untuk Amazon DynamoDB
- Menggunakan ketentuan kebijakan IAM untuk kontrol akses terperinci

Validasi kepatuhan oleh industri untuk DynamoDB

Untuk mempelajari apakah an Layanan AWS berada dalam lingkup program kepatuhan tertentu, lihat Layanan AWS di Lingkup oleh Program Kepatuhan Layanan AWS dan pilih program kepatuhan yang Anda minati. Untuk informasi umum, lihat Program AWS Kepatuhan Program AWS .

Anda dapat mengunduh laporan audit pihak ketiga menggunakan AWS Artifact. Untuk informasi selengkapnya, lihat Mengunduh Laporan di AWS Artifact .

Tanggung jawab kepatuhan Anda saat menggunakan Layanan AWS ditentukan oleh sensitivitas data Anda, tujuan kepatuhan perusahaan Anda, dan hukum dan peraturan yang berlaku. AWS menyediakan sumber daya berikut untuk membantu kepatuhan:

- Kepatuhan dan Tata Kelola Keamanan – Panduan implementasi solusi ini membahas pertimbangan arsitektur serta memberikan langkah-langkah untuk menerapkan fitur keamanan dan kepatuhan.
- Referensi Layanan yang Memenuhi Syarat HIPAA — Daftar layanan yang memenuhi syarat HIPAA. Tidak semua memenuhi Layanan AWS syarat HIPAA.
- AWS Sumber Daya AWS — Kumpulan buku kerja dan panduan ini mungkin berlaku untuk industri dan lokasi Anda.
- AWS Panduan Kepatuhan Pelanggan - Memahami model tanggung jawab bersama melalui lensa kepatuhan. Panduan ini merangkum praktik terbaik untuk mengamankan Layanan AWS dan memetakan panduan untuk kontrol keamanan di berbagai kerangka kerja (termasuk Institut Standar dan Teknologi Nasional (NIST), Dewan Standar Keamanan Industri Kartu Pembayaran (PCI), dan Organisasi Internasional untuk Standardisasi (ISO)).
- Mengevaluasi Sumber Daya dengan Aturan dalam Panduan AWS Config Pengembang — AWS Config Layanan menilai seberapa baik konfigurasi sumber daya Anda mematuhi praktik internal, pedoman industri, dan peraturan.
- AWS Security Hub— Ini Layanan AWS memberikan pandangan komprehensif tentang keadaan keamanan Anda di dalamnya AWS. Security Hub menggunakan kontrol keamanan untuk sumber daya AWS Anda serta untuk memeriksa kepatuhan Anda terhadap standar industri keamanan dan praktik terbaik. Untuk daftar layanan dan kontrol yang didukung, lihat Referensi kontrol Security Hub.
- Amazon GuardDuty — Ini Layanan AWS mendeteksi potensi ancaman terhadap beban kerja Akun AWS, kontainer, dan data Anda dengan memantau lingkungan Anda untuk aktivitas mencurigakan dan berbahaya. GuardDuty dapat membantu Anda mengatasi berbagai persyaratan kepatuhan, seperti PCI DSS, dengan memenuhi persyaratan deteksi intrusi yang diamanatkan oleh kerangka kerja kepatuhan tertentu.
- AWS Audit ManagerIni Layanan AWS membantu Anda terus mengaudit AWS penggunaan Anda untuk menyederhanakan cara Anda mengelola risiko dan kepatuhan terhadap peraturan dan standar industri.

Ketahanan dan pemulihan bencana di Amazon DynamoDB

Infrastruktur AWS global dibangun di sekitar AWS Wilayah dan Zona Ketersediaan. AWS Wilayah menyediakan beberapa Availability Zone yang terpisah secara fisik dan terisolasi, yang terhubung dengan latensi rendah, throughput tinggi, dan jaringan yang sangat redundan. Dengan Zona Ketersediaan, Anda dapat merancang serta mengoperasikan aplikasi dan basis data yang secara otomatis melakukan failover di antara Zona Ketersediaan tanpa gangguan. Zona Ketersediaan memiliki ketersediaan lebih tinggi, toleransi kesalahan lebih baik, dan lebih dapat diskalakan dibandingkan infrastruktur pusat data tunggal atau beberapa pusat data tradisional.

Jika Anda harus melakukan replikasi data atau aplikasi Anda pada jarak geografis yang lebih luas, gunakan Wilayah Lokal AWS . Wilayah AWS Lokal adalah pusat data tunggal yang dirancang untuk melengkapi AWS Wilayah yang ada. Seperti semua AWS Wilayah, Wilayah AWS Lokal benar-benar terisolasi dari AWS Wilayah lain.

Untuk informasi selengkapnya tentang AWS Wilayah dan Availability Zone, lihat infrastruktur AWS global.

Selain infrastruktur AWS global, Amazon DynamoDB menawarkan beberapa fitur untuk membantu mendukung ketahanan data dan kebutuhan pencadangan Anda.

Pencadangan dan pemulihan sesuai permintaan

DynamoDB menyediakan kemampuan pencadangan sesuai permintaan. Hal ini memungkinkan Anda membuat pencadangan penuh dari tabel untuk retensi dan pengarsipan jangka panjang. Untuk informasi selengkapnya, lihat Pencadangan dan pemulihan sesuai permintaan untuk DynamoDB.

Point-in-time pemulihan

Point-in-time pemulihan membantu melindungi tabel DynamoDB Anda dari operasi tulis atau hapus yang tidak disengaja. Dengan pemulihan titik waktu, Anda tidak perlu khawatir dengan membuat, mengelola, atau menjadwalkan pencadangan sesuai permintaan. Untuk informasi selengkapnya, lihat Point-in-time pemulihan untuk DynamoDB.

Tabel global yang disinkronkan di seluruh wilayah AWS

DynamoDB menyebarkan data dan lalu lintas untuk tabel Anda secara otomatis melalui sejumlah server yang cukup untuk menangani throughput dan persyaratan penyimpanan Anda, sekaligus mempertahankan performa yang konsisten dan cepat. Semua data Anda disimpan di solid-state

disk (SSDs) dan secara otomatis direplikasi di beberapa Availability Zone di suatu AWS Wilayah, menyediakan ketersediaan tinggi bawaan dan daya tahan data. Anda dapat menggunakan tabel global untuk menjaga tabel DynamoDB tetap sinkron di seluruh Wilayah. AWS

Keamanan infrastruktur di Amazon DynamoDB

Sebagai layanan terkelola, Amazon DynamoDB dilindungi oleh AWS prosedur keamanan jaringan global yang dijelaskan dalam perlindungan Infrastruktur yang terletak di Well-Architected Framework. AWS

Anda menggunakan panggilan API yang AWS dipublikasikan untuk mengakses DynamoDB melalui jaringan. Klien dapat menggunakan TLS (Transport Layer Security) versi 1.2 atau 1.3. Klien juga harus mendukung cipher suite dengan perfect forward secrecy (PFS) seperti Ephemeral Diffie-Hellman (DHE) atau Elliptic Curve Diffie-Hellman Ephemeral (ECDHE). Sebagian besar sistem modern seperti Java 7 dan sistem yang lebih baru mendukung mode ini. Selain itu, permintaan harus ditandatangani menggunakan ID kunci akses dan kunci akses rahasia yang dikaitkan dengan pengguna utama IAM. Atau Anda dapat menggunakan AWS Security Token Service (AWS STS) untuk membuat kredensial keamanan sementara untuk menandatangani permintaan.

Anda juga dapat menggunakan titik akhir virtual private cloud (VPC) untuk DynamoDB untuk mengaktifkan instans Amazon EC2 di VPC Anda untuk menggunakan alamat IP pribadi mereka untuk mengakses DynamoDB tanpa eksposur ke internet publik. Untuk informasi selengkapnya, lihat Menggunakan titik akhir Amazon VPC untuk mengakses DynamoDB.

Menggunakan titik akhir Amazon VPC untuk mengakses DynamoDB

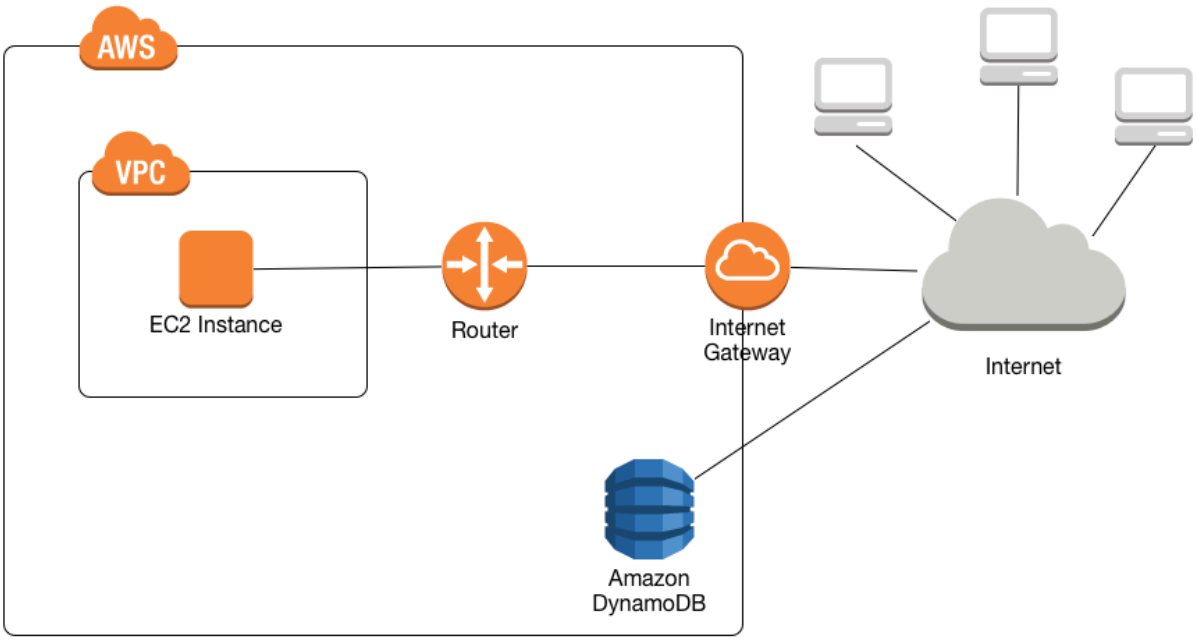
Untuk alasan keamanan, banyak AWS pelanggan menjalankan aplikasi mereka dalam lingkungan Amazon Virtual Private Cloud (Amazon VPC). Dengan Amazon VPC, Anda dapat meluncurkan EC2 instans Amazon ke cloud pribadi virtual, yang secara logis terisolasi dari jaringan lain—termasuk internet publik. Dengan Amazon VPC, Anda dapat mengontrol rentang alamat IP, subnet, tabel perutean, gateway jaringan, dan pengaturan keamanan.

> **Note**
>
> Jika Anda membuat Akun AWS setelah 4 Desember 2013, maka Anda sudah memiliki VPC default di masing-masing. Wilayah AWS VPC default siap untuk Anda gunakan—Anda dapat segera menggunakannya tanpa harus melakukan langkah-langkah konfigurasi tambahan.

Untuk informasi selengkapnya, lihat VPC Default dan Subnet Default di Panduan Pengguna Amazon VPC.

Untuk mengakses internet publik, VPC Anda harus memiliki gateway internet—router virtual yang menghubungkan VPC Anda ke internet. Ini memungkinkan aplikasi yang berjalan di Amazon EC2 di VPC Anda untuk mengakses sumber daya internet, seperti Amazon DynamoDB.

Secara default, komunikasi ke dan dari DynamoDB menggunakan protokol HTTPS, yang melindungi lalu lintas jaringan menggunakan enkripsi SSL/TLS. Diagram berikut menunjukkan EC2 instance Amazon dalam VPC yang mengakses DynamoDB, dengan meminta DynamoDB menggunakan gateway internet daripada titik akhir VPC.



Banyak pelanggan memiliki kekhawatiran yang sah mengenai privasi dan keamanan saat mengirim dan menerima data melalui internet publik. Anda dapat mengatasi masalah ini menggunakan jaringan privat virtual (VPN) untuk merutekan semua lalu lintas jaringan DynamoDB melalui infrastruktur jaringan perusahaan Anda. Namun, pendekatan ini dapat menimbulkan tantangan bandwidth dan ketersediaan.

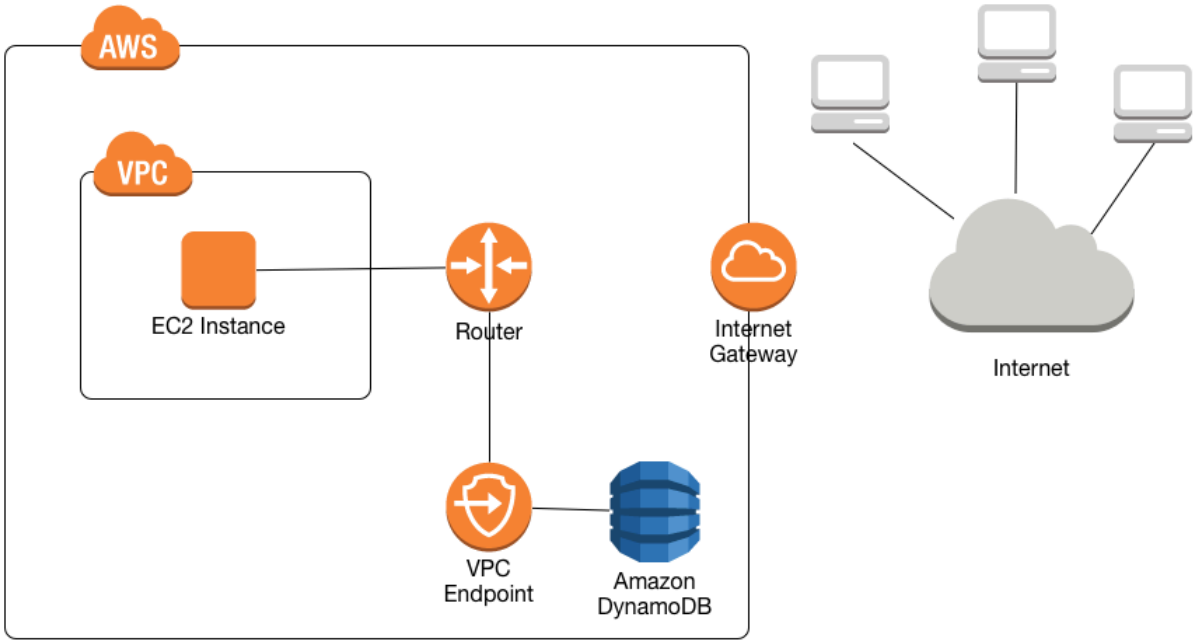
Titik akhir VPC untuk DynamoDB dapat mengatasi tantangan ini. Titik akhir VPC untuk DynamoDB memungkinkan EC2 instans Amazon di VPC Anda menggunakan alamat IP pribadinya untuk

mengakses DynamoDB tanpa eksposur ke internet publik. EC2 Instans Anda tidak memerlukan alamat IP publik, dan Anda tidak memerlukan gateway internet, perangkat NAT, atau gateway pribadi virtual di VPC Anda. Anda menggunakan kebijakan titik akhir untuk mengontrol akses ke DynamoDB. Lalu lintas antara VPC Anda dan AWS layanan tidak meninggalkan jaringan Amazon.

> **Note**
>
> Bahkan ketika Anda menggunakan alamat IP publik, semua komunikasi VPC antara instance dan layanan yang dihosting tetap pribadi di AWS dalam jaringan. AWS Paket yang berasal dari AWS jaringan dengan tujuan di jaringan tetap berada di AWS jaringan AWS global, kecuali lalu lintas ke atau dari Wilayah AWS Tiongkok.

Saat Anda membuat titik akhir VPC untuk DynamoDB, setiap permintaan ke titik akhir DynamoDB dalam Wilayah (misalnya, dynamodb.us-west-2.amazonaws.com) dirutekan ke titik akhir DynamoDB privat dalam jaringan Amazon. Anda tidak perlu memodifikasi aplikasi yang berjalan pada EC2 instance di VPC Anda. Nama titik akhir tetap sama, namun rute ke DynamoDB tetap sepenuhnya berada dalam jaringan Amazon, dan tidak mengakses internet publik.

Diagram berikut menunjukkan bagaimana sebuah EC2 instance dalam VPC dapat menggunakan titik akhir VPC untuk mengakses DynamoDB.



Untuk informasi selengkapnya, lihat the section called “Tutorial: Menggunakan titik akhir VPC untuk DynamoDB”.

Berbagi titik akhir Amazon VPC dan DynamoDB

Untuk mengaktifkan akses ke layanan DynamoDB melalui titik akhir gateway subnet VPC, Anda harus memiliki izin akun pemilik untuk subnet VPC tersebut.

Setelah titik akhir gateway subnet VPC diberikan akses ke DynamoDB, akun AWS mana pun yang memiliki akses ke subnet tersebut dapat menggunakan DynamoDB. Hal ini berarti semua pengguna akun dalam subnet VPC dapat menggunakan tabel DynamoDB apa pun yang dapat mereka akses. Ini termasuk tabel DynamoDB yang terkait dengan akun yang berbeda dari subnet VPC. Pemilik subnet VPC masih dapat membatasi pengguna tertentu dalam subnet untuk menggunakan layanan DynamoDB melalui titik akhir gateway, sesuai kebijakannya.

Tutorial: Menggunakan titik akhir VPC untuk DynamoDB

Bagian ini memandu Anda dalam menyiapkan dan menggunakan titik akhir VPC untuk DynamoDB.

Topik

- Langkah 1: Luncurkan EC2 instans Amazon

- Langkah 2: Konfigurasikan EC2 instans Amazon Anda
- Langkah 3: Membuat titik akhir VPC untuk DynamoDB
- Langkah 4: (Opsional) Hapus

Langkah 1: Luncurkan EC2 instans Amazon

Pada langkah ini, Anda meluncurkan EC2 instans Amazon di VPC Amazon default Anda. Anda kemudian dapat membuat dan menggunakan titik akhir VPC untuk DynamoDB.

1. Buka EC2 konsol Amazon di https://console.aws.amazon.com/ec2/.
2. Pilih Luncurkan Instans, dan lakukan hal berikut:

   #### Langkah 1: Pilih Amazon Machine Image (AMI)

   - Di bagian atas daftar AMIs, buka Amazon Linux AMI dan pilih Pilih.

   #### Langkah 2: Pilih Jenis Instans

   - Di bagian atas daftar jenis instans, piliht2.micro.
   - Pilih Berikutnya: Konfigurasikan Detail Instans.

   #### Langkah 3: Konfigurasikan Detail Instans

   - Buka Jaringan, lalu pilih VPC default Anda.

     Pilih Berikutnya: Tambahkan Penyimpanan.

   #### Langkah 4: Tambahkan Penyimpanan

   - Lewati langkah ini dengan memilih Selanjutnya: Instans Tag.

   #### Langkah 5: Instans Tag

   - Lewati langkah ini dengan memilih Berikutnya: Konfigurasikan Grup Keamanan.

   #### Langkah 6: Konfigurasikan Grup Keamanan

- Pilih Pilih grup keamanan yang ada.
- Dalam daftar grup keamanan, pilih default. Ini adalah grup keamanan default untuk VPC Anda.
- Pilih Berikutnya: Tinjau dan Luncurkan.

Langkah 7: Tinjau Peluncuran Instans

- Pilih Luncurkan.

3. Di jendela Pilih pasangan kunci yang ada atau buat pasangan kunci baru, lakukan salah satu hal berikut:

   - Jika Anda tidak memiliki EC2 key pair Amazon, pilih Create a new key pair dan ikuti petunjuknya. Anda akan diminta untuk mengunduh file kunci pribadi (file.pem); Anda akan memerlukan file ini nanti ketika Anda masuk ke instance Amazon EC2 Anda.
   - Jika Anda sudah memiliki EC2 key pair Amazon yang sudah ada, buka Select a key pair dan pilih key pair Anda dari daftar. Anda harus sudah memiliki file kunci pribadi (file.pem) yang tersedia untuk masuk ke instance Amazon EC2 Anda.

4. Ketika Anda telah mengonfigurasi pasangan kunci, pilih Luncurkan Instans.
5. Kembali ke halaman beranda EC2 konsol Amazon dan pilih instance yang Anda luncurkan. Pada panel bawah, di tab Deskripsi, cari DNS Publik untuk instans Anda. Sebagai contoh: `ec2-00-00-00-00.us-east-1.compute.amazonaws.com`.

   Catat nama DNS publik ini, karena Anda akan memerlukannya di langkah berikutnya dalam tutorial ini (Langkah 2: Konfigurasikan EC2 instans Amazon Anda).

> **Note**
>
> Ini akan memakan waktu beberapa menit agar EC2 instans Amazon Anda tersedia. Sebelum Anda melanjutkan ke langkah berikutnya, pastikan Status Instans dalam kondisi `running` dan semua Pemeriksaan Status telah lulus.

Langkah 2: Konfigurasikan EC2 instans Amazon Anda

Ketika EC2 instans Amazon Anda tersedia, Anda akan dapat masuk ke dalamnya dan menyiapkannya untuk penggunaan pertama.

> **Note**
>
> Langkah-langkah berikut mengasumsikan bahwa Anda terhubung ke EC2 instans Amazon Anda dari komputer yang menjalankan Linux. Untuk cara lain untuk terhubung, lihat Connect to Your Linux Instance di Panduan EC2 Pengguna Amazon.

1. Anda harus mengotorisasi lalu lintas SSH masuk ke instans Amazon Anda. EC2 Untuk melakukan ini, Anda akan membuat grup EC2 keamanan baru, dan kemudian menetapkan grup keamanan ke EC2 instans Anda.

   a. Pada panel navigasi, pilih Grup Keamanan.

   b. Pilih Buat Grup Keamanan. Di jendela Buat Grup Keamanan, lakukan hal berikut:

      - Nama grup keamanan—ketikkan nama untuk grup keamanan Anda. Misalnya: `my-ssh-access`
      - Deskripsi—ketikkan deskripsi singkat untuk grup keamanan.
      - VPC—pilih VPC default Anda.
      - Di bagian Aturan grup keamanan, pilih Tambahkan Aturan dan lakukan hal berikut:
        - Jenis—pilihlah SSH.
        - Sumber—pilih IP Saya.

      Jika pengaturan sudah sesuai keinginan Anda, pilih Buat.

   c. Di panel navigasi, pilih Instans.

   d. Pilih EC2 instans Amazon yang Anda luncurkanLangkah 1: Luncurkan EC2 instans Amazon.

   e. Pilih Tindakan –> Jaringan –> Ubah Grup Keamanan.

   f. Di Ubah Grup Keamanan, pilih grup keamanan yang Anda buat sebelumnya dalam prosedur ini (misalnya: `my-ssh-access`). Saat ini grup keamanan `default` juga harus dipilih. Bila pengaturan sesuai keinginan Anda, pilih Tetapkan Grup Keamanan.

2. Gunakan `ssh` perintah untuk masuk ke EC2 instans Amazon Anda, seperti pada contoh berikut.

   ```
   ssh -i my-keypair.pem ec2-user@public-dns-name
   ```

   Anda perlu menentukan file kunci privat Anda (file .pem) dan nama DNS publik dari instans Anda. (Lihat Langkah 1: Luncurkan EC2 instans Amazon).

ID login adalah `ec2-user`. Tidak diperlukan kata sandi.

3. Konfigurasikan AWS kredensyal Anda seperti yang ditunjukkan pada contoh berikut. Masukkan ID kunci akses, kunci rahasia, dan nama Wilayah default AWS saat diminta.

```
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]:
```

Anda sekarang siap untuk membuat titik akhir VPC untuk DynamoDB.

Langkah 3: Membuat titik akhir VPC untuk DynamoDB

Pada langkah ini, Anda akan membuat titik akhir VPC untuk DynamoDB dan mengujinya untuk memastikannya berfungsi.

1. Sebelum memulai, verifikasi bahwa Anda dapat berkomunikasi dengan DynamoDB menggunakan titik akhir publiknya.

```
aws dynamodb list-tables
```

Outputnya akan menampilkan daftar tabel DynamoDB yang Anda miliki saat ini. (Jika Anda tidak memiliki tabel apa pun, daftarnya akan kosong.).

2. Verifikasi bahwa DynamoDB adalah layanan yang tersedia untuk membuat titik akhir VPC di Wilayah saat ini. AWS (Perintah ditampilkan dalam teks tebal, diikuti dengan contoh output.)

```
aws ec2 describe-vpc-endpoint-services
```

```
{
    "ServiceNames": [
        "com.amazonaws.us-east-1.s3",
        "com.amazonaws.us-east-1.dynamodb"
    ]
}
```

Dalam contoh output, DynamoDB adalah salah satu layanan yang tersedia, sehingga Anda dapat melanjutkan dengan membuat titik akhir VPC untuk layanan tersebut.

3. Tentukan pengidentifikasi VPC Anda.

```
aws ec2 describe-vpcs
```

```
{
    "Vpcs": [
        {
            "VpcId": "vpc-0bbc736e",
            "InstanceTenancy": "default",
            "State": "available",
            "DhcpOptionsId": "dopt-8454b7e1",
            "CidrBlock": "172.31.0.0/16",
            "IsDefault": true
        }
    ]
}
```

Dalam contoh output, ID VPC adalah `vpc-0bbc736e`.

4. Buat titik akhir VPC. Untuk parameter `--vpc-id`, tentukan ID VPC dari langkah sebelumnya. Gunakan parameter `--route-table-ids` untuk mengasosiasikan titik akhir dengan tabel rute Anda.

```
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. Verifikasi bahwa Anda dapat mengakses DynamoDB melalui titik akhir VPC.

```
aws dynamodb list-tables
```

Jika mau, Anda dapat mencoba beberapa AWS CLI perintah lain untuk DynamoDB. Untuk informasi selengkapnya, lihat Referensi Perintah AWS AWS CLI.

Langkah 4: (Opsional) Hapus

Jika Anda ingin menghapus sumber daya yang Anda buat dalam tutorial ini, ikuti prosedur berikut:

Untuk menghapus titik akhir VPC Anda untuk DynamoDB

1. Masuk ke EC2 instans Amazon Anda.
2. Tentukan ID titik akhir VPC.

```
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"
    }
}
```

Dalam contoh output, ID titik akhir VPC adalah `vpce-9b15e2f2`.

3. Hapus titik akhir VPC.

```
aws ec2 delete-vpc-endpoints --vpc-endpoint-ids vpce-9b15e2f2
```

```
{
    "Unsuccessful": []
}
```

Array kosong `[]` menunjukkan keberhasilan (tidak ada permintaan yang gagal).

Untuk menghentikan instans Amazon EC2 Anda

1. Buka EC2 konsol Amazon di https://console.aws.amazon.com/ec2/.
2. Di panel navigasi, pilih Instans.
3. Pilih EC2 instans Amazon Anda.
4. Pilih Tindakan, Status Instans, Akhiri.
5. Dalam jendela konfirmasi, pilih Ya, Hentikan.

AWS PrivateLink untuk DynamoDB

Dengan AWS PrivateLink untuk DynamoDB, Anda dapat menyediakan antarmuka titik akhir Amazon VPC (titik akhir antarmuka) di cloud pribadi virtual Anda (Amazon VPC). Titik akhir ini dapat diakses langsung dari aplikasi yang ada di tempat melalui VPN dan AWS Direct Connect, atau berbeda dengan peering Wilayah AWS VPC Amazon. Menggunakan AWS PrivateLink dan antarmuka endpoint, Anda dapat menyederhanakan konektivitas jaringan pribadi dari aplikasi Anda ke DynamoDB.

Aplikasi di VPC Anda tidak memerlukan alamat IP publik untuk berkomunikasi dengan DynamoDB menggunakan titik akhir antarmuka VPC untuk operasi DynamoDB. Endpoint antarmuka diwakili oleh satu atau lebih antarmuka jaringan elastis (ENIs) yang diberi alamat IP pribadi dari subnet di VPC Amazon Anda. Permintaan ke DynamoDB melalui titik akhir antarmuka tetap berada di jaringan Amazon. Anda juga dapat mengakses titik akhir antarmuka di VPC Amazon dari aplikasi lokal AWS Direct Connect melalui AWS Virtual Private Network atau ().AWS VPNUntuk informasi selengkapnya tentang cara menghubungkan VPC Amazon dengan jaringan lokal, lihat Panduan AWS Direct Connect Pengguna dan Panduan Pengguna.AWS Site-to-Site VPN

Untuk informasi umum tentang titik akhir antarmuka, lihat Antarmuka Amazon VPC endpoints AWS PrivateLink() di Panduan.AWS PrivateLink AWS PrivateLink juga didukung untuk titik akhir Amazon DynamoDB Streams. Untuk informasi selengkapnya, lihat the section called “AWS PrivateLink untuk DynamoDB Streams”.

Topik

- Jenis titik akhir Amazon VPC untuk Amazon DynamoDB
- Pertimbangan saat menggunakan AWS PrivateLink untuk Amazon DynamoDB
- Membuat titik akhir Amazon VPC
- Mengakses titik akhir antarmuka Amazon DynamoDB
- Mengakses tabel DynamoDB dan mengontrol operasi API dari titik akhir antarmuka DynamoDB
- Memperbarui konfigurasi DNS on-premise
- Membuat kebijakan endpoint Amazon VPC untuk DynamoDB
- AWS PrivateLink untuk DynamoDB Streams

Jenis titik akhir Amazon VPC untuk Amazon DynamoDB

Anda dapat menggunakan dua jenis titik akhir Amazon VPC untuk mengakses Amazon DynamoDB: titik akhir gateway dan titik akhir antarmuka (dengan menggunakan). AWS PrivateLinkTitik akhir gateway adalah gateway yang Anda tentukan dalam tabel rute untuk mengakses DynamoDB dari VPC Amazon Anda melalui jaringan. AWS Titik akhir antarmuka memperluas fungsionalitas titik akhir gateway dengan menggunakan alamat IP pribadi untuk merutekan permintaan ke DynamoDB dari dalam VPC Amazon Anda, di tempat, atau dari VPC Amazon di tempat lain dengan menggunakan peering Amazon VPC atau. Wilayah AWS AWS Transit Gateway Untuk informasi lebih lanjut, lihat Apa itu peering Amazon VPC? dan Transit Gateway vs Pengintip VPC Amazon.

Titik akhir antarmuka kompatibel dengan titik akhir gateway. Jika Anda memiliki titik akhir gateway yang ada di VPC Amazon, Anda dapat menggunakan kedua jenis titik akhir di VPC Amazon yang sama.

Titik akhir Gateway untuk DynamoDB	Endpoint antarmuka untuk DynamoDB
Dalam kedua kasus, lalu lintas jaringan Anda tetap berada di AWS jaringan.	
Menggunakan alamat IP publik Amazon DynamoDB	Gunakan alamat IP pribadi dari Amazon VPC Anda untuk mengakses Amazon DynamoDB
Jangan izinkan mengakses dari on-premise	Izinkan mengakses dari on-premise
Jangan izinkan akses dari yang lain Wilayah AWS	Izinkan akses dari titik akhir VPC Amazon di titik lain dengan Wilayah

Titik akhir Gateway untuk DynamoDB	Endpoint antarmuka untuk DynamoDB
	AWS menggunakan peering Amazon VPC atau AWS Transit Gateway
Tidak ditagih	Ditagih

Untuk informasi selengkapnya tentang titik akhir gateway, lihat titik akhir Gateway Amazon VPC di Panduan.AWS PrivateLink

Pertimbangan saat menggunakan AWS PrivateLink untuk Amazon DynamoDB

Pertimbangan Amazon VPC berlaku untuk Amazon AWS PrivateLink DynamoDB. Untuk informasi selengkapnya, lihat Pertimbangan antarmuka titik akhir dan AWS PrivateLink kuota dalam AWS PrivateLink Panduan. Selain itu, larangan berikut juga berlaku.

AWS PrivateLink untuk Amazon DynamoDB tidak mendukung hal berikut:

- Keamanan Lapisan Pengangkutan (TLS) 1.1
- Layanan Sistem Nama Domain Pribadi dan Hybrid (DNS)

Anda dapat mengirimkan hingga 50.000 permintaan per detik untuk setiap AWS PrivateLink titik akhir yang Anda aktifkan.

> **Note**
>
> Batas waktu konektivitas jaringan ke AWS PrivateLink titik akhir tidak berada dalam lingkup respons kesalahan DynamoDB dan perlu ditangani dengan tepat oleh aplikasi Anda yang terhubung ke titik akhir. PrivateLink

Membuat titik akhir Amazon VPC

Untuk membuat titik akhir antarmuka VPC Amazon, lihat Membuat titik akhir VPC Amazon di Panduan.AWS PrivateLink

Mengakses titik akhir antarmuka Amazon DynamoDB

Saat Anda membuat titik akhir antarmuka, DynamoDB menghasilkan dua jenis nama DNS DynamoDB spesifik titik akhir: Regional dan Zonal.

- Nama DNS Regional mencakup ID titik akhir VPC Amazon yang unik, pengenal layanan, Wilayah AWS dan namanya. `vpce.amazonaws.com` Misalnya, untuk ID titik akhir VPC Amazon*vpce-1a2b3c4d*, nama DNS yang dihasilkan mungkin mirip dengan. *vpce-1a2b3c4d-5e6f*`.dynamodb.us-east-1.vpce.amazonaws.com`
- Nama DNS Zonal mencakup Zona Ketersediaan—misalnya, *vpce-1a2b3c4d-5e6f*`-us-east-1a.dynamodb.us-east-1.vpce.amazonaws.com`. Anda dapat menggunakan opsi ini jika arsitektur Anda mengisolasi Zona Ketersediaan. Contoh, Anda bisa menggunakannya untuk kontainer kesalahan atau untuk mengurangi biaya transfer data Regional.

> **Note**
>
> Untuk mencapai keandalan yang optimal, sebaiknya gunakan layanan Anda di minimal tiga zona ketersediaan.

Mengakses tabel DynamoDB dan mengontrol operasi API dari titik akhir antarmuka DynamoDB

Anda dapat menggunakan AWS CLI atau AWS SDKs untuk mengakses tabel DynamoDB dan mengontrol operasi API melalui titik akhir antarmuka DynamoDB.

AWS CLI contoh

Untuk mengakses tabel DynamoDB atau operasi API kontrol DynamoDB melalui titik akhir antarmuka DynamoDB dalam perintah, gunakan parameter dan. AWS CLI `--region` `--endpoint-url`

Contoh: Buat titik akhir VPC

```
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
```

Contoh: Memodifikasi titik akhir VPC

```
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
```

Contoh: Daftar tabel menggunakan URL endpoint

Dalam contoh berikut, ganti Region `us-east-1` dan nama DNS dari `vpce-1a2b3c4d-5e6f.dynamodb.us-east-1.vpce.amazonaws.com` ID endpoint VPC dengan informasi Anda sendiri.

```
aws dynamodb --region us-east-1 --endpoint https://vpce-1a2b3c4d-5e6f.dynamodb.us-
east-1.vpce.amazonaws.com list-tables
```

AWS Contoh SDK

Untuk mengakses tabel DynamoDB atau operasi API kontrol DynamoDB melalui titik akhir antarmuka DynamoDB saat menggunakan, perbarui versi Anda ke versi terbaru. AWS SDKs SDKs Kemudian, konfigurasikan klien Anda untuk menggunakan URL titik akhir untuk mengakses tabel atau operasi API kontrol DynamoDB melalui titik akhir antarmuka DynamoDB.

SDK for Python (Boto3)

Contoh: Gunakan URL endpoint untuk mengakses tabel DynamoDB

Dalam contoh berikut, ganti Wilayah `us-east-1` dan ID titik akhir VPC `https://vpce-1a2b3c4d-5e6f.dynamodb.us-east-1.vpce.amazonaws.com` dengan informasi Anda sendiri.

```
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

Contoh: Gunakan URL endpoint untuk mengakses tabel DynamoDB

Dalam contoh berikut, ganti Wilayah `us-east-1` dan ID titik akhir VPC `https://vpce-1a2b3c4d-5e6f.dynamodb.us-east-1.vpce.amazonaws.com` dengan informasi Anda sendiri.

```
//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

Contoh: Gunakan URL endpoint untuk mengakses tabel DynamoDB

Dalam contoh berikut, ganti ID endpoint Region us-east-1 dan https://vpce-1a2b3c4d-5e6f.dynamodb.us-east-1.vpce.amazonaws.com VPC dengan informasi Anda sendiri.

```
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()
```

Memperbarui konfigurasi DNS on-premise

Saat menggunakan nama DNS khusus titik akhir untuk mengakses titik akhir antarmuka DynamoDB, Anda tidak perlu memperbarui penyelesai DNS lokal Anda. Anda dapat menyelesaikan nama DNS spesifik titik akhir dengan alamat IP pribadi titik akhir antarmuka dari domain DNS DynamoDB publik.

Menggunakan titik akhir antarmuka untuk mengakses DynamoDB tanpa titik akhir gateway atau gateway internet di Amazon VPC

Titik akhir antarmuka di VPC Amazon Anda dapat merutekan aplikasi VPC di Amazon dan aplikasi lokal ke DynamoDB melalui jaringan Amazon, seperti yang diilustrasikan dalam diagram berikut.

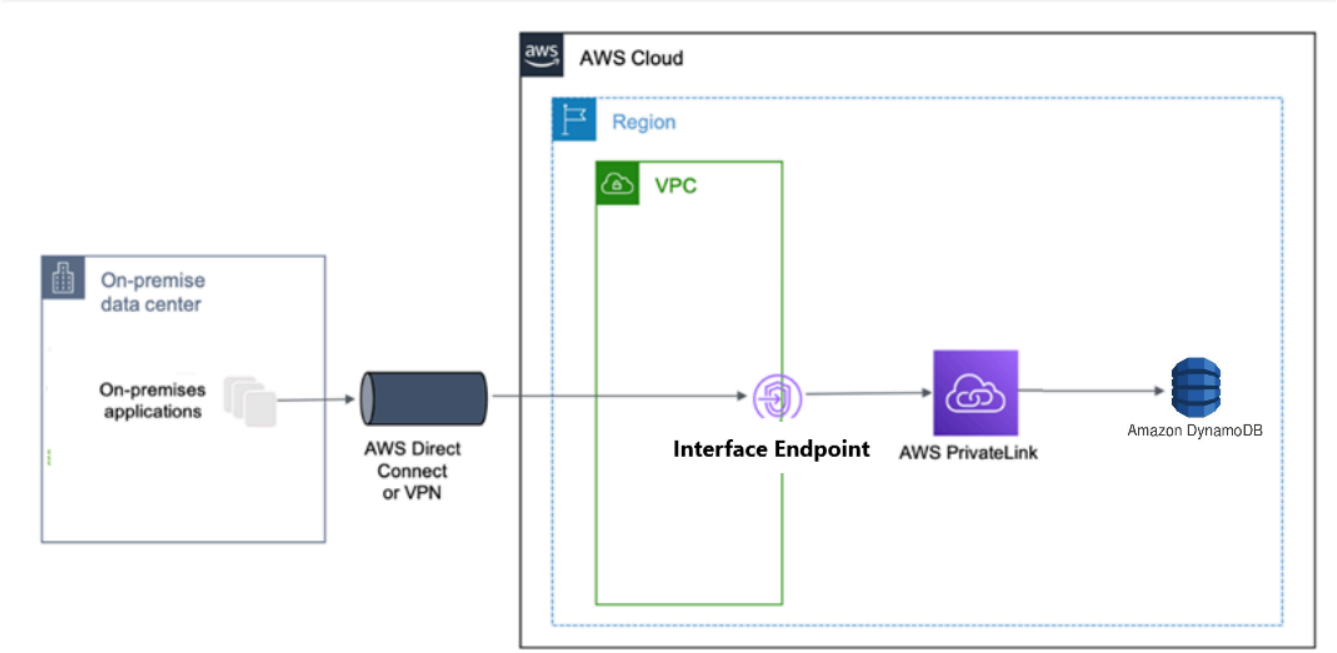


Diagram ini menggambarkan hal sebagai berikut:

- Jaringan lokal Anda menggunakan AWS Direct Connect atau menyambung AWS VPN ke Amazon VPC A.
- Aplikasi Anda lokal dan di Amazon VPC A menggunakan nama DNS khusus titik akhir untuk mengakses DynamoDB melalui titik akhir antarmuka DynamoDB.
- Aplikasi lokal mengirim data ke titik akhir antarmuka di Amazon VPC melalui AWS Direct Connect (atau). AWS VPN AWS PrivateLink memindahkan data dari titik akhir antarmuka ke DynamoDB melalui jaringan. AWS
- Aplikasi VPC di Amazon juga mengirim lalu lintas ke titik akhir antarmuka. AWS PrivateLink memindahkan data dari titik akhir antarmuka ke DynamoDB melalui jaringan. AWS

Menggunakan titik akhir gateway dan titik akhir antarmuka bersama-sama di Amazon VPC yang sama untuk mengakses DynamoDB

Anda dapat membuat titik akhir antarmuka dan mempertahankan titik akhir gateway yang ada di VPC Amazon yang sama, seperti yang ditunjukkan diagram berikut. Dengan mengambil pendekatan ini, Anda mengizinkan aplikasi VPC di Amazon untuk terus mengakses DynamoDB melalui titik akhir gateway, yang tidak ditagih. Kemudian, hanya aplikasi lokal Anda yang akan menggunakan titik akhir antarmuka untuk mengakses DynamoDB. Untuk mengakses DynamoDB dengan cara ini, Anda harus memperbarui aplikasi lokal Anda untuk menggunakan nama DNS khusus titik akhir untuk DynamoDB.

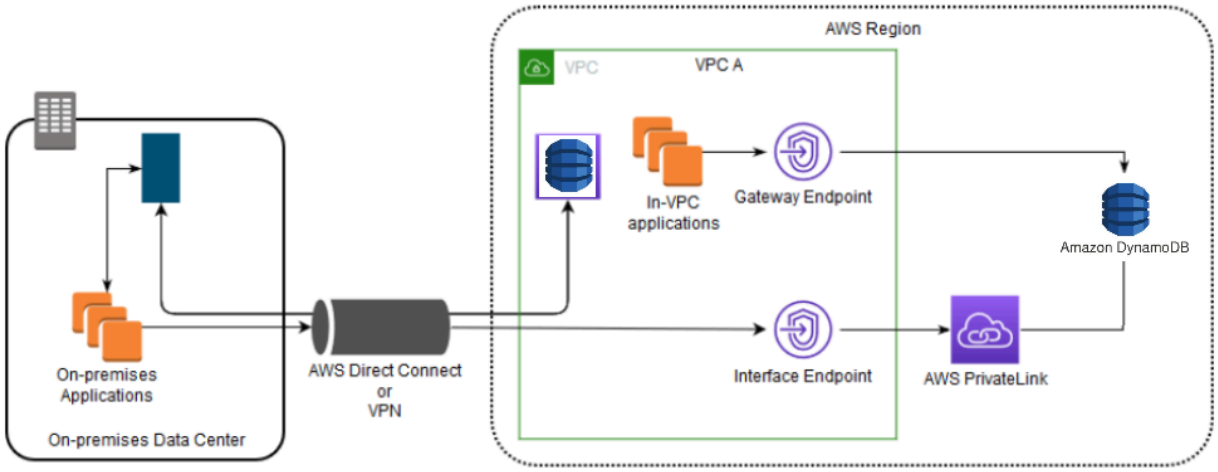


Diagram ini menggambarkan hal sebagai berikut:

- Aplikasi lokal menggunakan nama DNS khusus titik akhir untuk mengirim data ke titik akhir antarmuka dalam VPC Amazon melalui (atau). AWS Direct Connect AWS VPN AWS PrivateLink memindahkan data dari titik akhir antarmuka ke DynamoDB melalui jaringan. AWS
- Menggunakan nama DynamoDB Regional default, aplikasi VPC di Amazon mengirim data ke titik akhir gateway yang terhubung ke DynamoDB melalui jaringan. AWS

Untuk informasi selengkapnya tentang titik akhir gateway, lihat Gateway titik akhir VPC Amazon di Panduan Pengguna Amazon VPC.

Membuat kebijakan endpoint Amazon VPC untuk DynamoDB

Anda dapat melampirkan kebijakan endpoint ke endpoint Amazon VPC yang mengontrol akses ke DynamoDB. Kebijakan titik akhir menentukan informasi berikut:

- Prinsip AWS Identity and Access Management (IAM) yang dapat melakukan tindakan
- Tindakan-tindakan yang dapat dilakukan
- Sumber daya yang padanya tindakan dapat dilakukan

Topik

- Contoh: Membatasi akses ke tabel tertentu dari titik akhir VPC Amazon

Contoh: Membatasi akses ke tabel tertentu dari titik akhir VPC Amazon

Anda dapat membuat kebijakan endpoint yang membatasi akses hanya ke tabel DynamoDB tertentu. Jenis kebijakan ini berguna jika Anda memiliki kebijakan lain Layanan AWS di VPC Amazon Anda yang menggunakan tabel. Kebijakan tabel berikut membatasi akses hanya ke file. *DOC-EXAMPLE-TABLE* Untuk menggunakan kebijakan endpoint ini, ganti *DOC-EXAMPLE-TABLE* dengan nama tabel Anda.

```
{
"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:::DOC-EXAMPLE-TABLE",
                   "arn:aws:dynamodb:::DOC-EXAMPLE-TABLE/*"]
    }
  ]
}
```

AWS PrivateLink untuk DynamoDB Streams

Dengan AWS PrivateLink Amazon DynamoDB Streams, Anda dapat menyediakan antarmuka titik akhir Amazon VPC (titik akhir antarmuka) di cloud pribadi virtual Anda (Amazon VPC). Titik akhir ini dapat diakses langsung dari aplikasi yang ada di tempat melalui VPN dan AWS Direct Connect, atau berbeda dengan peering Wilayah AWS VPC Amazon. Menggunakan AWS PrivateLink dan

antarmuka endpoint, Anda dapat menyederhanakan konektivitas jaringan pribadi dari aplikasi Anda ke DynamoDB Streams.

Aplikasi di Amazon VPC Anda tidak memerlukan alamat IP publik untuk berkomunikasi dengan DynamoDB Streams menggunakan titik akhir antarmuka Amazon VPC untuk operasi DynamoDB Streams. Endpoint antarmuka diwakili oleh satu atau lebih antarmuka jaringan elastis (ENIs) yang diberi alamat IP pribadi dari subnet di VPC Amazon Anda. Permintaan ke DynamoDB Streams melalui titik akhir antarmuka tetap berada di jaringan Amazon. Anda juga dapat mengakses titik akhir antarmuka di VPC Amazon Anda dari aplikasi lokal AWS Direct Connect melalui AWS Virtual Private Network atau AWS (VPN). Untuk informasi selengkapnya tentang cara menghubungkan Anda AWS Virtual Private Network dengan jaringan lokal, lihat Panduan AWS Direct Connect Pengguna dan Panduan AWS Site-to-Site VPN Pengguna.

Untuk informasi umum tentang titik akhir antarmuka, lihat Antarmuka Amazon VPCAWS PrivateLink endpoint ().

> **Note**
>
> Hanya titik akhir antarmuka yang didukung untuk DynamoDB Streams. Titik akhir Gateway tidak didukung.

Topik

- Pertimbangan saat menggunakan AWS PrivateLink untuk Amazon DynamoDB Streams
- Membuat titik akhir Amazon VPC
- Mengakses titik akhir antarmuka Amazon DynamoDB Streams
- Mengakses operasi API DynamoDB Streams dari titik akhir antarmuka DynamoDB Streams
- AWS Contoh SDK
- Membuat kebijakan endpoint Amazon VPC untuk DynamoDB Streams

Pertimbangan saat menggunakan AWS PrivateLink untuk Amazon DynamoDB Streams

Pertimbangan Amazon VPC berlaku untuk Amazon AWS PrivateLink DynamoDB Streams. Untuk informasi selengkapnya, lihat pertimbangan dan AWS PrivateLink kuota titik akhir antarmuka. Pembatasan berikut berlaku.

AWS PrivateLink untuk Amazon DynamoDB Streams tidak mendukung hal berikut:

- Keamanan Lapisan Pengangkutan (TLS) 1.1
- Layanan Sistem Nama Domain Pribadi dan Hybrid (DNS)

> **Note**
>
> Batas waktu konektivitas jaringan ke AWS PrivateLink titik akhir tidak berada dalam lingkup respons kesalahan DynamoDB Streams dan perlu ditangani dengan tepat oleh aplikasi Anda yang terhubung ke titik akhir. AWS PrivateLink

Membuat titik akhir Amazon VPC

Untuk membuat titik akhir antarmuka VPC Amazon, lihat Membuat titik akhir VPC Amazon di Panduan.AWS PrivateLink

Mengakses titik akhir antarmuka Amazon DynamoDB Streams

Saat Anda membuat titik akhir antarmuka, DynamoDB menghasilkan dua jenis nama DNS DynamoDB Streams khusus titik akhir: Regional dan Zonal.

- Nama DNS Regional mencakup ID titik akhir VPC Amazon yang unik, pengenal layanan, Wilayah AWS dan namanya. `vpce.amazonaws.com` Misalnya, untuk ID titik akhir VPC Amazon*vpce-1a2b3c4d*, nama DNS yang dihasilkan mungkin mirip dengan. *vpce-1a2b3c4d-5e6f*`.streams.dynamodb.us-east-1.vpce.amazonaws.com`
- Nama DNS Zonal mencakup Zona Ketersediaan—misalnya, *vpce-1a2b3c4d-5e6f*`-us-east-1a.streams.dynamodb.us-east-1.vpce.amazonaws.com`. Anda dapat menggunakan opsi ini jika arsitektur Anda mengisolasi Zona Ketersediaan. Contoh, Anda bisa menggunakannya untuk kontainer kesalahan atau untuk mengurangi biaya transfer data Regional.

Mengakses operasi API DynamoDB Streams dari titik akhir antarmuka DynamoDB Streams

Anda dapat menggunakan AWS CLI atau AWS SDKs untuk mengakses operasi API DynamoDB Streams melalui titik akhir antarmuka DynamoDB Streams.

AWS CLI contoh

Untuk mengakses DynamoDB Streams atau operasi API melalui titik akhir antarmuka DynamoDB Streams dalam perintah, gunakan parameter dan. AWS CLI `--region` `--endpoint-url`

Contoh: Buat titik akhir VPC

```
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
```

Contoh: Memodifikasi titik akhir VPC

```
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
```

Contoh: Daftar aliran menggunakan URL endpoint

Dalam contoh berikut, ganti Region `us-east-1` dan nama DNS dari `vpce-1a2b3c4d-5e6f.streams.dynamodb.us-east-1.vpce.amazonaws.com` ID endpoint VPC dengan informasi Anda sendiri.

```
aws dynamodbstreams --region us-east-1 –endpoint https://
vpce-1a2b3c4d-5e6f.streams.dynamodb.us-east-1.vpce.amazonaws.com list-streams
```

AWS Contoh SDK

Untuk mengakses operasi API DynamoDB Streams Amazon melalui titik akhir antarmuka DynamoDB Streams saat menggunakan, perbarui ke versi terbaru. AWS SDKs SDKs Kemudian, konfigurasikan klien Anda untuk menggunakan URL endpoint untuk operasi DynamoDB Streams API melalui titik akhir antarmuka DynamoDB Streams.

SDK for Python (Boto3)

Contoh: Gunakan URL endpoint untuk mengakses aliran DynamoDB

Dalam contoh berikut, ganti Wilayah `us-east-1` dan ID titik akhir VPC `https://vpce-1a2b3c4d-5e6f.streams.dynamodb.us-east-1.vpce.amazonaws.com` dengan informasi Anda sendiri.

```
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

Contoh: Gunakan URL endpoint untuk mengakses aliran DynamoDB

Dalam contoh berikut, ganti Wilayah `us-east-1` dan ID titik akhir VPC `https://vpce-1a2b3c4d-5e6f.streams.dynamodb.us-east-1.vpce.amazonaws.com` dengan informasi Anda sendiri.

```
//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

Contoh: Gunakan URL endpoint untuk mengakses aliran DynamoDB

Dalam contoh berikut, ganti Wilayah `us-east-1` dan ID titik akhir VPC `https://vpce-1a2b3c4d-5e6f.streams.dynamodb.us-east-1.vpce.amazonaws.com` dengan informasi Anda sendiri.

```
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()
```

Membuat kebijakan endpoint Amazon VPC untuk DynamoDB Streams

Anda dapat melampirkan kebijakan endpoint ke endpoint Amazon VPC yang mengontrol akses ke DynamoDB Streams. Kebijakan titik akhir menentukan informasi berikut:

- Prinsip AWS Identity and Access Management (IAM) yang dapat melakukan tindakan
- Tindakan-tindakan yang dapat dilakukan
- Sumber daya yang padanya tindakan dapat dilakukan

Topik

- Contoh: Membatasi akses ke aliran tertentu dari titik akhir VPC Amazon

Contoh: Membatasi akses ke aliran tertentu dari titik akhir VPC Amazon

Anda dapat membuat kebijakan endpoint yang membatasi akses hanya ke DynamoDB Streams tertentu. Jenis kebijakan ini berguna jika Anda memiliki kebijakan lain Layanan AWS di Amazon VPC yang menggunakan DynamoDB Streams. Kebijakan aliran berikut membatasi akses hanya ke aliran yang *2025-02-20T11:22:33.444* dilampirkan*DOC-EXAMPLE-TABLE*. Untuk menggunakan kebijakan endpoint ini, ganti *DOC-EXAMPLE-TABLE* dengan nama tabel Anda dan *2025-02-20T11:22:33.444* dengan label stream.

```
{
"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:::DOC-EXAMPLE-TABLE/
stream/2025-02-20T11:22:33.444"]
```

```
    }
  ]
}
```

> **Note**
>
> Titik akhir Gateway tidak didukung di DynamoDB Streams.

Analisis konfigurasi dan kerentanan di Amazon DynamoDB

AWS menangani tugas-tugas keamanan dasar seperti sistem operasi tamu (OS) dan patching database, konfigurasi firewall, dan pemulihan bencana. Prosedur ini telah ditinjau dan disertifikasi oleh pihak ketiga yang sesuai. Untuk detail selengkapnya, lihat sumber daya berikut:

- Validasi kepatuhan untuk Amazon DynamoDB
- Model tanggung jawab bersama
- Amazon Web Services: Gambaran umum proses keamanan(laporan resmi)

Praktik terbaik keamanan berikut ini juga membahas analisis konfigurasi dan kerentanan di Amazon DynamoDB:

- Memantau kepatuhan DynamoDB dengan Aturan AWS Config
- Memantau konfigurasi DynamoDB dengan AWS Config

Praktik Terbaik Keamanan untuk Amazon DynamoDB

Amazon DynamoDB menyediakan sejumlah fitur keamanan untuk dipertimbangkan ketika Anda mengembangkan dan menerapkan kebijakan keamanan Anda sendiri. Praktik terbaik berikut adalah pedoman umum dan tidak mewakili solusi keamanan yang lengkap. Karena praktik terbaik ini mungkin tidak sesuai atau tidak memadai untuk lingkungan Anda, perlakukan itu sebagai pertimbangan yang bermanfaat, bukan sebagai resep.

Topik

- Praktik terbaik keamanan pencegahan di DynamoDB
- Praktik terbaik keamanan detektif DynamoDB

Praktik terbaik keamanan pencegahan di DynamoDB

Praktik terbaik berikut dapat membantu Anda mengantisipasi dan mencegah insiden keamanan di Amazon DynamoDB.

Enkripsi saat diam

DynamoDB mengenkripsi semua data pengguna saat diam yang disimpan dalam tabel, indeks, aliran, dan cadangan menggunakan kunci enkripsi yang disimpan dalam AWS Key Management Service (AWS KMS). Hal ini memberi lapisan perlindungan data tambahan dengan mengamankan data Anda dari akses yang tidak sah ke penyimpanan dasar.

Anda dapat menentukan apakah DynamoDB harus menggunakan Kunci milik AWS (tipe enkripsi default), atau kunci Kunci yang dikelola AWS yang dikelola pelanggan untuk mengenkripsi data pengguna. Untuk informasi selengkapnya, lihat Enkripsi Amazon DynamoDB saat Diam.

Menggunakan peran IAM untuk mengautentikasi akses ke DynamoDB

Untuk pengguna, aplikasi, dan AWS layanan lain untuk mengakses DynamoDB, mereka harus menyertakan kredensyal yang AWS valid dalam permintaan API mereka. AWS Anda tidak boleh menyimpan AWS kredensyal secara langsung di aplikasi atau EC2 instance. Ini adalah kredensial jangka panjang yang tidak dirotasi secara otomatis, dan karenanya dapat menimbulkan dampak bisnis yang signifikan jika dibobol. Peran IAM memungkinkan Anda memperoleh kunci akses sementara yang dapat digunakan untuk mengakses AWS layanan dan sumber daya.

Untuk informasi selengkapnya, lihat Manajemen Identitas dan Akses untuk Amazon DynamoDB.

Menggunakan kebijakan IAM untuk otorisasi dasar DynamoDB

Saat memberikan izin, Anda memutuskan siapa yang mendapatkannya, DynamoDB mana APIs mereka mendapatkan izin, dan tindakan spesifik yang ingin Anda izinkan pada sumber daya tersebut. Menerapkan akses hak akses paling rendah adalah hal mendasar dalam mengurangi risiko keamanan dan dampak yang dapat disebabkan oleh kesalahan atau niat jahat.

Memberikan kebijakan izin kepada identitas IAM (yaitu, pengguna, grup, dan peran) dan dengan demikian memberikan izin untuk melakukan operasi pada sumber daya DynamoDB.

Anda dapat melakukan hal ini dengan cara berikut:

- AWS Kebijakan terkelola (telah ditentukan)
- Kebijakan yang dikelola pelanggan

Menggunakan ketentuan kebijakan IAM untuk kontrol akses ketat

Ketika memberikan izin di DynamoDB, Anda dapat menetapkan syarat yang menentukan bagaimana kebijakan izin berlaku. Menerapkan akses hak akses paling rendah adalah hal mendasar dalam mengurangi risiko keamanan dan dampak yang dapat disebabkan oleh kesalahan atau niat jahat.

Anda dapat menetapkan syarat saat memberikan izin menggunakan kebijakan IAM. Misalnya, Anda dapat melakukan hal berikut:

- Memberi izin kepada pengguna akan akses hanya-baca pada item dan atribut tertentu dalam tabel atau indeks sekunder.
- Memberi izin kepada pengguna akan akses hanya-tulis pada atribut tertentu dalam tabel, berdasarkan identitas pengguna tersebut.

Untuk informasi selengkapnya, lihat Menggunakan Syarat Kebijakan IAM untuk Kontrol Akses Ketat.

Menggunakan kebijakan dan titik akhir VPC untuk mengakses DynamoDB

Jika Anda hanya memerlukan akses ke DynamoDB dari dalam cloud privat virtual (VPC), Anda harus menggunakan titik akhir VPC untuk membatasi akses hanya dari VPC yang diperlukan. Melakukan hal ini akan mencegah lalu lintas melintasi internet terbuka dan tunduk pada lingkungan tersebut.

Menggunakan titik akhir VPC untuk DynamoDB akan memungkinkan Anda untuk mengontrol dan membatasi akses menggunakan hal berikut:

- Kebijakan titik akhir VPC – Kebijakan ini diterapkan pada titik akhir VPC DynamoDB. Kebijakan tersebut memungkinkan Anda untuk mengontrol dan membatasi akses API ke tabel DynamoDB.
- Kebijakan IAM – Dengan menggunakan syarat `aws:sourceVpce` pada kebijakan yang diberikan kepada pengguna, grup, atau peran, Anda dapat memberlakukan bahwa semua akses ke tabel DynamoDB adalah melalui titik akhir VPC tertentu.

Untuk informasi selengkapnya, lihat Titik Akhir untuk Amazon DynamoDB.

Pertimbangkan enkripsi di sisi klien

Kami menyarankan Anda merencanakan strategi enkripsi Anda sebelum menerapkan tabel Anda di DynamoDB. Jika Anda menyimpan data sensitif atau rahasia di DynamoDB, pertimbangkan

untuk menyertakan enkripsi di sisi klien dalam paket Anda. Dengan cara ini, Anda dapat mengenkripsi data sedekat mungkin dengan asalnya, dan memastikan perlindungannya sepanjang siklus hidupnya. Mengenkripsi data bergerak dan data diam Anda yang sensitif membantu memastikan bahwa data plaintext Anda tidak tersedia untuk pihak ketiga mana pun.

AWS Database Encryption SDK untuk DynamoDB adalah pustaka perangkat lunak yang membantu Anda melindungi data tabel Anda sebelum mengirimkannya ke DynamoDB. Pustaka ini mengenkripsi, menandatangani, memverifikasi, dan mendekripsi item tabel DynamoDB Anda. Anda mengontrol atribut mana yang dienkripsi dan ditandatangani.

Pertimbangan Utama Utama

Jangan gunakan nama sensitif atau data plaintext sensitif di Kunci Utama untuk tabel dan Indeks Sekunder Global. Nama kunci akan muncul dalam definisi tabel Anda. Misalnya, nama Kunci Utama dapat diakses oleh siapa saja yang memiliki izin untuk menelepon DescribeTable. Nilai kunci dapat muncul di log Anda AWS CloudTraildan lainnya. Selain itu, DynamoDB menggunakan nilai-nilai kunci untuk mendistribusikan data dan permintaan rute AWS dan administrator dapat mengamati nilai-nilai untuk menjaga kesehatan layanan.

Jika Anda perlu menggunakan data sensitif dalam tabel atau nilai kunci GSI, sebaiknya gunakan enkripsi end-to-end klien. Ini memungkinkan Anda untuk melakukan referensi nilai kunci ke data Anda sambil memastikan bahwa itu tidak pernah muncul tidak terenkripsi di log terkait DynamoDB Anda. Salah satu cara untuk mencapai ini adalah dengan menggunakan AWS Database Encryption SDK untuk DynamoDB, tetapi itu tidak diperlukan. Jika Anda menggunakan solusi Anda sendiri, kami harus selalu menggunakan algoritma enkripsi yang cukup aman. Anda tidak boleh menggunakan opsi non-kriptografi seperti hash, karena mereka tidak dianggap cukup aman dalam kebanyakan situasi.

Jika nama kunci Primary Key Anda sensitif, sebaiknya gunakan `pk` dan `sk` sebagai gantinya. Ini adalah praktik terbaik umum yang membuat desain Partition Key Anda fleksibel.

Selalu konsultasikan dengan pakar keamanan atau tim AWS akun Anda jika Anda khawatir tentang pilihan yang tepat.

Praktik terbaik keamanan detektif DynamoDB

Praktik terbaik berikut untuk Amazon DynamoDB dapat membantu mendeteksi potensi kelemahan dan insiden keamanan.

Gunakan AWS CloudTrail untuk memantau penggunaan kunci KMS AWS terkelola

Jika Anda menggunakan enkripsi Kunci yang dikelola AWSfor saat istirahat, penggunaan kunci ini masuk AWS CloudTrail. CloudTrail memberikan visibilitas ke aktivitas pengguna dengan merekam tindakan yang diambil pada akun Anda. CloudTrail mencatat informasi penting tentang setiap tindakan, termasuk siapa yang membuat permintaan, layanan yang digunakan, tindakan yang dilakukan, parameter untuk tindakan, dan elemen respons yang dikembalikan oleh AWS layanan. Informasi ini membantu Anda melacak perubahan yang dibuat pada AWS sumber daya Anda dan memecahkan masalah operasional. CloudTrail membuatnya lebih mudah untuk memastikan kepatuhan terhadap kebijakan internal dan standar peraturan.

Anda dapat menggunakan CloudTrail untuk mengaudit penggunaan kunci. CloudTrail membuat file log yang berisi riwayat panggilan AWS API dan peristiwa terkait untuk akun Anda. File log ini mencakup semua permintaan AWS KMS API yang dibuat menggunakan AWS Management Console AWS SDKs,, dan alat baris perintah, selain yang dibuat melalui AWS layanan terintegrasi. Anda dapat menggunakan file log ini untuk mendapatkan informasi tentang waktu penggunaan kunci KMS, operasi yang diminta, identitas pemohon, alamat IP asal permintaan, dan seterusnya. Untuk informasi selengkapnya, lihat Pencatatan Panggilan API AWS KMS dengan AWS CloudTrail dan Panduan Pengguna AWS CloudTrail.

Memantau operasi DynamoDB menggunakan CloudTrail

CloudTrail dapat memantau peristiwa bidang kontrol dan peristiwa bidang data. Operasi bidang kontrol memungkinkan Anda membuat dan mengelola tabel DynamoDB. Operasi ini juga memungkinkan Anda bekerja dengan indeks, aliran, dan objek lain yang tergantung pada tabel. Operasi bidang data memungkinkan Anda membuat, membaca, memperbarui, dan menghapus tindakan (juga disebut CRUD) pada data dalam tabel. Beberapa operasi bidang data juga memungkinkan Anda membaca data dari indeks sekunder. Untuk mengaktifkan pencatatan peristiwa bidang data di CloudTrail, Anda harus mengaktifkan pencatatan aktivitas API bidang data CloudTrail. Lihat Pencatatan log peristiwa data untuk jejak untuk informasi selengkapnya.

Ketika aktivitas terjadi di DynamoDB, aktivitas tersebut direkam dalam CloudTrail suatu peristiwa bersama dengan peristiwa layanan AWS lainnya dalam riwayat peristiwa. Untuk informasi selengkapnya, lihat Pencatatan Operasi DynamoDB Menggunakan AWS CloudTrail. Anda dapat melihat, mencari, dan mengunduh acara terbaru di AWS akun Anda. Untuk informasi selengkapnya, lihat Melihat CloudTrail Acara dengan Riwayat Acara di Panduan AWS CloudTrail Pengguna.

Untuk catatan peristiwa yang sedang berlangsung di AWS akun Anda, termasuk acara untuk DynamoDB, buat jejak. Jejak memungkinkan CloudTrail untuk mengirimkan file log ke bucket

Amazon Simple Storage Service (Amazon S3). Secara default, saat Anda membuat jejak di konsol, jejak tersebut berlaku untuk semua AWS Wilayah. Jejak mencatat peristiwa dari semua Wilayah di partisi AWS dan mengirimkan file log ke bucket S3 yang Anda tentukan. Selain itu, Anda dapat mengonfigurasi AWS layanan lain untuk menganalisis lebih lanjut dan menindaklanjuti data peristiwa yang dikumpulkan dalam CloudTrail log.

Menggunakan DynamoDB Streams untuk memantau operasi bidang data

DynamoDB terintegrasi AWS Lambda dengan sehingga Anda dapat membuat pemicu—potongan kode yang secara otomatis merespons peristiwa di DynamoDB Streams. Dengan pemicu, Anda dapat membangun aplikasi yang bereaksi terhadap modifikasi data di tabel DynamoDB.

Jika Anda mengaktifkan DynamoDB Streams pada tabel, Anda dapat mengaitkan aliran Amazon Resource Name (ARN) dengan fungsi Lambda yang Anda tulis. Segera setelah item dalam tabel diubah, catatan baru muncul di aliran tabel. AWS Lambda polling aliran dan memanggil fungsi Lambda Anda secara serempak saat mendeteksi catatan aliran baru. Fungsi Lambda dapat melakukan tindakan apa pun yang Anda tentukan, seperti mengirim notifikasi atau memulai alur kerja.

Sebagai contoh, lihat Tutorial: Menggunakan AWS Lambda Dengan Amazon DynamoDB Streams. Contoh ini menerima input peristiwa DynamoDB, memproses pesan yang dikandungnya, dan menulis beberapa data peristiwa yang masuk ke Amazon Logs. CloudWatch

Memantau konfigurasi DynamoDB dengan AWS Config

Anda dapat terus memantau dan merekam perubahan konfigurasi pada sumber daya AWS Anda menggunakan AWS Config. Anda juga dapat menggunakan AWS Config inventaris AWS sumber daya Anda. Ketika terdeteksi perubahan dari keadaan sebelumnya, pemberitahuan Amazon Simple Notification Service (Amazon SNS) dapat dikirimkan untuk Anda guna meninjau dan mengambil tindakan. Ikuti panduan dalam Menyiapkan AWS Config dengan Konsol, memastikan bahwa jenis sumber daya DynamoDB disertakan.

Anda dapat mengonfigurasi AWS Config untuk mengalirkan perubahan konfigurasi dan pemberitahuan ke topik Amazon SNS. Misalnya, ketika sumber daya diperbarui, Anda bisa mendapatkan pemberitahuan yang dikirim ke email Anda, sehingga Anda dapat melihat perubahan tersebut. Anda juga dapat diberi tahu saat AWS Config mengevaluasi aturan kustom atau terkelola terhadap sumber daya Anda.

Sebagai contoh, lihat Pemberitahuan yang AWS Config Mengirim ke topik Amazon SNS di Panduan AWS Config Pengembang.

Pantau kepatuhan DynamoDB dengan aturan AWS Config

AWS Config terus melacak perubahan konfigurasi yang terjadi di antara sumber daya Anda. Hal itu memeriksa apakah perubahan ini melanggar salah satu syarat dalam aturan Anda. Jika sumber daya melanggar aturan, AWS Config tandai sumber daya dan aturan sebagai tidak patuh.

Dengan menggunakan AWS Config untuk mengevaluasi konfigurasi sumber daya Anda, Anda dapat menilai seberapa baik konfigurasi sumber daya Anda mematuhi praktik internal, pedoman industri, dan peraturan. AWS Config menyediakan aturan AWS terkelola, yang merupakan aturan yang telah ditentukan sebelumnya dan dapat disesuaikan yang AWS Config digunakan untuk mengevaluasi apakah AWS sumber daya Anda mematuhi praktik terbaik umum.

Tandai sumber daya DynamoDB Anda untuk identifikasi dan otomatisasi

Anda dapat menetapkan metadata ke AWS sumber daya Anda dalam bentuk tag. Setiap tanda adalah label sederhana yang terdiri dari kunci yang ditetapkan pelanggan dan nilai opsional yang memudahkan untuk mengelola, mencari, dan memfilter sumber daya.

Penandaan memungkinkan implementasi kontrol berkelompok. Meskipun tidak ada jenis tanda yang melekat, tanda memungkinkan Anda untuk mengelompokkan sumber daya berdasarkan tujuan, pemilik, lingkungan, atau kriteria lainnya. Berikut ini beberapa contohnya:

- Keamanan – Digunakan untuk menentukan persyaratan seperti enkripsi.
- Kerahasiaan – Sebuah pengidentifikasi untuk dukungan sumber daya tingkat kerahasiaan data tertentu.
- Lingkungan – Digunakan untuk membedakan antara pengembangan, pengujian, dan infrastruktur produksi.

Untuk informasi selengkapnya, lihat Strategi Penandaan AWS dan Penandaan untuk DynamoDB.

Pantau penggunaan Amazon DynamoDB karena berkaitan dengan praktik terbaik keamanan dengan menggunakan. AWS Security Hub

Hub Keamanan menggunakan kontrol keamanan untuk mengevaluasi konfigurasi sumber daya dan standar keamanan guna membantu Anda mematuhi berbagai kerangka kerja kepatuhan.

Untuk informasi selengkapnya tentang menggunakan Hub Keamanan guna mengevaluasi sumber daya DynamoDB, lihat kontrol Amazon DynamoDB di Panduan Pengguna AWS Security Hub .

Memantau dan masuk ke DynamoDB

Pemantauan adalah bagian penting dari menjaga keandalan, ketersediaan, dan kinerja DynamoDB dan solusi Anda. AWS Anda harus mengumpulkan data pemantauan dari semua bagian AWS solusi Anda sehingga Anda dapat dengan mudah men-debug kegagalan multi-titik.

Topik

- Rencana pemantauan
- Acuan dasar performa
- Layanan terintegrasi
- Alat-alat pemantauan otomatis
- Memantau metrik di DynamoDB dengan Amazon CloudWatch
- Pencatatan log operasi DynamoDB menggunakan AWS CloudTrail
- Menganalisis akses data menggunakan wawasan CloudWatch kontributor untuk DynamoDB

Rencana pemantauan

Sebelum Anda mulai memantau DynamoDB, buat rencana pemantauan yang mencakup jawaban atas pertanyaan-pertanyaan berikut:

- Apa saja sasaran pemantauan Anda?
- Sumber daya apa yang akan Anda pantau?
- Seberapa sering Anda akan memantau sumber daya ini?
- Alat pemantauan apa yang akan Anda gunakan?
- Siapa yang akan melakukan tugas pemantauan?
- Siapa yang harus diberi tahu saat terjadi kesalahan?

Acuan dasar performa

Tetapkan dasar untuk kinerja DynamoDB normal di lingkungan Anda, dengan mengukur kinerja pada berbagai waktu dan dalam kondisi beban yang berbeda. Saat memantau DynamoDB, Anda harus mempertimbangkan untuk menyimpan data pemantauan historis. Data yang disimpan ini akan memberi Anda dasar untuk membandingkan data performa saat ini, mengidentifikasi pola performa

normal dan anomali performa, dan merancang metode untuk mengatasi masalah. Untuk menetapkan baseline, Anda harus, setidaknya, menggunakan item berikut:

- Jumlah unit kapasitas baca atau tulis yang digunakan selama periode waktu tertentu, sehingga Anda dapat melacak jumlah penggunaan throughput yang Anda sediakan.
- Permintaan yang melampaui kapasitas baca atau tulis yang disediakan pada tabel selama jangka waktu tertentu, sehingga Anda dapat menentukan permintaan mana yang melampaui kuota throughput yang disediakan pada tabel.
- Kesalahan sistem, sehingga Anda dapat menentukan apakah ada permintaan yang menghasilkan kesalahan.

Layanan terintegrasi

DynamoDB secara otomatis memonitor tabel Anda atas nama Anda dan melaporkan metrik melalui Amazon. CloudWatch Selain itu, DynamoDB terintegrasi dengan yang Layanan AWS berikut ini untuk membantu Anda memantau dan memecahkan masalah sumber daya DynamoDB Anda.

- AWS CloudTrail menangkap panggilan API dan peristiwa terkait yang dibuat oleh atau atas nama Anda Akun AWS dan mengirimkan file log ke bucket Amazon S3 yang Anda tentukan. Untuk informasi selengkapnya, lihat Pencatatan log operasi DynamoDB menggunakan AWS CloudTrail.
- Contributor Insights adalah alat diagnostik untuk mengidentifikasi kunci yang paling sering diakses dan dibatasi dalam tabel atau indeks Anda secara sekilas. Untuk informasi selengkapnya, lihat Menganalisis akses data menggunakan wawasan CloudWatch kontributor untuk DynamoDB.

Alat-alat pemantauan otomatis

AWS menyediakan berbagai alat yang dapat Anda gunakan untuk memantau DynamoDB. Sebaiknya Anda mengotomatisasi tugas pemantauan sebanyak mungkin. Anda dapat menggunakan alat pemantauan otomatis berikut untuk memantau DynamoDB dan melapor saat terjadi masalah:

- CloudWatch Alarm Amazon — Tonton satu metrik selama periode waktu yang Anda tentukan, dan lakukan satu atau beberapa tindakan berdasarkan nilai metrik relatif terhadap ambang batas tertentu selama beberapa periode waktu.

  Tindakannya adalah pemberitahuan yang dikirim ke topik Amazon Simple Notification Service (Amazon SNS) atau kebijakan Amazon Auto EC2 Scaling. CloudWatch Alarm Amazon tidak

memanggil tindakan hanya karena mereka berada dalam keadaan tertentu; negara harus telah berubah dan dipertahankan untuk sejumlah periode tertentu. Untuk informasi selengkapnya, lihat Memantau metrik di DynamoDB dengan Amazon CloudWatch.

- AWS CloudTrail Pemantauan log - Bagikan file log antar akun, pantau file AWS CloudTrail log secara real time dengan mengirimkannya ke AWS CloudTrail Log, menulis aplikasi pemrosesan log di Java, dan validasi bahwa file log Anda tidak berubah setelah pengiriman oleh AWS CloudTrail. Untuk informasi selengkapnya, lihat Apa itu CloudWatch Log Amazon di Panduan AWS CloudTrail Pengguna.

Memantau metrik di DynamoDB dengan Amazon CloudWatch

Anda dapat memantau DynamoDB CloudWatch menggunakan, yang mengumpulkan dan memproses data mentah dari DynamoDB menjadi metrik yang dapat dibaca, mendekati real-time. Statistik ini disimpan untuk jangka waktu tertentu, sehingga Anda dapat mengakses informasi historis untuk perspektif yang lebih baik tentang kinerja aplikasi atau layanan web Anda. Secara default, data metrik DynamoDB dikirim secara otomatis. CloudWatch Untuk informasi selengkapnya, lihat Apa itu Amazon CloudWatch? dan Retensi metrik di Panduan CloudWatch Pengguna Amazon.

Topik

- Bagaimana cara menggunakan metrik DynamoDB?
- Melihat metrik di konsol CloudWatch
- Melihat metrik di AWS CLI
- Dimensi dan Metrik DynamoDB
- Membuat CloudWatch alarm di DynamoDB

Bagaimana cara menggunakan metrik DynamoDB?

Metrik yang dilaporkan oleh DynamoDB memberikan informasi yang dapat Anda analisis dengan berbagai cara. Daftar berikut menunjukkan beberapa penggunaan umum untuk metrik. Daftar ini berisi saran agar Anda dapat memulai, bukan daftar komprehensif.

Bagaimana cara menggunakan metrik DynamoDB?

Bagaimana saya dapat melakukannya?	Metrik terkait
Bagaimana saya bisa memantau tingkat penghapusan TTL di meja saya?	Anda dapat memantau `TimeToLiveDeletedItemCount` selama periode waktu yang ditentukan, untuk melacak tingkat penghapusan TTL di tabel Anda. Untuk contoh aplikasi tanpa server yang menggunakan `TimeToLiveDeletedItemCount` metrik, lihat Mengarsipkan item secara otomatis ke S3 menggunakan DynamoDB time to live (TTL) with dan Amazon Data Firehose. AWS Lambda
Bagaimana saya bisa menentukan berapa banyak throughput yang disediakan saya digunakan?	Anda dapat memantau `ConsumedReadCapacityUnits` atau `ConsumedWriteCapacityUnits` selama periode waktu yang ditentukan, untuk melacak jumlah throughput yang disediakan yang sedang digunakan.
Bagaimana cara menentukan permintaan mana yang melebihi kuota throughput yang disediakan dari sebuah tabel?	`ThrottledRequests` bertambah satu jika ada peristiwa dalam permintaan yang melampaui kuota throughput yang disediakan. Untuk mendapatkan wawasan tentang peristiwa mana yang melakukan throttling terhadap permintaan, bandingkan `ThrottledRequests` dengan metrik `ReadThrottleEvents` dan `WriteThrottleEvents` untuk tabel dan indeksnya.
Bagaimana saya bisa menentukan apakah ada kesalahan sistem yang terjadi?	Anda dapat memantau `SystemErrors` untuk menentukan apakah setiap permintaan menghasilkan kode HTTP 500 (kesalahan server). Biasanya, metrik ini sama dengan nol. Jika tidak, Anda mungkin harus menyelidikinya.
Bagaimana saya bisa memantau nilai latensi untuk operasi tabel saya?	Anda dapat memantau `SuccessfulRequestLatency` dengan melacak latensi rata-rata dan latensi median melalui metrik persentil (p50). Lonjakan latensi sesekali tidak perlu dikhawatirkan. Namun, jika latensi rata-rata atau p50 (median) tinggi, mungkin ada masalah mendasar yang harus Anda selesaikan. Untuk informasi selengkapnya, lihat Memecahkan masalah latensi di Amazon DynamoDB.

Melihat metrik di konsol CloudWatch

Metrik dikelompokkan berdasarkan namespace layanan terlebih dahulu dan kemudian oleh berbagai kombinasi dimensi dalam setiap namespace.

Untuk melihat metrik di konsol CloudWatch

1. Buka CloudWatch konsol di https://console.aws.amazon.com/cloudwatch/.
2. Di panel navigasi, pilih Metrik, Semua metrik.
3. Pilih namespace DynamoDB. Anda juga dapat memilih namespace Penggunaan untuk melihat metrik penggunaan DynamoDB. Untuk informasi selengkapnya tentang metrik penggunaan, lihat metrik penggunaan AWS.
4. Tab Browse menampilkan semua metrik di namespace.
5. (Opsional) Untuk menambahkan grafik metrik ke CloudWatch dasbor, pilih Tindakan, Tambahkan ke dasbor.

Melihat metrik di AWS CLI

Untuk mendapatkan informasi metrik menggunakan AWS CLI, gunakan CloudWatch perintah`list-metrics`. Dalam contoh berikut, Anda mencantumkan semua metrik di namespace `AWS/DynamoDB`.

```
aws cloudwatch list-metrics --namespace "AWS/DynamoDB"
```

Untuk mendapatkan statistik metrik, gunakan perintah `get-metric-statistics`. Perintah berikut mendapatkan `ConsumedReadCapacityUnits` statistik untuk tabel `ProductCatalog` selama periode 24 jam tertentu, dengan perincian 5 menit.

```
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
```

Contoh output muncul sebagai berikut:

```
{
    "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 "
}
```


Dimensi dan Metrik DynamoDB

Ketika Anda berinteraksi dengan DynamoDB, ia mengirimkan metrik dan dimensi ke. CloudWatch

Output DynamoDB menggunakan throughput yang disediakan untuk periode satu menit. Penskalaan otomatis terpicu ketika kapasitas yang Anda konsumsi melanggar pemanfaatan target yang dikonfigurasi selama dua menit berturut-turut. CloudWatch alarm mungkin memiliki penundaan singkat hingga beberapa menit sebelum memicu penskalaan otomatis. Penundaan ini memastikan evaluasi CloudWatch metrik yang akurat. Jika lonjakan throughput yang dikonsumsi terpisah lebih dari satu menit, penskalaan otomatis mungkin tidak terpicu. Demikian pula, peristiwa penurunan skala dapat terjadi ketika 15 titik data berturut-turut lebih rendah dari pemanfaatan target. Dalam kedua kasus tersebut, setelah pemicu penskalaan otomatis, UpdateTableAPI dipanggil. Kemudian dibutuhkan beberapa menit untuk memperbarui kapasitas yang disediakan untuk tabel atau indeks. Selama periode ini, setiap permintaan yang melebihi kapasitas tabel yang disediakan sebelumnya akan dibatasi.

Melihat metrik dan dimensi

CloudWatch menampilkan metrik berikut untuk DynamoDB:

Metrik DynamoDB

> **Note**
>
> Amazon CloudWatch menggabungkan metrik ini pada interval satu menit:
>
> - `ConditionalCheckFailedRequests`
> - `ConsumedReadCapacityUnits`
> - `ConsumedWriteCapacityUnits`
> - `ReadThrottleEvents`
> - `ReturnedBytes`
> - `ReturnedItemCount`
> - `ReturnedRecordsCount`
> - `SuccessfulRequestLatency`
> - `SystemErrors`
> - `TimeToLiveDeletedItemCount`
> - `ThrottledRequests`
> - `TransactionConflict`
> - `UserErrors`
> - `WriteThrottleEvents`
>
> Untuk semua metrik DynamoDB lainnya, granularitas agregasi adalah lima menit.

Tidak semua statistik, seperti Average atau Sum, berlaku untuk setiap metrik. Namun, semua nilai ini tersedia melalui konsol Amazon DynamoDB, atau dengan menggunakan konsol,, atau CloudWatch untuk semua AWS CLI metrik AWS SDKs .

Dalam daftar berikut, setiap metrik memiliki serangkaian statistik valid yang berlaku metrik tersebut.

Daftar Metrik yang Tersedia

- AccountMaxReads

- AccountMaxTableLevelReads
- AccountMaxTableLevelWrites
- AccountMaxWrites
- AccountProvisionedReadCapacityUtilization
- AccountProvisionedWriteCapacityUtilization
- AgeOfOldestUnreplicatedRecord
- ConditionalCheckFailedRequests
- ConsumedChangeDataCaptureUnits
- ConsumedReadCapacityUnits
- ConsumedWriteCapacityUnits
- FailedToReplicateRecordCount
- MaxProvisionedTableReadCapacityUtilization
- MaxProvisionedTableWriteCapacityUtilization
- OnDemandMaxReadRequestUnits
- OnDemandMaxWriteRequestUnits
- OnlineIndexConsumedWriteCapacity
- OnlineIndexPercentageProgress
- OnlineIndexThrottleEvents
- PendingReplicationCount
- ProvisionedReadCapacityUnits
- ProvisionedWriteCapacityUnits
- ReadThrottleEvents
- ReplicationLatency
- ReturnedBytes
- ReturnedItemCount
- ReturnedRecordsCount
- SuccessfulRequestLatency
- SystemErrors
- TimeToLiveDeletedItemCount
- ThrottledPutRecordCount

- ThrottledRequests
- TransactionConflict
- UserErrors
- WriteThrottleEvents

AccountMaxReads

Jumlah maksimum unit kapasitas baca yang dapat digunakan oleh suatu akun. Batas ini tidak berlaku untuk tabel sesuai permintaan atau indeks sekunder global.

Unit: `Count`

Statistik Valid:

- `Maximum` – Jumlah maksimum unit kapasitas baca yang dapat digunakan oleh suatu akun.

AccountMaxTableLevelReads

Jumlah maksimum unit kapasitas baca yang dapat digunakan oleh tabel atau indeks sekunder global suatu akun. Untuk tabel sesuai permintaan, batas ini membatasi unit permintaan baca maksimum yang dapat digunakan tabel atau indeks sekunder global.

Unit: `Count`

Statistik Valid:

- `Maximum` – Jumlah maksimum unit kapasitas baca yang dapat digunakan oleh tabel atau indeks sekunder global suatu akun.

AccountMaxTableLevelWrites

Jumlah maksimum unit kapasitas tulis yang dapat digunakan oleh tabel atau indeks sekunder global suatu akun. Untuk tabel sesuai permintaan, batas ini membatasi unit permintaan tulis maksimum yang dapat digunakan tabel atau indeks sekunder global.

Unit: `Count`

Statistik Valid:

- `Maximum` – Jumlah maksimum unit kapasitas tulis yang dapat digunakan oleh tabel atau indeks sekunder global suatu akun.

AccountMaxWrites

Jumlah maksimum unit kapasitas tulis yang dapat digunakan oleh suatu akun. Batas ini tidak berlaku untuk tabel sesuai permintaan atau indeks sekunder global.

Unit: `Count`

Statistik Valid:

- `Maximum` – Jumlah maksimum unit kapasitas tulis yang dapat digunakan oleh suatu akun.

AccountProvisionedReadCapacityUtilization

Persentase unit kapasitas baca yang disediakan dan digunakan oleh suatu akun.

Unit: `Percent`

Statistik Valid:

- `Maximum` – Persentase maksimum unit kapasitas baca yang disediakan dan digunakan oleh suatu akun.
- `Minimum` – Persentase minimum unit kapasitas baca yang disediakan dan digunakan oleh suatu akun.
- `Average` – Persentase rata-rata unit kapasitas baca yang disediakan dan digunakan oleh suatu akun. Metrik diterbitkan selama interval lima menit. Oleh karena itu, jika Anda dengan cepat menyesuaikan unit kapasitas baca yang disediakan, statistik ini mungkin tidak mencerminkan rata-rata sebenarnya.

AccountProvisionedWriteCapacityUtilization

Persentase unit kapasitas tulis yang disediakan dan digunakan oleh suatu akun.

Unit: `Percent`

Statistik Valid:

- `Maximum` – Persentase maksimum unit kapasitas tulis yang disediakan dan digunakan oleh suatu akun.
- `Minimum` – Persentase minimum unit kapasitas tulis yang disediakan dan digunakan oleh suatu akun.
- `Average` – Persentase rata-rata unit kapasitas tulis yang disediakan dan digunakan oleh suatu akun. Metrik diterbitkan selama interval lima menit. Oleh karena itu, jika Anda dengan cepat menyesuaikan unit kapasitas tulis yang disediakan, statistik ini mungkin tidak mencerminkan rata-rata sebenarnya.

AgeOfOldestUnreplicatedRecord

Waktu yang berlalu sejak catatan akan direplikasi ke Kinesis data stream yang muncul pertama kali di tabel DynamoDB.

Unit: `Milliseconds`

Dimensi: `TableName, DelegatedOperation`

Statistik Valid:

- `Maximum`.
- `Minimum`.
- `Average`.

ConditionalCheckFailedRequests

Jumlah upaya gagal untuk melakukan penulisan bersyarat. Operasi `PutItem`, `UpdateItem`, dan `DeleteItem` memungkinkan Anda memberikan syarat logis yang harus bernilai true sebelum operasi dapat dilanjutkan. Jika syarat ini bernilai false, `ConditionalCheckFailedRequests` bertambah satu. `ConditionalCheckFailedRequests` juga bertambah satu untuk pernyataan Perbarui dan Hapus PartiQL, dengan syarat logis disediakan dan syarat tersebut bernilai false.

> **Note**
>
> Penulisan bersyarat yang gagal akan menghasilkan kesalahan HTTP 400 (Permintaan Buruk). Peristiwa ini tercermin dalam metrik `ConditionalCheckFailedRequests`, tapi tidak di metrik `UserErrors`.

Unit: `Count`

Dimensi: `TableName`

Statistik Valid:

- `Minimum`
- `Maximum`
- `Average`
- `SampleCount`
- `Sum`

ConsumedChangeDataCaptureUnits

Jumlah unit tangkapan data perubahan yang digunakan.

Unit: `Count`

Dimensi: `TableName, DelegatedOperation`

Statistik Valid:

- `Minimum`
- `Maximum`
- `Average`

ConsumedReadCapacityUnits

Jumlah unit kapasitas baca yang digunakan selama jangka waktu tertentu untuk kapasitas yang tersedia dan sesuai permintaan, sehingga Anda dapat melacak berapa banyak throughput yang digunakan. Anda dapat mengambil total kapasitas baca yang digunakan untuk tabel dan semua indeks sekunder globalnya, atau untuk indeks sekunder global tertentu. Untuk informasi selengkapnya, lihat Mode Kapasitas Baca/Tulis.

Dimensi `TableName` mengembalikan `ConsumedReadCapacityUnits` untuk tabel, tetapi tidak untuk setiap indeks sekunder global. Guna melihat `ConsumedReadCapacityUnits` untuk indeks sekunder global, Anda harus menentukan `TableName` dan `GlobalSecondaryIndexName`.

> **Note**
>
> Ini berarti bahwa lonjakan konsumsi kapasitas yang pendek dan intens yang berlangsung hanya satu detik mungkin tidak tercermin secara akurat dalam CloudWatch grafik, yang berpotensi mengarah ke tingkat konsumsi yang lebih rendah untuk menit itu.
> Gunakan statistik `Sum` untuk menghitung throughput yang digunakan. Misalnya, dapatkan nilai `Sum` dalam rentang waktu satu menit, dan bagi nilai tersebut dengan jumlah detik dalam satu menit (60) untuk menghitung rata-rata `ConsumedReadCapacityUnits` per detik. Anda dapat membandingkan nilai terhitung dengan nilai throughput yang disediakan yang Anda berikan kepada DynamoDB.

Unit: `Count`

Dimensi: `TableName, GlobalSecondaryIndexName`

Statistik Valid:

- `Minimum` – Jumlah minimum unit kapasitas baca yang digunakan oleh setiap permintaan individu pada tabel atau indeks.
- `Maximum` – Jumlah maksimum unit kapasitas baca yang digunakan oleh setiap permintaan individu pada tabel atau indeks.
- `Average` – Rata-rata kapasitas baca per permintaan yang digunakan.

  > **Note**
  >
  > Nilai `Average` dipengaruhi oleh periode tidak aktif dengan nilai sampel akan menjadi nol.

- `Sum` – Total unit kapasitas baca yang digunakan. Ini adalah statistik yang paling berguna untuk metrik `ConsumedReadCapacityUnits`.
- `SampleCount`— mewakili frekuensi di mana metrik dipancarkan. Bahkan tabel dengan lalu lintas nol akan `SampleCount` dipancarkan secara teratur, tetapi nilai sampel akan selalu nol.

  > **Note**
  >
  > Nilai `SampleCount` dipengaruhi oleh periode tidak aktif dengan nilai sampel akan menjadi nol.

ConsumedWriteCapacityUnits

Jumlah unit kapasitas tulis yang digunakan selama jangka waktu tertentu untuk kapasitas yang tersedia dan sesuai permintaan, sehingga Anda dapat melacak berapa banyak throughput yang digunakan. Anda dapat mengambil total kapasitas tulis yang digunakan untuk tabel dan semua indeks sekunder globalnya, atau untuk indeks sekunder global tertentu. Untuk informasi selengkapnya, lihat Mode Kapasitas Baca/Tulis.

Dimensi `TableName` mengembalikan `ConsumedWriteCapacityUnits` untuk tabel, tetapi tidak untuk setiap indeks sekunder global. Guna melihat `ConsumedWriteCapacityUnits` untuk indeks sekunder global, Anda harus menentukan `TableName` dan `GlobalSecondaryIndexName`.

> **Note**
>
> Gunakan statistik `Sum` untuk menghitung throughput yang digunakan. Misalnya, dapatkan `Sum` nilai selama rentang satu menit, dan bagi dengan jumlah detik dalam satu menit (60) untuk menghitung rata-rata `ConsumedWriteCapacityUnits` per detik (mengakui bahwa rata-rata ini tidak menyoroti lonjakan besar tetapi singkat dalam aktivitas menulis yang terjadi selama menit itu). Anda dapat membandingkan nilai terhitung dengan nilai throughput yang disediakan yang Anda berikan kepada DynamoDB.

Unit: `Count`

Dimensi: `TableName, GlobalSecondaryIndexName`

Statistik Valid:

- `Minimum` – Jumlah minimum unit kapasitas tulis yang digunakan oleh setiap permintaan individu pada tabel atau indeks.
- `Maximum` – Jumlah maksimum unit kapasitas tulis yang digunakan oleh setiap permintaan individu pada tabel atau indeks.
- `Average` – Rata-rata kapasitas tulis per permintaan yang digunakan.

> **Note**
>
> Nilai `Average` dipengaruhi oleh periode tidak aktif dengan nilai sampel akan menjadi nol.

- `Sum` – Total unit kapasitas tulis yang digunakan. Ini adalah statistik yang paling berguna untuk metrik `ConsumedWriteCapacityUnits`.
- `SampleCount`— mewakili frekuensi di mana metrik dipancarkan. Bahkan tabel dengan lalu lintas nol akan `SampleCount` dipancarkan secara teratur, tetapi nilai sampel akan selalu nol.

> **Note**
>
> Nilai `SampleCount` dipengaruhi oleh periode tidak aktif dengan nilai sampel akan menjadi nol.

FailedToReplicateRecordCount

Jumlah catatan yang gagal direplikasi DynamoDB ke Kinesis data stream Anda.

Unit: `Count`

Dimensions: `TableName`, `DelegatedOperation`

Statistik Valid:

- `Sum`

MaxProvisionedTableReadCapacityUtilization

Persentase kapasitas baca yang disediakan dan digunakan oleh tabel atau indeks sekunder global baca tertinggi yang disediakan dari sebuah akun.

Unit: `Percent`

Statistik Valid:

- `Maximum` – Persentase maksimum unit kapasitas baca yang disediakan dan digunakan oleh tabel atau indeks sekunder global baca tertinggi yang disediakan dari sebuah akun.
- `Minimum` – Persentase minimum unit kapasitas baca yang disediakan dan digunakan oleh tabel atau indeks sekunder global baca tertinggi yang disediakan dari sebuah akun.
- `Average` – Persentase rata-rata unit kapasitas baca yang disediakan dan digunakan oleh tabel atau indeks sekunder global baca tertinggi yang disediakan dari akun. Metrik diterbitkan selama interval lima menit. Oleh karena itu, jika Anda dengan cepat menyesuaikan unit kapasitas baca yang disediakan, statistik ini mungkin tidak mencerminkan rata-rata sebenarnya.

MaxProvisionedTableWriteCapacityUtilization

Persentase kapasitas tulis yang disediakan dan digunakan oleh tabel atau indeks sekunder global penulisan tertinggi yang disediakan dari sebuah akun.

Unit: `Percent`

Statistik Valid:

- `Maximum` – Persentase maksimum unit kapasitas tulis yang disediakan dan digunakan oleh tabel atau indeks sekunder global tulis tertinggi yang disediakan dari sebuah akun.
- `Minimum` – Persentase minimum unit kapasitas tulis yang disediakan dan digunakan oleh tabel atau indeks sekunder global tulis tertinggi yang disediakan dari sebuah akun.
- `Average` – Persentase rata-rata unit kapasitas tulis yang disediakan dan digunakan oleh tabel atau indeks sekunder global tulis tertinggi yang disediakan dari akun. Metrik diterbitkan selama interval lima menit. Oleh karena itu, jika Anda dengan cepat menyesuaikan unit kapasitas tulis yang disediakan, statistik ini mungkin tidak mencerminkan rata-rata sebenarnya.

OnDemandMaxReadRequestUnits

Jumlah unit permintaan baca sesuai permintaan yang ditentukan untuk tabel atau indeks sekunder global.

`OnDemandMaxReadRequestUnits`Untuk melihat tabel, Anda harus menentukan`TableName`. Guna melihat `OnDemandMaxReadRequestUnits` untuk indeks sekunder global, Anda harus menentukan `TableName` dan `GlobalSecondaryIndexName`.

Unit: Hitungan

Dimensions: `TableName`, `GlobalSecondaryIndexName`

Statistik Valid:

- `Minimum`— Pengaturan terendah untuk unit permintaan baca sesuai permintaan. Jika Anda menggunakan `UpdateTable` untuk meningkatkan unit permintaan baca, metrik ini menunjukkan nilai terendah berdasarkan permintaan `ReadRequestUnits` selama periode waktu ini.
- `Maximum`— Pengaturan tertinggi untuk unit permintaan baca sesuai permintaan. Jika Anda menggunakan `UpdateTable` untuk mengurangi unit permintaan baca, metrik ini menunjukkan nilai tertinggi sesuai permintaan `ReadRequestUnits` selama periode waktu ini.

- `Average`— Rata-rata unit permintaan baca berdasarkan permintaan. Metrik `OnDemandMaxReadRequestUnits` diterbitkan pada interval lima menit. Oleh karena itu, jika Anda dengan cepat menyesuaikan unit permintaan baca sesuai permintaan, statistik ini mungkin tidak mencerminkan rata-rata sebenarnya.

OnDemandMaxWriteRequestUnits

Jumlah unit permintaan tulis sesuai permintaan yang ditentukan untuk tabel atau indeks sekunder global.

`OnDemandMaxWriteRequestUnits`Untuk melihat tabel, Anda harus menentukan`TableName`. Guna melihat `OnDemandMaxWriteRequestUnits` untuk indeks sekunder global, Anda harus menentukan `TableName` dan `GlobalSecondaryIndexName`.

Unit: `Count`

Dimensions: `TableName`, `GlobalSecondaryIndexName`

Statistik Valid:

- `Minimum`— Pengaturan terendah untuk unit permintaan tulis sesuai permintaan. Jika Anda menggunakan `UpdateTable` untuk meningkatkan unit permintaan tulis, metrik ini menunjukkan nilai terendah sesuai permintaan `WriteRequestUnits` selama periode waktu ini.
- `Maximum`— Pengaturan tertinggi untuk unit permintaan tulis sesuai permintaan. Jika Anda menggunakan `UpdateTable` untuk mengurangi unit permintaan tulis, metrik ini menunjukkan nilai tertinggi sesuai permintaan `WriteRequestUnits` selama periode waktu ini.
- `Average`— Rata-rata unit permintaan tulis berdasarkan permintaan. Metrik `OnDemandMaxWriteRequestUnits` diterbitkan pada interval lima menit. Oleh karena itu, jika Anda dengan cepat menyesuaikan unit permintaan tulis sesuai permintaan, statistik ini mungkin tidak mencerminkan rata-rata sebenarnya.

OnlineIndexConsumedWriteCapacity

Jumlah unit kapasitas tulis digunakan saat menambahkan indeks sekunder global baru ke tabel. Jika kapasitas tulis indeks terlalu rendah, aktivitas tulis masuk selama fase backfill mungkin akan mengalami throttling. Hal ini dapat meningkatkan waktu yang dibutuhkan untuk membuat indeks. Anda harus memantau statistik ini saat indeks sedang dibuat untuk menentukan apakah kapasitas tulis indeks kurang tersedia.

Anda dapat menyesuaikan kapasitas tulis indeks menggunakan operasi `UpdateTable`, meskipun indeks sedang dibuat.

`ConsumedWriteCapacityUnits`Metrik untuk indeks tidak menyertakan throughput tulis yang digunakan selama pembuatan indeks.

> **Note**
>
> Metrik ini mungkin tidak ditampilkan jika fase backfill indeks sekunder global yang baru selesai dengan cepat (kurang dari beberapa menit), yang mungkin terjadi jika tabel dasar memiliki sedikit atau tidak ada item untuk di-backfill dalam indeks.

Unit: `Count`

Dimensi: `TableName, GlobalSecondaryIndexName`

Statistik Valid:

- `Minimum`
- `Maximum`
- `Average`
- `SampleCount`
- `Sum`

OnlineIndexPercentageProgress

Persentase penyelesaian ketika indeks sekunder global baru sedang ditambahkan ke tabel. DynamoDB pertama-tama harus mengalokasikan sumber daya untuk indeks baru, lalu melakukan backfill atribut dari tabel ke indeks. Untuk tabel besar, proses ini mungkin memerlukan waktu lama. Anda harus memantau statistik ini untuk melihat kemajuan relatif saat DynamoDB membuat indeks.

Unit: `Count`

Dimensi: `TableName, GlobalSecondaryIndexName`

Statistik Valid:

- `Minimum`

- `Maximum`
- `Average`
- `SampleCount`
- `Sum`

OnlineIndexThrottleEvents

Jumlah peristiwa pelambatan menulis yang terjadi saat menambahkan indeks sekunder global baru ke tabel. Peristiwa ini menunjukkan bahwa pembuatan indeks akan memakan waktu lebih lama, karena aktivitas tulis masuk melampaui throughput tulis yang disediakan pada indeks.

Anda dapat menyesuaikan kapasitas tulis indeks menggunakan operasi `UpdateTable`, meskipun indeks sedang dibuat.

`WriteThrottleEvents`Metrik untuk indeks tidak menyertakan peristiwa throttle apa pun yang terjadi selama pembuatan indeks.

Unit: `Count`

Dimensi: `TableName, GlobalSecondaryIndexName`

Statistik Valid:

- `Minimum`
- `Maximum`
- `Average`
- `SampleCount`
- `Sum`

PendingReplicationCount

Metrik untuk Versi tabel global 2017.11.29 (Legacy) (hanya tabel global). Jumlah pembaruan item yang ditulis ke satu tabel replika, tetapi belum ditulis ke replika lain dalam tabel global.

Unit: `Count`

Dimensi: `TableName, ReceivingRegion`

Statistik Valid:

- Average
- Sample Count
- Sum

ProvisionedReadCapacityUnits

Jumlah unit kapasitas baca yang disediakan untuk tabel atau indeks sekunder global. Dimensi TableName mengembalikan ProvisionedReadCapacityUnits untuk tabel, tetapi tidak untuk setiap indeks sekunder global. Guna melihat ProvisionedReadCapacityUnits untuk indeks sekunder global, Anda harus menentukan TableName dan GlobalSecondaryIndexName.

Unit:Count

Dimensi: TableName, GlobalSecondaryIndexName

Statistik Valid:

- Minimum – Pengaturan terendah untuk kapasitas baca yang disediakan. Jika Anda menggunakan UpdateTable untuk menambah kapasitas baca, metrik ini menunjukkan nilai terendah ReadCapacityUnits yang disediakan selama periode waktu ini.
- Maximum – Pengaturan tertinggi untuk kapasitas baca yang disediakan. Jika Anda menggunakan UpdateTable untuk mengurangi kapasitas baca, metrik ini menunjukkan nilai tertinggi ReadCapacityUnits yang disediakan selama periode waktu ini.
- Average – Rata-rata kapasitas baca yang disediakan. Metrik ProvisionedReadCapacityUnits diterbitkan pada interval lima menit. Oleh karena itu, jika Anda dengan cepat menyesuaikan unit kapasitas baca yang disediakan, statistik ini mungkin tidak mencerminkan rata-rata sebenarnya.

ProvisionedWriteCapacityUnits

Jumlah unit kapasitas tulis yang disediakan untuk tabel atau indeks sekunder global.

Dimensi TableName mengembalikan ProvisionedWriteCapacityUnits untuk tabel, tetapi tidak untuk setiap indeks sekunder global. Guna melihat ProvisionedWriteCapacityUnits untuk indeks sekunder global, Anda harus menentukan TableName dan GlobalSecondaryIndexName.

Unit: Count

Dimensi: `TableName, GlobalSecondaryIndexName`

Statistik Valid:

- `Minimum` – Pengaturan terendah untuk kapasitas tulis yang disediakan. Jika Anda menggunakan `UpdateTable` untuk menambah kapasitas tulis, metrik ini menunjukkan nilai terendah `WriteCapacityUnits` yang disediakan selama periode waktu ini.
- `Maximum` – Pengaturan tertinggi untuk kapasitas tulis yang disediakan. Jika Anda menggunakan `UpdateTable` untuk mengurangi kapasitas tulis, metrik ini menunjukkan nilai tertinggi `WriteCapacityUnits` yang disediakan selama periode waktu ini.
- `Average` – Rata-rata kapasitas tulis yang disediakan. Metrik `ProvisionedWriteCapacityUnits` diterbitkan pada interval lima menit. Oleh karena itu, jika Anda dengan cepat menyesuaikan unit kapasitas tulis yang disediakan, statistik ini mungkin tidak mencerminkan rata-rata sebenarnya.

ReadThrottleEvents

Permintaan untuk DynamoDB yang melampaui unit kapasitas baca yang disediakan untuk tabel atau indeks sekunder global.

Suatu permintaan dapat mengakibatkan beberapa peristiwa. Sebagai contoh, `BatchGetItem` yang membaca 10 item diproses sebagai 10 peristiwa `GetItem`. Untuk setiap peristiwa, `ReadThrottleEvents` bertambah satu jika peristiwa tersebut mengalami throttling. Metrik `ThrottledRequests` untuk seluruh `BatchGetItem` tidak bertambah kecuali jika 10 peristiwa `GetItem` mengalami throttling.

Dimensi `TableName` mengembalikan `ReadThrottleEvents` untuk tabel, tetapi tidak untuk setiap indeks sekunder global. Guna melihat `ReadThrottleEvents` untuk indeks sekunder global, Anda harus menentukan `TableName` dan `GlobalSecondaryIndexName`.

Unit: `Count`

Dimensi: `TableName, GlobalSecondaryIndexName`

Statistik Valid:

- `SampleCount`
- `Sum`

ReplicationLatency

(Metrik ini adalah untuk tabel global DynamoDB.) Waktu yang berlalu antara item yang diperbarui yang muncul di aliran DynamoDB untuk satu tabel replika, dan item tersebut muncul di replika lain di tabel global.

Unit: `Milliseconds`

Dimensi: `TableName, ReceivingRegion`

Statistik Valid:

- `Average`
- `Minimum`
- `Maximum`

ReturnedBytes

Jumlah byte yang dikembalikan oleh operasi `GetRecords` (Amazon DynamoDB Streams) selama periode waktu yang ditentukan.

Unit: `Bytes`

Dimensi: `Operation, StreamLabel, TableName`

Statistik Valid:

- `Minimum`
- `Maximum`
- `Average`
- `SampleCount`
- `Sum`

ReturnedItemCount

Jumlah item yang dikembalikan oleh operasi `Query`, `Scan` atau `ExecuteStatement` (pilih) selama periode waktu yang ditentukan.

Jumlah item yang dikembalikan belum tentu sama dengan jumlah item yang dievaluasi. Sebagai contoh, anggaplah Anda meminta `Scan` pada suatu tabel atau indeks yang memiliki 100 item,

tetapi menentukan `FilterExpression` yang mempersempit hasil sehingga hanya 15 item yang dikembalikan. Dalam hal ini, respons dari `Scan` akan berisi `ScanCount` sebanyak 100 dan `Count` sebesar 15 item yang dikembalikan.

Unit: `Count`

Dimensi: `TableName, Operation`

Statistik Valid:

- `Minimum`
- `Maximum`
- `Average`
- `SampleCount`
- `Sum`

ReturnedRecordsCount

Jumlah catatan aliran yang dikembalikan oleh operasi `GetRecords` (Amazon DynamoDB Streams) selama periode waktu yang ditentukan.

Unit: `Count`

Dimensi: `Operation, StreamLabel, TableName`

Statistik Valid:

- `Minimum`
- `Maximum`
- `Average`
- `SampleCount`
- `Sum`

SuccessfulRequestLatency

Latensi permintaan yang berhasil untuk DynamoDB atau Amazon DynamoDB Streams selama jangka waktu tertentu. `SuccessfulRequestLatency` dapat memberikan dua jenis informasi:

- Waktu yang telah berlalu untuk permintaan yang berhasil (`Minimum`,,, `Maximum` `SumAverage`, atau`Percentile`).
- Jumlah permintaan yang berhasil (`SampleCount`).

`SuccessfulRequestLatency`mencerminkan aktivitas hanya dalam DynamoDB atau Amazon DynamoDB Streams, dan tidak mempertimbangkan latensi jaringan atau aktivitas sisi klien.

> **Note**
>
> Untuk menganalisis nilai persentil kustom (seperti p99.9), Anda dapat memasukkan persentil yang diinginkan secara manual (misalnya, p99.9) di bidang statistik metrik. CloudWatch Ini memungkinkan Anda untuk mengevaluasi distribusi latensi di luar persentil default yang tercantum dalam dropdown.

Unit: `Milliseconds`

Dimensi: `TableName, Operation, StreamLabel`

Statistik Valid:

- `Minimum`
- `Maximum`
- `Sum`
- `Average`
- `Percentile`
- `SampleCount`

SystemErrors

Permintaan untuk DynamoDB atau Amazon DynamoDB Streams yang menghasilkan kode status HTTP 500 selama jangka waktu tertentu. HTTP 500 biasanya menunjukkan kesalahan layanan internal.

Unit: `Count`

Dimensi: `TableName, Operation`

Statistik Valid:

- `Sum`
- `SampleCount`

TimeToLiveDeletedItemCount

Jumlah item yang dihapus oleh Waktu untuk Beroperasi (TTL) selama jangka waktu yang ditentukan. Metrik ini membantu Anda memantau laju penghapusan TTL di tabel Anda.

Unit: `Count`

Dimensi: TableName

Statistik Valid:

- `Sum`

ThrottledPutRecordCount

Jumlah catatan yang dibatasi oleh Kinesis data stream Anda karena kapasitas Kinesis Data Streams tidak memadai.

Unit: `Count`

Dimensi: TableName, DelegatedOperation

Statistik Valid:

- `Minimum`
- `Maximum`
- `Average`
- `SampleCount`

ThrottledRequests

Permintaan untuk DynamoDB yang melampaui batas throughput yang disediakan pada sumber daya (seperti tabel atau indeks).

`ThrottledRequests` bertambah satu jika ada peristiwa dalam permintaan yang melampaui batas throughput yang disediakan. Misalnya, jika Anda memperbarui item dalam tabel dengan indeks sekunder global, ada beberapa peristiwa—penulisan ke tabel, dan penulisan ke setiap indeks. Jika satu atau beberapa peristiwa ini mengalami throttling, maka `ThrottledRequests` bertambah satu.

> **Note**
>
> Dalam permintaan batch (`BatchGetItem` atau `BatchWriteItem`), `ThrottledRequests` bertambah hanya jika setiap permintaan dalam batch mengalami throttling.
> Jika setiap permintaan individual dalam batch mengalami throttling, salah satu metrik berikut bertambah:
>
> - `ReadThrottleEvents` – Untuk peristiwa `GetItem` yang mengalami throttling dalam `BatchGetItem`.
> - `WriteThrottleEvents` – Untuk peristiwa `PutItem` atau `DeleteItem` yang mengalami throttling dalam `BatchWriteItem`.

Untuk mendapatkan wawasan tentang peristiwa mana yang melakukan throttling terhadap permintaan, bandingkan `ThrottledRequests` dengan `ReadThrottleEvents` dan `WriteThrottleEvents` untuk tabel dan indeksnya.

> **Note**
>
> Permintaan yang mengalami throttling akan menghasilkan kode status HTTP 400. Semua peristiwa tersebut tercermin dalam metrik `ThrottledRequests`, tapi tidak di metrik `UserErrors`.

Unit: `Count`

Dimensi: `TableName, Operation`

Statistik Valid:

- `Sum`
- `SampleCount`

TransactionConflict

Permintaan tingkat item ditolak karena konflik transaksional antara permintaan serentak pada item yang sama. Untuk informasi selengkapnya, lihat Penanganan Konflik Transaksi di DynamoDB.

Unit: `Count`

Dimensi: `TableName`

Statistik Valid:

- `Sum` – Jumlah permintaan tingkat item yang ditolak karena konflik transaksi.

  > **Note**
  >
  > Jika beberapa permintaan tingkat item dalam suatu panggilan ke `TransactWriteItems` atau `TransactGetItems` ditolak, `Sum` bertambah satu untuk setiap permintaan `Put`, `Update`, `Delete`, atau `Get` tingkat item.

- `SampleCount` – Jumlah permintaan yang ditolak karena konflik transaksi.

  > **Note**
  >
  > Jika beberapa permintaan tingkat item dalam suatu panggilan ke `TransactWriteItems` atau `TransactGetItems` ditolak, `SampleCount` hanya bertambah satu.

- `Min` – Jumlah minimum permintaan tingkat item yang ditolak dalam panggilan ke `TransactWriteItems`, `TransactGetItems`, `PutItem`, `UpdateItem`, atau `DeleteItem`.
- `Max` – Jumlah maksimum permintaan tingkat item yang ditolak dalam panggilan ke `TransactWriteItems`, `TransactGetItems`, `PutItem`, `UpdateItem`, atau `DeleteItem`.
- `Average` – Jumlah rata-rata permintaan tingkat item yang ditolak dalam panggilan ke `TransactWriteItems`, `TransactGetItems`, `PutItem`, `UpdateItem`, atau `DeleteItem`.

UserErrors

Permintaan ke DynamoDB atau Amazon DynamoDB Streams yang menghasilkan kode status HTTP 400 selama jangka waktu tertentu. HTTP 400 biasanya menunjukkan kesalahan di sisi klien, seperti kombinasi parameter yang tidak valid, upaya untuk memperbarui tabel tidak ada, atau tanda tangan permintaan yang salah.

Beberapa contoh pengecualian yang akan mencatat metrik yang terkait dengan `UserErrors` adalah:

- `ResourceNotFoundException`
- `ValidationException`
- `TransactionConflict`

Semua peristiwa tersebut tercermin dalam metrik `UserErrors`, kecuali yang berikut ini:

- ProvisionedThroughputExceededException— Lihat `ThrottledRequests` metrik di bagian ini.
- ConditionalCheckFailedException— Lihat `ConditionalCheckFailedRequests` metrik di bagian ini.

`UserErrors`mewakili agregat kesalahan HTTP 400 untuk permintaan DynamoDB atau Amazon DynamoDB Streams untuk Wilayah saat ini dan akun saat ini. AWS AWS

Unit: `Count`

Statistik Valid:

- `Sum`
- `SampleCount`

WriteThrottleEvents

Permintaan untuk DynamoDB yang melampaui unit kapasitas tulis yang disediakan untuk tabel atau indeks sekunder global.

Suatu permintaan dapat mengakibatkan beberapa peristiwa. Misalnya, permintaan `PutItem` pada tabel dengan tiga indeks sekunder global akan menghasilkan empat peristiwa—penulisan tabel, dan masing-masing dari tiga penulisan indeks. Untuk setiap peristiwa, metrik `WriteThrottleEvents` bertambah satu jika peristiwa tersebut mengalami throttling. Untuk satu permintaan `PutItem`, jika salah satu peristiwa mengalami throttling, `ThrottledRequests` juga bertambah satu. Untuk `BatchWriteItem`, metrik `ThrottledRequests` untuk seluruh `BatchWriteItem` tidak bertambah kecuali jika semua `PutItem` individual atau peristiwa `DeleteItem` mengalami throttling.

Dimensi `TableName` mengembalikan `WriteThrottleEvents` untuk tabel, tetapi tidak untuk setiap indeks sekunder global. Guna melihat `WriteThrottleEvents` untuk indeks sekunder global, Anda harus menentukan `TableName` dan `GlobalSecondaryIndexName`.

Unit: `Count`

Dimensi: `TableName, GlobalSecondaryIndexName`

Statistik Valid:

- `Sum`
- `SampleCount`

Metrik penggunaan

Metrik penggunaan CloudWatch memungkinkan Anda mengelola penggunaan secara proaktif dengan memvisualisasikan metrik di CloudWatch konsol, membuat dasbor khusus, mendeteksi perubahan aktivitas dengan deteksi CloudWatch anomali, dan mengonfigurasi alarm yang mengingatkan Anda saat penggunaan mendekati ambang batas.

DynamoDB juga mengintegrasikan metrik penggunaan ini dengan Kuota Layanan. Anda dapat menggunakan CloudWatch untuk mengelola penggunaan akun Anda atas kuota layanan Anda. Untuk informasi selengkapnya, lihat Memvisualisasikan kuota layanan dan mengatur alarm

Daftar Metrik Penggunaan yang Tersedia

- AccountProvisionedWriteCapacityUnits
- AccountProvisionedReadCapacityUnits
- TableCount

AccountProvisionedWriteCapacityUnits

Jumlah unit kapasitas tulis yang disediakan untuk semua tabel dan indeks sekunder global pada account.

Unit: `Count`

Statistik Valid:

- `Minimum` – Jumlah terendah unit kapasitas tulis yang disediakan selama jangka waktu tertentu.

- `Maximum` – Jumlah tertinggi unit kapasitas tulis yang disediakan selama jangka waktu tertentu.
- `Average` – Jumlah rata-rata unit kapasitas tulis yang disediakan pada akun selama jangka waktu tertentu.

Metrik ini diterbitkan pada interval lima menit. Oleh karena itu, jika Anda dengan cepat menyesuaikan unit kapasitas tulis yang disediakan, statistik ini mungkin tidak mencerminkan rata-rata sebenarnya.

AccountProvisionedReadCapacityUnits

Jumlah unit kapasitas baca yang disediakan untuk semua tabel dan indeks sekunder global pada account.

Unit: `Count`

Statistik Valid:

- `Minimum` – Jumlah terendah unit kapasitas baca yang disediakan selama jangka waktu tertentu.
- `Maximum` – Jumlah tertinggi unit kapasitas baca yang disediakan selama jangka waktu tertentu.
- `Average` – Jumlah rata-rata unit kapasitas baca yang disediakan pada akun selama jangka waktu tertentu.

Metrik ini diterbitkan pada interval lima menit. Oleh karena itu, jika Anda dengan cepat menyesuaikan unit kapasitas baca yang disediakan, statistik ini mungkin tidak mencerminkan rata-rata sebenarnya.

TableCount

Jumlah tabel aktif pada akun.

Unit: `Count`

Statistik Valid:

- `Minimum` – Jumlah tabel terendah selama periode waktu tertentu.
- `Maximum` – Jumlah tabel tertinggi selama periode waktu tertentu.
- `Average` – Jumlah tabel rata-rata selama periode waktu tertentu.

Memahami metrik dan dimensi DynamoDB

Metrik untuk DynamoDB dikualifikasikan berdasarkan nilai akun, nama tabel, nama indeks sekunder global, atau operasi. Anda dapat menggunakan CloudWatch konsol untuk mengambil data DynamoDB sepanjang salah satu dimensi dalam tabel di bawah ini.

Daftar Dimensi yang Tersedia

- DelegatedOperation
- GlobalSecondaryIndexName
- Operasi
- OperationType
- Kata Kerja
- ReceivingRegion
- StreamLabel
- TableName

DelegatedOperation

Dimensi ini membatasi data pada operasi yang dilakukan DynamoDB atas nama Anda. Operasi berikut ditangkap:

- Ubah penangkapan data untuk Kinesis Data Streams.

GlobalSecondaryIndexName

Dimensi ini membatasi data untuk indeks sekunder global pada tabel. Jika menentukan `GlobalSecondaryIndexName`, Anda juga harus menentukan `TableName`.

Operasi

Dimensi ini membatasi data ke salah satu operasi DynamoDB berikut:

- `PutItem`
- `DeleteItem`
- `UpdateItem`

- `GetItem`
- `BatchGetItem`
- `Scan`
- `Query`
- `BatchWriteItem`
- `TransactWriteItems`
- `TransactGetItems`
- `ExecuteTransaction`
- `BatchExecuteStatement`
- `ExecuteStatement`

Selain itu, Anda dapat membatasi data untuk operasi Amazon DynamoDB Streams berikut:

- `GetRecords`

OperationType

Dimensi ini membatasi data ke salah satu jenis operasi berikut:

- `Read`
- `Write`

Dimensi ini diteruskan untuk permintaan `ExecuteTransaction` dan `BatchExecuteStatement`.

Kata Kerja

Dimensi ini membatasi data ke salah satu kata kerja DynamoDB PartiQL berikut:

- Sisipkan: `PartiQLInsert`
- Pilih: `PartiQLSelect`
- Perbarui: `PartiQLUpdate`
- Hapus: `PartiQLDelete`

Dimensi ini diteruskan untuk operasi `ExecuteStatement`.

ReceivingRegion

Dimensi ini membatasi data ke AWS wilayah tertentu. Ini digunakan dengan metrik yang berasal dari tabel replika dalam tabel global DynamoDB.

StreamLabel

Dimensi ini membatasi data untuk label aliran tertentu. Ini digunakan dengan metrik yang berasal dari operasi `GetRecords` Amazon DynamoDB Streams.

TableName

Dimensi ini membatasi data ke tabel tertentu. Nilai ini dapat berupa nama tabel apa pun di wilayah saat ini dan AWS akun saat ini.

Membuat CloudWatch alarm di DynamoDB

CloudWatch Alarm mengawasi metrik tunggal selama periode waktu tertentu, dan melakukan satu atau lebih tindakan tertentu, berdasarkan nilai metrik relatif terhadap ambang batas dari waktu ke waktu. Tindakan ini adalah notifikasi yang dikirim ke topik Amazon SNS atau kebijakan Penskalaan Otomatis. Anda juga dapat menambahkan alarm ke dasbor sehingga Anda dapat memantau dan menerima peringatan tentang AWS sumber daya dan aplikasi Anda di beberapa wilayah. Tidak ada batasan jumlah alarm yang dapat Anda buat. CloudWatch alarm tidak memanggil tindakan hanya karena mereka berada dalam keadaan tertentu; negara harus telah berubah dan dipertahankan untuk sejumlah periode tertentu. Untuk daftar alarm DynamoDB yang direkomendasikan, lihat Alarm yang direkomendasikan.

> **Note**
>
> Anda harus menentukan semua dimensi yang diperlukan saat membuat CloudWatch alarm Anda, karena tidak CloudWatch akan menggabungkan metrik untuk dimensi yang hilang. Membuat CloudWatch alarm dengan dimensi yang hilang tidak akan menghasilkan kesalahan, saat membuat alarm.

Asumsikan Anda memiliki tabel yang disediakan dengan lima unit kapasitas baca. Anda ingin diberi tahu sebelum Anda mengkonsumsi seluruh kapasitas baca yang disediakan, jadi Anda memutuskan untuk membuat CloudWatch alarm untuk mendapatkan pemberitahuan ketika kapasitas yang dikonsumsi mencapai 80% dari apa yang telah Anda sediakan untuk tabel. Anda dapat membuat alarm di CloudWatch konsol atau menggunakan. AWS CLI

Membuat alarm di CloudWatch konsol

Untuk membuat alarm di CloudWatch konsol

1. Masuk ke AWS Management Console dan buka CloudWatch konsol di https://console.aws.amazon.com/cloudwatch/.
2. Pada panel navigasi, pilih Alarm, Semua alarm.
3. Pilih Buat alarm.
4. Temukan baris dengan tabel yang ingin Anda pantau dan **`ConsumeReadCapacityUnits`**di kolom Nama Metrik. Pilih kotak centang di sebelah baris ini dan pilih Pilih metrik.
5. Di bawah Tentukan metrik dan kondisi, untuk Statistik pilih Jumlah. Pilih Periode 1 menit.
6. Pada Ketentuan, tentukan hal-hal berikut:
   a. Untuk Jenis ambang batas, pilih Statis.
   b. Untuk Kapanpun **`ConsumedReadCapacityUnits`** ada, pilih Greater/Equal dan tentukan ambang batas sebagai 240.
7. Pilih Berikutnya.
8. Di bawah Pemberitahuan, pilih **`In alarm`**dan pilih topik SNS untuk memberi tahu saat alarm dalam `ALARM` status.
9. Setelah selesai, silakan pilih Berikutnya.
10. Masukkan nama dan deskripsi untuk alarm dan pilih Berikutnya.
11. Pada Pratinjau dan buat, konfirmasikan bahwa informasi dan ketentuannya sudah sesuai keinginan Anda, kemudian pilih Buat alarm.

Membuat alarm di 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
```

Uji 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
```

Lebih banyak AWS CLI contoh

Prosedur berikut menjelaskan bagaimana Anda diberi tahu jika Anda memiliki permintaan yang melebihi kuota yang disediakan melalui tabel.

1. Buat topik Amazon SNS. `arn:aws:sns:us-east-1:123456789012:requests-exceeding-throughput` Untuk informasi selengkapnya, lihat Menyiapkan Amazon Simple Notification Service.
2. Buat 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. Uji 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
```

Prosedur berikut menjelaskan bagaimana Anda diberi tahu jika Anda mendapatkan kesalahan sistem.

1. Buat topik Amazon SNS. `arn:aws:sns:us-east-1:123456789012:notify-on-system-errors` Untuk informasi selengkapnya, lihat Menyiapkan Amazon Simple Notification Service.
2. Buat 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. Uji 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
```

Pencatatan log operasi DynamoDB menggunakan AWS CloudTrail

DynamoDB terintegrasi AWS CloudTrail dengan, layanan yang menyediakan catatan tindakan yang diambil oleh pengguna, peran, atau AWS layanan di DynamoDB. CloudTrail menangkap semua panggilan API untuk DynamoDB sebagai peristiwa. Panggilan yang ditangkap mencakup panggilan dari konsol DynamoDB dan panggilan kode ke operasi DynamoDB API, menggunakan PartiQL dan API klasik. Jika Anda membuat jejak, Anda dapat mengaktifkan pengiriman acara secara terus menerus ke bucket Amazon S3, termasuk CloudTrail peristiwa untuk DynamoDB. Jika Anda tidak mengonfigurasi jejak, Anda masih dapat melihat peristiwa terbaru di CloudTrail konsol dalam Riwayat acara. Dengan menggunakan informasi yang dikumpulkan oleh CloudTrail, Anda dapat menentukan permintaan yang dibuat ke DynamoDB, alamat IP dari mana permintaan dibuat, siapa yang membuat permintaan, kapan dibuat, dan detail tambahan.

Untuk pemantauan dan peringatan yang kuat, Anda juga dapat mengintegrasikan CloudTrail peristiwa dengan Amazon CloudWatch Logs. Untuk meningkatkan analisis aktivitas layanan DynamoDB dan mengidentifikasi perubahan aktivitas untuk AWS akun, Anda dapat melakukan AWS CloudTrail kueri log menggunakan Amazon Athena. Misalnya, Anda dapat menggunakan kueri untuk mengidentifikasi tren dan mengisolasi aktivitas lebih lanjut berdasarkan atribut seperti alamat IP sumber atau pengguna.

Untuk mempelajari selengkapnya CloudTrail, termasuk cara mengonfigurasi dan mengaktifkannya, lihat Panduan AWS CloudTrail Pengguna.

Topik

- Informasi DynamoDB di CloudTrail
- Memahami entri file log DynamoDB

Informasi DynamoDB di CloudTrail

CloudTrail diaktifkan di AWS akun Anda saat Anda membuat akun. Ketika aktivitas peristiwa yang didukung terjadi di DynamoDB, aktivitas tersebut direkam dalam CloudTrail suatu peristiwa bersama dengan peristiwa layanan AWS lainnya dalam riwayat Acara. Anda dapat melihat, mencari, dan mengunduh acara terbaru di AWS akun Anda. Untuk informasi selengkapnya, lihat Bekerja dengan riwayat CloudTrail Acara.

Untuk catatan peristiwa yang sedang berlangsung di AWS akun Anda, termasuk acara untuk DynamoDB, buat jejak. Jejak memungkinkan CloudTrail untuk mengirimkan file log ke bucket

Amazon S3. Secara default, saat Anda membuat jejak di konsol, jejak tersebut berlaku untuk semua AWS Wilayah. Jejak mencatat peristiwa dari semua Wilayah di AWS partisi dan mengirimkan file log ke bucket Amazon S3 yang Anda tentukan. Selain itu, Anda dapat mengonfigurasi AWS layanan lain untuk menganalisis lebih lanjut dan menindaklanjuti data peristiwa yang dikumpulkan dalam CloudTrail log. Untuk informasi selengkapnya, lihat berikut:

- Gambaran umum untuk membuat jejak
- CloudTrail layanan dan integrasi yang didukung
- Mengonfigurasi notifikasi Amazon SNS untuk CloudTrail
- Menerima file CloudTrail log dari beberapa wilayah dan Menerima file CloudTrail log dari beberapa akun

Mengontrol peristiwa pesawat di CloudTrail

Tindakan API berikut dicatat secara default sebagai peristiwa dalam CloudTrail file:

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

Peristiwa bidang data DynamoDB di CloudTrail

Untuk mengaktifkan pencatatan tindakan API berikut dalam CloudTrail file, Anda harus mengaktifkan pencatatan aktivitas API bidang data di CloudTrail. Lihat Pencatatan log peristiwa data untuk jejak untuk informasi selengkapnya.

Peristiwa bidang data dapat difilter berdasarkan jenis sumber daya, untuk kontrol terperinci atas panggilan DynamoDB API yang ingin Anda log dan bayar secara selektif. CloudTrail Misalnya, dengan menentukan `AWS::DynamoDB::Stream` sebagai tipe sumber daya, Anda hanya dapat mencatat panggilan ke aliran DynamoDB. APIs Untuk tabel dengan aliran yang diaktifkan, bidang sumber daya dalam peristiwa bidang data berisi `AWS::DynamoDB::Stream` dan `AWS::DynamoDB::Table`. Jika Anda menentukan `AWS::DynamoDB::Table` sebagai jenis sumber daya, tabel DynamoDB dan peristiwa aliran DynamoDB akan dicatat secara default. Anda dapat menambahkan filter tambahan untuk mengecualikan peristiwa aliran, jika tidak ingin peristiwa aliran dicatat. Untuk informasi selengkapnya, lihat DataResource di dalam Referensi API AWS CloudTrail .

Amazon DynamoDB

- BatchExecuteStatement
- BatchGetItem
- BatchWriteItem
- DeleteItem
- ExecuteStatement
- ExecuteTransaction
- GetItem
- PutItem

- Kueri
- Scan
- TransactGetItems
- TransactWriteItems
- UpdateItem

> **Note**
>
> DynamoDB Time to Live tindakan pesawat data tidak dicatat oleh CloudTrail

DynamoDB Streams

- GetRecords
- GetShardIterator

Memahami entri file log DynamoDB

Trail adalah konfigurasi yang memungkinkan pengiriman peristiwa sebagai file log ke bucket Amazon S3 yang Anda tentukan. CloudTrail file log berisi satu atau lebih entri log. Sebuah peristiwa mewakili permintaan tunggal dari sumber apa pun dan mencakup informasi tentang tindakan yang diminta, tanggal dan waktu tindakan, parameter permintaan, dan sebagainya.

Setiap entri kejadian atau log berisi informasi tentang siapa yang membuat permintaan tersebut. Informasi identitas membantu Anda menentukan hal berikut:

- Baik permintaan tersebut dibuat dengan kredensial pengguna atau root.
- Apakah permintaan tersebut dibuat dengan kredensial keamanan sementara untuk satu peran atau pengguna gabungan.
- Apakah permintaan itu dibuat oleh AWS layanan lain.

> **Note**
>
> Nilai atribut non kunci akan disunting dalam CloudTrail log tindakan menggunakan API PartiQL, dan tidak akan muncul di log tindakan menggunakan API klasik.

Untuk informasi selengkapnya, lihat Elemen userIdentity CloudTrail .

Contoh berikut menunjukkan CloudTrail log dari jenis acara ini:

Amazon DynamoDB

- UpdateTable
- DeleteTable
- CreateCluster
- PutItem (Berhasil)
- UpdateItem (Gagal)
- TransactWriteItems (Berhasil)
- TransactWriteItems (Dengan 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 (Berhasil)

```
{
    "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 (Gagal)


```
{
    "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` (Berhasil)

```
{
    "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 (Dengan **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"
        }
    ]
}
```

Menganalisis akses data menggunakan wawasan CloudWatch kontributor untuk DynamoDB

Amazon CloudWatch Contributor Insights untuk Amazon DynamoDB adalah alat diagnostik untuk mengidentifikasi kunci yang paling sering diakses dan dibatasi dalam tabel atau indeks Anda secara sekilas. Alat ini menggunakan wawasan CloudWatch kontributor.

Dengan mengaktifkan CloudWatch Contributor Insights for DynamoDB pada tabel atau indeks sekunder global, Anda dapat melihat item yang paling banyak diakses dan dibatasi dalam sumber daya tersebut.

> **Note**
>
> CloudWatch biaya berlaku untuk Contributor Insights untuk DynamoDB. Untuk informasi selengkapnya tentang harga, lihat CloudWatchharga Amazon.

Topik

- CloudWatch wawasan kontributor untuk DynamoDB: Cara kerjanya

- Memulai CloudWatch Contributor Insights for DynamoDB
- Menggunakan IAM dengan wawasan CloudWatch kontributor untuk DynamoDB

CloudWatch wawasan kontributor untuk DynamoDB: Cara kerjanya

Amazon DynamoDB terintegrasi CloudWatch dengan Contributor Insights untuk memberikan informasi tentang item yang paling banyak diakses dan dibatasi dalam tabel atau indeks sekunder global. DynamoDB memberikan informasi ini kepada Anda CloudWatch melalui aturan Contributor Insights, laporan, dan grafik data laporan.

CloudWatch Contributor Insights for DynamoDB dirancang agar tidak memiliki dampak kinerja pada tabel DynamoDB Anda.

Untuk informasi selengkapnya tentang Wawasan CloudWatch Kontributor, lihat Menggunakan Wawasan Kontributor untuk menganalisis data kardinalitas tinggi di Panduan Pengguna Amazon. CloudWatch

Bagian berikut menjelaskan konsep inti dan perilaku CloudWatch Contributor Insights untuk DynamoDB.

Topik

- CloudWatch wawasan kontributor untuk aturan DynamoDB
- Memahami wawasan CloudWatch kontributor untuk grafik DynamoDB
- Interaksi dengan fitur DynamoDB lain
- CloudWatch wawasan kontributor untuk penagihan DynamoDB

CloudWatch wawasan kontributor untuk aturan DynamoDB

Saat Anda mengaktifkan CloudWatch Contributor Insights for DynamoDB pada tabel atau indeks sekunder global, DynamoDB akan membuat aturan berikut atas nama Anda:

- Item yang paling sering diakses (kunci partisi) — Mengidentifikasi kunci partisi dari item yang paling sering diakses di tabel Anda atau indeks sekunder global.

  CloudWatch format nama aturan: `DynamoDBContributorInsights-PKC-[resource_name]-[creationtimestamp]`

- Item yang paling sering mengalami throttling (kunci partisi) — Mengidentifikasi kunci partisi dari item yang paling sering mengalami throttling di tabel atau indeks sekunder global Anda.

  CloudWatch format nama aturan: `DynamoDBContributorInsights-PKT-[resource_name]-[creationtimestamp]`

> **Note**
>
> Saat mengaktifkan Contributor Insights di tabel DynamoDB, Anda masih tunduk pada batas aturan Contributor Insights. Untuk informasi selengkapnya, lihat CloudWatch service quotas.

Jika tabel atau indeks sekunder global Anda memiliki kunci urutan, DynamoDB juga membuat aturan berikut khusus untuk kunci urutan:

- Kunci yang paling sering diakses (kunci partisi dan urutan) — Mengidentifikasi kunci partisi dan urutan dari item yang paling sering diakses di tabel atau indeks sekunder global Anda.

  CloudWatch format nama aturan: `DynamoDBContributorInsights-SKC-[resource_name]-[creationtimestamp]`

- Kunci yang paling sering mengalami throttling (kunci partisi dan urutan) — Mengidentifikasi kunci partisi dan urutan dari item yang paling sering mengalami throttling di tabel atau indeks sekunder global Anda.

  CloudWatch format nama aturan: `DynamoDBContributorInsights-SKT-[resource_name]-[creationtimestamp]`

> **Note**
>
> - Anda tidak dapat menggunakan CloudWatch konsol atau secara langsung mengubah atau APIs menghapus aturan yang dibuat oleh CloudWatch Contributor Insights for DynamoDB. Menonaktifkan CloudWatch Contributor Insights untuk DynamoDB pada tabel atau indeks sekunder global secara otomatis menghapus aturan yang dibuat untuk tabel tersebut atau indeks sekunder global.
> - Bila Anda menggunakan GetInsightRuleReportoperasi dengan aturan CloudWatch Contributor Insights yang dibuat oleh DynamoDB, hanya dan mengembalikan statistik

yang berguna. `MaxContributorValue` `Maximum` Statistik lain dalam daftar ini tidak mengembalikan nilai yang berarti.

- CloudWatch Contributor Insights untuk DynamoDB memiliki batas 25 kontributor. Meminta lebih dari 25 kontributor akan mengembalikan kesalahan.

Anda dapat membuat CloudWatch Alarm menggunakan CloudWatch Contributor Insights for DynamoDB rules. Hal ini memungkinkan Anda menerima pemberitahuan jika ada item yang melebihi atau mencapai ambang batas tertentu untuk `ConsumedThroughputUnits` atau `ThrottleCount`. Untuk informasi selengkapnya, lihat Menyetel alarm pada data metrik Contributor Insights.

Memahami wawasan CloudWatch kontributor untuk grafik DynamoDB

CloudWatch Contributor Insights for DynamoDB menampilkan dua jenis grafik pada DynamoDB dan konsol: Item Paling Banyak Diakses dan Item Paling Dibatasi. CloudWatch

Item paling sering diakses

Gunakan grafik ini untuk mengidentifikasi item yang paling sering diakses dalam tabel atau indeks sekunder global. Grafik ini menampilkan `ConsumedThroughputUnits` pada sumbu y dan waktu pada sumbu x. Setiap kunci N teratas ditampilkan dalam warnanya sendiri, dengan keterangan ditampilkan di bawah sumbu x.

DynamoDB mengukur frekuensi akses utama menggunakan `ConsumedThroughputUnits`, yang mengukur gabungan lalu lintas baca dan tulis. `ConsumedThroughputUnits` didefinisikan sebagai berikut:

- Disediakan — (3 x unit kapasitas tulis yang disediakan) + untit kapasitas tulis yang digunakan
- Sesuai permintaan — (3 x unit permintaan tulis) + unit permintaan baca

Pada konsol DynamoDB, setiap titik data dalam grafik mewakili `ConsumedThroughputUnits` maksimum selama periode 1 menit. Misalnya, nilai grafik 180.000 `ConsumedThroughputUnits` menunjukkan bahwa item diakses terus-menerus pada throughput maksimum per item sebanyak 1.000 permintaan unit tulis atau 3.000 unit permintaan baca untuk rentang 60 detik dalam periode 1 menit tersebut (3.000 x 60 detik). Dengan kata lain, nilai bergrafik mewakili menit lalu lintas tertinggi dalam setiap periode 1 menit. Anda dapat mengubah perincian waktu `ConsumedThroughputUnits` metrik (misalnya, untuk melihat metrik 5 menit, bukan 1 menit) di konsol. CloudWatch

Jika Anda melihat beberapa baris berklaster rapat tanpa outlier jelas, ini menunjukkan bahwa beban kerja Anda relatif seimbang di item pada periode waktu tertentu. Jika Anda melihat titik terisolasi dalam grafik, bukan garis terhubung, ini menunjukkan item yang sering diakses hanya untuk jangka waktu singkat.

Jika tabel atau indeks sekunder global Anda memiliki kunci urutan, DynamoDB membuat dua grafik: satu untuk kunci partisi yang paling sering diakses dan satu untuk pasangan kunci partisi + urutan yang paling sering diakses. Anda dapat melihat lalu lintas pada tingkat kunci partisi di grafik khusus kunci partisi. Anda dapat melihat lalu lintas pada tingkat item di grafik kunci partisi + urutan.

Item yang paling sering mengalami throttling

Gunakan grafik ini untuk mengidentifikasi item yang paling sering mengalami throttling dalam tabel atau indeks sekunder global. Grafik ini menampilkan `ThrottleCount` pada sumbu y dan waktu pada sumbu x. Masing-masing tombol N atas ditampilkan dalam warnanya sendiri, dengan legenda ditampilkan di bawah sumbu x.

DynamoDB mengukur frekuensi throttling menggunakan `ThrottleCount`, yang merupakan jumlah kesalahan `ProvisionedThroughputExceededException`, `ThrottlingException`, dan `RequestLimitExceeded`.

Penulisan throttling yang disebabkan oleh kapasitas tulis yang tidak memadai untuk indeks sekunder global tidak diukur. Anda dapat menggunakan grafik Item Paling Sering Diakses pada indeks sekunder global untuk mengidentifikasi pola akses tidak seimbang yang dapat menyebabkan throttling tulis. Untuk informasi selengkapnya, lihat Pertimbangan throughput yang disediakan untuk Indeks Sekunder Global.

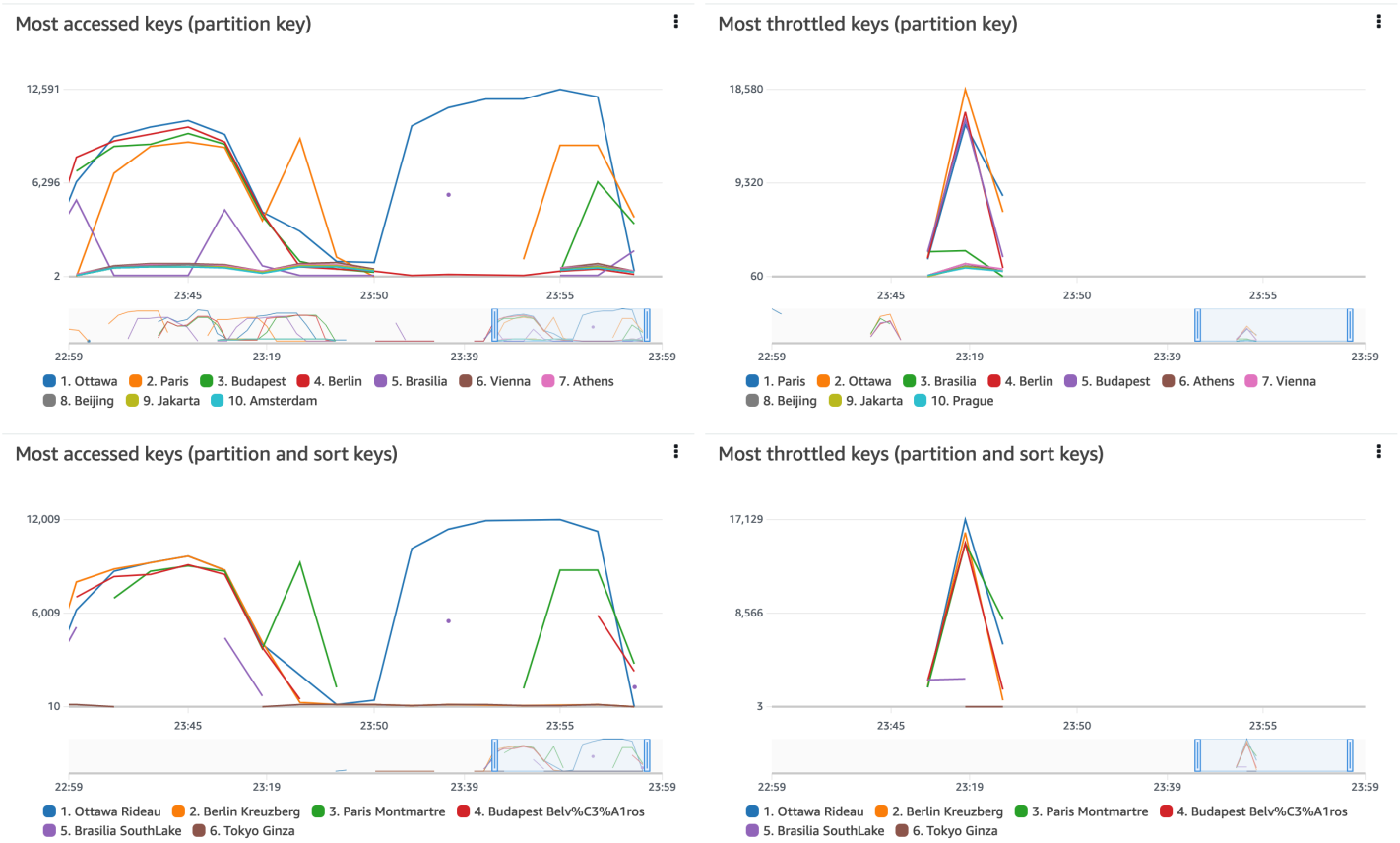
Pada konsol DynamoDB, setiap titik data dalam grafik menunjukkan jumlah peristiwa throttling selama periode 1 menit.

Jika Anda tidak melihat data dalam grafik ini, artinya permintaan Anda tidak mengalami throttling. Jika Anda melihat titik terisolasi dalam grafik, bukan garis yang terhubung, ini menunjukkan bahwa item sering mengalami throttling untuk jangka waktu singkat.

Jika tabel atau indeks sekunder global Anda memiliki kunci urutan, DynamoDB membuat dua grafik: satu untuk kunci partisi yang paling sering mengalami throttling dan satu untuk pasangan kunci partisi + urutan yang paling sering mengalami throttling. Anda dapat melihat jumlah throttling pada tingkat kunci partisi pada grafik khusus kunci partisi, dan jumlah throttling pada tingkat item pada grafik kunci partisi + urutan.

Contoh laporan

Berikut ini adalah contoh laporan yang dihasilkan untuk tabel dengan kunci partisi dan kunci urutan.



Interaksi dengan fitur DynamoDB lain

Bagian berikut menjelaskan bagaimana CloudWatch Contributor Insights for DynamoDB berperilaku dan berinteraksi dengan beberapa fitur lain di DynamoDB.

Tabel global

CloudWatch Contributor Insights for DynamoDB memonitor replika tabel global sebagai tabel yang berbeda. Grafik Contributor Insights untuk replika di satu AWS Wilayah mungkin tidak menunjukkan pola yang sama dengan Wilayah lain. Hal ini karena data tulis direplikasi di semua replika dalam tabel global, tetapi setiap replika dapat melayani lalu lintas baca yang terikat Wilayah.

DynamoDB Accelerator (DAX)

CloudWatch Contributor Insights untuk DynamoDB tidak menampilkan respons cache DAX. Wawasan Kontributor tersebut hanya menampilkan respons untuk mengakses tabel atau indeks sekunder global.

> **Note**
>
> CloudWatch DynamoDB Contributor Insights tidak mendukung permintaan PartiQL.

Enkripsi diam

CloudWatch Contributor Insights untuk DynamoDB tidak memengaruhi cara kerja enkripsi di DynamoDB. Data kunci utama yang dipublikasikan di CloudWatch dienkripsi dengan file. Kunci milik AWS Namun, DynamoDB juga mendukung dan kunci Kunci yang dikelola AWS yang dikelola pelanggan.

CloudWatch Contributor Insights for DynamoDB menampilkan kunci partisi dan kunci sortir (jika ada) dari item yang sering diakses dan dibatasi. Meskipun CloudWatch Contributor Insights bekerja dengan tabel DynamoDB terenkripsi, penting untuk dicatat bahwa Contributor Insights menggunakan konteks enkripsi milik Amazon sendiri, yang terpisah dari enkripsi tabel yang dikonfigurasi.

Jika kunci utama tabel DynamoDB berisi informasi sensitif dan kebijakan keamanan organisasi Anda memerlukan kontrol penuh atas proses enkripsi, CloudWatch mengaktifkan Contributor Insights mungkin tidak cocok.

Kontrol akses detail

CloudWatch Contributor Insights for DynamoDB tidak berfungsi secara berbeda untuk tabel dengan kontrol akses halus (FGAC). Dengan kata lain, setiap pengguna yang memiliki CloudWatch izin yang sesuai dapat melihat kunci utama yang dilindungi FGAC dalam grafik Contributor Insights. CloudWatch

Jika kunci utama tabel berisi data yang dilindungi FGAC yang tidak ingin Anda publikasikan CloudWatch, Anda tidak boleh mengaktifkan CloudWatch Contributor Insights for DynamoDB untuk tabel tersebut.

Kontrol akses

Anda mengontrol akses ke CloudWatch Contributor Insights for DynamoDB menggunakan (AWS Identity and Access Management IAM) dengan membatasi izin bidang kontrol DynamoDB dan izin bidang data. CloudWatch Untuk informasi selengkapnya lihat, Menggunakan IAM dengan CloudWatch Contributor Insights for DynamoDB.

CloudWatch wawasan kontributor untuk penagihan DynamoDB

Biaya untuk Wawasan CloudWatch Kontributor untuk DynamoDB muncul di bagian CloudWatchtagihan bulanan Anda. Biaya ini dihitung berdasarkan jumlah peristiwa DynamoDB yang diproses. Untuk tabel dan indeks sekunder global dengan CloudWatch Contributor Insights for DynamoDB diaktifkan, setiap item yang ditulis atau dibaca melalui operasi bidang data mewakili satu peristiwa.

Jika tabel atau indeks sekunder global menyertakan kunci urutan, setiap item yang dibaca atau ditulis mewakili dua peristiwa. Hal ini karena DynamoDB mengidentifikasi kontributor teratas dari rangkaian waktu terpisah: satu untuk kunci partisi saja, dan satu untuk pasangan kunci partisi dan urutan.

Sebagai contoh, misalkan aplikasi Anda melakukan operasi DynamoDB berikut: `GetItem`, `PutItem`, dan `BatchWriteItem` yang memasukkan lima item

- Jika tabel atau indeks sekunder global Anda hanya memiliki kunci partisi, hasilnya adalah 7 peristiwa (1 untuk `GetItem`, 1 untuk `PutItem`, dan 5 untuk `BatchWriteItem`).
- Jika tabel atau indeks sekunder global Anda memiliki kunci partisi dan kunci urutan, hasilnya adalah 14 peristiwa (2 untuk `GetItem`, 2 untuk `PutItem`, dan 10 untuk `BatchWriteItem`).
- Operasi `Query` selalu menghasilkan 1 peristiwa, terlepas dari jumlah item yang dikembalikan.

Tidak seperti fitur DynamoDB lainnya CloudWatch , Contributor Insights untuk penagihan DynamoDB tidak bervariasi berdasarkan hal berikut:

- Mode kapasitas (disediakan vs. sesuai permintaan)
- Baik jika Anda melakukan permintaan baca atau tulis
- Ukuran (KB) item yang dibaca atau ditulis

Memulai CloudWatch Contributor Insights for DynamoDB

Bagian ini menjelaskan cara menggunakan Amazon CloudWatch Contributor Insights dengan konsol Amazon DynamoDB atau (). AWS Command Line Interface AWS CLI

Dalam contoh berikut, Anda menggunakan tabel DynamoDB yang didefinisikan dalam tutorial Mulai menggunakan DynamoDB.

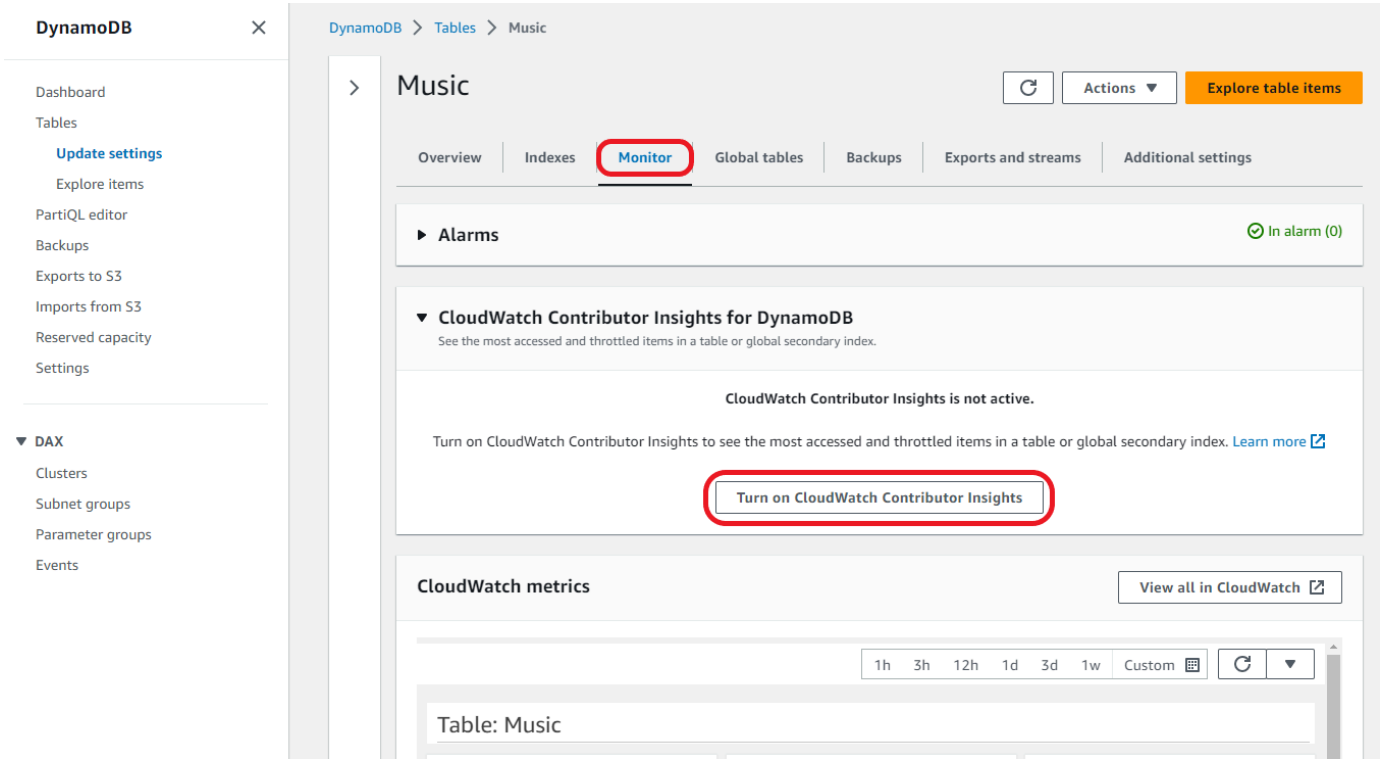
Topik

- Menggunakan Wawasan Kontributor (konsol)
- Menggunakan Wawasan Kontributor (AWS CLI)

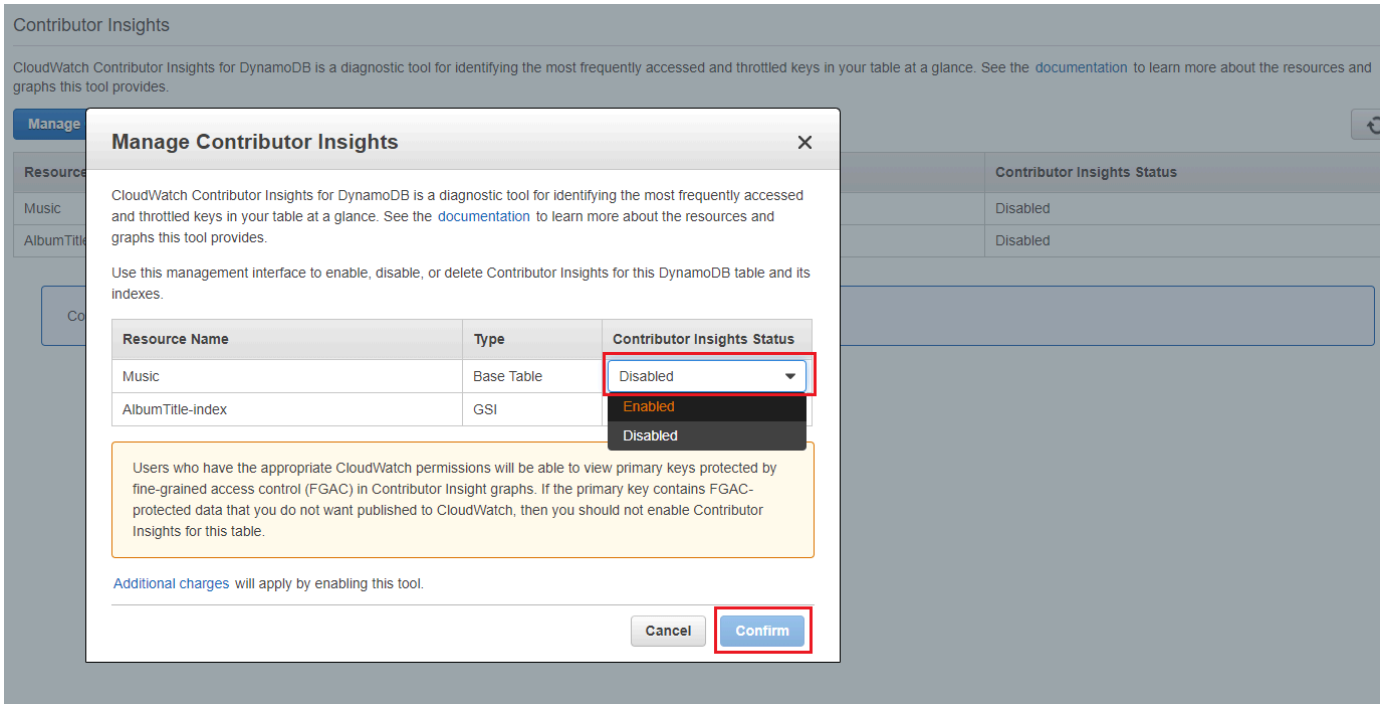
Menggunakan Wawasan Kontributor (konsol)

Untuk menggunakan Contributor Insights di konsol

1. Masuk ke AWS Management Console dan buka konsol DynamoDB di. https://console.aws.amazon.com/dynamodb/
2. Di panel navigasi di sisi kiri konsol, pilih Tabel.
3. Pilih tabel `Music`.
4. Pilih tab Pantau.
5. Pilih Aktifkan Wawasan CloudWatch Kontributor.

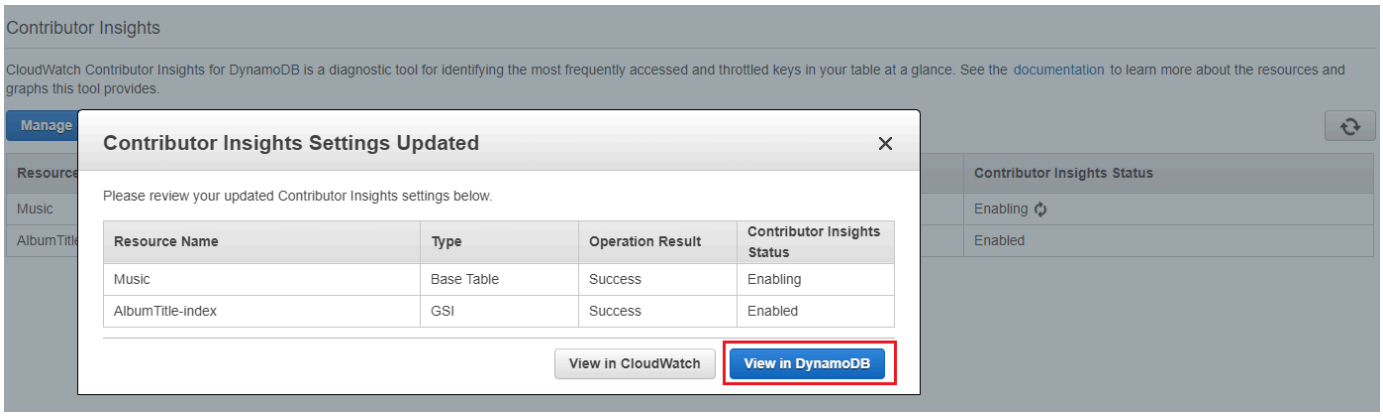


6. Di kotak dialog Kelola Wawasan Kontributor, di bagian Status Wawasan Kontributor, pilih Diaktifkan untuk tabel dasar `Music` dan indeks sekunder global `AlbumTitle-index`. Kemudian pilih Konfirmasi.

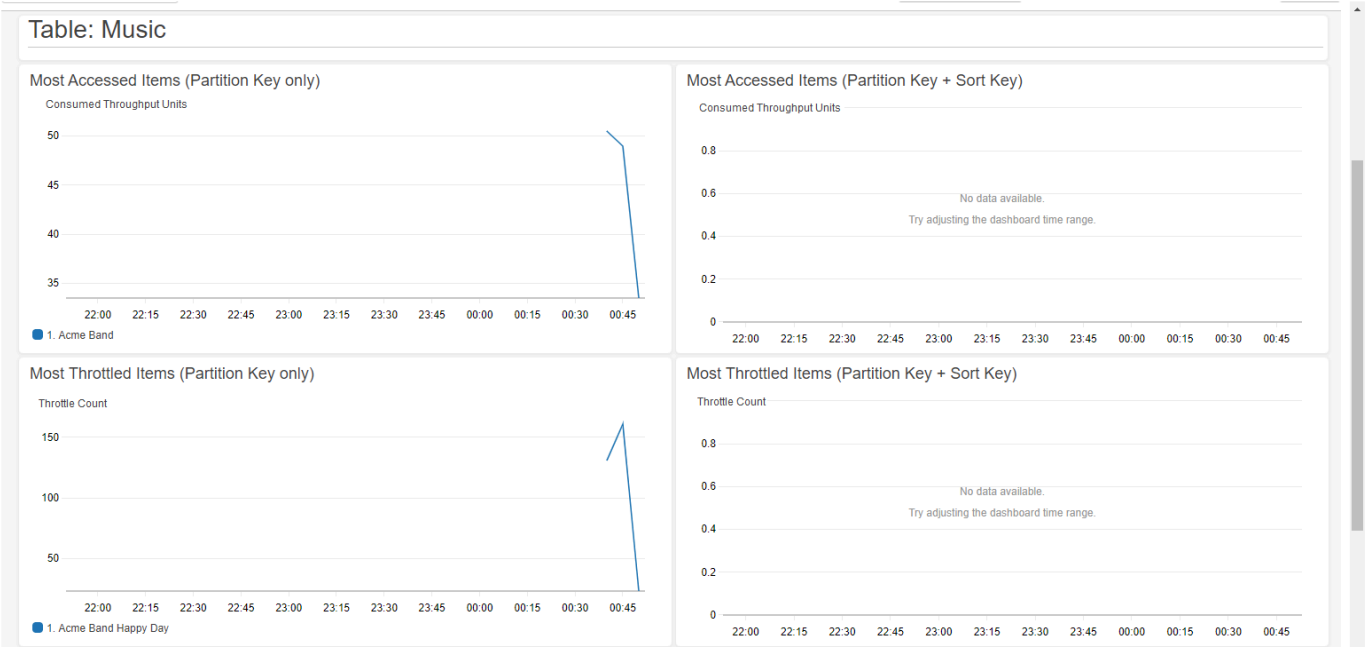


Jika operasi gagal, lihat DescribeContributorInsights FailureExceptiondi Referensi Amazon DynamoDB API untuk alasan yang mungkin.

7. Pilih Lihat di DynamoDB.



8. Grafik Wawasan Kontributor sekarang ditampilkan di tab Wawasan Kontributor untuk tabel `Music`.

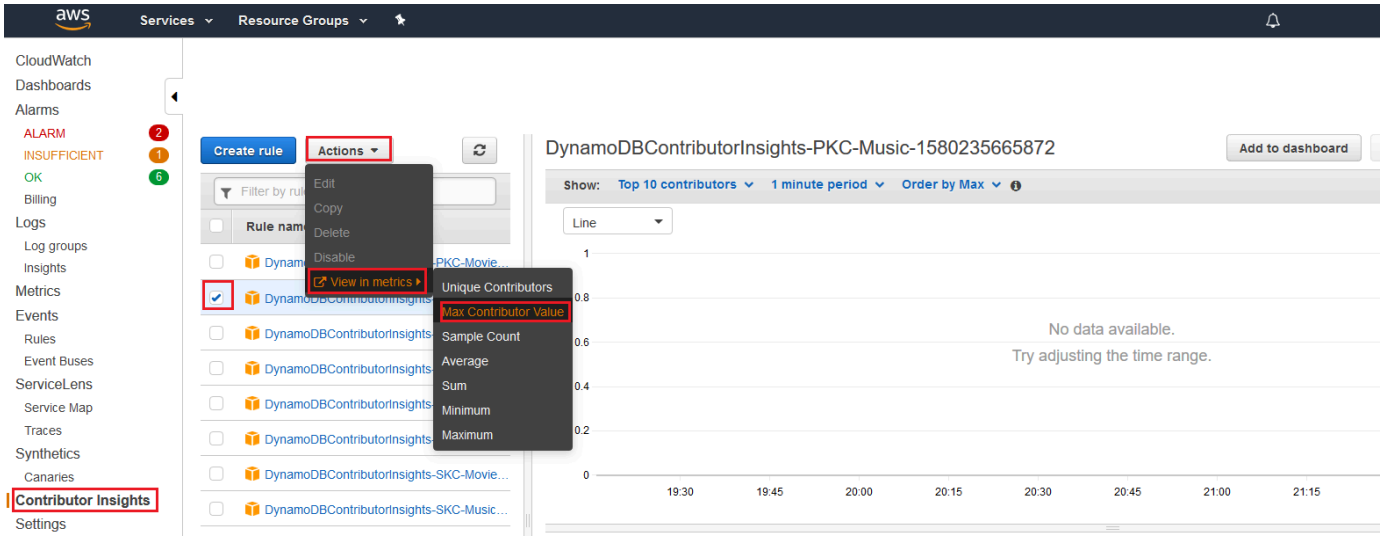


Membuat CloudWatch alarm

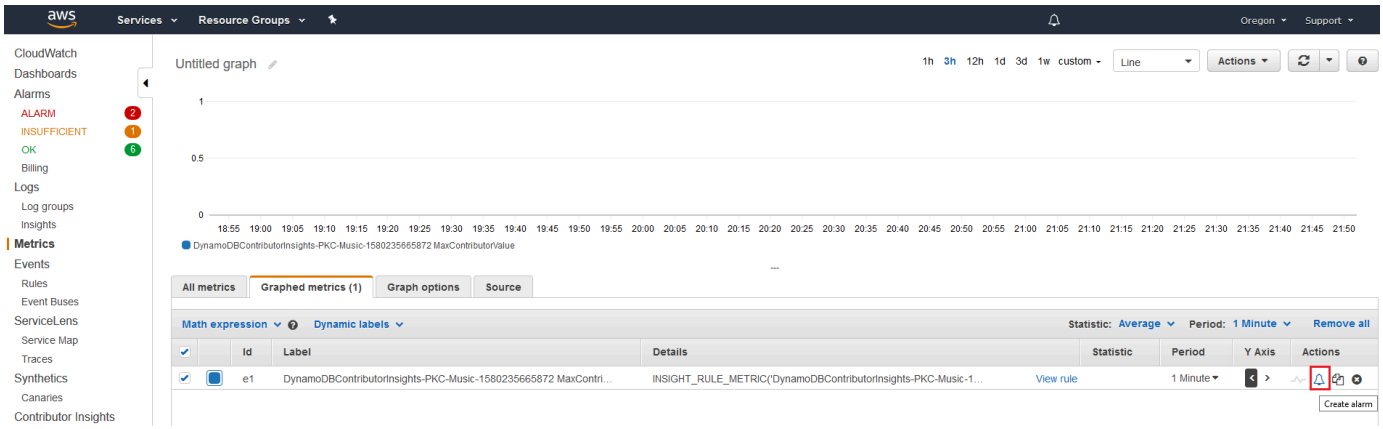
Ikuti langkah-langkah ini untuk membuat CloudWatch alarm dan diberi tahu ketika kunci partisi apa pun menghabiskan lebih dari 50.000. ConsumedThroughputUnits

1. Masuk ke AWS Management Console dan buka CloudWatch konsol di https://console.aws.amazon.com/cloudwatch/
2. Di panel navigasi di sisi kiri konsol, pilih Wawasan Kontributor.
3. Pilih aturan Dynamo DBContributor Insights-PKC-Music.
4. Pilih drop down Tindakan.
5. Pilih Lihat dalam metrik.
6. Pilih Nilai Kontributor Maks.

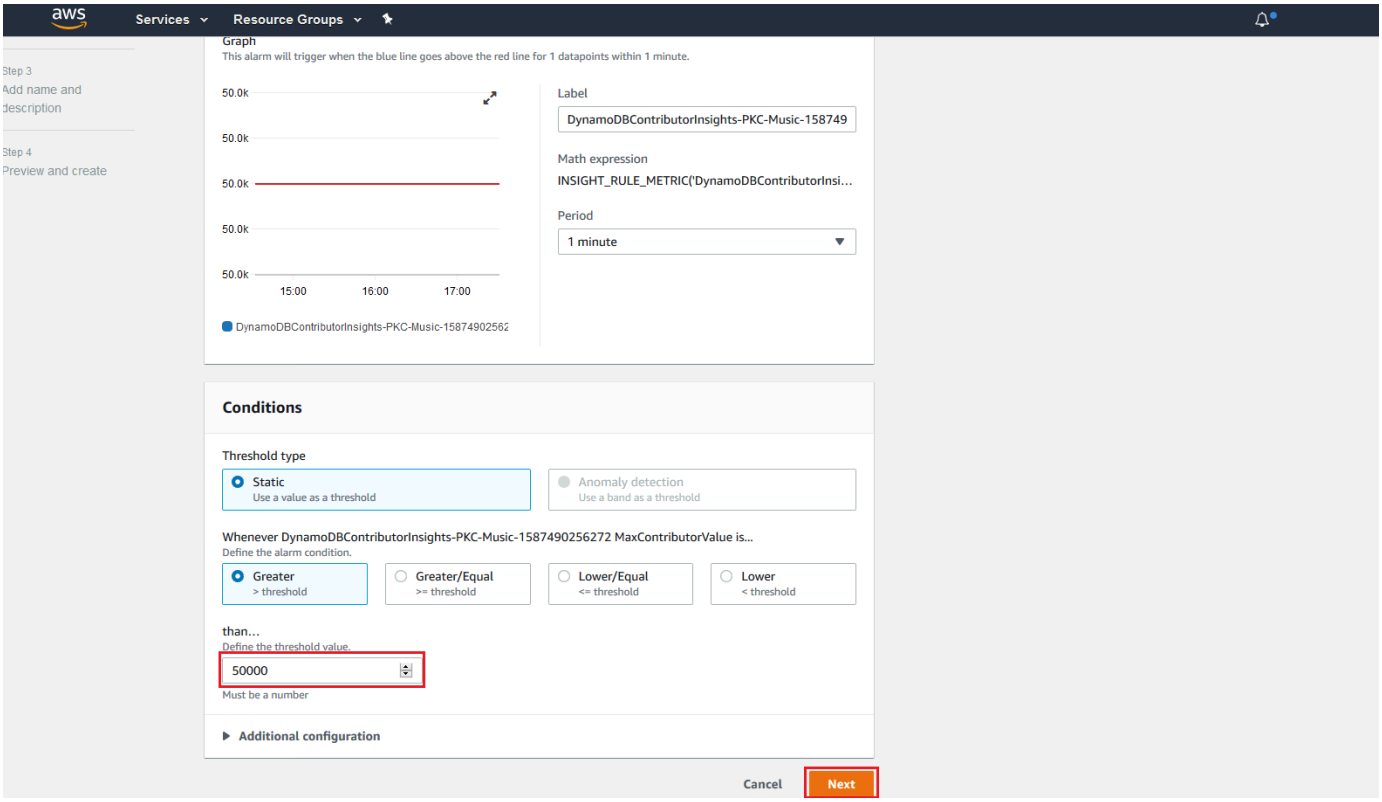
> **Note**
>
> Hanya `Max Contributor Value` dan `Maximum` yang mengembalikan statistik yang berguna. Statistik lain dalam daftar ini tidak mengembalikan nilai yang berarti.



7. Di kolom Tindakan, Pilih Buat Alarm.



8. Masukkan nilai 50000 untuk ambang batas dan pilih Selanjutnya.



9. Lihat Menggunakan CloudWatch alarm Amazon untuk detail tentang cara mengonfigurasi notifikasi untuk alarm.

Menggunakan Wawasan Kontributor (AWS CLI)

Untuk menggunakan Wawasan Kontributor di AWS CLI

1. Aktifkan CloudWatch Contributor Insights untuk DynamoDB pada tabel dasar. `Music`

```
aws dynamodb update-contributor-insights --table-name Music --contributor-insights-action=ENABLE
```

2. Aktifkan Wawasan Kontributor untuk DynamoDB di indeks sekunder global `AlbumTitle-index`.

```
aws dynamodb update-contributor-insights --table-name Music --index-name
 AlbumTitle-index --contributor-insights-action=ENABLE
```

3. Dapatkan status dan aturan untuk tabel `Music` dan semua indeksnya.

```
aws dynamodb describe-contributor-insights --table-name Music
```

4. Nonaktifkan CloudWatch Contributor Insights untuk DynamoDB pada indeks sekunder global. `AlbumTitle-index`

```
aws dynamodb update-contributor-insights --table-name Music --index-name
 AlbumTitle-index --contributor-insights-action=DISABLE
```

5. Dapatkan status tabel `Music` dan semua indeksnya.

```
aws dynamodb list-contributor-insights --table-name Music
```

Menggunakan IAM dengan wawasan CloudWatch kontributor untuk DynamoDB

Pertama kali Anda mengaktifkan Amazon CloudWatch Contributor Insights untuk Amazon DynamoDB, DynamoDB secara otomatis membuat peran terkait layanan (IAM) untuk Anda. AWS Identity and Access Management Peran ini,`AWSServiceRoleForDynamoDBCloudWatchContributorInsights`, memungkinkan DynamoDB CloudWatch mengelola aturan Contributor Insights atas nama Anda. Jangan hapus peran tertaut layanan ini. Jika Anda menghapusnya, semua aturan terkelola tidak akan lagi dibersihkan saat Anda menghapus tabel atau indeks sekunder global.

Untuk informasi selengkapnya tentang peran tertaut layanan, lihat Menggunakan peran tertaut layanan di Panduan Pengguna IAM.

Izin berikut diperlukan:

- Untuk mengaktifkan atau menonaktifkan CloudWatch Contributor Insights for DynamoDB, Anda harus memiliki `dynamodb:UpdateContributorInsights` izin pada tabel atau indeks.
- Untuk melihat CloudWatch Contributor Insights untuk grafik DynamoDB, Anda harus memiliki izin. `cloudwatch:GetInsightRuleReport`
- Untuk menjelaskan CloudWatch Contributor Insights for DynamoDB untuk tabel atau indeks DynamoDB tertentu, Anda harus memiliki izin. `dynamodb:DescribeContributorInsights`
- Untuk mencantumkan CloudWatch Contributor Insights untuk status DynamoDB untuk setiap tabel dan indeks sekunder global, Anda harus memiliki izin. `dynamodb:ListContributorInsights`

Contoh: Mengaktifkan atau menonaktifkan wawasan CloudWatch kontributor untuk DynamoDB

Kebijakan IAM berikut memberikan izin untuk mengaktifkan atau menonaktifkan CloudWatch Contributor Insights for DynamoDB.

```
{
    "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/*"
        }
    ]
}
```

Untuk tabel yang dienkripsi oleh kunci KMS, pengguna harus memiliki izin kms: Dekripsi untuk memperbarui Wawasan Kontributor.

```
{
    "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"
            ],
        }
    ]
}
```

Contoh: Ambil laporan aturan wawasan CloudWatch kontributor

Kebijakan IAM berikut memberikan izin untuk mengambil laporan aturan CloudWatch Contributor Insights.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "cloudwatch:GetInsightRuleReport"
            ],
            "Resource": "arn:aws:cloudwatch:*:*:insight-rule/
DynamoDBContributorInsights*"
        }
    ]
}
```

Contoh: Terapkan wawasan CloudWatch kontributor secara selektif untuk izin DynamoDB berdasarkan sumber daya

Kebijakan IAM berikut memberikan izin untuk memperbolehkan tindakan `ListContributorInsights` dan `DescribeContributorInsights` serta menolak tindakan `UpdateContributorInsights` untuk indeks sekunder global tertentu.

```
{
    "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"
        }
    ]
}
```

Menggunakan peran terkait layanan untuk CloudWatch Contributor Insights untuk DynamoDB

CloudWatch Contributor Insights for DynamoDB menggunakan AWS Identity and Access Management peran terkait layanan (IAM). Peran terkait layanan adalah jenis unik peran IAM yang ditautkan langsung ke CloudWatch Contributor Insights for DynamoDB. Peran terkait layanan telah ditentukan sebelumnya oleh CloudWatch Contributor Insights for DynamoDB dan menyertakan semua izin yang diperlukan layanan untuk memanggil layanan lain atas nama Anda. AWS

Peran terkait layanan membuat pengaturan CloudWatch Contributor Insights untuk DynamoDB lebih mudah karena Anda tidak perlu menambahkan izin yang diperlukan secara manual. CloudWatch Contributor Insights for DynamoDB mendefinisikan izin dari peran terkait layanannya, dan kecuali

ditentukan lain, hanya Contributor Insights untuk DynamoDB yang dapat mengambil perannya. CloudWatch Izin yang ditentukan mencakup kebijakan kepercayaan dan kebijakan izin, serta bahwa kebijakan izin tidak dapat dilampirkan ke entitas IAM lainnya.

Untuk informasi tentang layanan lain yang mendukung peran terkait layanan, lihat Layanan AWS yang Berfungsi dengan IAM dan cari layanan yang memiliki Ya di kolom Peran Terkait Layanan. Pilih Ya bersama tautan untuk melihat dokumentasi peran tertaut layanan untuk layanan tersebut.

Izin peran terkait layanan untuk CloudWatch Contributor Insights untuk DynamoDB

CloudWatch Contributor Insights untuk DynamoDB menggunakan peran terkait layanan bernama. AWSServiceRoleForDynamoDBCloudWatchContributorInsights Tujuan dari peran terkait layanan adalah untuk mengizinkan Amazon DynamoDB mengelola CloudWatch aturan Amazon Contributor Insights yang dibuat untuk tabel DynamoDB dan indeks sekunder global, atas nama Anda.

`AWSServiceRoleForDynamoDBCloudWatchContributorInsights` peran terkait layanan memercayakan layanan berikut untuk menjalankan peran tersebut:

- `contributorinsights.dynamodb.amazonaws.com`

Kebijakan izin peran memungkinkan CloudWatch Contributor Insights for DynamoDB menyelesaikan tindakan berikut pada sumber daya yang ditentukan:

- Tindakan: `Create and manage Insight Rules` pada `DynamoDBContributorInsights`

Anda harus mengonfigurasikan izin untuk mengizinkan entitas IAM (seperti pengguna, grup, atau peran) untuk membuat, menyunting, atau menghapus peran terhubung dengan layanan. Untuk informasi selengkapnya, silakan lihat Izin Peran Tertaut Layanan di Panduan Pengguna IAM.

Membuat peran terkait layanan untuk CloudWatch Contributor Insights untuk DynamoDB

Anda tidak perlu membuat peran terkait layanan secara manual. Saat Anda mengaktifkan Contributor Insights di AWS Management Console, the, atau AWS API AWS CLI, CloudWatch Contributor Insights for DynamoDB akan membuat peran terkait layanan untuk Anda.

Jika Anda menghapus peran tertaut layanan ini, dan ingin membuatnya lagi, Anda dapat mengulangi proses yang sama untuk membuat kembali peran tersebut di akun Anda. Saat Anda mengaktifkan Contributor Insights, CloudWatch Contributor Insights for DynamoDB membuat peran terkait layanan untuk Anda lagi.

Mengedit peran terkait layanan untuk CloudWatch Contributor Insights for DynamoDB

CloudWatch Contributor Insights for DynamoDB tidak mengizinkan Anda mengedit peran terkait layanan. `AWSServiceRoleForDynamoDBCloudWatchContributorInsights` Setelah Anda membuat peran terkait layanan, Anda tidak dapat mengubah nama peran karena berbagai entitas mungkin mereferensikan peran tersebut. Namun, Anda dapat mengedit penjelasan peran menggunakan IAM. Untuk informasi selengkapnya, lihat Mengedit peran tertaut layanan dalam Panduan Pengguna IAM.

Menghapus peran terkait layanan untuk CloudWatch Contributor Insights for DynamoDB

Anda tidak perlu menghapus peran `AWSServiceRoleForDynamoDBCloudWatchContributorInsights` secara manual. Saat Anda menonaktifkan Contributor Insights di AWS Management Console, the, atau AWS API AWS CLI, CloudWatch Contributor Insights for DynamoDB membersihkan sumber daya.

Anda juga dapat menggunakan konsol IAM, AWS CLI atau AWS API untuk menghapus peran terkait layanan secara manual. Untuk melakukannya, Anda harus membersihkan sumber daya untuk peran tertaut layanan terlebih dahulu, lalu Anda dapat menghapusnya secara manual.

> **Note**
>
> Jika layanan CloudWatch Contributor Insights for DynamoDB menggunakan peran saat Anda mencoba menghapus sumber daya, maka penghapusan mungkin gagal. Jika hal itu terjadi, tunggu beberapa menit dan coba mengoperasikannya lagi.

Untuk menghapus peran terkait layanan secara manual menggunakan IAM

Gunakan konsol IAM, the AWS CLI, atau AWS API untuk menghapus peran `AWSServiceRoleForDynamoDBCloudWatchContributorInsights` terkait layanan. Untuk informasi selengkapnya, lihat Menghapus peran tertaut layanan dalam Panduan Pengguna IAM.

Praktik terbaik untuk mendesain dan merancang dengan DynamoDB

Gunakan bagian ini untuk menemukan rekomendasi dengan cepat untuk memaksimalkan kinerja dan meminimalkan biaya throughput saat bekerja dengan DynamoDB.

Topik

- Desain NoSQL untuk DynamoDB
- Menggunakan DynamoDB Well-Architected Lens untuk mengoptimalkan beban kerja DynamoDB Anda
- Praktik terbaik untuk merancang dan menggunakan kunci partisi secara efektif di DynamoDB
- Praktik terbaik untuk menggunakan kunci sortir untuk mengatur data di DynamoDB
- Praktik terbaik untuk menggunakan indeks sekunder di DynamoDB
- Praktik terbaik untuk menyimpan item dan atribut besar di DynamoDB
- Praktik terbaik untuk penanganan data deret waktu di DynamoDB
- Praktik terbaik untuk mengelola many-to-many hubungan dalam tabel DynamoDB
- Praktik terbaik untuk kueri dan pemindaian data di DynamoDB
- Praktik terbaik untuk desain tabel DynamoDB
- Praktik terbaik untuk desain tabel global DynamoDB
- Praktik terbaik untuk mengelola bidang kontrol di DynamoDB
- Praktik terbaik untuk menggunakan operasi data massal di DynamoDB
- Praktik terbaik untuk menerapkan kontrol versi di DynamoDB
- Praktik terbaik untuk memahami laporan AWS penagihan dan penggunaan Anda di DynamoDB
- Migrasi tabel DynamoDB dari satu akun ke akun lainnya
- Panduan preskriptif untuk mengintegrasikan DAX dengan aplikasi DynamoDB
- Pertimbangan saat menggunakan AWS PrivateLink untuk Amazon DynamoDB

Desain NoSQL untuk DynamoDB

Sistem basis data NoSQL seperti Amazon DynamoDB menggunakan model alternatif untuk manajemen data, seperti pasangan nilai kunci atau penyimpanan dokumen. Saat Anda beralih dari sistem manajemen basis data relasional ke sistem basis data NoSQL seperti DynamoDB, penting untuk memahami perbedaan utama dan pendekatan desain tertentu.

Topik

- Perbedaan antara desain data relasional dan NoSQL
- Dua konsep utama untuk desain NoSQL
- Mendekati desain NoSQL
- NoSQL Workbench untuk DynamoDB

Perbedaan antara desain data relasional dan NoSQL

Sistem basis data relasional (RDBMS) dan basis data NoSQL memiliki keunggulan dan kelemahan yang berbeda:

- Di RDBMS, data dapat dikueri secara fleksibel, tetapi kueri relaitf mahal dan tidak dapat diskalakan dengan baik dalam situasi lalu lintas tinggi (lihat Langkah pertama untuk memodelkan data relasional di DynamoDB).
- Dalam basis data NoSQL seperti DynamoDB, data dapat dikueri secara efisien dalam sejumlah cara terbatas, di luar itu kueri bisa jadi mahal dan lambat.

Perbedaan ini membuat desain basis data menjadi berbeda di antara kedua sistem:

- Di RDBMS, Anda mendesain untuk fleksibilitas tanpa perlu mengkhawatirkan detail penerapan atau performa. Optimasi kueri umumnya tidak memengaruhi desain skema, tetapi normalisasi itu penting.
- Di DynamoDB, Anda mendesain skema secara spesifik untuk membuat kueri yang paling umum dan penting seefisien dan seterjangkau mungkin. Struktur data Anda disesuaikan dengan kebutuhan spesifik kasus penggunaan bisnis Anda.

Dua konsep utama untuk desain NoSQL

Desain NoSQL membutuhkan pola pikir yang berbeda dari desain RDBMS. Untuk RDBMS, Anda dapat melanjutkan dan membuat model data yang dinormalisasi tanpa memikirkan pola akses. Anda kemudian dapat memperluasnya nanti ketika ada pertanyaan dan persyaratan kueri baru. Anda dapat mengatur setiap jenis data ke dalam tabelnya sendiri.

Perbedaan desain NoSQL

- Sebaliknya, Anda tidak boleh mulai merancang skema untuk DynamoDB sampai Anda mengetahui pertanyaan yang perlu dijawab. Memahami masalah bisnis dan kasus penggunaan aplikasi di awal sangat penting.
- Anda harus mempertahankan tabel sesedikit mungkin dalam aplikasi DynamoDB. Memiliki lebih sedikit tabel membuat segala sesuatunya lebih terukur, memerlukan lebih sedikit manajemen izin, dan mengurangi overhead untuk aplikasi DynamoDB Anda. Hal ini juga dapat membantu menjaga biaya pencadangan tetap rendah secara keseluruhan.

Mendekati desain NoSQL

Langkah pertama dalam mendesain aplikasi DynamoDB adalah mengidentifikasi pola kueri tertentu yang harus dipenuhi oleh sistem.

Secara khusus, penting untuk memahami tiga properti dasar dari pola akses aplikasi Anda sebelum memulai:

- Ukuran data: Mengetahui jumlah data yang akan disimpan dan diminta pada satu waktu akan membantu menentukan cara paling efektif untuk mempartisi data.
- Bentuk data: Alih-alih membentuk kembali data saat kueri diproses (seperti yang dilakukan sistem RDBMS), basis data NoSQL mengatur data sehingga bentuknya dalam basis data tersebut sesuai dengan apa yang akan dikueri. Ini adalah faktor kunci dalam meningkatkan kecepatan dan skalabilitas.
- Kecepatan data: DynamoDB menskalakan dengan meningkatkan jumlah partisi fisik yang tersedia untuk memproses kueri, dan dengan mendistribusikan data secara efisien ke seluruh partisi tersebut. Mengetahui sebelumnya apa beban kueri puncak mungkin membantu menentukan cara mempartisi data ke I/O kapasitas penggunaan terbaik.

Setelah mengidentifikasi persyaratan kueri tertentu, Anda bisa mengatur data menurut prinsip umum yang mengatur performa:

- Menyimpan data terkait bersama-sama. Penelitian telah menunjukkan bahwa prinsip 'lokalitas referensi', menjaga data terkait bersama-sama di satu tempat, merupakan faktor kunci dalam meningkatkan kinerja dan waktu respons dalam sistem NoSQL, seperti yang ditemukan penting untuk mengoptimalkan tabel perutean bertahun-tahun yang lalu.

  Aturan umumnya, Anda harus mempertahankan tabel sesedikit mungkin dalam aplikasi DynamoDB.

  Pengecualian adalah kasus yang melibatkan data deret waktu bervolume tinggi, atau set data yang memiliki pola akses yang sangat berbeda. Tabel tunggal dengan indeks terbalik biasanya dapat mengaktifkan kueri sederhana untuk membuat dan mengambil struktur data hierarki kompleks yang diperlukan oleh aplikasi Anda.
- Menggunakan urutan. Item terkait dapat dikelompokkan bersama dan dikueri secara efisien jika desain utamanya menyebabkan item tersebut disortir bersama. Ini adalah strategi desain NoSQL yang penting.
- Mendistribusikan kueri. Penting juga bahwa volume kueri yang tinggi tidak difokuskan pada satu bagian dari database, di mana mereka dapat melebihi I/O kapasitas. Sebagai gantinya, Anda harus merancang kunci data untuk mendistribusikan lalu lintas secara merata di seluruh partisi sebanyak mungkin, menghindari hot spot.
- Menggunakan indeks sekunder global. Dengan membuat indeks sekunder global tertentu, Anda dapat mengaktifkan kueri yang berbeda dari yang dapat didukung oleh tabel utama Anda, dan itu masih cepat dan relatif murah.

Prinsip-prinsip umum ini diterjemahkan ke dalam beberapa pola desain umum yang dapat Anda gunakan untuk memodelkan data secara efisien di DynamoDB.

NoSQL Workbench untuk DynamoDB

NoSQL Workbench untuk DynamoDB adalah aplikasi GUI sisi klien lintas platform yang dapat Anda gunakan untuk pengembangan dan operasi basis data modern. Ini tersedia untuk Windows, macOS, dan Linux. NoSQL Workbench adalah alat pengembangan visual yang menyediakan fitur pemodelan data, visualisasi data, pembuatan data sampel, dan pengembangan kueri untuk membantu Anda mendesain, membuat, mengkueri, dan mengelola tabel DynamoDB. Dengan NoSQL Workbench untuk DynamoDB, Anda dapat membuat model data baru dari, atau mendesain model

berdasarkan, model data yang sudah ada yang memenuhi pola akses data aplikasi Anda. Anda juga dapat mengimpor dan mengekspor model data yang didesain pada akhir proses. Untuk informasi selengkapnya, lihat Membangun Model Data dengan NoSQL Workbench.

Menggunakan DynamoDB Well-Architected Lens untuk mengoptimalkan beban kerja DynamoDB Anda

Bagian ini menjelaskan Amazon DynamoDB Well-Architected Lens, kumpulan prinsip desain, dan panduan untuk mendesain beban kerja DynamoDB yang dirancang dengan baik.

Mengoptimalkan biaya pada tabel DynamoDB

Bagian ini mencakup praktik terbaik tentang cara mengoptimalkan biaya untuk tabel DynamoDB yang ada. Anda harus melihat strategi berikut untuk mengetahui strategi pengoptimalan biaya yang paling sesuai dengan kebutuhan Anda dan mendekatinya secara berulang. Setiap strategi akan memberikan gambaran umum tentang hal-hal yang mungkin memengaruhi biaya Anda, tanda-tanda yang harus dicari, dan panduan preskriptif tentang cara menguranginya.

Topik

- Evaluasi biaya Anda di tingkat tabel
- Evaluasi mode kapasitas tabel DynamoDB Anda
- Evaluasi pengaturan penskalaan otomatis tabel DynamoDB Anda
- Evaluasi pilihan kelas tabel DynamoDB Anda
- Identifikasi sumber daya Anda yang tidak digunakan di DynamoDB
- Evaluasi pola penggunaan tabel DynamoDB Anda
- Evaluasi penggunaan aliran DynamoDB Anda
- Evaluasi kapasitas yang disediakan untuk penyediaan ukuran yang tepat di tabel DynamoDB

Evaluasi biaya Anda di tingkat tabel

Alat Cost Explorer yang ditemukan di dalamnya AWS Management Console memungkinkan Anda melihat biaya yang dipecah berdasarkan jenisnya, seperti biaya baca, tulis, penyimpanan, dan cadangan. Anda juga dapat melihat biaya-biaya ini dirangkum berdasarkan periode seperti bulan atau hari.

Salah satu tantangan yang dapat dihadapi administrator adalah ketika biaya hanya pada satu tabel tertentu perlu ditinjau. Beberapa data ini tersedia melalui konsol DynamoDB atau melalui panggilan ke `DescribeTable` API, tetapi Cost Explorer tidak, secara default, memungkinkan Anda untuk memfilter atau mengelompokkan berdasarkan biaya yang terkait dengan tabel tertentu. Bagian ini akan menunjukkan cara menggunakan penandaan untuk melakukan analisis biaya tabel individual di Cost Explorer.

Topik

- Cara melihat biaya tabel DynamoDB tunggal
- Tampilan default Cost Explorer
- Cara menggunakan dan menerapkan tanda tabel di Cost Explorer

Cara melihat biaya tabel DynamoDB tunggal

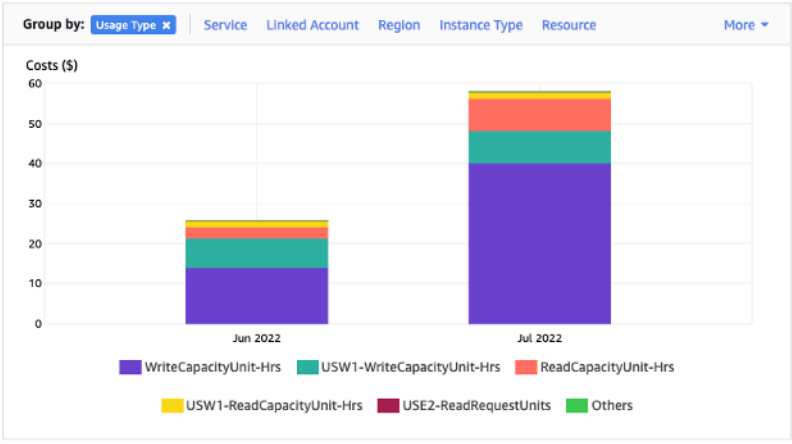
Baik Amazon AWS Management Console DynamoDB dan `DescribeTable` API akan menampilkan informasi tentang satu tabel, termasuk skema kunci utama, indeks apa pun di tabel, dan ukuran dan jumlah item tabel dan indeks apa pun. Ukuran tabel, ditambah ukuran indeks, dapat digunakan untuk menghitung biaya penyimpanan bulanan untuk tabel Anda. Misalnya, $0,25 per GB di wilayah us-east-1.

Jika tabel dalam mode kapasitas yang disediakan, pengaturan RCU dan WCU saat ini juga dikembalikan. Ini dapat digunakan untuk menghitung biaya baca dan tulis saat ini untuk tabel, tetapi biaya ini dapat berubah, terutama jika tabel telah dikonfigurasi dengan Penskalaan Otomatis.

> **Note**
>
> Jika tabel berada dalam mode kapasitas sesuai permintaan, maka `DescribeTable` tidak akan membantu memperkirakan biaya throughput, karena penagihannya berdasarkan penggunaan aktual, tidak disediakan dalam satu periode.

Tampilan default Cost Explorer

Tampilan default Cost Explorer menyediakan bagan yang menunjukkan biaya sumber daya yang digunakan seperti throughput dan penyimpanan. Anda dapat memilih untuk mengelompokkan biaya berdasarkan periode, seperti total berdasarkan bulan atau hari. Biaya penyimpanan, baca, tulis, dan fitur lainnya dapat dipecah dan dibandingkan juga.



Cara menggunakan dan menerapkan tanda tabel di Cost Explorer

Secara default, Cost Explorer tidak menyediakan ringkasan biaya untuk satu tabel tertentu, karena Cost Explorer akan menggabungkan biaya beberapa tabel menjadi total. Namun, Anda dapat menggunakan penandaan sumber daya AWS untuk mengidentifikasi setiap tabel dengan tanda metadata. Tanda adalah pasangan nilai kunci yang dapat Anda gunakan untuk berbagai tujuan, seperti mengidentifikasi semua sumber daya milik proyek atau departemen. Untuk contoh ini, kita akan menganggap Anda memiliki tabel bernama MyTable.

1. Tetapkan tag dengan kunci table_name dan nilai. MyTable
2. Aktifkan tanda di dalam Cost Explorer lalu filter pada nilai tanda untuk mendapatkan lebih banyak visibilitas ke setiap biaya tabel.

> **Note**
>
> Tanda mungkin akan mulai muncul di Cost Explorer setelah satu atau dua hari

Anda dapat mengatur sendiri tag metadata di konsol, atau melalui otomatisasi seperti AWS CLI atau SDK. AWS Pertimbangkan untuk mewajibkan tanda table_name ditetapkan sebagai bagian dari proses pembuatan tabel baru organisasi Anda. Untuk tabel yang ada, tersedia utilitas Python yang akan menemukan dan menerapkan tanda ini ke semua tabel yang ada di wilayah tertentu di akun Anda. Lihat Tagger Tabel Eponymous GitHub untuk lebih jelasnya.

Evaluasi mode kapasitas tabel DynamoDB Anda

Bagian ini memberikan gambaran umum tentang cara memilih mode kapasitas yang sesuai untuk tabel DynamoDB Anda. Setiap mode disesuaikan untuk memenuhi kebutuhan beban kerja yang berbeda dalam hal respons terhadap perubahan throughput, serta cara penagihan penggunaan tersebut. Anda harus menyeimbangkan faktor-faktor ini saat membuat keputusan.

Topik

- Mode kapasitas tabel yang tersedia
- Kapan harus memilih mode kapasitas sesuai permintaan
- Kapan harus mode kapasitas yang disediakan
- Faktor lain yang perlu dipertimbangkan saat memilih mode kapasitas tabel

Mode kapasitas tabel yang tersedia

Saat membuat tabel DynamoDB, Anda harus memilih mode kapasitas sesuai permintaan atau yang disediakan. Anda dapat beralih di antara mode kapasitas setiap 24 jam sekali. Satu-satunya pengecualian untuk ini adalah jika Anda mengalihkan tabel mode yang disediakan ke mode sesuai permintaan, Anda dapat beralih kembali ke mode yang disediakan dalam periode 24 jam yang sama.

Mode kapasitas sesuai permintaan

Mode kapasitas sesuai permintaan dirancang untuk menghilangkan kebutuhan untuk merencanakan atau menyediakan kapasitas tabel DynamoDB Anda. Dalam mode ini, tabel Anda akan langsung mengakomodasi permintaan ke tabel Anda tanpa perlu menaikkan atau menurunkan skala sumber daya apa pun (hingga dua kali lipat throughput puncak tabel sebelumnya).

DynamoDB on-demand pay-per-request menawarkan harga untuk permintaan baca dan tulis sehingga Anda hanya membayar untuk apa yang Anda gunakan.

Tabel kapasitas yang disediakan

Mode kapasitas yang disediakan adalah model yang lebih tradisional di mana Anda harus menentukan berapa banyak kapasitas tabel yang tersedia untuk permintaan baik secara langsung atau dengan bantuan penskalaan otomatis. Karena kapasitas tertentu disediakan untuk tabel pada waktu tertentu, penagihan didasarkan pada total kapasitas yang disediakan daripada jumlah permintaan yang dikonsumsi. Melebihi kapasitas yang dialokasikan juga dapat menyebabkan tabel menolak permintaan dan mengurangi pengalaman pengguna aplikasi Anda.

Mode kapasitas yang disediakan memerlukan pemantauan konstan untuk menemukan keseimbangan antara tidak penyediaan berlebihan atau kurang penyediaan tabel untuk menjaga pembatasan tetap rendah dan biaya tetap disetel.

Kapan harus memilih mode kapasitas sesuai permintaan

Saat mengoptimalkan biaya, mode sesuai permintaan adalah pilihan terbaik Anda ketika Anda memiliki beban kerja yang mirip dengan grafik berikut.

Faktor-faktor berikut berkontribusi pada jenis beban kerja ini:

- Pola lalu lintas yang berkembang dari waktu ke waktu
- Volume permintaan variabel (dihasilkan dari beban kerja batch)
- Waktu permintaan yang tidak dapat diprediksi (mengakibatkan lonjakan lalu lintas)
- Turun ke nol atau di bawah 30% dari puncak selama satu jam tertentu

Untuk beban kerja dengan faktor-faktor di atas, menggunakan penskalaan otomatis untuk mempertahankan kapasitas yang cukup di atas meja untuk merespons lonjakan lalu lintas kemungkinan akan menyebabkan tabel dilebih-lebihkan dan biaya lebih dari yang diperlukan atau tabel sedang disediakan dan permintaan dibatasi secara tidak perlu. Mode kapasitas on-demand adalah pilihan yang lebih baik karena dapat menangani lalu lintas yang berfluktuasi tanpa mengharuskan Anda untuk memprediksi atau menyesuaikan kapasitas.

Dengan model pay-per-request harga mode on-demand, Anda tidak perlu khawatir tentang kapasitas idle karena Anda hanya membayar untuk throughput yang benar-benar Anda gunakan. Anda ditagih per permintaan baca atau tulis yang dikonsumsi, sehingga biaya Anda secara langsung mencerminkan penggunaan aktual Anda, sehingga mudah untuk menyeimbangkan biaya dan kinerja. Secara opsional, Anda juga dapat mengonfigurasi throughput baca atau tulis maksimum (atau keduanya) per detik untuk tabel sesuai permintaan individu dan indeks sekunder global untuk

membantu menjaga biaya dan penggunaan tetap terbatas. Untuk informasi selengkapnya, lihat throughput maksimum untuk tabel sesuai permintaan.

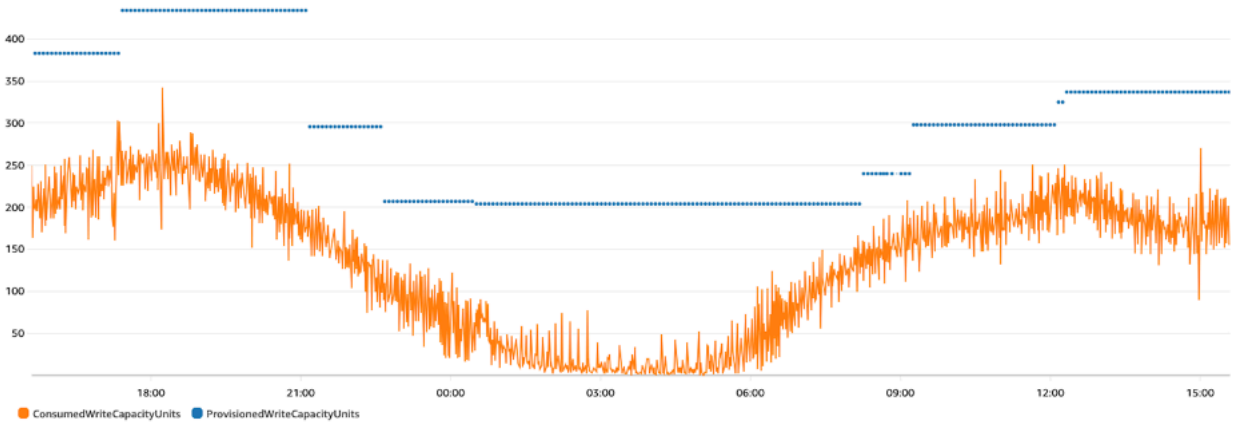
Kapan harus mode kapasitas yang disediakan

Beban kerja yang ideal untuk mode kapasitas yang disediakan adalah beban kerja dengan pola penggunaan yang lebih stabil dan dapat diprediksi seperti grafik di bawah ini.

> **Note**
>
> Sebaiknya tinjau metrik pada periode berbutir halus, seperti 14 hari, sebelum mengambil tindakan terhadap kapasitas yang Anda berikan.

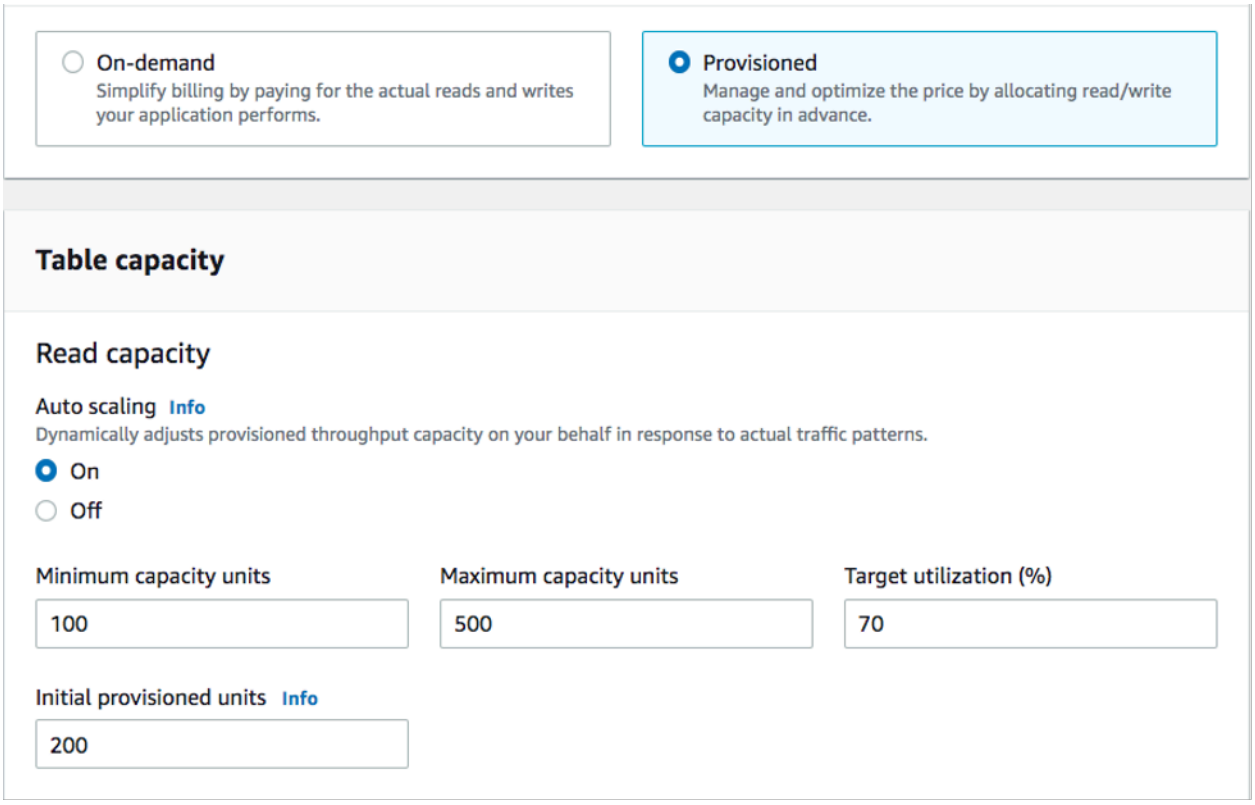
Faktor-faktor berikut berkontribusi pada jenis beban kerja ini:

- Lalu lintas stabil, dapat diprediksi, dan siklus untuk jam atau hari tertentu
- Semburan lalu lintas jangka pendek yang terbatas



Karena volume lalu lintas dalam satu jam atau hari tertentu lebih stabil, Anda dapat mengatur kapasitas tabel yang disediakan relatif dekat dengan kapasitas konsumsi tabel yang sebenarnya. Pengoptimalan biaya pada kapasitas tabel yang disediakan pada akhirnya merupakan upaya untuk membuat kapasitas yang disediakan (garis biru) sedekat mungkin dengan kapasitas yang digunakan (garis oranye) tanpa meningkatkan `ThrottledRequests` pada tabel. Jarak antara kedua garis tersebut merupakan kapasitas yang terbuang serta jaminan terhadap pengalaman pengguna yang buruk akibat throttling. Jika Anda dapat memprediksi persyaratan throughput aplikasi Anda dan Anda lebih suka prediktabilitas biaya untuk mengontrol kapasitas baca dan tulis, maka Anda mungkin ingin terus menggunakan tabel yang disediakan.

DynamoDB menyediakan penskalaan otomatis untuk kapasitas tabel yang disediakan yang akan menyeimbangkan kapasitas tersebut secara otomatis atas nama Anda. Ini memungkinkan Anda memantau kapasitas yang digunakan sepanjang hari dan mengatur kapasitas tabel berdasarkan beberapa variabel. Saat menggunakan penskalaan otomatis, tabel Anda akan disediakan secara berlebihan dan Anda perlu menyempurnakan rasio antara jumlah throttle versus unit kapasitas yang disediakan secara berlebihan agar sesuai dengan kebutuhan beban kerja Anda.



Unit kapasitas minimum

Anda dapat mengatur kapasitas tabel minimum untuk membatasi throttling, tetapi tindakan ini tidak akan mengurangi biaya tabel. Jika tabel Anda memiliki periode penggunaan rendah yang diikuti dengan lonjakan penggunaan tinggi secara tiba-tiba, mengatur ke kapasitas minimum dapat mencegah penskalaan otomatis mengatur kapasitas tabel terlalu rendah.

Unit kapasitas maksimum

Anda dapat mengatur kapasitas tabel maksimum untuk membatasi penskalaan tabel yang lebih tinggi dari yang dimaksudkan. Pertimbangkan untuk menerapkan jumlah maksimum untuk tabel Pengembgangan atau Pengujian dengan pengujian beban skala besar tidak diinginkan. Anda dapat menetapkan maksimum untuk tabel apa pun, tetapi pastikan untuk mengevaluasi pengaturan ini

secara berkala terhadap garis dasar tabel saat menggunakannya dalam Produksi untuk mencegah throttling yang tidak disengaja.

Pemanfaatan target

Menetapkan pemanfaatan target pada tabel adalah cara utama pengoptimalan biaya untuk kapasitas tabel yang disediakan. Menetapkan nilai persentase yang lebih rendah di sini akan meningkatkan jumlah tabel yang disediakan secara berlebihan, sehingga meningkatkan biaya, tetapi mengurangi risiko throttling. Menetapkan nilai persentase yang lebih tinggi akan mengurangi jumlah tabel yang disediakan secara berlebihan, tetapi meningkatkan risiko throttling.

Faktor lain yang perlu dipertimbangkan saat memilih mode kapasitas tabel

Saat memutuskan antara dua mode, ada beberapa faktor lain yang perlu dipertimbangkan.

Pemanfaatan kapasitas yang disediakan

Untuk menentukan kapan mode sesuai permintaan akan lebih murah daripada kapasitas yang disediakan, akan sangat membantu untuk melihat pemanfaatan kapasitas yang disediakan, yang mengacu pada seberapa efisien sumber daya yang dialokasikan (atau “disediakan) digunakan. Mode on-demand lebih murah untuk beban kerja dengan pemanfaatan kapasitas rata-rata yang disediakan di bawah 35%. Dalam banyak kasus, bahkan untuk beban kerja dengan pemanfaatan kapasitas yang disediakan lebih tinggi dari 35%, akan lebih hemat biaya untuk menggunakan mode sesuai permintaan terutama jika beban kerja memiliki periode aktivitas rendah yang dicampur dengan puncak sesekali.

Kapasitas terpesan

Untuk kapasitas tabel yang disediakan, DynamoDB menawarkan kemampuan untuk membeli kapasitas terpesan untuk kapasitas baca dan tulis Anda (unit kapasitas tulis yang direplikasi (RWCU) dan tabel Standard-IA saat ini tidak memenuhi syarat). Kapasitas cadangan menawarkan diskon signifikan atas harga kapasitas standar yang disediakan.

Saat memutuskan antara dua mode tabel, pertimbangkan seberapa besar pengaruh diskon tambahan ini terhadap biaya tabel. Dalam beberapa kasus, mungkin lebih murah untuk menjalankan beban kerja yang relatif tidak dapat diprediksi dapat lebih murah untuk dijalankan pada tabel kapasitas yang disediakan berlebihan dengan kapasitas cadangan.

Meningkatkan prediktabilitas beban kerja Anda

Dalam beberapa situasi, beban kerja tampaknya memiliki pola yang dapat diprediksi dan tidak dapat diprediksi. Meskipun ini dapat dengan mudah didukung dengan tabel sesuai permintaan, biaya kemungkinan akan lebih terjangkau jika pola beban kerja yang tidak dapat diprediksi dapat ditingkatkan.

Salah satu penyebab paling umum dari pola ini adalah impor batch. Jenis lalu lintas ini sering kali dapat melebihi kapasitas dasar tabel sedemikian rupa sehingga throttling akan terjadi jika tabel tersebut dijalankan. Agar beban kerja seperti ini tetap berjalan pada kapasitas tabel yang disediakan, pertimbangkan opsi berikut:

- Jika batch terjadi pada waktu yang dijadwalkan, Anda dapat menjadwalkan peningkatan kapasitas penskalaan otomatis sebelum batch dijalankan
- Jika batch terjadi secara acak, pertimbangkan untuk memperpanjang waktu berjalan daripada mengeksekusi secepat mungkin
- Tambahkan periode kenaikan ke impor di mana kecepatan impor mulai kecil tetapi perlahan-lahan meningkat selama beberapa menit sampai penskalaan otomatis memiliki kesempatan untuk mulai menyesuaikan kapasitas tabel

Evaluasi pengaturan penskalaan otomatis tabel DynamoDB Anda

Bagian ini memberikan gambaran umum tentang cara mengevaluasi pengaturan penskalaan otomatis pada tabel DynamoDB Anda. Penskalaan otomatis Amazon DynamoDB adalah fitur yang mengelola tabel dan throughput indeks sekunder global (GSI) berdasarkan lalu lintas aplikasi dan metrik pemanfaatan target Anda. Ini memastikan tabel Anda atau GSIs akan memiliki kapasitas yang diperlukan untuk pola aplikasi Anda.

Layanan penskalaan AWS otomatis akan memantau pemanfaatan tabel Anda saat ini dan membandingkannya dengan nilai pemanfaatan target:. `TargetValue` Fitur ini akan memberi tahu Anda jika sudah waktunya untuk menambah atau mengurangi kapasitas yang dialokasikan.

Topik

- Memahami pengaturan penskalaan otomatis Anda
- Cara mengidentifikasi tabel dengan pemanfaatan target rendah (<= 50%)
- Cara mengatasi beban kerja dengan varian musiman
- Cara mengatasi lonjakan beban kerja dengan pola yang tidak diketahui
- Cara mengatasi beban kerja dengan aplikasi tertaut

Memahami pengaturan penskalaan otomatis Anda

Mendefinisikan nilai yang benar untuk pemanfaatan target, langkah awal, dan nilai akhir adalah aktivitas yang memerlukan keterlibatan dari tim operasi Anda. Hal ini memungkinkan Anda untuk menentukan nilai dengan benar berdasarkan penggunaan aplikasi historis, yang akan digunakan untuk memicu kebijakan penskalaan otomatis AWS . Pemanfaatan target adalah persentase total kapasitas yang perlu dicapai selama jangka waktu tertentu sebelum aturan penskalaan otomatis berlaku.

Ketika Anda menetapkan pemanfaatan target yang tinggi (target sekitar 90%), artinya lalu lintas Anda harus lebih tinggi dari 90% untuk jangka waktu tertentu sebelum penskalaan otomatis dimulai. Jangan menggunakan pemanfaatan target yang tinggi kecuali aplikasi Anda sangat konstan dan tidak menerima lonjakan lalu lintas.

Ketika Anda menetapkan pemanfaatan yang sangat rendah (target kurang dari 50%), artinya aplikasi Anda harus mencapai 50% dari kapasitas yang disediakan sebelum memicu kebijakan penskalaan otomatis. Kecuali lalu lintas aplikasi Anda tumbuh pada tingkat yang sangat agresif, ini biasanya diterjemahkan ke dalam kapasitas yang tidak terpakai dan sumber daya yang terbuang.

Cara mengidentifikasi tabel dengan pemanfaatan target rendah (<= 50%)

Anda dapat menggunakan AWS CLI atau AWS Management Console untuk memantau dan mengidentifikasi kebijakan penskalaan otomatis di sumber daya DynamoDB Anda: `TargetValues`

AWS CLI

1. Kembalikan seluruh daftar sumber daya dengan menjalankan perintah berikut:

   ```
   aws application-autoscaling describe-scaling-policies --service-namespace
    dynamodb
   ```

   Perintah ini akan mengembalikan seluruh daftar kebijakan penskalaan otomatis yang dikeluarkan ke sumber daya DynamoDB apa pun. Jika hanya ingin mengambil sumber daya dari tabel tertentu, Anda dapat menambahkan `–resource-id parameter`. Misalnya:

   ```
   aws application-autoscaling describe-scaling-policies --service-namespace
    dynamodb --resource-id "table/<table-name>”
   ```

2. Kembalikan hanya kebijakan penskalaan otomatis untuk GSI tertentu dengan menjalankan perintah berikut

```
aws application-autoscaling describe-scaling-policies --service-namespace
 dynamodb --resource-id "table/<table-name>/index/<gsi-name>”
```

Nilai-nilai yang kami minati untuk kebijakan penskalaan otomatis disorot di bawah ini. Kami ingin memastikan bahwa nilai target lebih besar dari 50% untuk menghindari penyediaan berlebih. Anda akan mendapatkan hasil yang mirip dengan berikut ini:

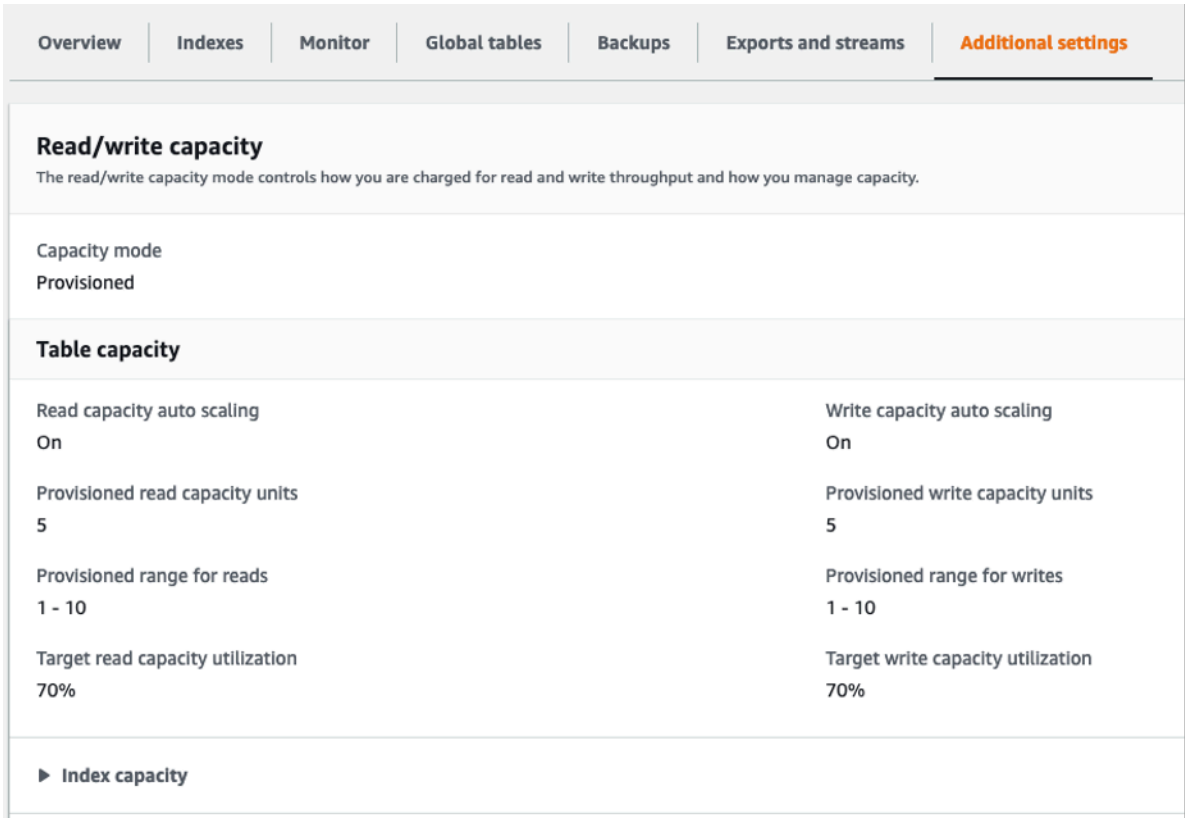
```
{
    "ScalingPolicies": [
        {
            "PolicyARN": "arn:aws:autoscaling:<region>:<account-
id>:scalingPolicy:<uuid>:resource/dynamodb/table/<table-name>/index/<index-
name>:policyName/$<full-gsi-name>-scaling-policy",
            "PolicyName": $<full-gsi-name>”,
            "ServiceNamespace": "dynamodb",
            "ResourceId": "table/<table-name>/index/<index-name>",
            "ScalableDimension": "dynamodb:index:WriteCapacityUnits",
            "PolicyType": "TargetTrackingScaling",
            "TargetTrackingScalingPolicyConfiguration": {
                "TargetValue": 70.0,
                "PredefinedMetricSpecification": {
                    "PredefinedMetricType": "DynamoDBWriteCapacityUtilization"
                }
            },
            "Alarms": [
                ...
            ],
            "CreationTime": "2022-03-04T16:23:48.641000+10:00"
        },
        {
            "PolicyARN": "arn:aws:autoscaling:<region>:<account-
id>:scalingPolicy:<uuid>:resource/dynamodb/table/<table-name>/index/<index-
name>:policyName/$<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"
        }
    ]
}
```

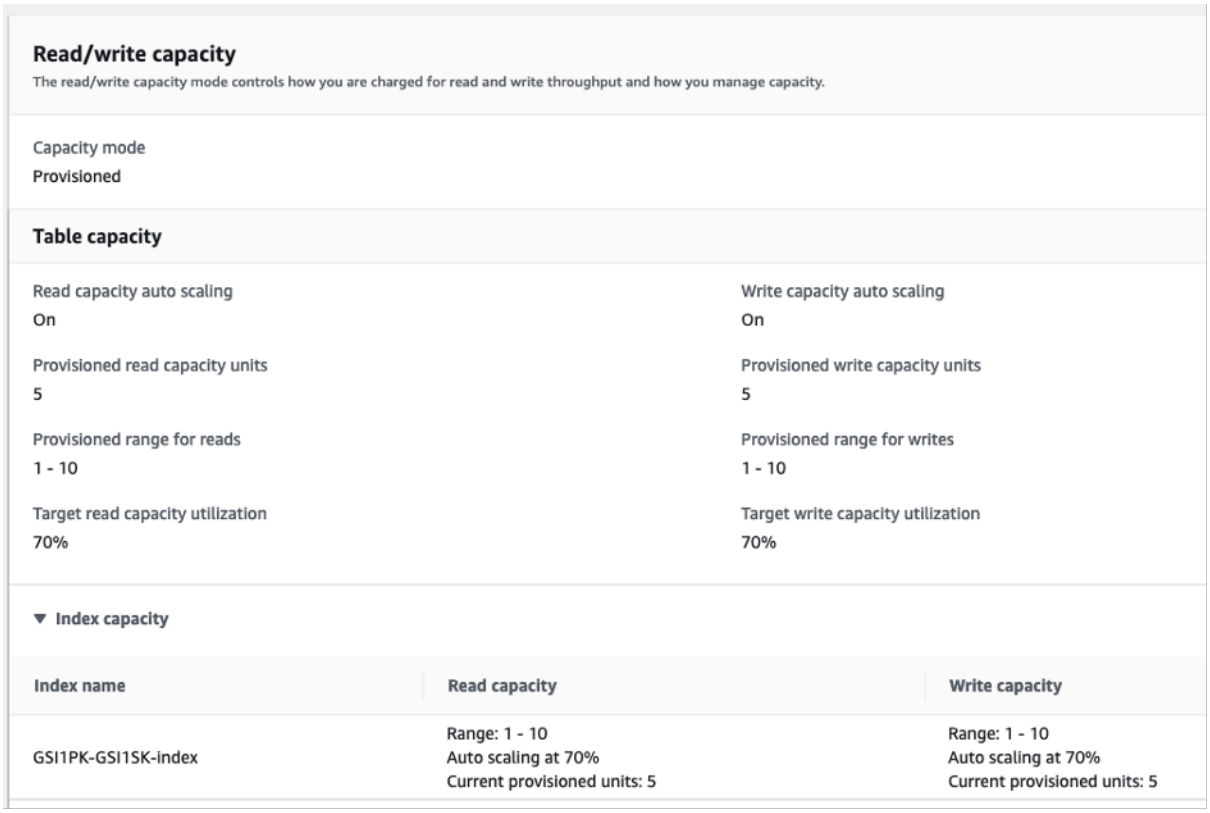
AWS Management Console

1. Masuk ke AWS Management Console dan buka konsol DynamoDB di. https://console.aws.amazon.com/dynamodb/

   Pilih yang sesuai Wilayah AWS jika perlu.
2. Di bilah navigasi kiri, pilih Tabel. Di halaman Tabel, pilih Nama tabel.
3. Pada halaman Detail tabel, pilih Pengaturan tambahan, lalu tinjau pengaturan penskalaan otomatis tabel Anda.



Untuk indeks, perluas bagian Kapasitas indeks untuk meninjau pengaturan penskalaan otomatis indeks.



Jika nilai pemanfaatan target Anda kurang dari atau sama dengan 50%, Anda harus mempelajari metrik pemanfaatan tabel Anda untuk mengetahui apakah nilai tersebut kurang tersedia atau disediakan secara berlebihan.

Cara mengatasi beban kerja dengan varian musiman

Pertimbangkan skenario berikut: aplikasi Anda sering kali beroperasi di bawah nilai rata-rata minimum, tetapi pemanfaatan targetnya rendah sehingga aplikasi Anda dapat bereaksi dengan cepat terhadap peristiwa yang terjadi pada jam-jam tertentu dalam sehari dan Anda memiliki kapasitas yang memadai serta menghindari throttling. Skenario ini umum terjadi ketika Anda memiliki aplikasi yang sangat sibuk selama jam kantor normal (9 pagi hingga 5 sore) tetapi kemudian berfungsi pada tingkat dasar setelah jam kerja. Karena beberapa pengguna akan mulai terhubung sebelum jam 9 pagi, aplikasi menggunakan ambang batas rendah ini untuk meningkatkan dengan cepat guna mencapai kapasitas yang diperlukan selama jam sibuk.

Skenario ini akan seperti berikut:

- Antara jam 5 sore dan 9 pagi `ConsumedWriteCapacity` unit tetap berada di antara 90 dan 100

- Pengguna mulai terhubung ke aplikasi sebelum jam 9 pagi dan unit kapasitas meningkat pesat (nilai maksimum yang Anda lihat adalah 1500 WCU)
- Rata-rata, penggunaan aplikasi Anda berkisar antara 800 hingga 1200 selama jam kerja

Jika skenario sebelumnya berlaku untuk Anda, sebaiknya gunakan penskalaan otomatis terjadwal, yaitu tabel Anda masih dapat mengonfigurasi aturan penskalaan otomatis aplikasi, tetapi dengan pemanfaatan target yang kurang agresif yang hanya menyediakan kapasitas tambahan pada interval tertentu yang Anda perlukan.

Anda dapat menggunakan AWS CLI untuk menjalankan langkah-langkah berikut untuk membuat aturan penskalaan otomatis terjadwal yang akan dijalankan berdasarkan waktu hari dan hari dalam seminggu.

1. Daftarkan GSI atau tabel DynamoDB Anda sebagai target yang dapat diskalakan dengan Application Auto Scaling. Target yang dapat diskalakan adalah sumber daya yang skalanya dapat diperkecil atau diperbesar oleh Application Auto Scaling .

   ```
   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. Siapkan tindakan terjadwal sesuai dengan kebutuhan Anda.

   Kita akan membutuhkan dua aturan untuk mencakup skenario: satu untuk meningkatkan skala dan satu lagi untuk menurunkan skala. Aturan pertama untuk meningkatkan skala tindakan terjadwal:

   ```
   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"
   ```

   Aturan kedua untuk menurunkan skala tindakan terjadwal:

```
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. Jalankan perintah berikut untuk memvalidasi bahwa kedua aturan telah diaktifkan:

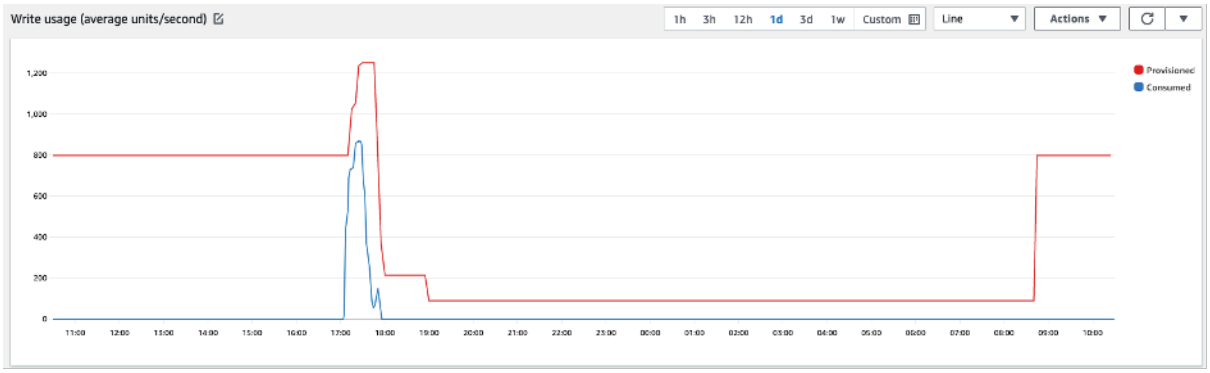
```
aws application-autoscaling describe-scheduled-actions --service-namespace dynamodb
```

Anda akan mendapatkan hasil seperti ini:

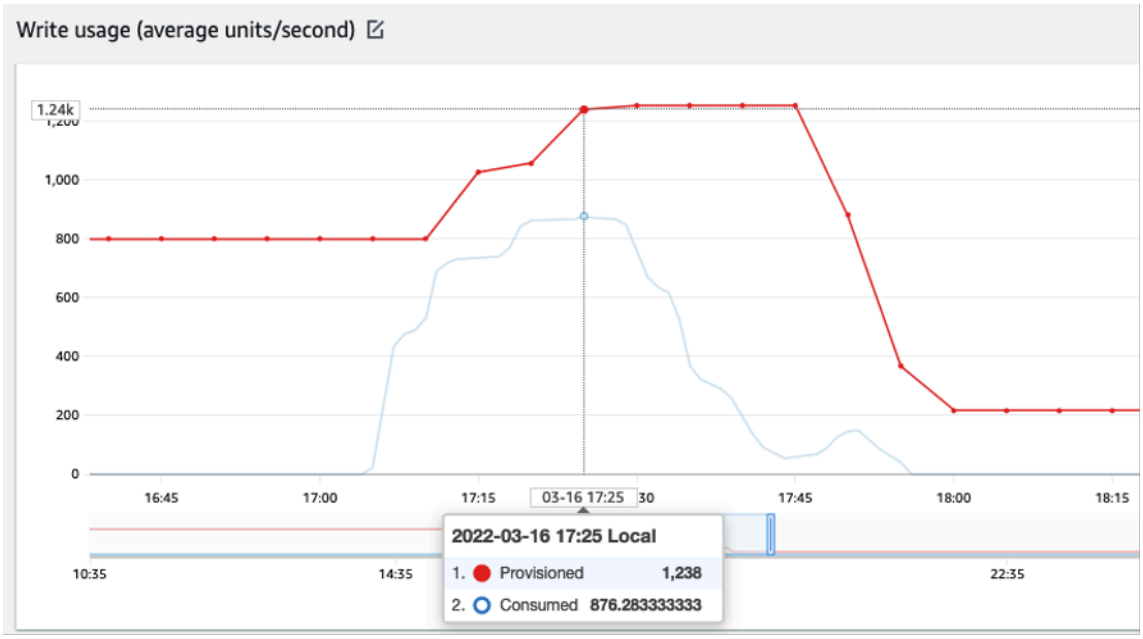
```
{
    "ScheduledActions": [
        {
            "ScheduledActionName": "my-5-8-scheduled-down-action",
            "ScheduledActionARN":
 "arn:aws:autoscaling:<region>:<account>:scheduledAction:<uuid>:resource/dynamodb/
table/<table-name>:scheduledActionName/my-5-8-scheduled-down-action",
            "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>",
```

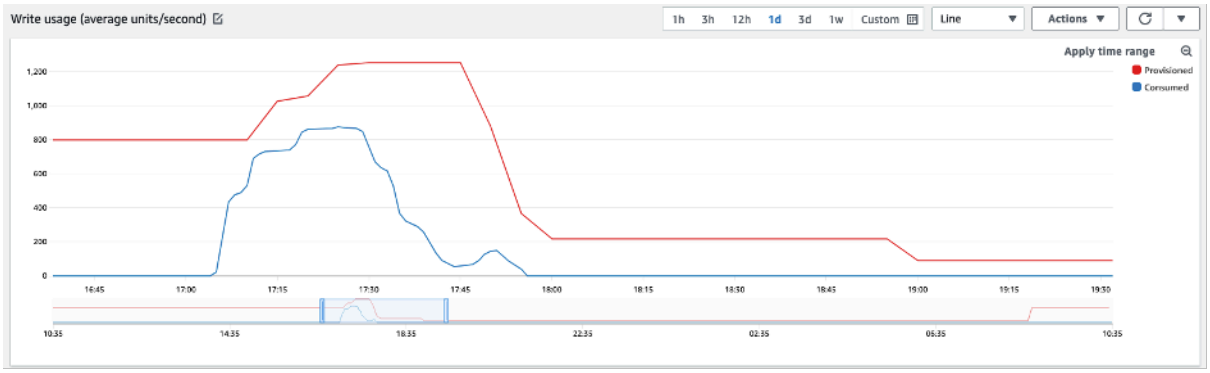
```
            "ScalableDimension": "dynamodb:table:WriteCapacityUnits",
            "ScalableTargetAction": {
                "MinCapacity": 800,
                "MaxCapacity": 1500
            },
            "CreationTime": "2022-03-15T17:28:57.816000+10:00"
        }
    ]
}
```

Gambar berikut menunjukkan beban kerja sampel yang selalu mempertahankan pemanfaatan target 70%. Perhatikan bagaimana aturan penskalaan otomatis masih tetap diterapkan dan throughput tidak akan berkurang.



Memperbesar, kita dapat melihat ada lonjakan dalam aplikasi yang memicu ambang batas penskalaan otomatis 70%, memaksa penskalaan otomatis untuk masuk dan memberikan kapasitas ekstra yang diperlukan untuk tabel. Tindakan penskalaan otomatis terjadwal akan memengaruhi nilai maksimum dan minimum, dan Anda bertanggung jawab untuk mengaturnya.





Cara mengatasi lonjakan beban kerja dengan pola yang tidak diketahui

Dalam skenario ini, aplikasi menggunakan pemanfaatan target yang sangat rendah karena Anda belum mengetahui pola aplikasi, dan Anda ingin memastikan beban kerja Anda tidak mengalami throttling.

Sebaiknya gunakan mode kapasitas sesuai permintaan. Tabel sesuai permintaan sangat cocok untuk lonjakan beban kerja yang tidak Anda ketahui pola lalu lintasnya. Dengan mode kapasitas sesuai permintaan, Anda membayar per permintaan atas pembacaan dan penulisan data yang dilakukan aplikasi Anda pada tabel Anda. Anda tidak perlu menentukan berapa banyak throughput baca dan tulis yang Anda harapkan untuk dijalankan oleh aplikasi Anda, karena DynamoDB langsung mengakomodasi beban kerja Anda saat beban kerja tersebut naik atau turun.

Cara mengatasi beban kerja dengan aplikasi tertaut

Dalam skenario ini, aplikasi bergantung pada sistem lain, seperti skenario pemrosesan batch yang dapat menghasilkan lonjakan besar dalam lalu lintas sesuai dengan peristiwa dalam logika aplikasi.

Pertimbangkan untuk mengembangkan logika penskalaan otomatis khusus yang bereaksi terhadap peristiwa tersebut saat Anda dapat meningkatkan `TargetValues` dan kapasitas tabel bergantung pada kebutuhan spesifik Anda. Anda bisa mendapatkan keuntungan dari Amazon EventBridge dan menggunakan kombinasi AWS layanan seperti Lambda dan Step Functions untuk menanggapi kebutuhan aplikasi spesifik Anda.

Evaluasi pilihan kelas tabel DynamoDB Anda

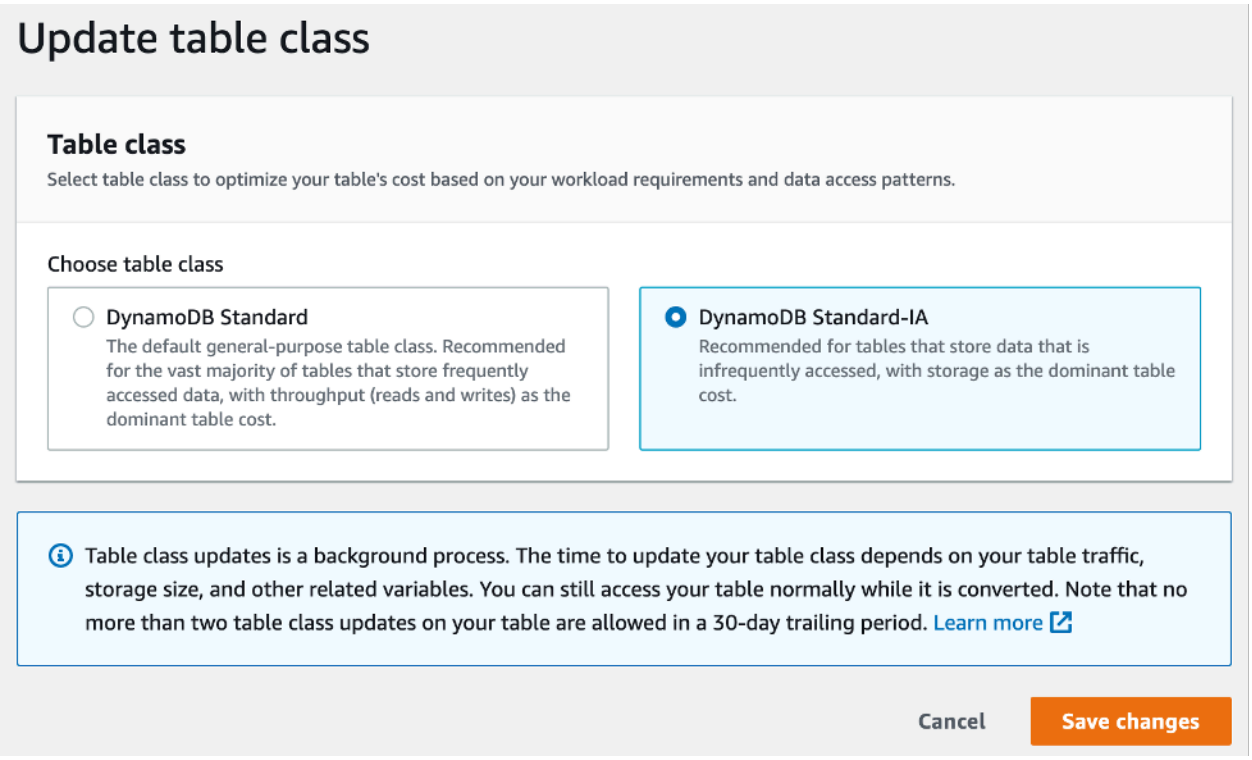
Bagian ini memberikan gambaran umum tentang cara memilih kelas tabel yang sesuai untuk tabel DynamoDB Anda. Dengan peluncuran kelas tabel Standard Infrequent-Access (Standard-IA), Anda sekarang memiliki kemampuan mengoptimalkan tabel untuk biaya penyimpanan yang lebih rendah atau biaya throughput yang lebih rendah.

Topik

- Kelas tabel yang tersedia
- Kapan harus memilih kelas tabel DynamoDB Standard
- Kapan harus memilih kelas tabel DynamoDB Standard-IA
- Faktor lain yang perlu dipertimbangkan saat memilih kelas tabel

Kelas tabel yang tersedia

Ketika membuat Tabel DynamoDB, Anda harus memilih DynamoDB Standard atau DynamoDB Standard-IA untuk kelas tabel. Kelas tabel dapat diubah dua kali dalam periode 30 hari, sehingga Anda selalu dapat mengubahnya di masa mendatang. Memilih salah satu kelas tabel tidak akan memengaruhi performa, ketersediaan, keandalan, atau daya tahan tabel.



Kelas tabel Standard

Kelas tabel Standard adalah opsi default untuk tabel baru. Opsi ini mempertahankan saldo penagihan asli DynamoDB yang menawarkan keseimbangan biaya throughput dan penyimpanan untuk tabel dengan data yang sering diakses.

Kelas tabel Standard-IA

Kelas tabel Standard-IA menawarkan biaya penyimpanan yang lebih rendah (~60% lebih rendah) untuk beban kerja yang memerlukan penyimpanan data jangka panjang dengan pembaruan atau pembacaan yang jarang. Karena kelas ini dioptimalkan untuk akses yang jarang, baca dan tulis akan dikenakan biaya yang sedikit lebih tinggi (~25% lebih tinggi) dibandingkan kelas tabel Standard.

Kapan harus memilih kelas tabel DynamoDB Standard

Kelas tabel DynamoDB Standard paling cocok untuk tabel yang biaya penyimpanannya sekitar 50% atau kurang dari keseluruhan biaya bulanan tabel. Keseimbangan biaya ini menunjukkan beban kerja yang secara berkala mengakses atau memperbarui item yang sudah disimpan dalam DynamoDB.

Kapan harus memilih kelas tabel DynamoDB Standard-IA

Kelas tabel DynamoDB Standard-IA paling cocok untuk tabel yang biaya penyimpanannya sekitar 50% atau lebih dari keseluruhan biaya bulanan tabel. Keseimbangan biaya ini menunjukkan beban

kerja yang membuat atau membaca lebih sedikit item per bulan daripada yang disimpan dalam penyimpanan.

Penggunaan umum untuk kelas tabel Standard-IA adalah memindahkan data yang jarang diakses ke tabel Standard-IA individual. Untuk informasi selengkapnya, lihat Mengoptimalkan biaya penyimpanan beban kerja Anda dengan kelas tabel Amazon DynamoDB Standard-IA.

Faktor lain yang perlu dipertimbangkan saat memilih kelas tabel

Saat memutuskan antara dua kelas tabel, ada beberapa faktor lain yang layak untuk dipertimbangkan sebagai bagian dari keputusan Anda.

Kapasitas terpesan

Pembelian kapasitas terpesan untuk tabel yang menggunakan kelas tabel Standard-IA saat ini tidak didukung. Saat beralih dari tabel Standard dengan kapasitas terpesan ke tabel Standard-IA tanpa kapasitas terpesan, Anda mungkin tidak melihat manfaat biaya.

Identifikasi sumber daya Anda yang tidak digunakan di DynamoDB

Bagian ini memberikan gambaran umum tentang cara mengevaluasi sumber daya yang Anda tidak terpakai secara berkala. Seiring berkembangnya persyaratan aplikasi Anda, Anda harus memastikan tidak ada sumber daya yang tidak terpakai dan berkontribusi terhadap biaya Amazon DynamoDB yang tidak diperlukan. Prosedur yang dijelaskan di bawah ini akan menggunakan CloudWatch metrik Amazon untuk mengidentifikasi sumber daya yang tidak digunakan dan akan membantu Anda mengidentifikasi dan mengambil tindakan terhadap sumber daya tersebut untuk mengurangi biaya.

Anda dapat memantau DynamoDB CloudWatch menggunakan, yang mengumpulkan dan memproses data mentah dari DynamoDB menjadi metrik yang dapat dibaca, mendekati real-time. Statistik ini disimpan selama jangka waktu tertentu, sehingga Anda dapat mengakses informasi historis untuk lebih memahami pemanfaatan Anda. Secara default, data metrik DynamoDB dikirim secara otomatis. CloudWatch Untuk informasi lebih lanjut, lihat Apa itu Amazon CloudWatch? dan Retensi metrik di Panduan CloudWatch Pengguna Amazon.

Topik

- Cara mengidentifikasi sumber daya yang tidak terpakai
- Mengidentifikasi sumber daya tabel yang tidak terpakai
- Membersihkan sumber daya tabel yang tidak terpakai
- Mengidentifikasi sumber daya GSI yang tidak terpakai

- Membersihkan sumber daya GSI yang tidak terpakai
- Membersihkan tabel global yang tidak terpakai
- Membersihkan cadangan atau point-in-time pemulihan yang tidak terpakai (PITR)

Cara mengidentifikasi sumber daya yang tidak terpakai

Untuk mengidentifikasi tabel atau indeks yang tidak digunakan, kita akan melihat CloudWatch metrik berikut selama periode 30 hari untuk memahami apakah ada pembacaan atau penulisan aktif di tabel atau pembacaan apa pun pada indeks sekunder global (): GSIs

ConsumedReadCapacityUnits

Jumlah unit kapasitas baca yang terpakai selama jangka waktu tertentu, sehingga Anda dapat melacak jumlah kapasitas terpakai yang telah Anda gunakan. Anda dapat mengambil total kapasitas baca yang digunakan untuk tabel dan semua indeks sekunder globalnya, atau untuk indeks sekunder global tertentu.

ConsumedWriteCapacityUnits

Jumlah unit kapasitas tulis yang terpakai selama jangka waktu tertentu, sehingga Anda dapat melacak jumlah kapasitas terpakai yang telah Anda gunakan. Anda dapat mengambil total kapasitas tulis yang digunakan untuk tabel dan semua indeks sekunder globalnya, atau untuk indeks sekunder global tertentu.

Mengidentifikasi sumber daya tabel yang tidak terpakai

Amazon CloudWatch adalah layanan pemantauan dan observabilitas yang menyediakan metrik tabel DynamoDB yang akan Anda gunakan untuk mengidentifikasi sumber daya yang tidak digunakan. CloudWatch metrik dapat dilihat melalui AWS Management Console maupun melalui. AWS Command Line Interface

AWS Command Line Interface

Untuk melihat metrik tabel Anda melalui AWS Command Line Interface, Anda dapat menggunakan perintah berikut.

1. Pertama, evaluasi pembacaan tabel Anda:

```
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>
```

Untuk menghindari kesalahan dalam mengidentifikasi tabel sebagai tidak terpakai, evaluasi metrik dalam jangka waktu yang lebih lama. Pilih rentang waktu mulai dan waktu berakhir yang sesuai, seperti 30 hari, dan periode yang sesuai, seperti 86400.

Dalam data yang dikembalikan, setiap Jumlah di atas 0 menunjukkan bahwa tabel yang Anda evaluasi menerima lalu lintas baca selama periode tersebut.

Hasil berikut menunjukkan tabel yang menerima lalu lintas baca pada periode yang dievaluasi:

```
        {
            "Timestamp": "2022-08-25T19:40:00Z",
            "Sum": 36023355.0,
            "Unit": "Count"
        },
        {
            "Timestamp": "2022-08-12T19:40:00Z",
            "Sum": 38025777.5,
            "Unit": "Count"
        },
```

Hasil berikut menunjukkan tabel yang tidak menerima lalu lintas baca pada periode yang dievaluasi:

```
        {
            "Timestamp": "2022-08-01T19:50:00Z",
            "Sum": 0.0,
            "Unit": "Count"
        },
        {
            "Timestamp": "2022-08-20T19:50:00Z",
            "Sum": 0.0,
            "Unit": "Count"
        },
```

2. Selanjutnya, evaluasi penulisan tabel Anda:

```
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>
```

Untuk menghindari kesalahan dalam mengidentifikasi tabel sebagai tidak terpakai, sebaiknya Anda mengevaluasi metrik dalam jangka waktu yang lebih lama. Pilih rentang waktu mulai dan waktu berakhir yang sesuai, seperti 30 hari, dan periode yang sesuai, seperti 86400.

Dalam data yang dikembalikan, setiap Jumlah di atas 0 menunjukkan bahwa tabel yang Anda evaluasi menerima lalu lintas baca selama periode tersebut.

Hasil berikut menunjukkan tabel yang menerima lalu lintas tulis pada periode yang dievaluasi:

```
        {
            "Timestamp": "2022-08-19T20:15:00Z",
            "Sum": 41014457.0,
            "Unit": "Count"
        },
        {
            "Timestamp": "2022-08-18T20:15:00Z",
            "Sum": 40048531.0,
            "Unit": "Count"
        },
```

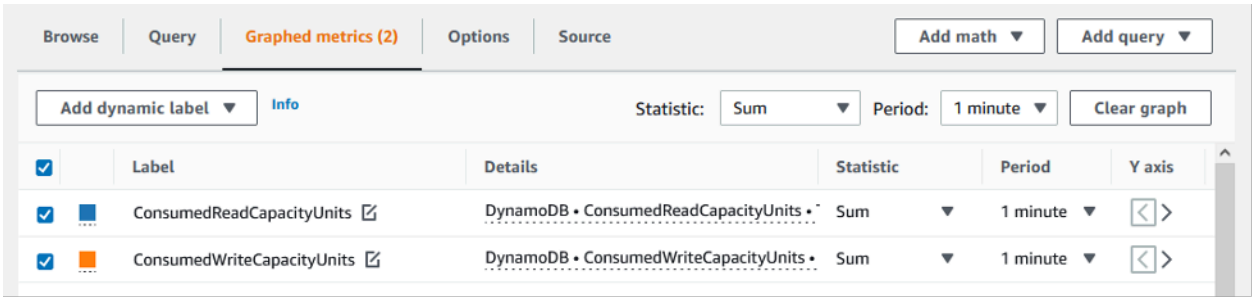
Hasil berikut menunjukkan tabel yang tidak menerima lalu lintas tulis pada periode yang dievaluasi:

```
        {
            "Timestamp": "2022-07-31T20:15:00Z",
            "Sum": 0.0,
            "Unit": "Count"
        },
        {
            "Timestamp": "2022-08-19T20:15:00Z",
            "Sum": 0.0,
            "Unit": "Count"
        },
```

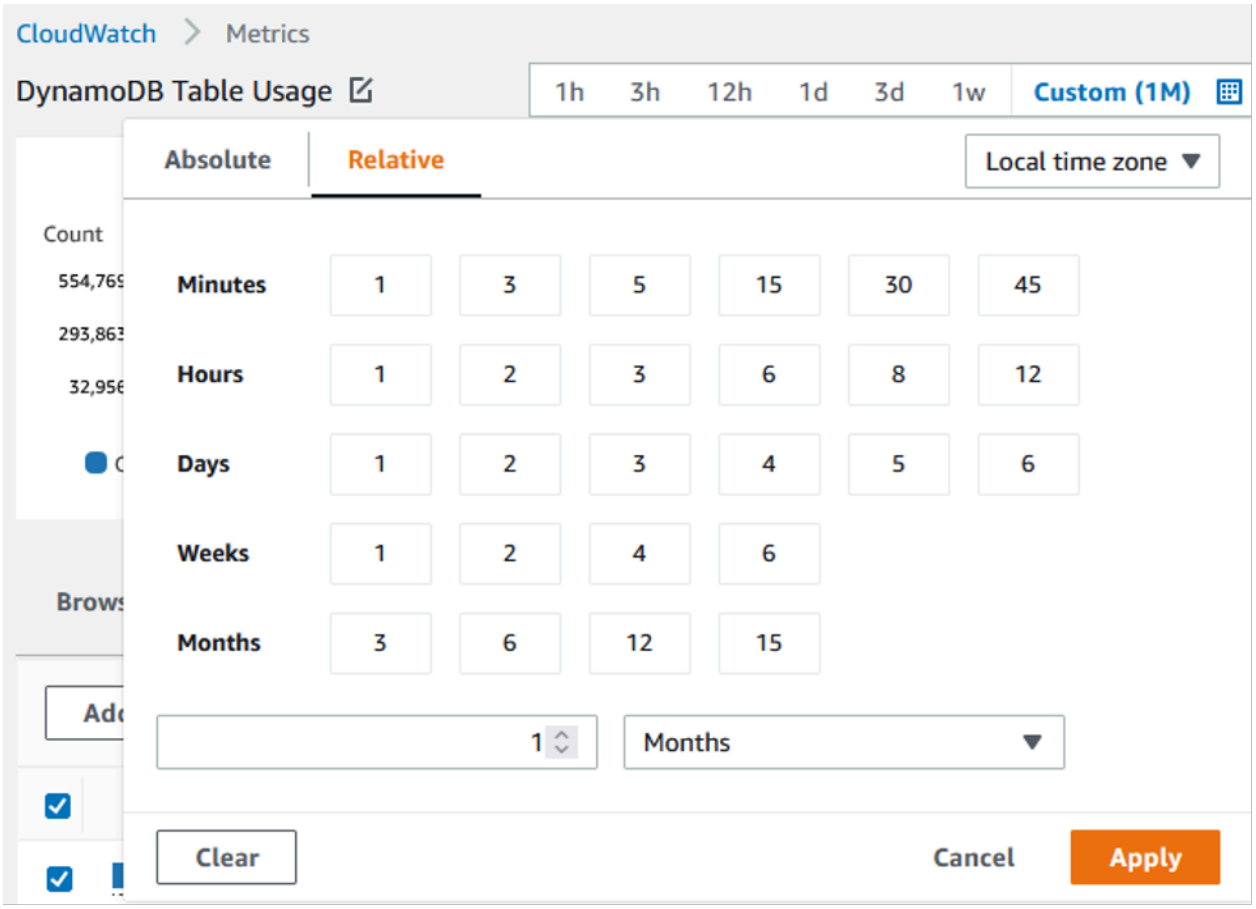
AWS Management Console

Langkah-langkah berikut akan memungkinkan Anda untuk mengevaluasi penggunaan sumber daya Anda melalui AWS Management Console.

1. Masuk ke AWS konsol dan arahkan ke halaman CloudWatch layanan di https://console.aws.amazon.com/cloudwatch/. Pilih AWS wilayah yang sesuai di kanan atas konsol, jika perlu.
2. Di bilah navigasi kiri, cari bagian Metrik lalu pilih Semua metrik.
3. Tindakan di atas akan membuka dashboard dengan dua panel. Di panel atas, Anda akan melihat metrik bergrafik saat ini. Di bagian bawah, Anda akan memilih metrik yang tersedia untuk dibuat grafik. Pilih DynamoDB di panel bawah.
4. Di panel pemilihan metrik DynamoDB, pilih kategori Metrik Tabel untuk menampilkan metrik tabel Anda di wilayah saat ini.
5. Identifikasi nama tabel Anda dengan menggulir ke bawah menu, lalu pilih metrik `ConsumedReadCapacityUnits` dan `ConsumedWriteCapacityUnits` untuk tabel Anda.
6. Pilih tab Metrik bergrafik (2) dan sesuaikan kolom Statistik menjadi Jumlah.

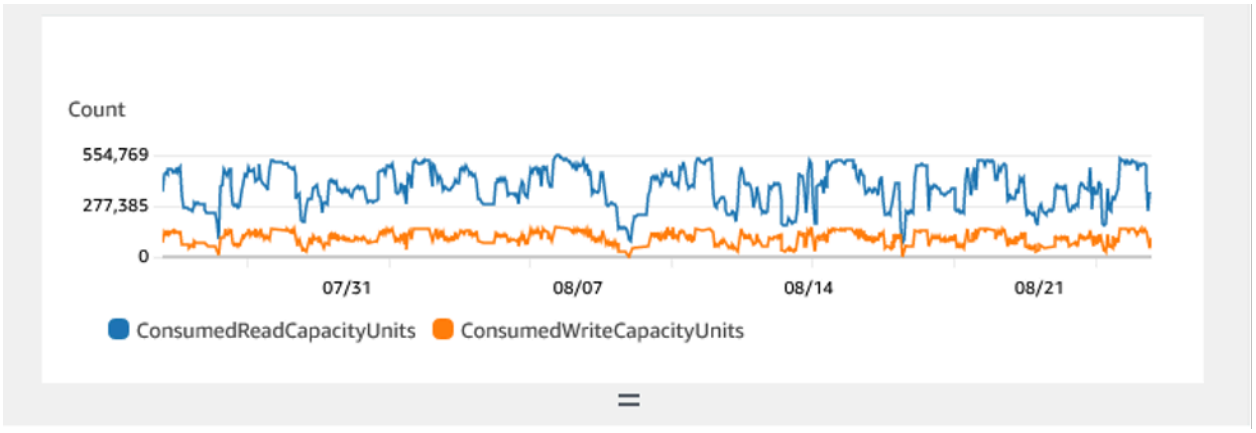


7. Untuk menghindari kesalahan dalam mengidentifikasi tabel sebagai tidak terpakai, sebaiknya Anda mengevaluasi metrik dalam jangka waktu yang lebih lama. Di bagian atas panel grafik, pilih kerangka waktu yang sesuai, seperti 1 bulan, untuk mengevaluasi tabel Anda. Pilih Kustom, pilih 1 Bulan di dropdown, lalu pilih Terapkan.

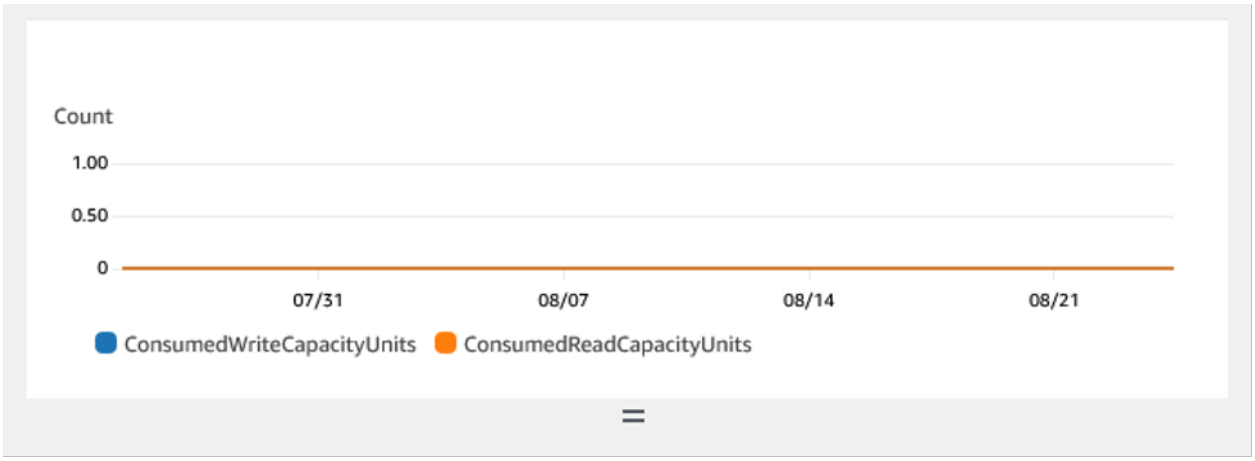


8. Evaluasi metrik bergrafik untuk tabel Anda guna menentukan apakah tabel sedang digunakan. Metrik di atas 0 menunjukkan bahwa tabel telah digunakan selama jangka waktu evaluasi. Grafik datar pada 0 untuk baca dan tulis menunjukkan tabel yang tidak terpakai.

   Gambar berikut menunjukkan tabel dengan lalu lintas baca:



   Gambar berikut menunjukkan tabel tanpa lalu lintas baca:



Membersihkan sumber daya tabel yang tidak terpakai

Jika Anda telah mengidentifikasi sumber daya tabel yang tidak terpakai, Anda dapat mengurangi biaya berkelanjutannya dengan cara berikut.

> **Note**
>
> Jika Anda telah mengidentifikasi tabel yang tidak terpakai tetapi masih ingin tetap tersedia jika tabel tersebut perlu diakses di masa mendatang, pertimbangkan untuk mengalihkannya ke mode sesuai permintaan. Atau, Anda dapat mempertimbangkan untuk membuat cadangan dan menghapus tabel sepenuhnya.

Mode kapasitas

DynamoDB mengenakan biaya untuk membaca, menulis, dan menyimpan data dalam tabel DynamoDB Anda.

DynamoDB memiliki dua mode kapasitas, yang dilengkapi dengan opsi penagihan khusus untuk memproses baca dan tulis pada tabel Anda: sesuai permintaan dan disediakan. Mode read/write kapasitas mengontrol bagaimana Anda dikenakan biaya untuk throughput baca dan tulis dan bagaimana Anda mengelola kapasitas.

Untuk tabel mode sesuai permintaan, Anda tidak perlu menentukan jumlah throughput baca dan tulis yang Anda harapkan untuk dijalankan oleh aplikasi Anda. DynamoDB membebankan biaya bagi Anda atas baca dan tulis yang dilakukan aplikasi Anda pada tabel Anda dalam hal unit permintaan baca dan unit permintaan tulis. Jika tidak ada aktivitas pada Anda, table/index Anda tidak membayar untuk throughput tetapi Anda masih akan dikenakan biaya penyimpanan.

Kelas tabel

DynamoDB juga menawarkan dua kelas tabel yang didesain untuk membantu Anda mengoptimalkan biaya. Kelas tabel DynamoDB Standard adalah defaultnya dan direkomendasikan untuk sebagian besar beban kerja. Kelas tabel DynamoDB Standard-Infrequent Access (DynamoDB Standard-IA) dioptimalkan untuk tabel yang biayanya didominasi oleh penyimpanan.

Jika tidak ada aktivitas di tabel atau indeks Anda, penyimpanan kemungkinan akan menjadi biaya yang dominan dan mengubah kelas tabel akan menghasilkan penghematan yang signifikan.

Menghapus tabel

Jika telah menemukan tabel yang tidak terpakai dan ingin menghapusnya, sebaiknya Anda mencadangkan atau mengekspor datanya terlebih dahulu.

Pencadangan yang diambil melalui AWS Backup dapat memanfaatkan tiering cold storage, yang selanjutnya mengurangi biaya. Lihat Menggunakan AWS Backup dengan DynamoDB dokumentasi untuk informasi tentang cara mengaktifkan pencadangan melalui AWS Backup serta dokumentasi Mengelola rencana cadangan untuk informasi tentang cara menggunakan siklus hidup untuk memindahkan cadangan Anda ke penyimpanan dingin.

Atau, Anda dapat mengekspor data tabel ke S3. Untuk melakukannya, lihat dokumentasi Mengekspor ke Amazon S3. Setelah data Anda diekspor, jika Anda ingin memanfaatkan S3 Glacier Instant Retrieval, S3 Glacier Flexile Retrieval, atau S3 Glacier Deep Archive untuk semakin mengurangi biaya, lihat Mengelola siklus hidup penyimpanan Anda.

Setelah tabel Anda telah dicadangkan, Anda dapat menghapusnya melalui AWS Management Console atau AWS Command Line Interface.

Mengidentifikasi sumber daya GSI yang tidak terpakai

Langkah-langkah untuk mengidentifikasi sekunder global yang tidak terpakai sama dengan langkah-langkah untuk mengidentifikasi tabel yang tidak terpakai. Karena DynamoDB mereplikasi item yang ditulis ke tabel dasar Anda ke dalam GSI jika item tersebut berisi atribut yang digunakan sebagai kunci partisi GSI, GSI yang tidak digunakan kemungkinan masih memiliki `ConsumedWriteCapacityUnits` di atas 0 jika tabel dasarnya digunakan. Dengan demikian, Anda hanya akan mengevaluasi metrik `ConsumedReadCapacityUnits` untuk menentukan apakah GSI Anda tidak terpakai.

Untuk melihat metrik GSI Anda melalui AWS AWS CLI, Anda dapat menggunakan perintah berikut untuk mengevaluasi bacaan tabel Anda:

```
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>
```

Untuk menghindari kesalahan dalam mengidentifikasi tabel sebagai tidak terpakai, sebaiknya Anda mengevaluasi metrik dalam jangka waktu yang lebih lama. Pilih rentang waktu mulai dan waktu berakhir yang sesuai, seperti 30 hari, dan periode yang sesuai, seperti 86400.

Dalam data yang dikembalikan, setiap Jumlah di atas 0 menunjukkan bahwa tabel yang Anda evaluasi menerima lalu lintas baca selama periode tersebut.

Hasil berikut menunjukkan GSI yang menerima lalu lintas baca pada periode yang dievaluasi:

```
        {
          "Timestamp": "2022-08-17T21:20:00Z",
          "Sum": 36319167.0,
          "Unit": "Count"
        },
        {
          "Timestamp": "2022-08-11T21:20:00Z",
          "Sum": 1869136.0,
          "Unit": "Count"
        },
```

Hasil berikut menunjukkan GSI yang menerima lalu lintas baca minimal pada periode yang dievaluasi:

```
        {
          "Timestamp": "2022-08-28T21:20:00Z",
          "Sum": 0.0,
          "Unit": "Count"
        },
        {
          "Timestamp": "2022-08-15T21:20:00Z",
          "Sum": 2.0,
          "Unit": "Count"
        },
```

Hasil berikut menunjukkan GSI yang tidak menerima lalu lintas baca pada periode yang dievaluasi:

```
        {
          "Timestamp": "2022-08-17T21:20:00Z",
          "Sum": 0.0,
          "Unit": "Count"
        },
        {
          "Timestamp": "2022-08-11T21:20:00Z",
          "Sum": 0.0,
          "Unit": "Count"
        },
```


Membersihkan sumber daya GSI yang tidak terpakai

Jika telah mengidentifikasi GSI yang tidak terpakai, Anda dapat menghapusnya. Karena semua data yang ada di GSI juga ada di tabel dasar, cadangan tambahan tidak diperlukan sebelum menghapus GSI. Jika GSI diperlukan lagi di masa mendatang, GASI dapat ditambahkan kembali ke tabel.

Jika telah mengidentifikasi GSI yang jarang digunakan, Anda harus mempertimbangkan perubahan desain di aplikasi Anda yang memungkinkan Anda untuk menghapusnya atau mengurangi biayanya. Misalnya, meskipun pindaian DynamoDB harus digunakan dengan hemat karena pindaian dapat mengonsumsi sejumlah besar sumber daya sistem, pindaian tersebut mungkin lebih hemat biaya daripada GSI jika pola akses yang didukungnya sangat jarang digunakan.

Selain itu, jika GSI diperlukan untuk mendukung pola akses yang jarang, pertimbangkan untuk memproyeksikan serangkaian atribut yang lebih terbatas. Meskipun mungkin memerlukan kueri berikutnya terhadap tabel dasar untuk mendukung pola akses yang jarang, hal ini berpotensi memberikan pengurangan yang signifikan dalam biaya penyimpanan dan penulisan.

Membersihkan tabel global yang tidak terpakai

Tabel global Amazon DynamoDB menyediakan solusi terkelola penuh untuk men-deploy basis data multi-Wilayah dan multi-aktif, tanpa harus membangun dan memelihara solusi replikasi Anda sendiri.

Tabel global ideal untuk menyediakan akses latensi rendah ke data yang dekat dengan pengguna dan juga wilayah sekunder untuk pemulihan bencana.

Jika opsi tabel global diaktifkan untuk sumber daya dalam upaya menyediakan akses latensi rendah ke data tetapi bukan bagian dari strategi pemulihan bencana Anda, validasi bahwa kedua replika secara aktif melayani lalu lintas baca dengan mengevaluasi metriknya. CloudWatch Jika satu replika tidak melayani lalu lintas baca, replika tersebut mungkin merupakan sumber daya yang tidak terpakai.

Jika tabel global adalah bagian dari strategi pemulihan bencana Anda, satu replika yang tidak menerima lalu lintas baca mungkin diharapkan di bawah suatu active/standby pola.

Membersihkan cadangan atau point-in-time pemulihan yang tidak terpakai (PITR)

DynamoDB menawarkan dua gaya cadangan. Point-in-timepemulihan menyediakan pencadangan berkelanjutan hingga 35 hari untuk membantu Anda melindungi dari penulisan atau penghapusan yang tidak disengaja sementara cadangan sesuai permintaan memungkinkan pembuatan snapshot yang dapat disimpan dalam jangka panjang. Anda dapat mengatur periode pemulihan ke nilai apa pun antara 1 dan 35 hari. Kedua jenis pencadangan ini memiliki biaya yang terkait dengannya.

Lihat dokumentasi untuk Backup dan restore untuk DynamoDB dan Point-in-time backup untuk DynamoDB untuk menentukan apakah tabel Anda mengaktifkan cadangan yang mungkin tidak diperlukan lagi.

Evaluasi pola penggunaan tabel DynamoDB Anda

Bagian ini memberikan gambaran umum tentang cara mengevaluasi apakah Anda menggunakan tabel DynamoDB secara efisien. Ada pola penggunaan tertentu yang tidak optimal untuk DynamoDB, dan pola tersebut memberikan ruang untuk pengoptimalan baik dari sudut pandang performa maupun biaya.

Topik

- Lakukan lebih sedikit operasi bacaan sangat konsisten
- Lakukan lebih sedikit transaksi untuk operasi baca
- Lakukan lebih sedikit pemindaian
- Gunakan nama atribut yang lebih pendek
- Aktifkan Waktu untuk Tayang (TTL)
- Ganti tabel global dengan cadangan lintas Wilayah

Lakukan lebih sedikit operasi bacaan sangat konsisten

DynamoDB memungkinkan Anda mengonfigurasi konsistensi baca berdasarkan permintaan. Permintaan baca pada akhirnya konsisten secara default. Bacaan akhir konsisten dikenakan 0,5 RCU untuk data hingga 4 KB.

Sebagian besar beban kerja terdistribusi bersifat fleksibel dan dapat menoleransi konsistensi akhir. Namun, mungkin ada pola akses yang membutuhkan bacaan sangat konsisten. Bacaan sangat

konsisten dikenakan 1 RCU untuk data hingga 4 KB, yang pada dasarnya menggandakan biaya baca Anda. DynamoDB memberi Anda fleksibilitas untuk menggunakan kedua model konsistensi pada tabel yang sama.

Anda dapat mengevaluasi beban kerja dan kode aplikasi untuk mengonfirmasi apakah bacaan sangat konsisten hanya digunakan jika diperlukan.

Lakukan lebih sedikit transaksi untuk operasi baca

DynamoDB memungkinkan Anda untuk mengelompokkan tindakan tertentu dengan cara tertentu, all-or-nothing yang berarti Anda memiliki kemampuan untuk mengeksekusi transaksi ACID dengan DynamoDB. Namun, seperti halnya basis data relasional, pendekatan ini tidak perlu diikuti untuk setiap tindakan.

Operasi baca transaksional hingga 4 KB mengkonsumsi 2 RCUs sebagai lawan dari default 0,5 RCUs untuk membaca jumlah data yang sama dengan cara yang akhirnya konsisten. Biaya digandakan dalam operasi tulis yang berarti, penulisan transaksional hingga 1 KB sama dengan 2. WCUs

Untuk menentukan apakah semua operasi pada tabel Anda adalah transaksi, CloudWatch metrik untuk tabel dapat disaring ke transaksi APIs. Jika transaksi APIs adalah satu-satunya grafik yang tersedia di bawah `SuccessfulRequestLatency` metrik untuk tabel, ini akan mengkonfirmasi bahwa setiap operasi adalah transaksi untuk tabel ini. Atau, jika tren pemanfaatan kapasitas keseluruhan sesuai dengan tren API transaksi, pertimbangkan untuk meninjau kembali desain aplikasi karena tampaknya didominasi oleh transaksional. APIs

Lakukan lebih sedikit pemindaian

Penggunaan operasi `Scan` yang ekstensif umumnya berasal dari kebutuhan untuk menjalankan kueri analitis pada data DynamoDB. Menjalankan operasi `Scan` yang sering pada tabel besar bisa menjadi tidak efisien dan mahal.

Alternatif yang lebih baik adalah dengan menggunakan fitur Ekspor ke S3 dan memilih titik waktu untuk mengekspor status tabel ke S3. AWS menawarkan layanan seperti Athena yang kemudian dapat digunakan untuk menjalankan kueri analitis pada data tanpa menghabiskan kapasitas apa pun dari tabel.

Frekuensi operasi `Scan` dapat ditentukan menggunakan statistik `SampleCount` di bagian metrik `SuccessfulRequestLatency` untuk`Scan`. Jika operasi `Scan` memang sangat sering, pola akses dan model data harus dievaluasi ulang.

Gunakan nama atribut yang lebih pendek

Ukuran total item di DynamoDB adalah jumlah dari panjang dan nilai nama atributnya. Memiliki nama atribut yang panjang tidak hanya berkontribusi terhadap biaya penyimpanan, tetapi juga dapat menyebabkan RCU/WCU konsumsi yang lebih tinggi. Sebaiknya Anda memilih nama atribut yang lebih pendek. Memiliki nama atribut yang lebih pendek dapat membantu membatasi ukuran item dalam batas 4KB/1KB berikutnya setelah itu Anda akan menggunakan tambahan untuk mengakses data. RCU/WCU

Aktifkan Waktu untuk Tayang (TTL)

Waktu untuk Tayang (TTL) dapat mengidentifikasi item yang lebih lama dari waktu kedaluwarsa yang telah Anda tetapkan pada suatu item dan menghapus item tersebut dari tabel. Jika data Anda bertambah seiring waktu dan data lama menjadi tidak relevan, mengaktifkan TTL pada tabel dapat membantu memangkas data Anda dan menghemat biaya penyimpanan.

Aspek lain yang berguna dari TTL adalah bahwa item yang kedaluwarsa terjadi pada aliran DynamoDB Anda, jadi daripada hanya menghapus data dari data Anda, Anda dapat menggunakan item tersebut dari aliran dan mengarsipkannya ke tingkat penyimpanan berbiaya lebih rendah. Selain itu, menghapus item melalui TTL tidak dikenakan biaya tambahan — tidak menggunakan kapasitas, dan tidak ada biaya tambahan untuk merancang aplikasi pembersihan.

Ganti tabel global dengan cadangan lintas Wilayah

Tabel global memungkinkan Anda mengelola beberapa tabel replika aktif di Wilayah berbeda — semuanya dapat menerima operasi tulis dan mereplikasi data satu sama lain. Sangat mudah untuk mengatur replika dan sinkronisasi dikelola untuk Anda. Replika menyatu ke keadaan konsisten menggunakan strategi penulis terakhir menang.

Jika Anda menggunakan tabel Global semata-mata sebagai bagian dari strategi failover atau pemulihan bencana (DR), Anda dapat menggantinya dengan salinan cadangan lintas Wilayah untuk sasaran titik pemulihan yang relatif ringan dan persyaratan sasaran waktu pemulihan. Jika Anda tidak memerlukan akses lokal yang cepat dan ketersediaan lima sembilan yang tinggi, mempertahankan replika tabel global mungkin bukan pendekatan terbaik untuk failover.

Sebagai alternatif, pertimbangkan untuk menggunakan AWS Backup untuk mengelola backup DynamoDB. Anda dapat menjadwalkan pencadangan rutin dan menyalinnya di seluruh Wilayah untuk memenuhi persyaratan DR dengan pendekatan yang lebih hemat biaya dibandingkan menggunakan tabel Global.

Evaluasi penggunaan aliran DynamoDB Anda

Bagian ini memberikan gambaran umum tentang cara mengevaluasi penggunaan DynamoDB Streams. Ada pola penggunaan tertentu yang tidak optimal untuk DynamoDB, dan pola tersebut memberikan ruang untuk pengoptimalan baik dari sudut pandang performa maupun biaya.

Anda memiliki dua integrasi streaming asli untuk kasus penggunaan berdasarkan peristiwa dan streaming:

- Amazon DynamoDB Streams
- Amazon Kinesis Data Streams

Halaman ini hanya akan fokus pada strategi pengoptimalan biaya untuk opsi ini. Jika Anda ingin mengetahui cara memilih di antara kedua opsi tersebut, lihat Opsi streaming untuk tangkapan data perubahan.

Topik

- Mengoptimalkan biaya untuk DynamoDB Streams
- Mengoptimalkan biaya untuk Kinesis Data Streams
- Strategi pengoptimalan biaya untuk kedua jenis penggunaan Streams

Mengoptimalkan biaya untuk DynamoDB Streams

Seperti yang disebutkan di halaman harga untuk DynamoDB Streams, terlepas dari mode kapasitas throughput tabel, DynamoDB membebankan biaya pada jumlah permintaan baca yang dibuat terhadap DynamoDB Stream tabel. Permintaan baca yang dibuat terhadap DynamoDB Stream berbeda dari permintaan baca yang dibuat terhadap tabel DynamoDB.

Setiap permintaan baca dalam aliran berbentuk panggilan API `GetRecords` yang dapat mengembalikan hingga 1000 catatan atau catatan senilai 1 MB sebagai respons, mana saja yang dicapai terlebih dahulu. Tak satu pun dari DynamoDB APIs Stream lainnya yang dibebankan dan DynamoDB Streams tidak dikenakan biaya karena idle. Dengan kata lain, jika tidak ada permintaan baca yang dibuat ke DynamoDB Stream, mengaktifkan DynamoDB Stream pada tabel tidak akan dikenakan biaya.

Berikut adalah beberapa aplikasi konsumen untuk DynamoDB Streams:

- AWS Lambda fungsi

- Aplikasi berbasis Amazon Kinesis Data Streams
- Aplikasi konsumen pelanggan yang dibuat menggunakan AWS SDK

Permintaan baca yang dibuat oleh konsumen DynamoDB Streams AWS berbasis Lambda gratis, sedangkan panggilan yang dilakukan oleh konsumen jenis lain dikenakan biaya. Setiap bulan, 2.500.000 permintaan baca pertama yang dibuat oleh konsumen non-Lambda juga tidak dikenakan biaya. Ini berlaku untuk semua permintaan baca yang dibuat ke DynamoDB Streams apa pun di AWS Akun untuk setiap Wilayah. AWS

Memantau penggunaan DynamoDB Streams

Biaya DynamoDB Streams pada konsol penagihan dikelompokkan bersama untuk semua DynamoDB Streams di seluruh Wilayah dalam Akun. AWS AWS Saat ini, penandaan DynamoDB Streams tidak didukung, sehingga tanda alokasi biaya tidak dapat digunakan untuk mengidentifikasi biaya mendetail untuk DynamoDB Streams. Volume panggilan `GetRecords` dapat diperoleh di tingkat DynamoDB Stream untuk menghitung biaya per aliran. Volume diwakili oleh `SuccessfulRequestLatency` metrik DynamoDB Stream dan CloudWatch statistiknya. `SampleCount` Metrik ini juga akan mencakup `GetRecords` panggilan yang dibuat oleh tabel global untuk melakukan replikasi yang sedang berlangsung serta panggilan yang dilakukan oleh konsumen AWS Lambda, yang keduanya tidak dikenakan biaya. Untuk informasi tentang CloudWatch metrik lain yang diterbitkan oleh DynamoDB Streams, lihat. Dimensi dan Metrik DynamoDB

Menggunakan AWS Lambda sebagai konsumen

Mengevaluasi jika menggunakan fungsi AWS Lambda sebagai konsumen untuk DynamoDB Streams layak karena dapat menghilangkan biaya yang terkait dengan pembacaan dari DynamoDB Stream. Di sisi lain, aplikasi konsumen berbasis SDK atau Adaptor DynamoDB Streams Kinesis akan dikenakan biaya pada jumlah panggilan `GetRecords` yang dilakukan terhadap DynamoDB Stream.

Invokasi fungsi Lambda akan dikenakan biaya berdasarkan harga Lambda standar, tetapi DynamoDB Streams tidak membebankan biaya apa pun. Lambda akan melakukan polling pada pecahan di DynamoDB Stream Anda untuk catatan dengan tingkat dasar sebanyak 4 kali per detik. Setelah catatan tersedia, Lambda menginvokasi fungsi Anda dan menunggu hasilnya. Jika pemrosesan berhasil, Lambda melanjutkan polling sampai menerima lebih banyak catatan.

Menyetel aplikasi konsumen berbasis Adaptor DynamoDB Streams Kinesis

Karena permintaan baca yang dibuat oleh konsumen berbasis non-Lambda dikenakan biaya untuk DynamoDB Streams, penting untuk menemukan keseimbangan antara persyaratan mendekati waktu nyata dan frekuensi aplikasi konsumen harus melakukan polling terhadap DynamoDB Stream.

Frekuensi polling pada DynamoDB Streams menggunakan aplikasi berbasis Adaptor DynamoDB Streams Kinesis ditentukan oleh konfigurasi nilai `idleTimeBetweenReadsInMillis`. Parameter ini menentukan waktu tunggu konsumen dalam milidetik sebelum memproses pecahan jika panggilan `GetRecords` sebelumnya yang dilakukan ke pecahan yang sama tidak menghasilkan catatan apa pun. Secara default, nilai untuk parameter ini adalah 1000 ms. Jika pemrosesan mendekati waktu nyata tidak diperlukan, parameter ini dapat ditingkatkan agar aplikasi konsumen melakukan lebih sedikit panggilan `GetRecords` dan mengoptimalkan panggilan DynamoDB Streams.

Mengoptimalkan biaya untuk Kinesis Data Streams

Jika Kinesis Data Stream ditetapkan sebagai tujuan untuk mengirimkan peristiwa pengambilan data perubahan untuk tabel DynamoDB, Kinesis Data Stream mungkin memerlukan manajemen ukuran terpisah yang akan memengaruhi biaya keseluruhan. DynamoDB mengenakan biaya dalam hal Change Data capture Units CDUs () di mana setiap unit terdiri dari ukuran item DynamoDB 1 KB yang dicoba oleh layanan DynamoDB ke Kinesis Data Stream tujuan.

Selain biaya oleh layanan DynamoDB, biaya Kinesis Data Stream standar akan dikenakan. Seperti yang disebutkan di halaman harga, harga layanan berbeda berdasarkan mode kapasitas - disediakan dan sesuai permintaan, yang berbeda dari mode kapasitas tabel DynamoDB dan ditentukan pengguna. Pada tingkat tinggi, Kinesis Data Streams membebankan tarif per jam berdasarkan mode kapasitas, serta data yang diserap ke dalam aliran oleh layanan DynamoDB. Mungkin ada biaya tambahan seperti pengambilan data (untuk mode sesuai permintaan), retensi data yang diperpanjang (melebihi default 24 jam), dan peningkatan pengambilan konsumen fan-out tergantung pada konfigurasi pengguna untuk Kinesis Data Stream.

Memantau penggunaan Kinesis Data Streams

Kinesis Data Streams untuk DynamoDB menerbitkan metrik dari DynamoDB selain Metrik Aliran Data Kinesis standar. CloudWatch Ada kemungkinan bahwa `Put` upaya oleh layanan DynamoDB dibatasi oleh layanan Kinesis karena kapasitas Kinesis Data Streams tidak mencukupi, atau oleh komponen dependen seperti layanan yang dapat dikonfigurasi untuk mengenkripsi data Kinesis Data Stream saat AWS KMS istirahat.

Untuk mempelajari lebih lanjut tentang CloudWatch metrik yang diterbitkan oleh layanan DynamoDB untuk Kinesis Data Stream, lihat. Memantau pengambilan data perubahan dengan Kinesis Data

Streams Untuk menghindari biaya tambahan percobaan ulang layanan karena throttling, penting untuk menentukan ukuran Kinesis Data Stream yang tepat dalam Mode yang Disediakan.

Memilih mode kapasitas yang tepat untuk Kinesis Data Streams

Kinesis Data Streams didukung dalam dua mode kapasitas — mode disediakan dan mode sesuai permintaan.

- Jika beban kerja yang melibatkan Kinesis Data Stream memiliki lalu lintas aplikasi yang dapat diprediksi, lalu lintas yang konsisten atau meningkat secara bertahap, atau lalu lintas yang dapat diperkirakan dengan akurat, maka mode yang disediakan untuk Kinesis Data Streams akan cocok dan lebih hemat biaya
- Jika beban kerja terhitung baru, memiliki lalu lintas aplikasi yang tidak dapat diprediksi, atau Anda memilih untuk tidak mengelola kapasitas, maka mode sesuai permintaan untuk Kinesis Data Streams akan cocok dan lebih hemat biaya

Praktik terbaik untuk mengoptimalkan biaya adalah dengan mengevaluasi apakah tabel DynamoDB yang terkait dengan Kinesis Data Stream memiliki pola lalu lintas yang dapat diprediksi yang dapat memanfaatkan mode yang disediakan untuk Kinesis Data Streams. Jika beban kerja baru, Anda dapat menggunakan mode sesuai permintaan untuk Kinesis Data Streams selama beberapa minggu awal, meninjau CloudWatch metrik untuk memahami pola lalu lintas, lalu mengalihkan Stream yang sama ke mode yang disediakan berdasarkan sifat beban kerja. Dalam kasus mode yang disediakan, estimasi jumlah pecahan dapat dilakukan dengan mengikuti pertimbangan manajemen pecahan untuk Kinesis Data Streams.

Evaluasi aplikasi konsumen Anda menggunakan Kinesis Data Streams untuk DynamoDB

Karena Kinesis Data Streams tidak membebankan biaya pada jumlah panggilan `GetRecords` seperti DynamoDB Streams, aplikasi konsumen dapat membuat panggilan sebanyak mungkin, selama frekuensinya berada di bawah batas throttling untuk `GetRecords`. Terkait mode sesuai permintaan untuk Kinesis Data Streams, pembacaan data dikenakan biaya per GB. Untuk mode yang disediakan pada Kinesis Data Streams, pembacaan tidak dikenakan biaya jika data berumur kurang dari 7 hari. Dalam kasus fungsi Lambda sebagai konsumen Kinesis Data Streams, Lambda melakukan polling pada setiap pecahan di Kinesis Stream Anda untuk catatan dengan tingkat dasar sebanyak sekali per detik.

Strategi pengoptimalan biaya untuk kedua jenis penggunaan Streams

Pemfilteran acara untuk konsumen AWS Lambda

Pemfilteran peristiwa Lambda memungkinkan Anda untuk membuang peristiwa berdasarkan kriteria filter agar tidak tersedia dalam batch invokasi fungsi Lambda. Ini mengoptimalkan biaya Lambda untuk memproses atau membuang catatan aliran yang tidak diinginkan dalam logika fungsi konsumen. Untuk mempelajari selengkapnya tentang mengonfigurasi pemfilteran peristiwa dan menulis kriteria pemfilteran Anda, lihat Pemfilteran peristiwa Lambda.

Tuning AWS Lambda konsumen

Biaya dapat lebih dioptimalkan dengan menyetel parameter konfigurasi Lambda seperti meningkatkan `BatchSize` untuk memproses lebih banyak per invokasi, mengaktifkan `BisectBatchOnFunctionError` untuk mencegah pemrosesan duplikat (yang menimbulkan biaya tambahan), dan mengatur `MaximumRetryAttempts` agar tidak mengalami terlalu banyak percobaan ulang. Secara default, invokasi Lambda konsumen yang gagal akan dicoba ulang tanpa batas waktu hingga catatan kedaluwarsa dari aliran, yaitu sekitar 24 jam untuk DynamoDB Streams dan dapat dikonfigurasi mulai dari 24 jam hingga 1 tahun untuk Kinesis Data Streams. Opsi konfigurasi Lambda tambahan yang tersedia termasuk yang disebutkan di atas untuk konsumen DynamoDB Stream dapat dilihat di panduan developer AWS Lambda.

Evaluasi kapasitas yang disediakan untuk penyediaan ukuran yang tepat di tabel DynamoDB

Bagian ini memberikan gambaran umum tentang cara mengevaluasi apakah Anda memiliki ukuran penyediaan yang tepat pada tabel DynamoDB Anda. Seiring berkembangnya beban kerja Anda, Anda harus memodifikasi prosedur operasional dengan tepat, terutama ketika tabel DynamoDB Anda dikonfigurasi dalam mode yang disediakan dan Anda memiliki risiko untuk menyediakan tabel secara berlebihan atau kurang.

Prosedur yang dijelaskan di bawah memerlukan informasi statistik yang harus diambil dari tabel DynamoDB yang mendukung aplikasi produksi Anda. Untuk memahami perilaku aplikasi Anda, Anda harus menentukan periode waktu yang cukup signifikan untuk mengambil data musiman dari aplikasi Anda. Misalnya, jika aplikasi Anda menunjukkan pola mingguan, menggunakan periode tiga minggu akan memberi Anda cukup ruang untuk menganalisis kebutuhan throughput aplikasi.

Jika Anda tidak tahu harus mulai dari mana, gunakan penggunaan data setidaknya selama satu bulan untuk penghitungan di bawah ini.

Saat mengevaluasi kapasitas, tabel DynamoDB dapat mengkonfigurasi Unit Kapasitas Baca RCUs () dan Unit Kapasitas Tulis (WCU) secara independen. Jika tabel Anda memiliki Indeks Sekunder

Global (GSI) yang dikonfigurasi, Anda harus menentukan throughput yang akan dikonsumsinya, yang juga akan independen dari RCUs dan WCUs dari tabel dasar.

> **Note**
>
> Indeks Sekunder Lokal (LSI) menggunakan kapasitas dari tabel dasar.

Topik

- Cara mengambil metrik penggunaan pada tabel DynamoDB Anda
- Cara mengidentifikasi tabel DynamoDB yang kurang tersedia
- Cara mengidentifikasi tabel DynamoDB yang disediakan secara berlebihan

Cara mengambil metrik penggunaan pada tabel DynamoDB Anda

Untuk mengevaluasi tabel dan kapasitas GSI, pantau CloudWatch metrik berikut dan pilih dimensi yang sesuai untuk mengambil informasi tabel atau GSI:

Unit Kapasitas Baca	Unit Kapasitas Tulis
`ConsumedReadCapacityUnits`	`ConsumedWriteCapacityUnits`
`ProvisionedReadCapacityUnits`	`ProvisionedWriteCapacityUnits`
`ReadThrottleEvents`	`WriteThrottleEvents`

Anda dapat melakukan ini baik melalui AWS CLI atau AWS Management Console.

AWS CLI

Sebelum kita mengambil metrik konsumsi tabel, kita harus mulai dengan menangkap beberapa titik data historis menggunakan API. CloudWatch

Mulailah dengan membuat dua file: `write-calc.json` dan `read-calc.json`. File-file ini akan mewakili penghitungan untuk tabel atau GSI. Anda harus memperbarui beberapa bidang, seperti yang ditunjukkan pada tabel di bawah ini, agar sesuai dengan lingkungan Anda.

Nama Bidang	Definisi	Contoh
`<table-name>`	Nama tabel yang akan dianalisis	SampleTable
`<period>`	Jangka waktu yang akan Anda gunakan untuk mengevaluasi pemanfaatan target, berdasarkan detik	Untuk periode 1 jam, Anda harus menentukan: 3600
`<start-time>`	Awal interval evaluasi Anda, ditentukan dalam format ISO86 01	2022-02-21T23:00:00
`<end-time>`	Akhir interval evaluasi Anda, ditentukan dalam format ISO86 01	2022-02-22T06:00:00

File penghitungan tulis akan mengambil jumlah WCU yang disediakan dan terpakai dalam rentang tanggal tertentu. Ini juga akan menghasilkan persentase penggunaan yang akan digunakan untuk analisis. Konten lengkap file `write-calc.json` akan seperti berikut:

```
{
  "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
}
```

File penghitungan baca menggunakan file serupa. File ini akan mengambil berapa banyak RCUs yang disediakan dan dikonsumsi selama periode waktu untuk rentang tanggal yang ditentukan. Konten file `read-calc.json` akan seperti berikut:

```
{
  "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
}
```

Setelah membuat file, Anda dapat mulai mengambil data penggunaan.

1. Untuk mengambil data penggunaan tulis, keluarkan perintah berikut:

   ```
   aws cloudwatch get-metric-data --cli-input-json file://write-calc.json
   ```

2. Untuk mengambil data penggunaan baca, keluarkan perintah berikut:

   ```
   aws cloudwatch get-metric-data --cli-input-json file://read-calc.json
   ```

Hasil untuk kedua kueri akan menjadi serangkaian titik data dalam format JSON yang akan digunakan untuk analisis. Hasil Anda akan tergantung pada jumlah titik data yang Anda tentukan, periode, dan data beban kerja spesifik Anda sendiri. Hasilnya akan seperti berikut:

```
{
    "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": []
}
```

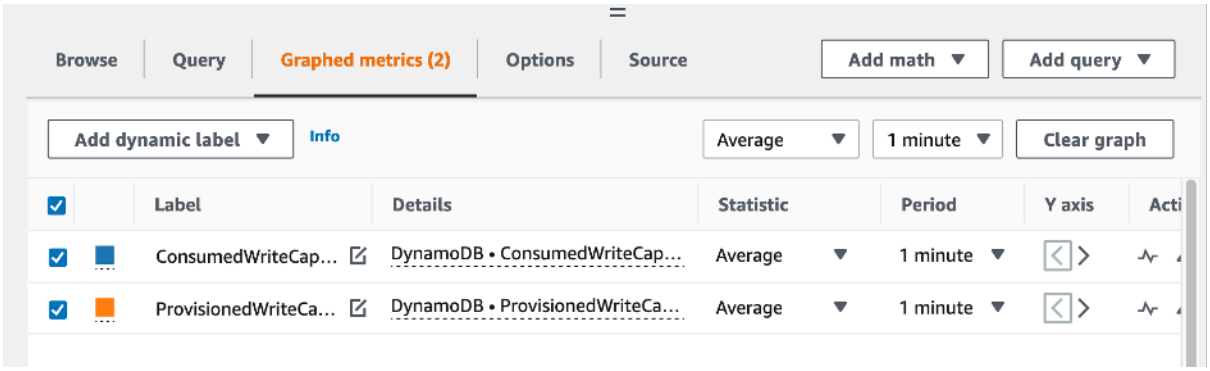
> **Note**
>
> Jika menentukan jangka waktu pendek dan rentang waktu yang lama, Anda mungkin perlu memodifikasi `MaxDatapoints` yang secara default ditetapkan ke 24 dalam skrip. Ini mewakili satu titik data per jam dan 24 per hari.

AWS Management Console

1. Masuk ke AWS Management Console dan arahkan ke halaman CloudWatch layanan. Pilih yang sesuai Wilayah AWS jika perlu.
2. Temukan bagian Metrik di bilah navigasi kiri dan pilih Semua metrik.
3. Ini akan membuka dasbor dengan dua panel. Panel atas akan menampilkan grafik, dan panel bawah akan menampilkan metrik yang ingin Anda grafik. Pilih DynamoDB.
4. Pilih Metrik Tabel. Ini akan menampilkan tabel di Wilayah Anda saat ini.
5. Gunakan kotak Pencarian untuk mencari nama tabel Anda dan memilih metrik operasi tulis: `ConsumedWriteCapacityUnits` dan `ProvisionedWriteCapacityUnits`

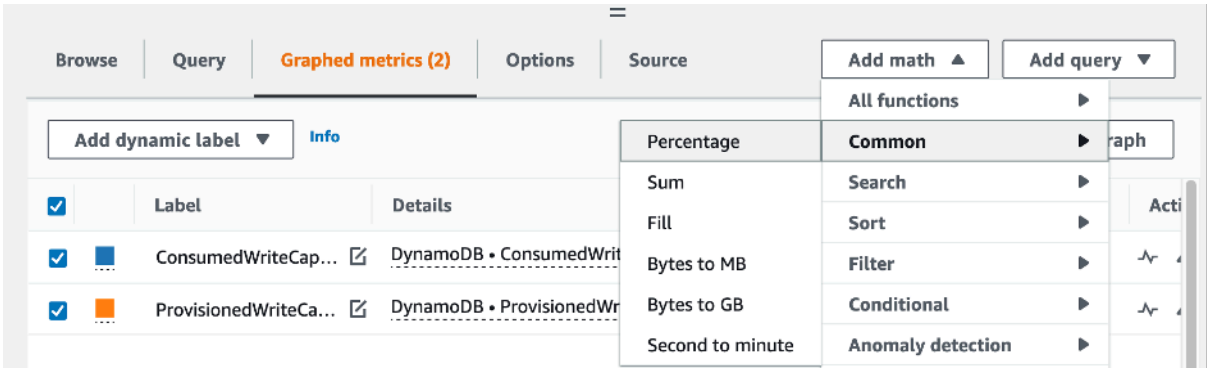
> **Note**
>
> Contoh ini membahas metrik operasi tulis, tetapi Anda juga dapat menggunakan langkah-langkah ini untuk membuat grafik metrik operasi baca.

6. Pilih tab Graphed metrics (2) untuk memodifikasi rumus. Secara default, CloudWatch memilih fungsi statistik Rata-rata untuk grafik.

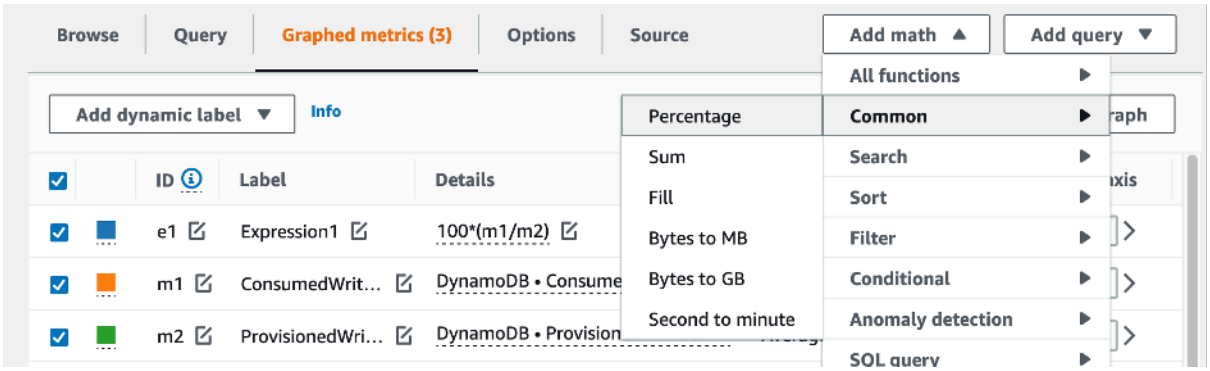


7. Saat memilih kedua metrik bergrafik (kotak centang di sebelah kiri) pilih menu Tambahkan perhitungan, diikuti oleh Umum, lalu pilih fungsi Persentase. Ulangi prosedur ini dua kali.

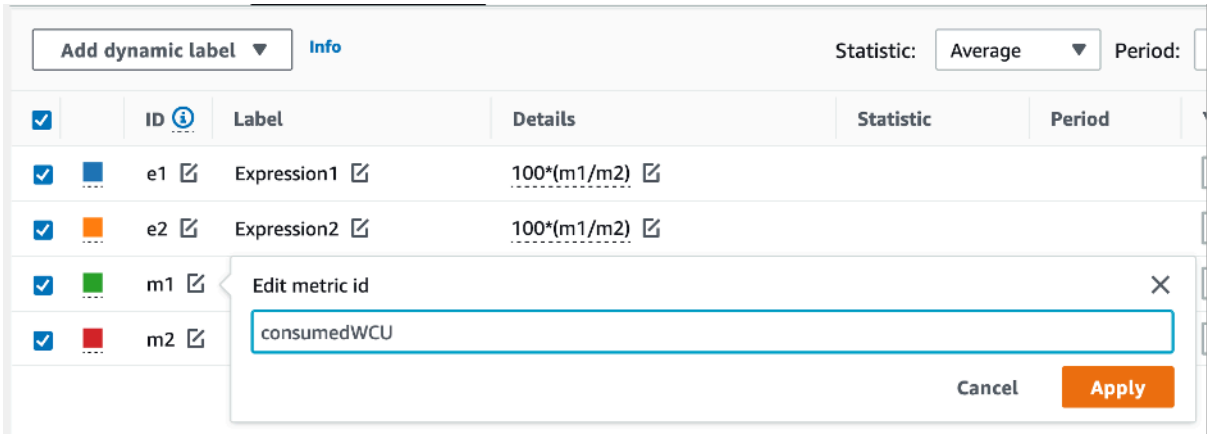
   Pertama kali memilih fungsi Persentase:



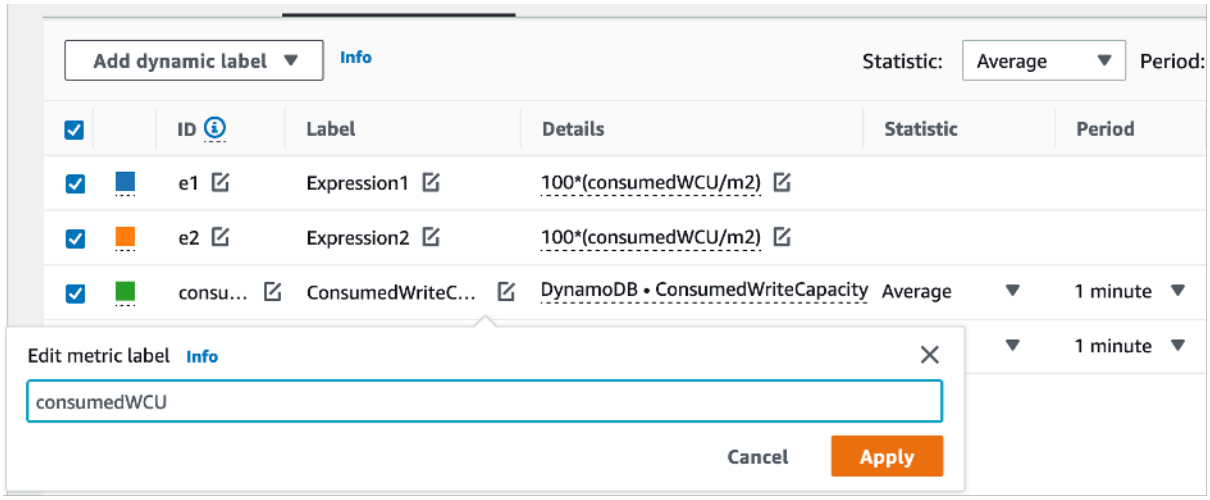
   Kedua kali memilih fungsi Persentase:



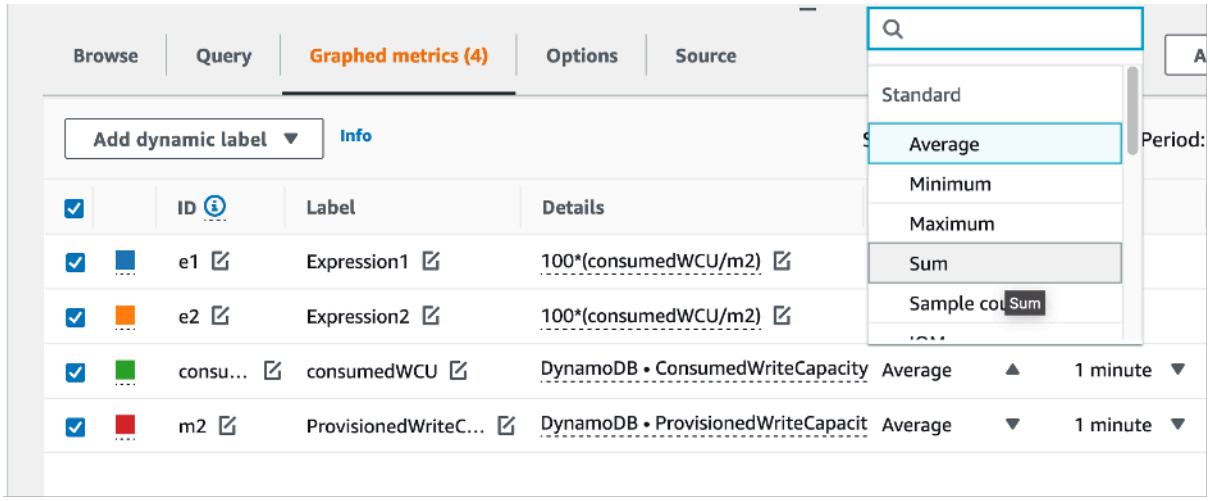
8. Pada titik ini, Anda memiliki empat metrik di menu bawah. Mari kita kerjakan penghitungan `ConsumedWriteCapacityUnits`. Agar konsisten, kita perlu mencocokkan nama untuk yang kita gunakan di AWS CLI bagian ini. Klik m1 ID dan ubah nilai ini menjadi consumedWCU.

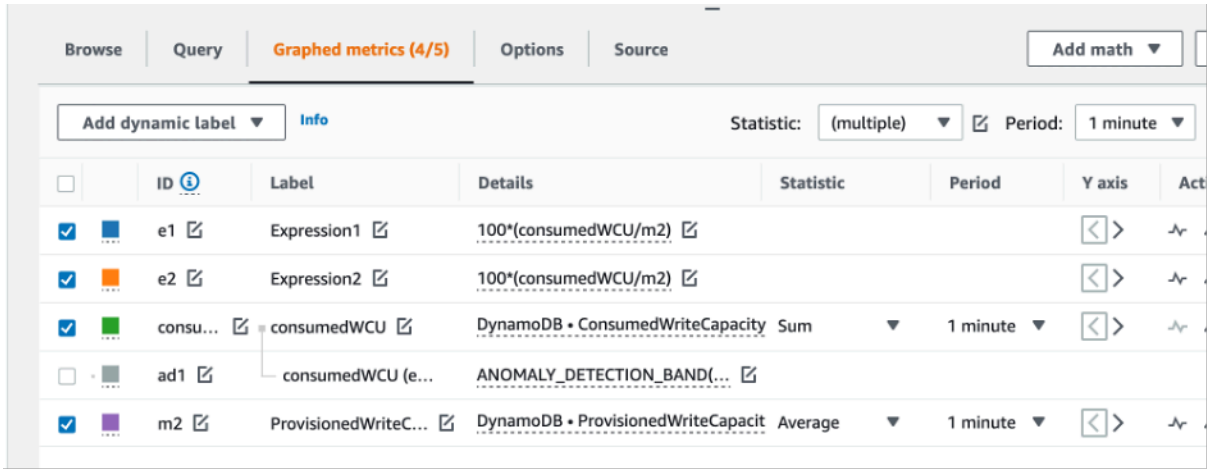


Ganti nama ConsumedWriteCapacityUnitlabel sebagai**`consumedWCU`**.

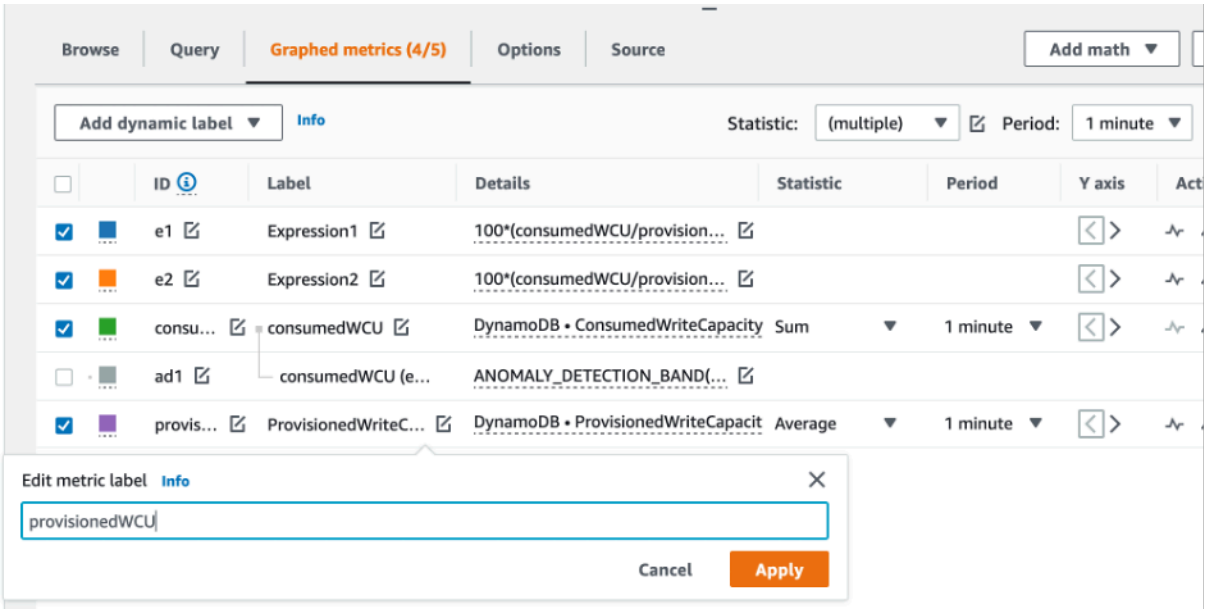


9. Ubah statistik dari Average menjadi Sum. Tindakan ini secara otomatis akan membuat metrik lain yang disebut ANOMALY_DETECTION_BAND. Untuk cakupan prosedur ini, mari kita abaikan dengan menghapus kotak centang pada metrik ad1 yang baru dibuat.

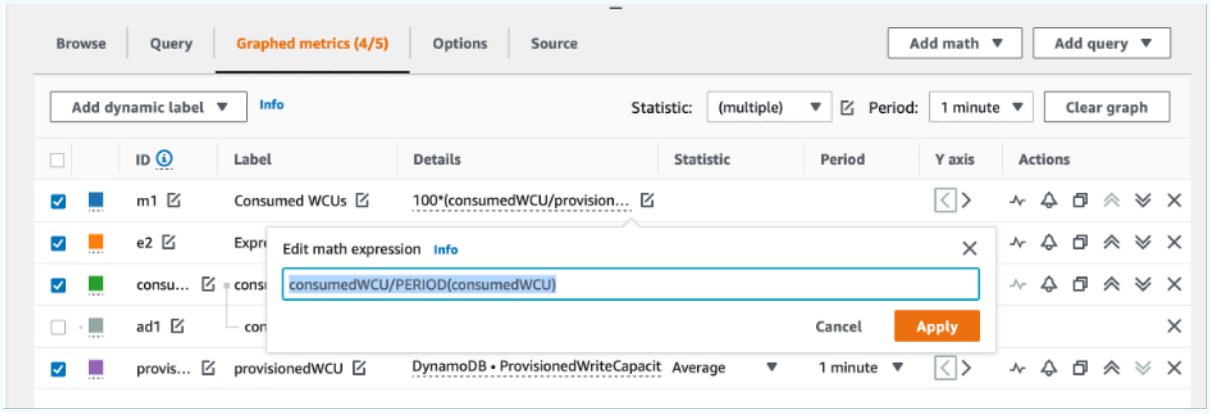




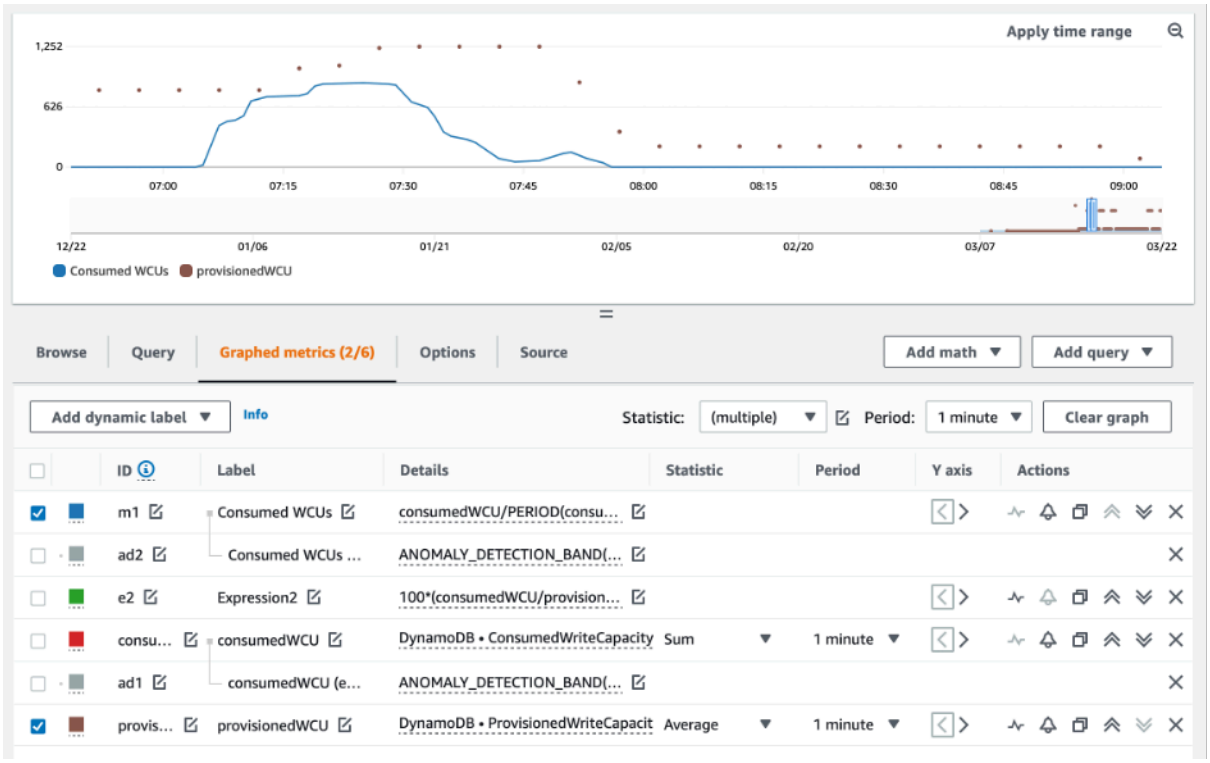
10. Ulangi langkah 8 untuk mengganti nama m2 ID menjadi ProvisionedWCU. Biarkan statistik diatur ke Average.



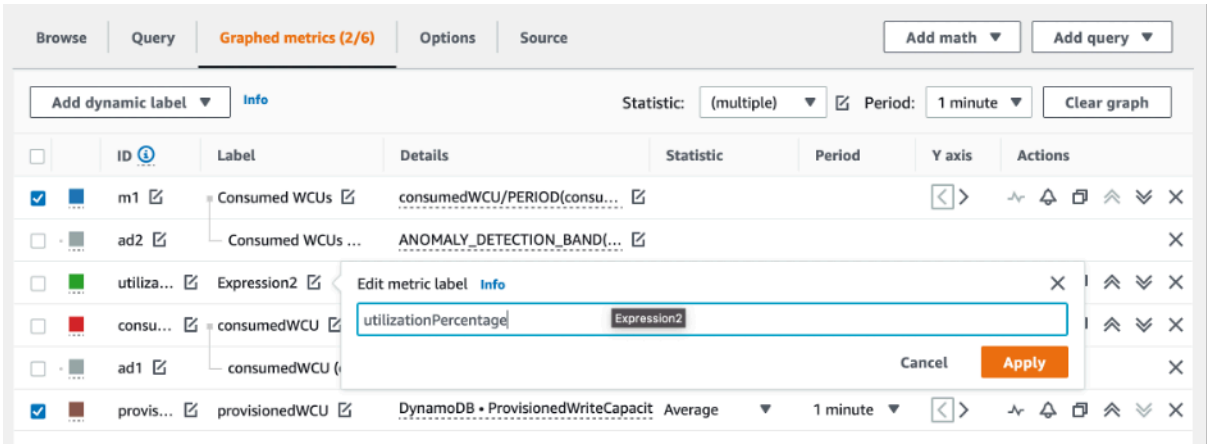
11. Pilih label Expression1 dan perbarui nilainya ke m1 dan label ke Consumed. WCUs

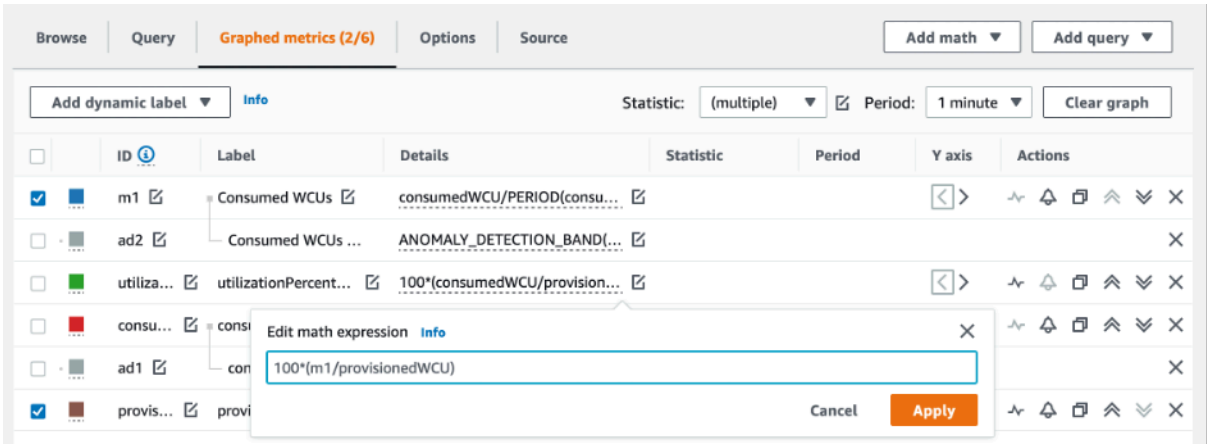
> **Note**
>
> Pastikan Anda hanya memilih m1 (kotak centang di sebelah kiri) dan ProvisionedWCU untuk memvisualisasikan data dengan benar. Perbarui rumus dengan mengklik Detail dan mengubah rumus menjadi consumedWCU/PERIOD(consumedWCU). Langkah ini mungkin juga menghasilkan metrik ANOMALY_DETECTION_BAND, tetapi kita dapat mengabaikannya untuk cakupan prosedur ini.
>
> 
>

12. Anda seharusnya sekarang memiliki dua grafik: satu yang menunjukkan Anda disediakan WCUs di atas meja dan satu lagi yang menunjukkan yang dikonsumsi. WCUs Bentuk grafiknya mungkin berbeda dengan gambar di bawah ini, tetapi Anda dapat menggunakannya sebagai referensi:



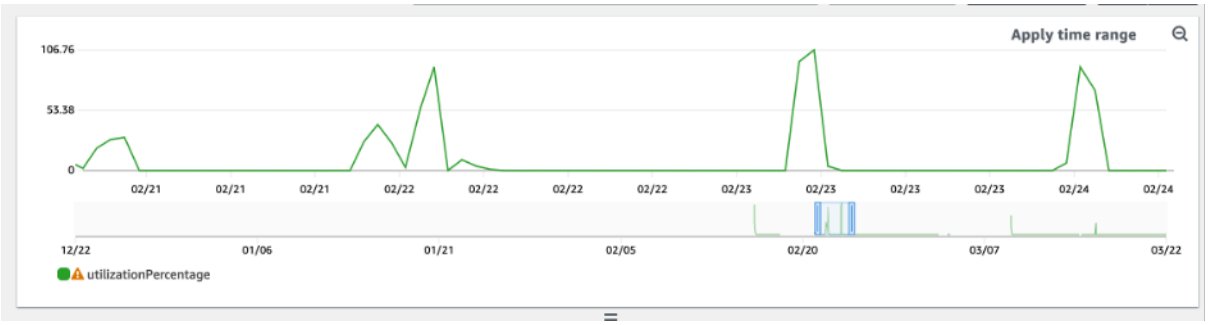
13. Perbarui rumus persentase dengan memilih grafik Expression2 (e2). Ganti nama label dan IDs UtilizationPercentage. Ganti nama rumus agar sesuai dengan 100*(m1/provisionedWCU).





14. Hapus centang dari semua metrik kecuali utilizationPercentage untuk memvisualisasikan pola penggunaan Anda. Interval default diatur ke 1 menit, tetapi Anda dapat mengubahnya sesuai kebutuhan.

Berikut adalah tampilan periode waktu yang lebih lama serta periode yang lebih dari 1 jam. Anda dapat melihat ada beberapa interval yang penggunaannya lebih tinggi dari 100%, tetapi beban kerja khusus ini memiliki interval yang lebih panjang dengan penggunaan nol.



Pada titik ini, Anda mungkin mendapatkan hasil yang berbeda dari gambar dalam contoh ini. Itu semua tergantung pada data dari beban kerja Anda. Interval dengan penggunaan lebih dari 100% rentan terhadap peristiwa throttling. DynamoDB menawarkan kapasitas lonjakan, tetapi segera setelah kapasitas lonjakan dilakukan, apa pun di atas 100% akan dibatasi.

Cara mengidentifikasi tabel DynamoDB yang kurang tersedia

Untuk sebagian besar beban kerja, tabel dianggap kurang tersedia jika terus-menerus menggunakan lebih dari 80% kapasitas yang disediakan.

Kapasitas burst adalah fitur DynamoDB yang memungkinkan pelanggan untuk sementara mengkonsumsi RCUs/WCUs lebih dari yang disediakan semula (lebih dari throughput yang disediakan per detik yang ditentukan dalam tabel). Kapasitas lonjakan diciptakan untuk menyerap peningkatan lalu lintas tiba-tiba karena peristiwa khusus atau lonjakan penggunaan. Kapasitas lonjakan ini tidak bertahan selamanya. Segera setelah tidak digunakan RCUs dan habis, Anda akan WCUs terhambat jika Anda mencoba untuk mengkonsumsi lebih banyak kapasitas daripada yang disediakan. Ketika lalu lintas aplikasi Anda mendekati tingkat penggunaan 80%, risiko throttling Anda jauh lebih tinggi.

Aturan tingkat penggunaan 80% bervariasi berdasarkan musim data dan pertumbuhan lalu lintas Anda. Pertimbangkan skenario berikut:

- Jika lalu lintas Anda stabil pada tingkat penggunaan ~90% selama 12 bulan terakhir, tabel Anda memiliki kapasitas yang tepat
- Jika lalu lintas aplikasi Anda tumbuh sebesar 8% setiap bulan dalam waktu kurang dari 3 bulan, Anda akan mencapai 100%
- Jika lalu lintas aplikasi Anda tumbuh sebesar 5% dalam waktu lebih dari 4 bulan, Anda masih akan mencapai 100%

Hasil dari kueri di atas memberikan gambaran tingkat penggunaan Anda. Gunakan hasil tersebut sebagai panduan untuk mengevaluasi lebih lanjut metrik lain yang dapat membantu Anda meningkatkan kapasitas tabel sesuai kebutuhan (misalnya: tingkat pertumbuhan bulanan atau mingguan). Bekerjalah dengan tim operasi Anda untuk menentukan persentase yang baik untuk beban kerja dan tabel Anda.

Ada skenario khusus, datanya miring ketika dianalisis setiap hari atau setiap minggu. Misalnya, dengan aplikasi musiman yang memiliki lonjakan penggunaan selama jam kerja (tetapi kemudian turun menjadi hampir nol di luar jam kerja), Anda bisa mendapatkan keuntungan dengan menjadwalkan penskalaan otomatis di mana Anda menentukan jam dalam sehari (dan hari dalam seminggu) untuk meningkatkan kapasitas yang disediakan dan kapan harus menguranginya. Alih-alih bertujuan untuk kapasitas yang lebih tinggi sehingga Anda dapat menutupi jam sibuk, Anda juga dapat memanfaatkan konfigurasi penskalaan otomatis tabel DynamoDB jika musim Anda kurang terasa.

> **Note**
>
> Saat membuat konfigurasi penskalaan otomatis DynamoDB untuk tabel dasar Anda, jangan lupa menyertakan konfigurasi lain untuk GSI yang terkait dengan tabel tersebut.

Cara mengidentifikasi tabel DynamoDB yang disediakan secara berlebihan

Hasil kueri yang diperoleh dari skrip di atas memberikan titik data yang diperlukan untuk melakukan beberapa analisis awal. Jika set data Anda menyajikan nilai penggunaan yang lebih rendah dari 20% untuk beberapa interval, tabel Anda mungkin disediakan secara berlebihan. Untuk menentukan lebih lanjut apakah Anda perlu mengurangi jumlah WCUs dan RCUS, Anda harus meninjau kembali bacaan lain dalam interval.

Ketika tabel Anda berisi beberapa interval penggunaan rendah, Anda benar-benar dapat mengambil manfaat dari menggunakan kebijakan penskalaan otomatis, baik dengan menjadwalkan penskalaan otomatis atau hanya mengonfigurasi kebijakan penskalaan otomatis default untuk tabel yang didasarkan pada pemanfaatan.

Jika Anda memiliki beban kerja dengan pemanfaatan rendah terhadap rasio throttle tinggi (Max (ThrottleEvents) /Min () dalam intervalThrottleEvents), ini bisa terjadi ketika Anda memiliki beban kerja yang sangat runcing di mana lalu lintas meningkat banyak selama beberapa hari (atau jam), tetapi secara umum lalu lintas secara konsisten rendah. Dalam skenario ini, mungkin bermanfaat untuk menggunakan penskalaan otomatis terjadwal.

AWS Well-Architected Framework membantu arsitek cloud membangun infrastruktur yang aman, berkinerja tinggi, tangguh, dan efisien untuk berbagai aplikasi dan beban kerja. Dibangun di sekitar enam pilar—keunggulan operasional, keamanan, keandalan, efisiensi kinerja, optimalisasi biaya, dan keberlanjutan—AWS Well-Architected memberikan pendekatan yang konsisten bagi pelanggan dan mitra untuk mengevaluasi arsitektur dan menerapkan desain yang dapat diskalakan.

AWS Well-Architected Lenses memperluas panduan yang ditawarkan oleh AWS Well-Architected untuk domain industri dan teknologi tertentu. Amazon DynamoDB Well-Architected Lens berfokus pada beban kerja DynamoDB. Ini memberikan praktik terbaik, prinsip desain, dan pertanyaan untuk menilai dan meninjau beban kerja DynamoDB. Menyelesaikan tinjauan Amazon DynamoDB Well-Architected Lens akan memberi Anda edukasi dan panduan seputar prinsip-prinsip desain yang direkomendasikan yang berkaitan dengan masing-masing pilar AWS Well-Architected. Panduan ini didasarkan pada pengalaman kami bekerja dengan pelanggan di berbagai industri, segmen, ukuran, dan geografi.

Sebagai hasil langsung dari tinjauan Well-Architected Lens, Anda akan menerima ringkasan rekomendasi yang dapat ditindaklanjuti untuk mengoptimalkan dan meningkatkan beban kerja DynamoDB Anda.

Melakukan tinjauan Amazon DynamoDB Well-Architected Lens

Ulasan DynamoDB Well-Architected Lens biasanya dilakukan AWS oleh Arsitek Solusi bersama dengan pelanggan, tetapi juga dapat dilakukan oleh pelanggan sebagai layanan mandiri. Meskipun kami merekomendasikan untuk meninjau keenam Pilar yang Dirancang dengan Baik sebagai bagian dari Amazon DynamoDB Well-Architected Lens, Anda juga dapat memutuskan untuk memprioritaskan fokus Anda pada satu atau beberapa pilar terlebih dahulu.

Informasi dan instruksi tambahan untuk melakukan tinjauan Amazon DynamoDB Well-Architected Lens tersedia di video ini dan halaman DynamoDB Well-Architected Lens. GitHub

Pilar Amazon DynamoDB Well-Architected Lens

Amazon DynamoDB Well-Architected Lens dibangun dengan enam pilar:

Pilar efisiensi performa

Pilar efisiensi performa mencakup kemampuan untuk menggunakan sumber daya komputasi secara efisien untuk memenuhi persyaratan sistem, dan untuk mempertahankan efisiensi tersebut seiring perubahan permintaan dan perkembangan teknologi.

Prinsip desain DynamoDB yang utama untuk pilar ini berkisar pada pemodelan data, pemilihan kunci partisi dan kunci urutan, serta penentuan indeks sekunder berdasarkan pola akses aplikasi. Pertimbangan tambahan termasuk memilih mode throughput optimal untuk beban kerja, penyetelan AWS SDK dan, bila sesuai, menggunakan strategi caching yang optimal. Untuk mempelajari selengkapnya tentang prinsip-prinsip desain ini, tonton video mendalam tentang pilar efisiensi performa DynamoDB Well-Architected Lens.

Pilar optimasi biaya

Pilar optimasi biaya berfokus menghindari biaya yang tidak perlu.

Topik utama yang dibahas meliputi pemahaman dan pengendalian pembelanjaan uang, pemilihan jenis sumber daya yang paling sesuai dan tepat, analisis pengeluaran dari waktu ke waktu, merancang model data untuk mengoptimalkan biaya atas pola akses khusus aplikasi, dan melakukan penskalaan untuk memenuhi kebutuhan bisnis tanpa pengeluaran berlebihan.

Prinsip desain pengoptimalan biaya utama untuk DynamoDB berkisar pada pemilihan mode kapasitas dan kelas tabel yang paling sesuai untuk tabel dan menghindari penyediaan kapasitas yang berlebihan menggunakan mode kapasitas sesuai permintaan, atau mode kapasitas yang disediakan dengan penskalaan otomatis. Pertimbangan tambahan termasuk pemodelan data yang efisien dan kueri untuk mengurangi jumlah kapasitas yang dikonsumsi, memesan porsi kapasitas yang dikonsumsi dengan harga diskon, meminimalkan ukuran barang, mengidentifikasi dan menghapus sumber daya yang tidak digunakan dan menggunakan TTL untuk secara otomatis menghapus data yang sudah tua tanpa biaya. Untuk mempelajari selengkapnya tentang prinsip-prinsip desain ini, tonton video mendalam tentang pilar pengoptimalan biaya DynamoDB Well-Architected Lens.

Lihat Optimasi biaya untuk informasi tambahan tentang praktik terbaik pengoptimalan biaya untuk DynamoDB.

Pilar keunggulan operasional

Pilar keunggulan operasional berfokus menjalankan dan memantau sistem guna memberikan nilai bisnis, dan terus meningkatkan proses dan prosedur. Topik utamanya meliputi mengotomatisasi perubahan, menanggapi peristiwa, dan mendefinisikan standar untuk mengelola operasi harian.

Prinsip desain keunggulan operasional utama untuk DynamoDB mencakup pemantauan metrik DynamoDB melalui CloudWatch AWS Config Amazon dan dan secara otomatis memperingatkan dan memulihkan ketika ambang batas yang telah ditentukan dilanggar, atau aturan yang tidak sesuai terdeteksi. Pertimbangan tambahan adalah mendefinisikan sumber daya DynamoDB melalui infrastruktur sebagai kode dan memanfaatkan tanda untuk pengorganisasian, identifikasi, dan penghitungan biaya sumber daya DynamoDB yang lebih baik. Untuk mempelajari selengkapnya tentang prinsip-prinsip desain ini, tonton video mendalam tentang pilar keunggulan operasional DynamoDB Well-Architected Lens.

Pilar keandalan

Pilar keandalan berfokus memastikan beban kerja menjalankan fungsinya dengan benar dan konsisten sesuai harapan. Beban kerja yang tangguh dapat pulih dengan cepat dari kegagalan memenuhi permintaan bisnis dan pelanggan. Topik utama yang dibahas meliputi desain sistem terdistribusi, perencanaan pemulihan, dan cara menangani perubahan.

Prinsip desain keandalan penting untuk DynamoDB berkisar pada pemilihan strategi pencadangan dan retensi berdasarkan persyaratan RPO dan RTO Anda, menggunakan tabel global DynamoDB untuk beban kerja multi-regional, atau skenario pemulihan bencana lintas wilayah dengan RTO

rendah, menerapkan logika coba lagi dengan dukungan eksponensial dalam aplikasi dengan mengonfigurasi dan menggunakan kemampuan ini di SDK, dan memantau metrik DynamoDB melalui Amazon dan secara otomatis memperingatkan dan memulihkan ketika ambang batas yang telah ditentukan dilanggar. AWS CloudWatch Untuk mempelajari selengkapnya tentang prinsip-prinsip desain ini, tonton video mendalam tentang pilar keandalan DynamoDB Well-Architected Lens.

Pilar keamanan

Pilar keamanan berfokus pada perlindungan informasi dan sistem. Topik utama yang dibahas meliputi kerahasiaan dan integritas data, mengidentifikasi dan mengelola “siapa yang dapat melakukan apa” dengan manajemen hak istimewa, melindungi sistem, dan menetapkan kontrol untuk mendeteksi peristiwa keamanan.

Prinsip desain keamanan utama untuk DynamoDB adalah mengenkripsi data bergerak dengan HTTPS, memilih jenis kunci untuk enkripsi data diam, serta menentukan kebijakan dan peran IAM untuk mengautentikasi, mengotorisasi, dan memberikan akses mendetail ke sumber daya DynamoDB. Pertimbangan tambahan termasuk mengaudit bidang kontrol DynamoDB dan operasi pesawat data melalui. AWS CloudTrail Untuk mempelajari selengkapnya tentang prinsip-prinsip desain ini, tonton video mendalam tentang pilar keamanan DynamoDB Well-Architected Lens.

Lihat Keamanan untuk informasi tambahan tentang keamanan untuk DynamoDB.

Pilar keberlanjutan

Pilar keberlanjutan berfokus meminimalkan dampak lingkungan dari menjalankan beban kerja cloud. Topik utama yang dibahas mencakup model tanggung jawab bersama untuk keberlanjutan, pemahaman terhadap dampak, dan memaksimalkan penggunaan untuk meminimalkan sumber daya yang dibutuhkan dan mengurangi dampak hilir.

Prinsip desain keberlanjutan utama untuk DynamoDB mencakup mengidentifikasi dan menghapus sumber daya DynamoDB yang tidak terpakai, menghindari penyediaan berlebihan melalui penggunaan mode kapasitas sesuai permintaan atau mode kapasitas yang disediakan dengan penskalaan otomatis, kueri yang efisien untuk mengurangi jumlah kapasitas yang digunakan dan pengurangan jejak penyimpanan dengan mengompresi data dan menghapus data lama melalui penggunaan TTL. Untuk mempelajari selengkapnya tentang prinsip-prinsip desain ini, tonton video mendalam tentang pilar keberlanjutan DynamoDB Well-Architected Lens.

Praktik terbaik untuk merancang dan menggunakan kunci partisi secara efektif di DynamoDB

Kunci primer yang secara unik mengidentifikasi setiap item dalam tabel Amazon DynamoDB bisa sederhana (hanya kunci partisi) atau komposit (kunci partisi dikombinasikan dengan kunci urutan).

Anda harus mendesain aplikasi Anda untuk aktivitas seragam di semua kunci partisi dalam tabel dan indeks sekundernya. Anda dapat menentukan pola akses yang diperlukan aplikasi Anda, serta unit baca dan tulis yang diperlukan setiap tabel dan indeks sekunder.

> **Note**
>
> Kapasitas adaptif berlaku untuk mode sesuai permintaan dan kapasitas yang disediakan.

Secara default, setiap partisi dalam tabel DynamoDB dirancang untuk memberikan kapasitas maksimum 3.000 unit baca per detik dan 1.000 unit tulis per detik. Satu unit baca mewakili satu operasi baca yang sangat konsisten per detik, atau dua operasi baca yang akhirnya konsisten per detik, untuk item berukuran hingga 4 KB. Satu unit tulis mewakili satu operasi tulis per detik untuk item berukuran hingga 1 KB.

Anda harus memperhitungkan ukuran item saat mengevaluasi batas throughput partisi untuk tabel Anda. Misalnya, jika tabel memiliki ukuran item 20 KB, satu operasi baca yang konsisten akan mengkonsumsi 5 unit baca. Ini berarti Anda dapat secara bersamaan mendorong 600 operasi baca yang konsisten per detik pada satu item tersebut sebelum mencapai batas partisi. Total throughput di semua partisi dalam tabel dapat dibatasi oleh throughput yang disediakan dalam mode yang disediakan, atau oleh batas throughput tingkat tabel dalam mode sesuai permintaan. Lihat Kuota Layanan untuk informasi selengkapnya.

Topik

- Merancang kunci partisi untuk mendistribusikan beban kerja Anda di DynamoDB
- Menggunakan sharding tulis untuk mendistribusikan beban kerja secara merata di tabel DynamoDB Anda
- Mendistribusikan aktivitas menulis secara efisien selama pengunggahan data di DynamoDB

Merancang kunci partisi untuk mendistribusikan beban kerja Anda di DynamoDB

Bagian kunci partisi pada kunci primer tabel menentukan partisi logis tempat data tabel disimpan. Pada gilirannya, hal ini akan memengaruhi partisi fisik yang mendasarinya. Desain kunci partisi yang tidak mendistribusikan I/O permintaan secara efektif dapat membuat partisi “panas” yang mengakibatkan pelambatan dan menggunakan kapasitas yang disediakan I/O secara tidak efisien.

Penggunaan optimal throughput yang disediakan pada tabel tidak hanya tergantung pada pola beban kerja masing-masing item, tetapi juga pada desain kunci partisi. Ini tidak berarti bahwa Anda harus mengakses semua nilai kunci partisi untuk mencapai tingkat throughput yang efisien, atau bahkan persentase nilai kunci partisi yang diakses harus tinggi. Artinya, semakin banyak nilai kunci partisi berbeda yang diakses oleh beban kerja Anda, semakin banyak permintaan tersebut akan tersebar di seluruh ruang yang dipartisi. Secara umum, Anda akan menggunakan throughput yang disediakan lebih efisien karena rasio nilai kunci partisi yang diakses ke jumlah total nilai kunci partisi meningkat.

Berikut ini adalah perbandingan efisiensi throughput yang disediakan dari beberapa skema kunci partisi umum.

Nilai kunci partisi	Keseragaman
User ID, aplikasi memiliki banyak pengguna.	Baik
Kode status, hanya ada beberapa kemungkin an kode status.	Buruk
Tanggal pembuatan item, dibulatkan ke periode waktu terdekat (misalnya, hari, jam, atau menit).	Buruk
ID Perangkat, setiap perangkat mengakses data pada interval yang relatif sama.	Baik
ID Perangkat, meskipun ada banyak perangkat yang dilacak, salah satunya jauh lebih populer daripada yang lainnya.	Buruk

Jika tabel tunggal hanya memiliki sejumlah kecil nilai kunci partisi, pertimbangkan untuk mendistribusikan operasi tulis Anda di nilai kunci partisi yang lebih berbeda. Dengan kata lain, susun elemen kunci primer untuk menghindari satu nilai kunci partisi "panas" (banyak diminta) yang memperlambat performa secara keseluruhan.

Misalnya, pertimbangkan tabel dengan kunci primer komposit. Kunci partisi mewakili tanggal pembuatan item, dibulatkan ke hari terdekat. Kunci urutan adalah pengidentifikasi item. Pada hari tertentu, katakanlah `2014-07-09`, semua item baru ditulis ke nilai kunci partisi tunggal (dan partisi fisik yang sesuai).

Jika tabel seluruhnya masuk ke dalam satu partisi (dengan mempertimbangkan pertumbuhan data Anda dari waktu ke waktu), dan jika persyaratan throughput baca dan tulis aplikasi Anda tidak melebihi kemampuan baca dan tulis dari satu partisi, aplikasi Anda tidak akan mengalami throttling yang tidak terduga sebagai akibat dari pembuatan partisi.

Untuk menggunakan NoSQL Workbench untuk DynamoDB guna membantu memvisualisasikan desain kunci partisi Anda, lihat Membangun Model Data dengan NoSQL Workbench.

Menggunakan sharding tulis untuk mendistribusikan beban kerja secara merata di tabel DynamoDB Anda

Salah satu cara untuk mendistribusikan penulisan dengan lebih baik di seluruh ruang kunci partisi di Amazon DynamoDB adalah dengan memperluas ruang tersebut. Hal ini dapat dilakukan dengan berbagai cara. Anda dapat menambahkan angka acak pada nilai kunci partisi untuk mendistribusikan item di antara partisi. Atau, Anda dapat menggunakan angka yang dihitung berdasarkan sesuatu yang Anda kueri.

Pembagian menggunakan akhiran acak

Salah satu strategi untuk mendistribusikan beban lebih merata di seluruh ruang kunci partisi adalah dengan menambahkan angka acak ke akhir nilai kunci partisi. Kemudian Anda mengacak penulisan di ruang yang lebih besar.

Misalnya, untuk kunci partisi yang menunjukkan tanggal hari ini, Anda dapat memilih nomor acak antara `1` dan `200` dan menggabungkannya sebagai akhiran untuk tanggal. Ini menghasilkan nilai kunci partisi seperti `2014-07-09.1`, `2014-07-09.2`, dan sebagainya, melalui `2014-07-09.200`. Karena Anda mengacak kunci partisi, penulisan ke tabel pada setiap hari tersebar merata di sejumlah partisi. Hal ini menghasilkan paralelisme yang lebih baik dan throughput keseluruhan yang lebih tinggi.

Namun, untuk membaca semua item pada hari tertentu, Anda harus mengkueri item untuk semua akhiran lalu menggabungkan hasilnya. Misalnya, Anda akan terlebih dahulu mengeluarkan permintaan `Query` untuk nilai kunci partisi `2014-07-09.1`. Kemudian mengeluarkan `Query` lainnya untuk `2014-07-09.2`, dan seterusnya, melalui `2014-07-09.200`. Terakhir, aplikasi Anda harus menggabungkan hasil dari semua permintaan `Query` tersebut.

Pembagian menggunakan akhiran terhitung

Strategi pengacakan dapat meningkatkan throughput tulis secara signifikan. Namun, sulit untuk membaca item tertentu karena Anda tidak mengetahui nilai akhiran mana yang digunakan saat menulis item tersebut. Untuk mempermudah membaca item satu per satu, Anda dapat menggunakan strategi yang berbeda. Alih-alih menggunakan angka acak untuk mendistribusikan item di antara partisi, gunakan angka yang dapat Anda hitung berdasarkan sesuatu yang ingin Anda kueri.

Pertimbangkan contoh sebelumnya, yaitu tabel menggunakan tanggal hari ini dalam kunci partisi. Sekarang anggaplah setiap item memiliki atribut `OrderId` yang dapat diakses, dan Anda sering kali perlu mencari item berdasarkan ID urutan selain tanggal. Sebelum aplikasi Anda menulis item ke tabel, aplikasi Anda dapat menghitung akhiran hash berdasarkan ID urutan dan menambahkannya ke tanggal kunci partisi. Penghitungannya dapat menghasilkan angka antara 1 dan 200 yang terdistribusi cukup merata, mirip dengan yang dihasilkan strategi acak.

Perhitungan sederhana mungkin sudah cukup, seperti produk nilai titik kode UTF-8 untuk karakter dalam ID urutan, modulo 200, + 1. Nilai kunci partisi kemudian akan menjadi tanggal yang digabungkan dengan hasil penghitungan.

Dengan strategi ini, penulisan tersebar merata di seluruh nilai kunci partisi, serta di partisi fisik. Anda dapat dengan mudah melakukan operasi `GetItem` untuk item dan tanggal tertentu karena Anda dapat menghitung nilai kunci partisi untuk nilai `OrderId` tertentu.

Untuk membaca semua item pada hari tertentu, Anda masih harus `Query`setiap kunci `2014-07-09.N` (dengan `N` adalah 1–200), dan aplikasi Anda kemudian harus menggabungkan semua hasilnya. Manfaatnya adalah Anda menghindari satu nilai kunci partisi "panas" yang mengambil semua beban kerja.

> **Note**
>
> Untuk strategi yang lebih efisien yang dirancang khusus untuk menangani data deret waktu volume tinggi, lihat Data deret waktu.

Mendistribusikan aktivitas menulis secara efisien selama pengunggahan data di DynamoDB

Biasanya, ketika Anda memuat data dari sumber data lain, Amazon DynamoDB mempartisi data tabel Anda pada beberapa server. Anda akan mendapatkan performa yang lebih baik jika mengunggah data ke semua server yang dialokasikan secara bersamaan.

Contoh, misalkan Anda ingin mengunggah pesan pengguna ke tabel DynamoDB yang menggunakan kunci primer komposit dengan `UserID` sebagai kunci partisi dan `MessageID` sebagai kunci urutan.

Saat mengunggah data, salah satu pendekatan yang dapat Anda lakukan adalah mengunggah semua item pesan untuk setiap pengguna, satu per satu:

UserID	MessageID
U1	1
U1	2
U1	...
U1	... hingga 100
U2	1
U2	2
U2	...
U2	... hingga 200

Masalahnya dalam kasus ini adalah Anda tidak mendistribusikan permintaan tulis ke DynamoDB di seluruh nilai kunci partisi Anda. Anda mengambil satu nilai kunci partisi pada satu waktu dan mengunggah semua itemnya sebelum melanjutkan ke nilai kunci partisi berikutnya dan melakukan hal yang sama.

Di balik layar, DynamoDB mempartisi data di tabel Anda di beberapa server. Untuk sepenuhnya menggunakan semua kapasitas throughput yang disediakan untuk tabel, Anda harus mendistribusikan beban kerja ke nilai kunci partisi Anda. Dengan mengarahkan pekerjaan unggahan

dalam jumlah yang tidak merata ke item yang semuanya memiliki nilai kunci partisi yang sama, Anda tidak sepenuhnya menggunakan semua sumber daya yang telah disediakan DynamoDB untuk tabel Anda.

Anda dapat mendistribusikan pekerjaan unggahan Anda menggunakan kunci urutan untuk memuat satu item dari setiap nilai kunci partisi, kemudian item lain dari setiap nilai kunci partisi, dan seterusnya:

UserID	MessageID
U1	1
U2	1
U3	1
...	...
U1	2
U2	2
U3	2
...	...

Setiap unggahan dalam urutan ini menggunakan nilai kunci partisi yang berbeda, sehingga lebih banyak server DynamoDB sibuk secara bersamaan dan meningkatkan performa throughput Anda.

Praktik terbaik untuk menggunakan kunci sortir untuk mengatur data di DynamoDB

Dalam tabel Amazon DynamoDB, kunci utama yang secara unik mengidentifikasi setiap item dalam tabel dapat terdiri dari kunci partisi dan kunci sortir.

Kunci urutan yang dirancang dengan baik memiliki dua manfaat utama:

- Kunci-kunci tersebut mengumpulkan informasi terkait di satu tempat sehingga dapat dikueri secara efisien. Desain kunci urutan yang cermat memungkinkan Anda mengambil grup item terkait

yang biasanya dibutuhkan menggunakan kueri rentang dengan operator seperti `begins_with`, `between`, >, <, dan sebagainya.

- Kunci sortir komposit memungkinkan Anda menentukan hubungan hierarkis (one-to-many) dalam data yang dapat Anda kueri di setiap tingkat hierarki.

  Misalnya, dalam tabel yang mencantumkan lokasi geografis, Anda dapat menyusun kunci urutan sebagai berikut.

  ```
  [country]#[region]#[state]#[county]#[city]#[neighborhood]
  ```

  Ini akan memungkinkan Anda membuat kueri rentang yang efisien untuk daftar lokasi di salah satu tingkat agregasi ini, dari `country`, hingga `neighborhood`, dan segala sesuatu di antaranya.

Menggunakan kunci urutan untuk kontrol version

Banyak aplikasi yang perlu menyimpan riwayat revisi tingkat item untuk tujuan audit atau kepatuhan dan agar dapat mengambil versi terbaru dengan mudah. Ada pola desain efektif yang dapat mencapai tujuan ini menggunakan prefiks kunci urutan:

- Untuk setiap item baru, buat dua salinan item: Satu berisi prefiks nomor versi nol (seperti `v0_`) di awal kunci urutan, dan satu lagi berisi prefiks nomor versi satu (seperti `v1_`).
- Setiap kali item diperbarui, gunakan prefiks versi berikutnya yang lebih tinggi di kunci urutan versi yang diperbarui, dan salin konten yang diperbarui ke item dengan prefiks versi nol. Artinya, versi terbaru item apa pun dapat ditemukan dengan mudah menggunakan prefiks nol.

Misalnya, produsen suku cadang dapat menggunakan skema seperti yang diilustrasikan di bawah ini.

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.

Item `Equipment_1` melewati serangkaian audit oleh berbagai auditor. Hasil dari setiap audit baru dicatat dalam item baru di tabel, dimulai dengan versi nomor satu, lalu menambahkan nomor untuk setiap revisi berturut-turut.

Ketika setiap revisi baru ditambahkan, lapisan aplikasi mengganti konten item versi nol (memiliki kunci urutan yang sama dengan `v0_Audit`) dengan konten revisi yang baru.

Setiap kali aplikasi perlu mengambil status audit terbaru, aplikasi dapat mengkueri prefiks kunci urutan `v0_`.

Jika aplikasi perlu mengambil seluruh riwayat revisi, aplikasi dapat mengkueri semua item pada kunci partisi item dan memfilter item `v0_`.

Desain ini juga berfungsi untuk audit di beberapa bagian peralatan, jika Anda menyertakan masing-masing bagian- IDs dalam kunci sortir setelah awalan kunci sortir.

Praktik terbaik untuk menggunakan indeks sekunder di DynamoDB

Indeks sekunder sering kali penting untuk mendukung pola kueri yang diperlukan aplikasi Anda. Pada saat yang sama, menggunakan indeks sekunder secara berlebihan atau secara tidak efisien dapat menambah biaya dan menurunkan performa yang tidak perlu.

Daftar Isi

- Pedoman umum tentang indeks sekunder di DynamoDB
  - Gunakan indeks secara efisien
  - Pilih proyeksi dengan hati-hati
  - Mengoptimalkan kueri yang sering dilakukan untuk menghindari pengambilan
  - Memahami batas ukuran kumpulan item ketika membuat indeks sekunder lokal
- Memanfaatkan indeks jarang
  - Contoh indeks jarang di DynamoDB
- Menggunakan Indeks Sekunder Global untuk kueri agregasi terwujud di DynamoDB
- Membebani Indeks Sekunder Global di DynamoDB
- Menggunakan sharding tulis Indeks Sekunder Global untuk kueri tabel selektif di DynamoDB
  - Desain pola
  - Strategi sharding

- Meminta GSI yang dipecah
- Pertimbangan eksekusi kueri paralel
- Contoh kode
- Menggunakan Indeks Sekunder Global untuk membuat replika yang pada akhirnya konsisten di DynamoDB

Pedoman umum tentang indeks sekunder di DynamoDB

Amazon DynamoDB mendukung dua jenis indeks sekunder:

- Indeks sekunder global (GSI)— Indeks dengan kunci partisi dan kunci urutan yang mungkin berbeda dari yang ada di tabel dasar. Indeks sekunder global dianggap "global" karena kueri pada indeks dapat menjangkau semua data di tabel dasar, di semua partisi. Indeks sekunder global tidak memiliki batasan ukuran dan memiliki pengaturan throughput tersendiri untuk aktivitas baca dan tulis yang terpisah dari yang ada di tabel.
- Indeks sekunder lokal (LSI)—Indeks yang memiliki kunci partisi yang sama dengan tabel dasar, tetapi kunci urutan yang berbeda. Indeks sekunder lokal bersifat "lokal" dalam arti bahwa setiap partisi indeks sekunder lokal dicakup ke partisi tabel dasar yang memiliki nilai kunci partisi yang sama. Dengan demikian, ukuran total item yang diindeks untuk setiap satu nilai kunci partisi tidak dapat melampaui 10 GB. Selain itu, indeks sekunder lokal memiliki pengaturan throughput yang disediakan yang sama untuk aktivitas baca dan tulis dengan tabel yang diindeks.

Setiap tabel di DynamoDB dapat memiliki hingga 20 indeks sekunder global (kuota default) dan 5 indeks sekunder lokal.

Indeks sekunder global sering kali lebih berguna daripada indeks sekunder lokal. Menentukan jenis indeks yang akan digunakan juga akan bergantung pada kebutuhan aplikasi Anda. Untuk perbandingan indeks sekunder global dan indeks sekunder lokal, dan informasi lebih lanjut tentang cara memilih di antara mereka, lihat. the section called “Bekerja dengan indeks”

Berikut ini adalah beberapa prinsip umum dan pola desain yang perlu diingat saat membuat indeks di DynamoDB:

Topik

- Gunakan indeks secara efisien
- Pilih proyeksi dengan hati-hati

- Mengoptimalkan kueri yang sering dilakukan untuk menghindari pengambilan
- Memahami batas ukuran kumpulan item ketika membuat indeks sekunder lokal

Gunakan indeks secara efisien

Pertahankan jumlah indeks seminimal mungkin. Jangan membuat indeks sekunder pada atribut yang tidak sering Anda kueri. Indeks yang jarang digunakan berkontribusi pada peningkatan penyimpanan dan I/O biaya tanpa meningkatkan kinerja aplikasi.

Pilih proyeksi dengan hati-hati

Karena indeks sekunder menggunakan penyimpanan dan throughput yang disediakan, Anda harus menjaga ukuran indeks sekecil mungkin. Selain itu, semakin kecil indeks, semakin besar keuntungan performa dibandingkan dengan kueri untuk tabel penuh. Jika kueri Anda biasanya hanya menampilkan sebagian kecil atribut, dan ukuran total atribut tersebut jauh lebih kecil dibandingkan keseluruhan item, hanya proyeksikan atribut yang sering Anda minta.

Jika Anda menginginkan lebih banyak aktivitas tulis pada tabel daripada aktivitas baca, ikuti praktik terbaik berikut:

- Pertimbangkan untuk memproyeksikan lebih sedikit atribut untuk meminimalkan ukuran item yang ditulis ke indeks. Namun, ini hanya berlaku jika ukuran atribut yang diproyeksikan lebih besar dari satu unit kapasitas tulis (1 KB). Sebagai contoh, jika ukuran entri indeks hanya 200 byte, DynamoDB membulatkannya hingga 1 KB. Dengan kata lain, selama item indeksnya kecil, Anda dapat memproyeksikan lebih banyak atribut tanpa biaya tambahan.
- Hindari memproyeksikan atribut yang Anda tahu akan jarang diperlukan dalam kueri. Setiap kali memperbarui atribut yang diproyeksikan dalam indeks, Anda juga dikenakan biaya tambahan untuk memperbarui indeks. Anda masih dapat mengambil atribut yang tidak diproyeksikan di `Query` dengan biaya yang lebih tinggi untuk throughput yang disediakan, tetapi biaya kueri mungkin jauh lebih rendah daripada biaya memperbarui indeks berulang kali.
- Tentukan `ALL` hanya jika Anda ingin kueri mengembalikan seluruh item tabel yang diurutkan berdasarkan kunci urutan yang berbeda. Memproyeksikan semua atribut akan menghilangkan kebutuhan untuk mengambil tabel, tetapi dalam banyak kasus, akan menggandakan biaya untuk penyimpanan dan aktivitas tulis.

Seimbangkan kebutuhan untuk menjaga indeks Anda sekecil mungkin terhadap kebutuhan untuk menjaga pengambilan tetap rendah, seperti yang dijelaskan di bagian berikutnya.

Mengoptimalkan kueri yang sering dilakukan untuk menghindari pengambilan

Untuk mendapatkan kueri tercepat dengan latensi serendah mungkin, proyeksikan semua atribut yang Anda harapkan akan dihasilkan oleh kueri tersebut. Khususnya, jika Anda mengkueri indeks sekunder lokal untuk atribut yang tidak diproyeksikan, DynamoDB otomatis mengambil atribut tersebut dari tabel, yang mengharuskan pembacaan seluruh item dari tabel. Ini memperkenalkan latensi dan I/O operasi tambahan yang dapat Anda hindari.

Ingatlah bahwa kueri "sesekali" sering kali dapat berubah menjadi kueri "penting". Jika ada atribut yang tidak ingin Anda proyeksikan karena Anda hanya ingin mengkuerinya sesekali, pertimbangkan apakah keadaan mungkin berubah dan Anda pada akhirnya mungkin menyesal tidak memproyeksikan atribut tersebut.

Untuk informasi selengkapnya tentang pengambilan tabel, lihat Pertimbangan throughput yang disediakan untuk Indeks Sekunder Lokal.

Memahami batas ukuran kumpulan item ketika membuat indeks sekunder lokal

Koleksi item adalah semua item dalam tabel dan indeks sekunder lokalnya yang memiliki kunci partisi yang sama. koleksi item tidak dapat melampaui 10 GB, sehingga kehabisan ruang dapat terjadi untuk nilai kunci partisi tertentu.

Ketika Anda menambahkan atau memperbarui item tabel, DynamoDB memperbarui semua indeks sekunder lokal yang terpengaruh. Jika atribut yang diindeks ditetapkan dalam tabel, indeks sekunder lokal juga bertambah.

Saat Anda membuat indeks sekunder lokal, perhatikan jumlah data yang akan ditulis ke dalamnya, dan jumlah item data tersebut yang memiliki nilai kunci partisi yang sama. Jika Anda memperkirakan bahwa jumlah item tabel dan indeks untuk nilai kunci partisi tertentu mungkin melebihi 10 GB, pertimbangkan apakah Anda harus menghindari pembuatan indeks.

Jika tidak dapat menghindari pembuatan indeks sekunder lokal, Anda harus mengantisipasi batas ukuran koleksi item dan mengambil tindakan sebelum batas tersebut terlampaui. Sebagai praktik terbaik, Anda harus memanfaatkan `ReturnItemCollectionMetrics`parameter saat menulis item untuk memantau dan memperingatkan ukuran koleksi item yang mendekati batas ukuran 10GB. Melebihi ukuran koleksi item maksimum akan mengakibatkan upaya penulisan yang gagal. Anda dapat mengurangi masalah ukuran koleksi item dengan memantau dan memberi tahu ukuran koleksi item sebelum berdampak pada aplikasi Anda.

> **Note**
>
> Setelah dibuat, Anda tidak dapat menghapus indeks sekunder lokal.

Untuk strategi bekerja dalam batas dan pengambilan tindakan korektif, lihat Batas ukuran kumpulan item.

Memanfaatkan indeks jarang

Untuk setiap item dalam tabel, DynamoDB menulis entri indeks yang sesuai hanya jika nilai kunci urutan indeks berada dalam item tersebut. Jika kunci urutan tidak muncul di setiap item tabel, atau jika kunci partisi indeks tidak ada dalam item, indeks dianggap jarang.

Indeks renggang berguna untuk kueri pada subbagian kecil tabel. Misalnya, anggaplah Anda memiliki tabel untuk menyimpan semua pesanan pelanggan Anda, dengan atribut kunci berikut ini:

- Kunci partisi: `CustomerId`
- Kunci urutan: `OrderId`

Untuk melacak pesanan terbuka, Anda dapat memasukkan atribut bernama `isOpen` dalam item pesanan yang belum dikirimkan. Kemudian, setelah pesanan dikirimkan, Anda dapat menghapus atribut tersebut. Jika Anda kemudian membuat indeks pada `CustomerId` (kunci partisi) dan `isOpen` (kunci urutan), hanya pesanan dengan `isOpen` yang ditetapkan yang akan muncul di dalamnya. Apabila Anda memiliki ribuan pesanan dan hanya sejumlah kecil yang terbuka, mengkueri indeks pesanan terbuka tersebut akan lebih cepat dan lebih murah daripada memindai seluruh tabel.

Daripada menggunakan jenis atribut seperti `isOpen`, Anda dapat menggunakan atribut dengan nilai yang menghasilkan urutan pesanan yang berguna dalam indeks. Misalnya, Anda dapat menggunakan atribut `OrderOpenDate` yang diatur ke tanggal ketika setiap pesanan dilakukan, lalu menghapusnya setelah pesanan dipenuhi. Dengan begitu, saat Anda mengkueri indeks jarang, item dikembalikan dan diurutkan berdasarkan tanggal ketika setiap pesanan dilakukan.

Contoh indeks jarang di DynamoDB

Indeks sekunder global bersifat jarang secara default. Ketika membuat indeks sekunder global, Anda menentukan kunci partisi dan, secara opsional, kunci urutan. Hanya item di tabel dasar yang berisi atribut tersebut yang muncul dalam indeks.

Dengan merancang indeks sekunder global menjadi jarang, Anda dapat menyediakannya dengan throughput tulis yang lebih rendah dibandingkan tabel dasar, tetapi tetap mencapai performa yang sangat baik.

Misalnya, aplikasi game dapat melacak semua skor setiap pengguna, tetapi umumnya hanya perlu mengkueri beberapa skor tinggi. Desain berikut menangani skenario ini secara efisien:

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)*

Di sini, Rick telah memainkan tiga permainan dan mencapai status `Champ` di salah satunya. Padma telah memainkan empat permainan dan mencapai status `Champ` di dua di antaranya. Perhatikan bahwa atribut `Award` hanya ada pada item tempat pengguna mendapat penghargaan. Indeks sekunder global terkait akan seperti berikut ini:

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

Indeks sekunder global hanya berisi skor tinggi yang sering dikueri, yang merupakan sebagian kecil item di tabel dasar.

Menggunakan Indeks Sekunder Global untuk kueri agregasi terwujud di DynamoDB

Mempertahankan metrik kunci dan agregasi yang mendekati waktu nyata selain data yang berubah dengan cepat menjadi semakin berharga bagi bisnis untuk mengambil keputusan dengan cepat. Misalnya, pustaka musik mungkin ingin menampilkan lagu-lagu yang paling banyak diunduh mendekati waktu nyata.

Pertimbangkan tata letak tabel pustaka musik berikut:

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)*			

Tabel dalam contoh ini menyimpan lagu dengan `songID` sebagai kunci partisi. Anda dapat mengaktifkan Amazon DynamoDB Streams pada tabel ini dan melampirkan fungsi Lambda ke alirannya sehingga saat setiap lagu diunduh, entri ditambahkan ke tabel dengan `Partition-Key=SongID` dan `Sort-Key=DownloadID`. Saat dilakukan, pembaruan ini memicu fungsi Lambda di DynamoDB Streams. Fungsi Lambda dapat mengagregasi dan mengelompokkan unduhan menurut `songID` dan memperbarui item tingkat atas, `Partition-Key=songID`, dan `Sort-Key=Month`. Perlu diingat bahwa jika eksekusi Lambda gagal tepat setelah menulis nilai agregat baru, eksekusi tersebut mungkin akan dicoba ulang dan mengagregasi nilai lebih dari sekali, sehingga Anda mendapatkan nilai perkiraan.

Untuk membaca pembaruan mendekati waktu nyata, dengan latensi milidetik satu digit, gunakan indeks sekunder global dengan ketentuan kueri `Month=2018-01`, `ScanIndexForward=False`, `Limit=1`.

Optimasi kunci lain yang digunakan di sini adalah bahwa indeks sekunder global merupakan indeks jarang dan tersedia hanya pada item yang perlu dikueri untuk mengambil data secara waktu nyata. Indeks sekunder global dapat melayani alur kerja tambahan yang memerlukan informasi tentang 10 lagu teratas yang populer, atau lagu apa pun yang diunduh pada bulan itu.

Membebani Indeks Sekunder Global di DynamoDB

Meskipun Amazon DynamoDB memiliki kuota default 20 sekunder indeks global per tabel, dalam praktiknya, Anda dapat mengindeks di lebih dari 20 bidang data. Berbeda dari tabel dalam sistem manajemen basis data relasional (RDBMS), yang skemanya seragam, tabel di DynamoDB dapat menyimpan berbagai jenis item data sekaligus. Selain itu, atribut yang sama pada item yang berbeda dapat berisi jenis informasi yang berbeda sepenuhnya.

Perhatikan contoh tata letak tabel DynamoDB berikut yang menyimpan berbagai jenis data.

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.

Atribut `Data`, yang lazim untuk semua item, memiliki konten yang berbeda tergantung pada item induknya. Jika Anda membuat indeks sekunder global untuk tabel yang menggunakan kunci urutan tabel sebagai kunci partisinya dan atribut `Data` sebagai kunci urutannya, Anda dapat membuat berbagai kueri menggunakan indeks sekunder global tunggal tersebut. Kueri tersebut dapat mencakup hal-hal berikut:

- Cari karyawan berdasarkan nama di indeks sekunder global, menggunakan `Employee_Name` sebagai nilai kunci partisi dan nama karyawan (misalnya `Murphy, John`) sebagai nilai kunci urutan.
- Gunakan indeks sekunder global untuk menemukan semua karyawan yang bekerja di gudang tertentu dengan mencari ID gudang (seperti `Warehouse_01`).
- Dapatkan daftar karyawan baru-baru ini, dengan mengkueri indeks sekunder global di `HR_confidential` sebagai nilai kunci partisi dan menggunakan rentang tanggal di nilai kunci urutan.

Menggunakan sharding tulis Indeks Sekunder Global untuk kueri tabel selektif di DynamoDB

Saat Anda perlu menanyakan data terbaru dalam jendela waktu tertentu, persyaratan DynamoDB untuk menyediakan kunci partisi untuk sebagian besar operasi baca dapat menghadirkan tantangan. Untuk mengatasi skenario ini, Anda dapat menerapkan pola kueri yang efektif menggunakan kombinasi sharding tulis dan Indeks Sekunder Global (GSI).

Pendekatan ini memungkinkan Anda untuk secara efisien mengambil dan menganalisis data sensitif waktu tanpa melakukan pemindaian tabel lengkap, yang dapat memakan banyak sumber daya dan mahal. Dengan merancang struktur tabel dan pengindeksan secara strategis, Anda dapat membuat solusi fleksibel yang mendukung pengambilan data berbasis waktu sambil mempertahankan kinerja optimal.

Topik

- Desain pola
- Strategi sharding
- Meminta GSI yang dipecah
- Pertimbangan eksekusi kueri paralel
- Contoh kode

Desain pola

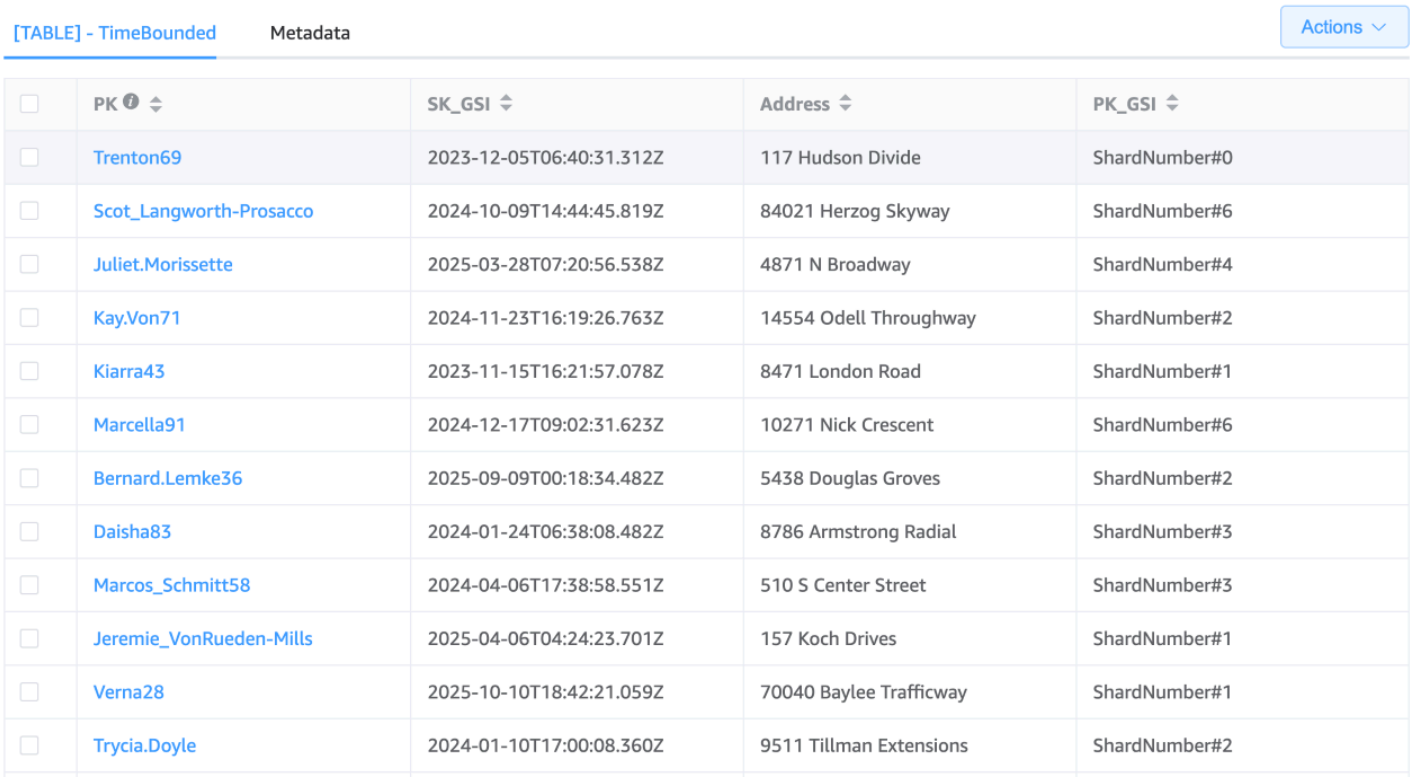
Saat bekerja dengan DynamoDB, Anda dapat mengatasi tantangan pengambilan data berbasis waktu dengan menerapkan pola canggih yang menggabungkan sharding tulis dan Indeks Sekunder Global untuk memungkinkan kueri yang fleksibel dan efisien di seluruh jendela data terbaru.

Struktur tabel

- Kunci Partisi (PK): “Nama Pengguna”

Struktur GSI

- Kunci Partisi GSI (PK_GSI): "#” ShardNumber
- Kunci Sortir GSI (SK_GSI): Stempel waktu ISO 8601 (misalnya, “2030-04-01T 12:00:00 Z”)



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

Strategi sharding

Dengan asumsi Anda memutuskan untuk menggunakan 10 pecahan, nomor pecahan Anda bisa berkisar dari 0 hingga 9. Saat mencatat aktivitas, Anda akan menghitung nomor pecahan (misalnya, dengan menggunakan fungsi hash pada ID pengguna dan kemudian mengambil modulus jumlah

pecahan) dan menambahkannya ke kunci partisi GSI. Metode ini mendistribusikan entri di berbagai pecahan, mengurangi risiko partisi panas.

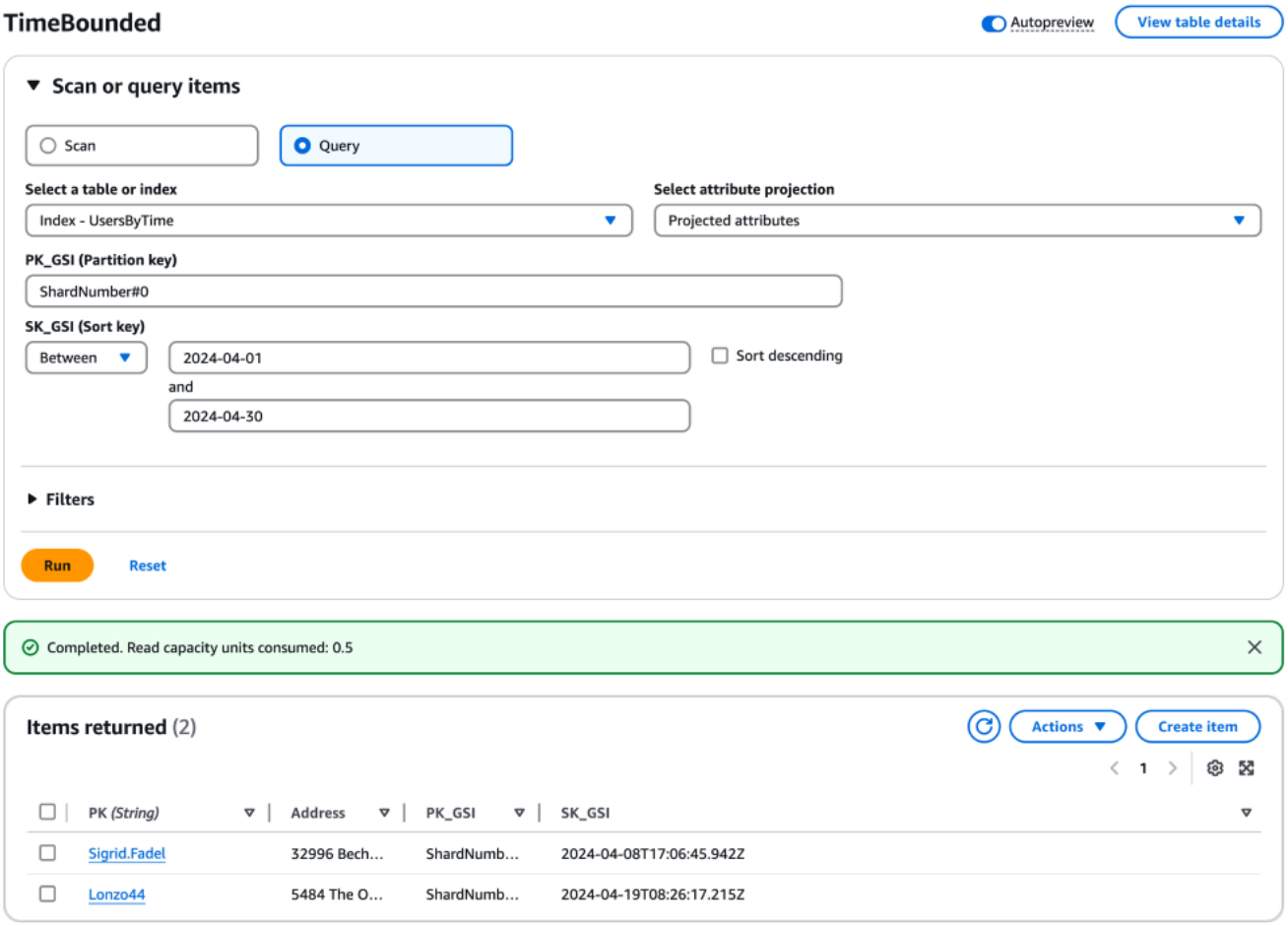
Meminta GSI yang dipecah

Query di semua pecahan untuk item dalam rentang waktu tertentu dalam tabel DynamoDB, di mana data di-sharded di beberapa kunci partisi, memerlukan pendekatan yang berbeda dari query partisi tunggal. Karena kueri DynamoDB terbatas pada satu kunci partisi pada satu waktu, Anda tidak dapat langsung melakukan kueri di beberapa pecahan dengan satu operasi kueri. Namun, Anda dapat mencapai hasil yang diinginkan melalui logika tingkat aplikasi dengan melakukan beberapa kueri, masing-masing menargetkan pecahan tertentu, dan kemudian menggabungkan hasilnya. Prosedur di bawah ini menjelaskan cara melakukan ini.

Untuk kueri dan agregat pecahan

1. Identifikasi kisaran nomor pecahan yang digunakan dalam strategi sharding Anda. Misalnya, jika Anda memiliki 10 pecahan, nomor pecahan Anda akan berkisar dari 0-9.
2. Untuk setiap pecahan, buat dan jalankan kueri untuk mengambil item dalam rentang waktu yang diinginkan. Kueri ini dapat dijalankan secara paralel untuk meningkatkan efisiensi. Gunakan tombol partisi dengan nomor pecahan dan tombol sortir dengan rentang waktu Anda untuk kueri ini. Berikut adalah contoh kueri untuk pecahan tunggal:

```
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"}
    }'
```



Anda akan mereplikasi kueri ini untuk setiap pecahan, menyesuaikan kunci partisi yang sesuai (misalnya, "ShardNumber#1 “," ShardNumber #2 “,...," ShardNumber #9 “).

3. Agregat hasil dari setiap kueri setelah semua kueri selesai. Lakukan agregasi ini dalam kode aplikasi Anda, gabungkan hasilnya menjadi satu kumpulan data yang mewakili item dari semua pecahan dalam rentang waktu yang ditentukan.

Pertimbangan eksekusi kueri paralel

Setiap kueri menggunakan kapasitas baca dari tabel atau indeks Anda. Jika Anda menggunakan throughput yang disediakan, pastikan tabel Anda disediakan dengan kapasitas yang cukup untuk menangani ledakan kueri paralel. Jika Anda menggunakan kapasitas sesuai permintaan, perhatikan implikasi biaya potensial.

Contoh kode

Untuk mengeksekusi query paralel di seluruh pecahan di DynamoDB menggunakan Python, Anda dapat menggunakan library boto3, yang merupakan Amazon Web Services SDK untuk Python. Contoh ini mengasumsikan Anda telah menginstal boto3 dan dikonfigurasi dengan kredenal yang sesuai. AWS

Kode Python berikut menunjukkan bagaimana melakukan query paralel di beberapa pecahan untuk rentang waktu tertentu. Ini menggunakan futures bersamaan untuk mengeksekusi kueri secara paralel, mengurangi waktu eksekusi keseluruhan dibandingkan dengan eksekusi sekuensial.

```
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)}")
```

Sebelum menjalankan kode ini, pastikan untuk mengganti `YourTableName` dan `YourGSIName` dengan tabel aktual dan nama GSI dari pengaturan DynamoDB Anda. Juga, sesuaikan `total_shards`time_start`,, dan `time_end` variabel sesuai dengan kebutuhan spesifik Anda.

Skrip ini menanyakan setiap pecahan untuk item dalam rentang waktu yang ditentukan dan mengumpulkan hasilnya.

Menggunakan Indeks Sekunder Global untuk membuat replika yang pada akhirnya konsisten di DynamoDB

Anda dapat menggunakan indeks sekunder global untuk membuat replika tabel yang akhirnya konsisten. Membuat replika dapat memungkinkan Anda melakukan hal berikut:

- Atur berbagai kapasitas baca yang disediakan untuk berbagai pembaca. Sebagai contoh, misalkan Anda memiliki dua aplikasi: Satu menangani kueri prioritas tinggi dan membutuhkan performa baca tingkat tertinggi, sedangkan satunya menangani kueri prioritas rendah yang dapat menoleransi throttling aktivitas baca.

  Jika kedua aplikasi ini membaca dari tabel yang sama, beban baca berat dari aplikasi prioritas rendah dapat menggunakan semua kapasitas baca yang tersedia untuk tabel. Hal ini dapat mengakibatkan throttling pada aktivitas baca aplikasi prioritas tinggi.

Sebagai gantinya, Anda dapat membuat replika melalui indeks sekunder global yang kapasitas bacanya dapat diatur secara terpisah dari tabel itu sendiri. Anda kemudian dapat menggunakan aplikasi prioritas rendah untuk mengkueri replika, bukan tabel.

- Hilangkan pembacaan dari tabel seluruhnya. Misalnya, Anda mungkin memiliki aplikasi yang menangkap aktivitas clickstream dalam jumlah besar dari situs web, dan Anda tidak ingin mengambil risiko pembacaan mengganggu aktivitas tersebut. Anda dapat mengisolasi tabel ini dan mencegah pembacaan oleh aplikasi lain (lihat Menggunakan ketentuan kebijakan IAM untuk kontrol akses terperinci), sekaligus membiarkan aplikasi lain membaca replika yang dibuat menggunakan indeks sekunder global.

Untuk membuat replika, siapkan indeks sekunder global yang memiliki skema kunci yang sama dengan tabel induk, dengan beberapa atau semua atribut non-kunci yang diproyeksikan ke dalamnya. Dalam aplikasi, Anda dapat mengarahkan beberapa atau semua aktivitas baca ke indeks sekunder global ini, bukan ke tabel induk. Anda kemudian dapat menyesuaikan kapasitas baca yang disediakan pada indeks sekunder global untuk menangani pembacaan tersebut tanpa mengubah kapasitas baca yang disediakan pada tabel induk.

Selalu ada penundaan penyebaran singkat antara penulisan ke tabel induk dan waktu ketika data yang ditulis muncul di indeks. Dengan kata lain, aplikasi Anda harus memperhitungkan bahwa replika indeks sekunder global pada akhirnya hanya konsisten dengan tabel induk.

Anda dapat membuat beberapa replika indeks sekunder global untuk mendukung pola pembacaan yang berbeda. Ketika Anda membuat replika, hanya proyeksikan atribut yang benar-benar dibutuhkan setiap pola baca. Suatu aplikasi kemudian dapat menggunakan lebih sedikit kapasitas baca yang disediakan untuk hanya memperoleh data yang diperlukan daripada harus membaca item dari tabel induk. optimasi ini dapat menghasilkan penghematan biaya yang signifikan dari waktu ke waktu.

Praktik terbaik untuk menyimpan item dan atribut besar di DynamoDB

Amazon DynamoDB membatasi ukuran setiap item yang Anda simpan dalam tabel hingga 400 KB (lihat). Kuota di Amazon DynamoDB Jika aplikasi Anda perlu menyimpan data dalam suatu item lebih dari yang diizinkan oleh batas ukuran DynamoDB, Anda dapat mencoba mengompresi satu atau lebih atribut besar atau memecah item menjadi beberapa item (diindeks secara efisien dengan kunci

pengurutan). Anda juga dapat menyimpan item sebagai objek di Amazon Simple Storage Service (Amazon S3) dan menyimpan pengidentifikasi objek Amazon S3 di item DynamoDB Anda.

Sebagai praktik terbaik, Anda harus memanfaatkan `ReturnConsumedCapacity`parameter saat menulis item untuk memantau dan memperingatkan ukuran item yang mendekati ukuran item maksimum 400 KB. Melebihi ukuran item maksimum akan mengakibatkan upaya penulisan yang gagal. DynamoDB akan mengembalikan kesalahan. ValidationException Pemantauan dan peringatan pada ukuran item akan memungkinkan Anda untuk mengurangi masalah ukuran item sebelum berdampak pada aplikasi Anda.

Mengompresi nilai atribut besar

Mengompresi nilai atribut yang besar dapat membuatnya sesuai dengan batas item di DynamoDB dan mengurangi biaya penyimpanan Anda. Algoritma kompresi seperti GZIP atau LZO menghasilkan output biner yang kemudian dapat Anda simpan dalam tipe `Binary` atribut dalam item.

Sebagai contoh, pertimbangkan tabel yang menyimpan pesan yang ditulis oleh pengguna forum. Pesan semacam itu sering mengandung string teks yang panjang, yang merupakan kandidat untuk kompresi. Meskipun kompresi dapat mengurangi ukuran item, sisi negatifnya adalah nilai atribut terkompresi tidak berguna untuk pemfilteran.

Untuk kode sampel yang menunjukkan cara mengompresi pesan tersebut di DynamoDB, lihat berikut ini:

- Contoh: Menangani atribut tipe biner menggunakan API AWS SDK untuk Java dokumen
- Contoh: Menangani atribut tipe biner menggunakan API AWS SDK untuk .NET tingkat rendah

Partisi vertikal

Solusi alternatif untuk menangani item besar adalah memecahnya menjadi potongan data yang lebih kecil dan mengaitkan semua item yang relevan dengan nilai kunci partisi. Anda kemudian dapat menggunakan string kunci sortir untuk mengidentifikasi informasi terkait yang disimpan di sampingnya. Dengan melakukan ini, dan memiliki beberapa item yang dikelompokkan berdasarkan nilai kunci partisi yang sama, Anda membuat koleksi item.

Untuk informasi lebih lanjut tentang pendekatan ini, lihat:

- Gunakan partisi vertikal untuk menskalakan data secara efisien di Amazon DynamoDB
- Menerapkan partisi vertikal di Amazon DynamoDB menggunakan AWS Glue

Menyimpan nilai atribut besar di Amazon S3

Seperti yang disebutkan sebelumnya, Anda juga dapat menggunakan Amazon S3 untuk menyimpan nilai atribut besar yang tidak dapat ditampung dalam item DynamoDB. Anda dapat menyimpannya sebagai objek di Amazon S3, lalu menyimpan pengidentifikasi objek di item DynamoDB Anda.

Anda juga dapat menggunakan dukungan metadata objek di Amazon S3 untuk menyediakan tautan kembali ke item induk di DynamoDB. Simpan nilai kunci primer item sebagai metadata Amazon S3 dari objek di Amazon S3. Tindakan ini sering kali membantu pemeliharaan objek Amazon S3.

Misalnya, perhatikan `ProductCatalog` tabelnya. Item dalam tabel ini menyimpan informasi tentang harga item, deskripsi, penulis buku, dan dimensi untuk produk lainnya. Jika Anda ingin menyimpan gambar produk yang terlalu besar untuk suatu item, Anda dapat menyimpan gambar tersebut di Amazon S3, bukan di DynamoDB.

Saat mengimplementasikan strategi ini, perhatikan hal berikut:

- DynamoDB tidak mendukung transaksi yang melintasi Amazon S3 dan DynamoDB. Oleh karena itu, aplikasi Anda harus menangani segala kegagalan, yang dapat mencakup pembersihan objek Amazon S3 yang tidak ada lagi.
- Amazon S3 membatasi panjang pengidentifikasi objek. Jadi, Anda harus mengatur data Anda sedemikian rupa sehingga tidak menghasilkan pengidentifikasi objek yang terlalu panjang atau melanggar batasan Amazon S3 lainnya.

Untuk informasi selengkapnya tentang cara menggunakan Amazon S3, lihat Panduan Pengguna Amazon Simple Storage Service.

Praktik terbaik untuk penanganan data deret waktu di DynamoDB

Prinsip desain umum di Amazon DynamoDB merekomendasikan agar Anda meminimalkan jumlah tabel yang Anda gunakan. Untuk sebagian besar aplikasi, hanya satu tabel yang Anda butuhkan. Namun, untuk data deret waktu, Anda sering kali dapat menanganinya menggunakan satu tabel per aplikasi per periode.

Pola desain untuk data deret waktu

Pertimbangkan skenario deret waktu yang umum, yaitu Anda ingin melacak peristiwa dalam jumlah besar. Pola akses tulis Anda adalah semua peristiwa yang direkam memiliki tanggal hari ini. Pola

akses baca Anda mungkin membaca peristiwa hari ini paling sering, peristiwa kemarin lebih jarang, dan peristiwa lama sangat sedikit. Salah satu cara untuk mengatasinya adalah dengan memasukkan tanggal dan waktu saat ini ke dalam kunci primer.

Pola desain berikut sering kali menangani skenario semacam ini dengan efektif:

- Buat satu tabel per periode, dilengkapi dengan kapasitas baca dan tulis yang diperlukan serta indeks yang diperlukan.
- Sebelum akhir setiap periode, buat tabel untuk periode berikutnya. Setelah periode saat ini berakhir, arahkan lalu lintas peristiwa ke tabel baru. Anda dapat memberikan nama pada tabel-tabel ini yang menentukan periode yang telah dicatatnya.
- Segera setelah tabel tidak lagi digunakan untuk menulis, kurangi kapasitas tulis yang disediakan ke nilai yang lebih rendah (misalnya, 1 WCU), dan sediakan kapasitas baca berapa pun yang sesuai. Kurangi kapasitas baca yang disediakan pada tabel sebelumnya seiring bertambahnya usia. Anda dapat mengarsipkan atau menghapus tabel yang isinya jarang atau tidak pernah diperlukan.

Tujuannya adalah untuk mengalokasikan sumber daya yang diperlukan untuk periode saat ini yang akan mengalami volume lalu lintas tertinggi dan mengurangi penyediaan tabel lama yang tidak digunakan secara aktif, sehingga menghemat biaya. Bergantung pada kebutuhan bisnis Anda, Anda dapat mempertimbangkan pembagian tulis untuk mendistribusikan lalu lintas secara merata ke kunci partisi logis. Untuk informasi selengkapnya, lihat Menggunakan sharding tulis untuk mendistribusikan beban kerja secara merata di tabel DynamoDB Anda.

Contoh tabel deret waktu

Berikut ini adalah contoh data deret waktu di mana tabel saat ini disediakan pada read/write kapasitas yang lebih tinggi dan tabel yang lebih lama diperkecil karena jarang diakses.

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	...
. . .	. . .	. . .	. . .	...

Praktik terbaik untuk mengelola many-to-many hubungan dalam tabel DynamoDB

Adjacency list adalah pola desain yang berguna untuk pemodelan many-to-many hubungan di Amazon DynamoDB. Secara lebih umum, daftar ini menyediakan cara untuk merepresentasikan data grafik (simpul dan edge) di DynamoDB.

Pola desain daftar kedekatan

Ketika entitas yang berbeda dari aplikasi memiliki many-to-many hubungan di antara mereka, hubungan dapat dimodelkan sebagai daftar kedekatan. Dalam pola ini, semua entitas tingkat atas (sama dengan simpul dalam model grafik) direpresentasikan menggunakan kunci partisi. Hubungan apa pun dengan entitas lain (edge dalam grafik) direpresentasikan sebagai item dalam partisi dengan mengatur nilai kunci urutan ke ID entitas target (simpul target).

Keunggulan pola ini antara lain duplikasi data yang minimal dan pola kueri yang disederhanakan untuk menemukan semua entitas (simpul) yang terkait dengan entitas target (memiliki edge terhadap simpul target).

Contoh aktual kegunaan pola ini adalah sistem faktur dengan faktur yang berisi beberapa tagihan. Satu tagihan bisa masuk ke beberapa faktur. Kunci partisi dalam contoh ini adalah `InvoiceID` atau `BillID`. Partisi `BillID` memiliki semua atribut yang khusus untuk tagihan. Partisi `InvoiceID` memiliki item yang menyimpan atribut khusus faktur, dan item untuk masing-masing `BillID` yang dimasukkan ke faktur.

Skemanya akan seperti berikut.

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...

Menggunakan skema sebelumnya, Anda dapat melihat bahwa semua tagihan dalam faktur dapat dikueri menggunakan kunci primer pada tabel. Untuk mencari semua faktur yang memuat bagian dari tagihan, buat indeks sekunder global pada kunci urutan tabel.

Proyeksi untuk indeks sekunder global akan seperti berikut.

GSI	Primary Key *Partition Key*	Projected Attributes...	
	Bill-4224663	**Bill_ID:** Bill-4224663 *(table primary key)*	Attributes of this bill...
		Inv_ID: Invoice-92551 *(table primary key)*	Attributes of this bill *in this invoice*..
	Bill-4224687	**Bill_ID:** Bill-4224687 *(table primary key)*	Attributes of this bill...
		Inv_ID: Invoice-92551 *(table primary key)*	Attributes of this bill *in this invoice*..
		Inv_ID: Invoice-92552 *(table primary key)*	Attributes of this bill *in this invoice*..
	Invoice-92551	**Inv_ID:** Invoice-92551 *(table primary key)*	Attributes of this invoice...
	Invoice-92552	**Inv_ID:** Invoice-92552 *(table primary key)*	Attributes of this invoice...

Pola grafik terwujud

Banyak aplikasi dibangun berdasarkan pemahaman peringkat di antara sesama aplikasi, hubungan umum di antara entitas, status entitas tetangga, dan jenis alur kerja gaya grafik lainnya. Untuk jenis aplikasi ini, pertimbangkan pola desain skema berikut.

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	

	Primary Key		Attributes		
	GSI PK	GSI 2 SK (TypeTarget)			
GSI 2	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	

Skema sebelumnya menunjukkan struktur data grafik yang didefinisikan oleh satu set partisi data yang berisi item yang menentukan edge dan simpul grafik. Item edge berisi atribut `Target` dan `Type`. Atribut ini digunakan sebagai bagian dari nama kunci komposit TypeTarget "" untuk mengidentifikasi item dalam partisi di tabel primer atau dalam indeks sekunder global kedua.

Indeks sekunder global pertama dibangun di atas atribut `Data`. Atribut ini menggunakan muatan berlebih indeks sekunder global seperti yang dijelaskan sebelumnya untuk mengindeks beberapa jenis atribut, yaitu `Dates`, `Names`, `Places`, dan `Skills`. Di sini, satu indeks sekunder global mengindeks empat atribut yang berbeda secara efektif.

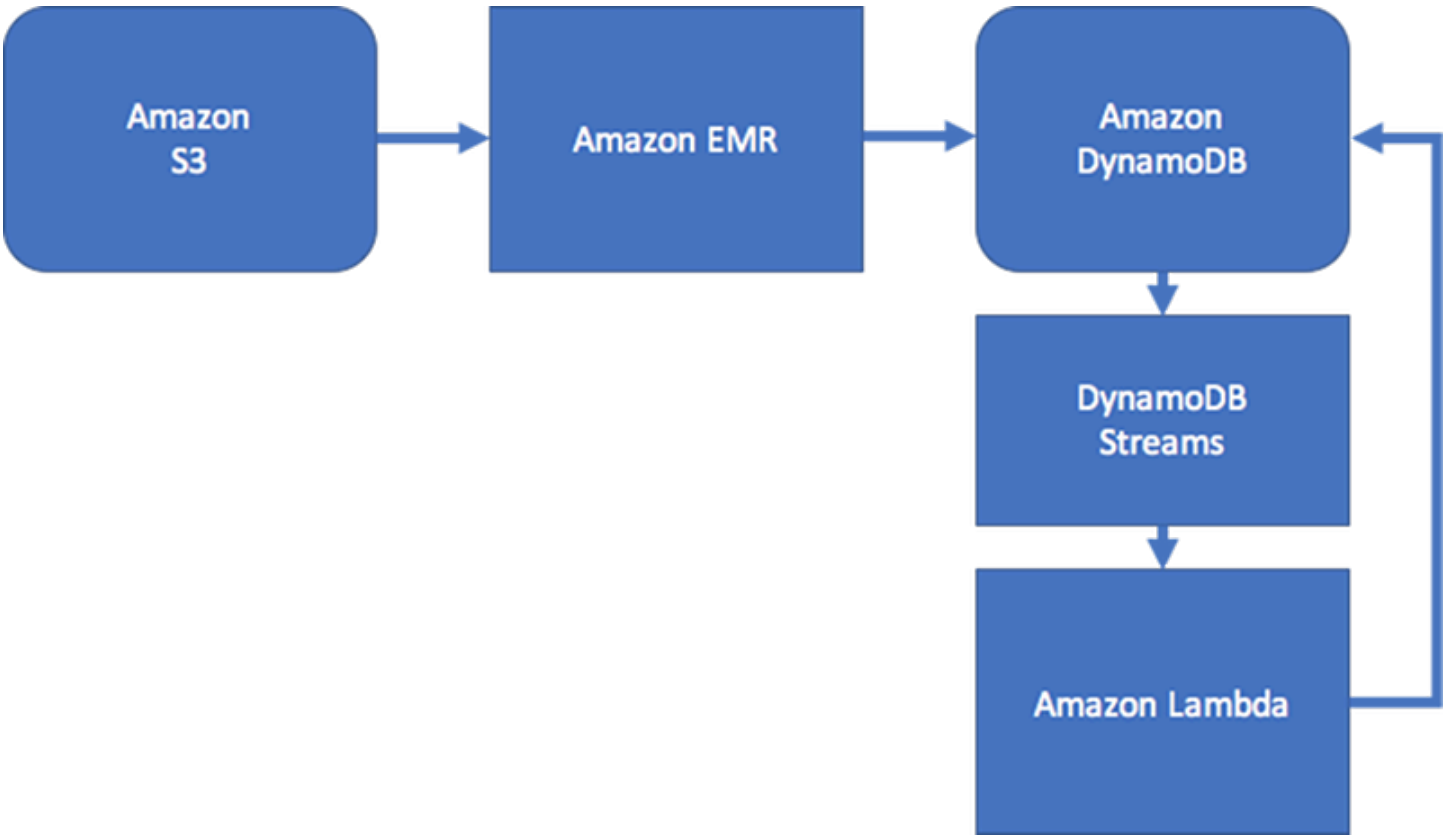
Saat memasukkan item ke dalam tabel, Anda dapat menggunakan strategi pembagian cerdas untuk mendistribusikan kumpulan item dengan agregasi besar (tanggal lahir, keterampilan) ke sebanyak mungkin partisi logis pada indeks sekunder global yang diperlukan untuk menghindari masalah baca/tulis yang panas.

Hasil dari kombinasi pola desain ini adalah penyimpanan data yang solid untuk alur kerja grafik waktu nyata yang sangat efisien. Alur kerja ini dapat memberikan status entitas tetangga performa tinggi

dan kueri agregasi edge untuk mesin rekomendasi, aplikasi jejaring sosial, peringkat simpul, agregasi subtree, dan kasus penggunaan grafik umum lainnya.

Jika kasus penggunaan Anda tidak sensitif terhadap konsistensi data waktu nyata, Anda dapat menggunakan proses Amazon EMR terjadwal untuk mengisi edge dengan agregasi ringkasan grafik yang relevan untuk alur kerja Anda. Anda dapat menggunakan proses terjadwal untuk menggabungkan hasil jika aplikasi Anda tidak perlu segera mengetahui kapan edge ditambahkan ke grafik.

Untuk mempertahankan beberapa tingkat konsistensi, desain dapat menyertakan Amazon DynamoDB Streams dan AWS Lambda untuk memproses pembaruan edge. Desain ini juga bisa menggunakan tugas Amazon EMR untuk memvalidasi hasil secara berkala. Pendekatan ini diilustrasikan dengan diagram berikut. Ini biasanya digunakan dalam aplikasi jejaring sosial, jika biaya kueri waktu nyata tinggi dan kebutuhan untuk mengetahui pembaruan pengguna individu rendah.



Manajemen layanan TI (IT service-management/ITSM) dan aplikasi keamanan umumnya perlu merespons secara waktu nyata terhadap perubahan status entitas yang terdiri dari agregasi edge yang kompleks. Aplikasi semacam itu membutuhkan sistem yang dapat mendukung agregasi beberapa simpul waktu nyata dari hubungan tingkat kedua dan ketiga, atau traversal edge yang

kompleks. Jika kasus penggunaan Anda memerlukan jenis alur kerja kueri grafik waktu nyata ini, sebaiknya pertimbangkan menggunakan Amazon Neptune untuk mengelola alur kerja ini.

> **Note**
>
> Jika Anda perlu mengkueri set data yang sangat terhubung atau mengeksekusi kueri yang perlu melintasi beberapa simpul (juga dikenal sebagai kueri multi-hop) dengan latensi milidetik, sebaiknya gunakan Amazon Neptune. Amazon Neptune adalah mesin basis data grafik berperforma tinggi yang dibuat khusus dan dioptimalkan untuk menyimpan miliaran hubungan dan mengkueri grafik dengan latensi milidetik.

Praktik terbaik untuk kueri dan pemindaian data di DynamoDB

Bagian ini membahas beberapa praktik terbaik untuk menggunakan operasi `Query` dan `Scan` di Amazon DynamoDB.

Pertimbangan performa untuk pemindaian

Secara umum, efisiensi operasi `Scan` lebih rendah daripada operasi lain di DynamoDB. Operasi `Scan` selalu memindai seluruh tabel atau indeks sekunder. Kemudian, operasi ini memfilter nilai-nilai untuk memberikan hasil yang Anda inginkan, yang pada dasarnya menambahkan langkah tambahan untuk menghapus data dari kumpulan hasil.

Jika memungkinkan, Anda harus menghindari penggunaan operasi `Scan` pada indeks atau tabel besar dengan filter yang menghapus banyak hasil. Selain itu, seiring dengan berkembangnya tabel atau indeks, operasi `Scan` melambat. Operasi `Scan` memeriksa setiap item untuk nilai yang diminta dan dapat menggunakan throughput yang disediakan untuk indeks atau tabel besar dalam satu operasi. Untuk waktu respons yang lebih cepat, rancang tabel dan indeks Anda agar aplikasi Anda dapat menggunakan `Query`, bukan `Scan`. (Untuk tabel, Anda juga dapat mempertimbangkan untuk menggunakan `GetItem` dan `BatchGetItem` APIs.)

Atau, Anda dapat merancang aplikasi Anda untuk menggunakan operasi `Scan` dengan cara yang meminimalkan dampak pada tingkat permintaan Anda. Ini dapat mencakup pemodelan jika penggunaan indeks sekunder global mungkin lebih efisien daripada operasi `Scan`. Informasi selengkapnya tentang proses ini ada di video berikut.

Memodelkan pola akses kecepatan rendah

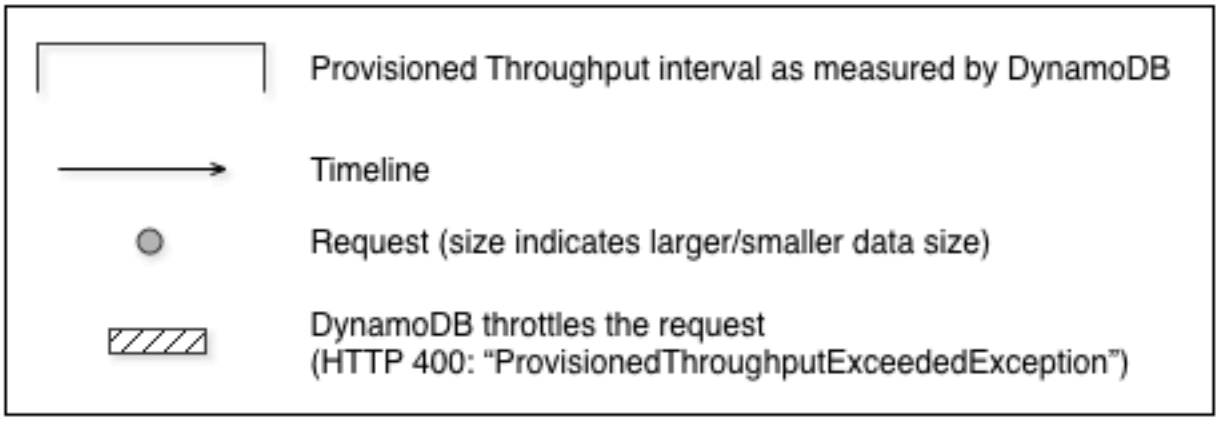
Menghindari lonjakan mendadak dalam aktivitas baca

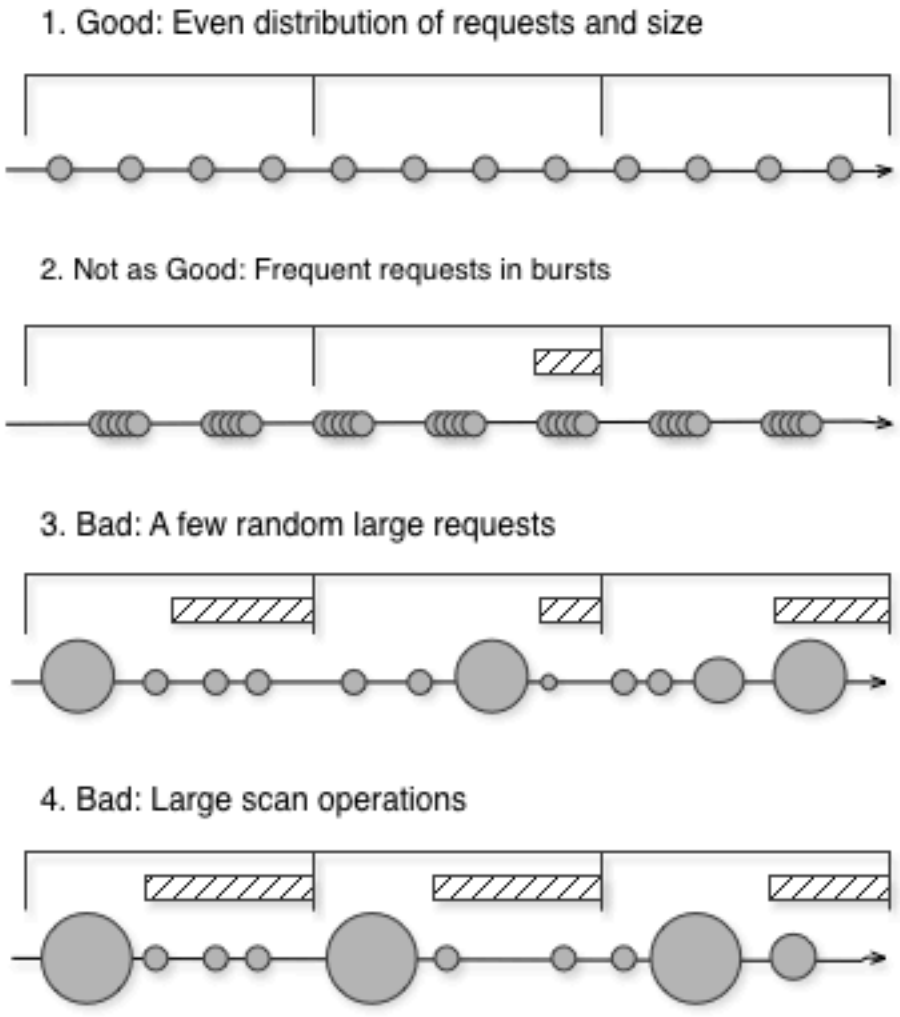
Jika Anda membuat tabel, Anda akan mengatur persyaratan unit kapasitas baca dan tulisnya. Untuk baca, unit kapasitasnya dinyatakan sebagai jumlah permintaan baca data 4 KB yang sangat konsisten per detik. Untuk bacaan akhir konsisten, unit kapasitas baca memiliki dua permintaan baca 4 KB per detik. Operasi `Scan` melakukan bacaan akhir konsisten default, dan dapat mengembalikan hingga 1 MB (satu halaman) data. Oleh karena itu, satu permintaan `Scan` dapat menggunakan (ukuran halaman 1 MB / ukuran item 4 KB) / 2 (bacaan akhir konsisten) = 128 operasi baca. Jika Anda meminta bacaan sangat konsisten, operasi `Scan` akan menggunakan throughput yang disediakan dua kali lebih banyak – 256 operasi baca.

Hal ini menunjukkan lonjakan tiba-tiba pada penggunaan, dibandingkan dengan kapasitas baca yang dikonfigurasi untuk tabel. Penggunaan unit kapasitas melalui pemindaian ini mencegah permintaan lain yang berpotensi lebih penting untuk tabel yang sama menggunakan unit kapasitas yang tersedia. Akibatnya, Anda mungkin mendapatkan pengecualian `ProvisionedThroughputExceeded` untuk permintaan tersebut.

Masalahnya bukan hanya peningkatan mendadak pada unit kapasitas yang digunakan `Scan`. Pemindaian juga kemungkinan akan menggunakan semua unit kapasitasnya dari partisi yang sama karena pemindaian meminta item baca yang berdampingan di partisi. Artinya, permintaan tersebut mengenai partisi yang sama, sehingga menyebabkan semua unit kapasitasnya terpakai, dan membatasi permintaan lain ke partisi tersebut. Jika permintaan untuk membaca data tersebar di beberapa partisi, operasi tidak akan membatasi partisi tertentu.

Diagram berikut mengilustrasikan dampak lonjakan penggunaan unit kapasitas secara mendadak oleh operasi `Query` dan `Scan`, serta dampaknya pada permintaan Anda lainnya terhadap tabel yang sama.





Seperti yang diilustrasikan di sini, lonjakan penggunaan dapat memengaruhi throughput yang disediakan pada tabel dalam beberapa cara:

1. Baik: Distribusi permintaan dan ukuran yang merata

2. Tidak cukup baik: Permintaan yang sering terjadi secara beruntun
3. Buruk: Beberapa permintaan besar acak
4. Buruk: Operasi pemindaian besar

Daripada menggunakan operasi `Scan` besar, Anda dapat menggunakan teknik berikut untuk meminimalkan dampak pemindaian pada throughput yang disediakan di tabel.

- Mengurangi ukuran halaman

  Karena operasi Pindai membaca seluruh halaman (secara default, 1 MB), Anda dapat mengurangi dampak operasi pemindaian dengan mengatur ukuran halaman yang lebih kecil. Operasi `Scan` menyediakan parameter Limit yang dapat Anda gunakan untuk mengatur ukuran halaman untuk permintaan Anda. Setiap permintaan `Query` atau `Scan` yang memiliki ukuran halaman yang lebih kecil menggunakan lebih sedikit operasi baca dan membuat "jeda" di antara setiap permintaan. Misalnya, katakanlah setiap item berukuran 4 KB dan Anda mengatur ukuran halamannya menjadi 40 item. Permintaan `Query` kemudian hanya akan menggunakan 20 operasi bacaan akhir konsisten atau 40 operasi bacaan sangat konsisten. Sejumlah besar operasi `Query` atau `Scan` yang lebih kecil akan memungkinkan permintaan penting Anda yang lain untuk berhasil tanpa mengalami throttling.

- Mengisolasi operasi pemindaian

  DynamoDB dirancang untuk skalabilitas yang mudah. Dengan demikian, aplikasi dapat membuat tabel untuk tujuan yang berbeda, bahkan mungkin menduplikasi konten di beberapa tabel. Anda ingin melakukan pemindaian pada tabel yang tidak mengambil lalu lintas "vital". Beberapa aplikasi menangani beban ini dengan memutar lalu lintas setiap jam antara dua tabel—satu untuk lalu lintas vital, dan satu lagi untuk pembukuan. Aplikasi lain dapat melakukan ini dengan melakukan setiap penulisan pada dua tabel: tabel "vital", dan tabel "bayangan".

Konfigurasikan aplikasi Anda untuk mencoba kembali permintaan apa pun yang menerima kode respons yang menunjukkan bahwa Anda telah melampaui throughput yang disediakan. Atau, tingkatkan throughput yang disediakan untuk tabel Anda menggunakan operasi `UpdateTable`.
Jika Anda mengalami lonjakan sementara dalam beban kerja yang menyebabkan throughput Anda melebihi, terkadang, melampaui tingkat yang disediakan, coba ulang permintaan dengan penundaan eksponensial. Untuk informasi selengkapnya tentang cara mengimplementasikan penundaan eksponensial, lihat Percobaan ulang kesalahan dan penundaan eksponensial.

Memanfaatkan pemindaian paralel

Banyak aplikasi mendapatkan keuntungan dari penggunaan operasi `Scan` paralel dibandingkan pemindaian berurutan. Misalnya, aplikasi yang memproses tabel data historis besar dapat melakukan pemindaian paralel jauh lebih cepat dibandingkan pemindaian berurutan. Beberapa thread pekerja dalam proses "sweeper" latar belakang dapat memindai tabel dengan prioritas rendah tanpa memengaruhi lalu lintas produksi. Dalam masing-masing contoh ini, `Scan` paralel digunakan sedemikian rupa sehingga tidak membuat aplikasi lain kekurangan sumber daya throughput yang disediakan.

Meskipun pemindaian paralel menguntungkan, pemindaian ini dapat memberikan tuntutan besar pada throughput yang disediakan. Dengan pemindaian paralel, aplikasi Anda memiliki beberapa pekerja yang semuanya menjalankan operasi `Scan` secara bersamaan. Operasi ini dapat dengan cepat menghabiskan seluruh kapasitas baca yang disediakan pada tabel Anda. Dalam hal ini, aplikasi lain yang perlu mengakses tabel mungkin akan mengalami throttling.

Pemindaian paralel dapat menjadi pilihan tepat jika kondisi berikut terpenuhi:

- Ukuran tabel sebesar 20 GB atau lebih.
- Throughput baca yang disediakan pada tabel tidak digunakan sepenuhnya.
- Operasi `Scan` yang berurutan terlalu lambat.

Memilih TotalSegments

Pengaturan terbaik untuk `TotalSegments` tergantung pada data spesifik Anda, pengaturan throughput yang disediakan pada tabel, dan kebutuhan performa Anda. Anda mungkin perlu bereksperimen untuk melakukannya dengan benar. Sebaiknya mulai dengan rasio sederhana, seperti satu segmen per 2 GB data. Misalnya, untuk tabel 30 GB, Anda dapat mengatur `TotalSegments` ke 15 (30 GB / 2 GB). Aplikasi Anda kemudian akan menggunakan 15 pekerja, masing-masing pekerja memindai segmen yang berbeda.

Anda juga dapat memilih nilai untuk `TotalSegments` yang didasarkan pada sumber daya klien. Anda dapat mengatur `TotalSegments` ke angka berapa pun dari 1 hingga 1000000, dan DynamoDB akan memungkinkan Anda memindai jumlah segmen tersebut. Misalnya, jika klien Anda membatasi jumlah thread yang dapat berjalan secara bersamaan, Anda dapat meningkatkan `TotalSegments` secara bertahap hingga mendapatkan performa `Scan` terbaik dengan aplikasi Anda.

Pantau pemindaian paralel Anda untuk mengoptimalkan penggunaan throughput yang disediakan, sekaligus memastikan bahwa aplikasi Anda yang lain tidak kekurangan sumber daya. Tingkatkan nilai `TotalSegments` jika Anda tidak menggunakan seluruh throughput yang disediakan tetapi masih mengalami throttling dalam permintaan `Scan` Anda. Kurangi nilai `TotalSegments` jika permintaan `Scan` menggunakan lebih banyak throughput yang disediakan daripada yang ingin Anda gunakan.

Praktik terbaik untuk desain tabel DynamoDB

Prinsip desain umum di Amazon DynamoDB merekomendasikan agar Anda meminimalkan jumlah tabel yang Anda gunakan. Dalam sebagian besar kasus, sebaiknya Anda menggunakan satu tabel. Namun, jika satu atau sejumlah kecil tabel tidak dapat digunakan, pedoman ini mungkin berguna.

- Batas per akun tidak dapat ditingkatkan di atas 10.000 tabel per akun. Jika aplikasi Anda memerlukan lebih banyak tabel, rencanakan untuk mendistribusikan tabel ke beberapa akun. Untuk informasi selengkapnya, lihat kuota layanan, akun, dan tabel di Amazon DynamoDB.
- Pertimbangkan batas bidang kontrol untuk operasi bidang kontrol bersamaan yang dapat memengaruhi manajemen tabel Anda.
- Bekerja dengan arsitek AWS solusi untuk memvalidasi pola desain Anda untuk desain multi-tenant.

Praktik terbaik untuk desain tabel global DynamoDB

Tabel global dibangun berdasarkan jejak global Amazon DynamoDB untuk memberi Anda basis data yang terkelola sepenuhnya, multi-Wilayah, dan multi-aktif yang memberikan performa baca dan tulis yang cepat serta lokal, untuk aplikasi global berskala besar. Dengan tabel global, data Anda akan direplikasi secara otomatis di seluruh AWS wilayah pilihan Anda. Karena tabel global menggunakan APIs DynamoDB yang ada, tidak diperlukan perubahan pada aplikasi Anda. Tidak ada biaya atau komitmen di muka untuk menggunakan tabel global, dan Anda hanya membayar untuk sumber daya yang Anda gunakan.

Topik

- Panduan preskriptif untuk desain tabel global DynamoDB
- Fakta kunci tentang desain tabel global DynamoDB
- Kasus penggunaan tabel global DynamoDB
- Tulis mode dengan tabel global DynamoDB

- Minta perutean dengan tabel global DynamoDB
- Mengevakuasi Wilayah dengan tabel global DynamoDB
- Perencanaan kapasitas throughput untuk tabel global DynamoDB
- Daftar periksa persiapan untuk tabel global DynamoDB dan Pertanyaan yang Sering Diajukan

Panduan preskriptif untuk desain tabel global DynamoDB

Penggunaan tabel global yang efisien memerlukan pertimbangan yang cermat terhadap faktor-faktor seperti mode tulis pilihan Anda, model perutean, dan proses evakuasi. Anda harus melengkapi aplikasi Anda di setiap Wilayah dan siap menyesuaikan rute atau melakukan evakuasi untuk menjaga kesehatan global. Sehingga Anda akan mendapatkan kumpulan data yang terdistribusi secara global dengan pembacaan dan penulisan latensi rendah serta perjanjian tingkat layanan 99,999%.

Fakta kunci tentang desain tabel global DynamoDB

- Ada dua versi tabel global: versi saat ini Tabel Global versi 2019.11.21 (Saat ini) (kadang-kadang disebut “V2"), dan Versi tabel global 2017.11.29 (Legacy) (kadang-kadang disebut “V1"). Panduan ini berfokus secara khusus pada versi terbaru, V2.
- Tanpa menggunakan tabel global, DynamoDB adalah layanan Regional. Ini memiliki ketersediaan tinggi dan secara intrinsik memiliki ketahanan terhadap kegagalan infrastruktur suatu Wilayah, termasuk kegagalan seluruh zona ketersediaan (AZ). Tabel DynamoDB Wilayah tunggal memiliki 99,99% ketersediaan https://aws.amazon.com/dynamodb/sla/Perjanjian Tingkat Layanan (SLA).
- Dengan penggunaan tabel global, DynamoDB memungkinkan tabel mereplikasi datanya di antara dua Wilayah atau lebih. Tabel DynamoDB multi-Wilayah memiliki 99,999% ketersediaan SLA. Dengan perencanaan yang tepat, tabel global dapat membantu menciptakan arsitektur yang tangguh dan menolak kegagalan Regional.
- Tabel global menggunakan model replikasi aktif-aktif. Dari perspektif DynamoDB, tabel di setiap Wilayah memiliki kedudukan yang sama untuk menerima permintaan baca dan tulis. Setelah menerima permintaan penulisan, tabel replika lokal akan mereplikasi penulisan tersebut ke Wilayah yang berpartisipasi lainnya di latar belakang.
- Item direplikasi satu per satu. Item yang diperbarui dalam satu transaksi mungkin tidak direplikasi bersama.
- Setiap partisi tabel di Wilayah sumber mereplikasi penulisannya secara paralel dengan setiap partisi lainnya. Urutan penulisan dalam Wilayah jarak jauh mungkin tidak cocok dengan urutan penulisan yang terjadi dalam Wilayah sumber. Untuk informasi selengkapnya tentang partisi tabel,

lihat postingan blog Penskalaan DynamoDB: Dampak partisi, hot key, dan pemisahan panas terhadap performa.

- Item yang baru ditulis biasanya disebarkan ke semua tabel replika dalam hitungan detik. Wilayah terdekat cenderung menyebarkan lebih cepat.
- Amazon CloudWatch menyediakan `ReplicationLatency` metrik untuk setiap pasangan Wilayah. Ini dihitung berdasarkan item yang tiba dan membandingkan waktu kedatangannya dengan waktu penulisan awal serta menghitung rata-ratanya. Pengaturan waktu disimpan CloudWatch di dalam Wilayah sumber. Melihat pengaturan waktu rata-rata dan maksimum dapat membantu menentukan jeda replikasi rata-rata dan terburuk. Tidak ada SLA pada latensi ini.
- Jika item yang sama diperbarui pada waktu yang hampir bersamaan (dalam jendela `ReplicationLatency` ini) di dua Wilayah yang berbeda, dan penulisan kedua terjadi sebelum penulisan pertama direplikasi, ada potensi konflik penulisan. Tabel global menyelesaikan konflik demikian dengan mekanisme penulis terakhir menang, berdasarkan timestamp penulisan. Penulisan pertama “kalah” dengan penulisan kedua. Konflik ini tidak dicatat dalam CloudWatch atau AWS CloudTrail.
- Setiap item memiliki timestamp tulis terakhir yang disimpan sebagai properti sistem privat. Pendekatan penulis terakhir menang diimplementasikan menggunakan penulisan bersyarat yang mengharuskan timestamp item masuk lebih besar daripada timestamp item yang ada.
- Tabel global akan mereplikasi semua item ke semua Wilayah yang berpartisipasi. Jika ingin memiliki cakupan replikasi yang berbeda, Anda dapat membuat beberapa tabel berbeda dan memasukkan Wilayah yang berpartisipasi yang berbeda ke masing-masing tabel.
- Penulisan akan diterima di Wilayah lokal meskipun Wilayah replika sedang offline atau `ReplicationLatency` berkembang. Tabel lokal terus mencoba mereplikasi item ke tabel jarak jauh hingga setiap item berhasil.
- Jika suatu Wilayah menjadi offline sepenuhnya, ketika kemudian kembali online, semua replikasi keluar dan masuk yang tertunda akan dicoba ulang. Tidak ada tindakan khusus yang diperlukan untuk mengembalikan sinkronisasi tabel. Mekanisme penulis terakhir menang memastikan data pada akhirnya akan menjadi konsisten.
- Anda dapat menambahkan Region baru ke tabel DynamoDB kapan saja. DynamoDB akan menangani sinkronisasi awal dan replikasi yang sedang berlangsung. Jika suatu Wilayah dihapus, meskipun itu adalah Wilayah asli, hanya tabel untuk Wilayah tersebut yang akan dihapus.
- DynamoDB tidak memiliki titik akhir global. Semua permintaan dibuat ke titik akhir regional, yang kemudian mengakses instans tabel global yang bersifat lokal di Wilayah tersebut.

- Panggilan ke DynamoDB tidak boleh dilakukan lintas Wilayah. Praktik terbaiknya adalah lapisan komputasi di satu Wilayah hanya mengakses titik akhir DynamoDB lokal untuk Wilayah tersebut secara langsung. Jika masalah terdeteksi dalam Wilayah, apakah masalah tersebut ada di lapisan DynamoDB atau di tumpukan sekitarnya, maka lalu lintas pengguna akhir harus diarahkan ke lapisan komputasi berbeda yang dihosting di Wilayah yang berbeda. Berkat replikasi tabel global, Wilayah yang berbeda sudah memiliki salinan lokal data yang sama untuk digunakan secara lokal. Dalam beberapa situasi, lapisan komputasi di satu Wilayah dapat meneruskan permintaan ke lapisan komputasi Wilayah lain untuk diproses, tetapi lapisan komputasi ini tidak boleh mengakses titik akhir DynamoDB jarak jauh secara langsung. Untuk informasi selengkapnya tentang kasus penggunaan khusus ini, lihat Perutean permintaan lapisan komputasi.

Kasus penggunaan tabel global DynamoDB

Tabel global memberikan manfaat umum ini:

- Pembacaan berlatensi rendah. Anda dapat menempatkan salinan data lebih dekat ke pengguna akhir untuk mengurangi latensi jaringan selama pembacaan. Cache tetap baru seperti nilai `ReplicationLatency`.
- Penulisan berlatensi lebih rendah. Anda dapat menulis ke wilayah terdekat untuk mengurangi latensi jaringan dan waktu yang dibutuhkan untuk menghasilkan penulisan. Lalu lintas tulis harus dirutekan dengan hati-hati untuk memastikan tidak ada konflik. Teknik perutean dibahas lebih detail di Minta perutean dengan tabel global DynamoDB.
- Peningkatan ketahanan dan pemulihan bencana. Anda dapat mengevakuasi suatu Wilayah (memindahkan beberapa atau semua permintaan ke Wilayah tersebut) jika Wilayah tersebut mengalami penurunan performa atau penghentian total, dengan sasaran titik pemulihan (RPO) dan sasaran waktu pemulihan (RTO) diukur dalam detik. Penggunaan tabel global juga meningkatkan SLA DynamoDB dari 99,99% menjadi 99,999%.
- Migrasi Wilayah yang mulus. Anda dapat menambahkan Wilayah baru lalu menghapus Wilayah lama untuk memigrasikan deployment dari satu Wilayah ke Wilayah lainnya, tanpa adanya waktu henti pada lapisan data. Misalnya, Anda dapat menggunakan tabel global DynamoDB agar sistem manajemen pesanan mencapai pemrosesan latensi rendah yang andal dalam skala tinggi sekaligus menjaga ketahanan terhadap kegagalan AZ dan Regional.

Tulis mode dengan tabel global DynamoDB

Tabel global selalu aktif-aktif di tingkat tabel. Namun, Anda dapat memperlakukannya sebagai aktif-pasif dengan mengontrol cara Anda merutekan permintaan tulis. Misalnya, Anda dapat memutuskan untuk merutekan permintaan tulis ke satu Wilayah untuk menghindari potensi konflik penulisan.

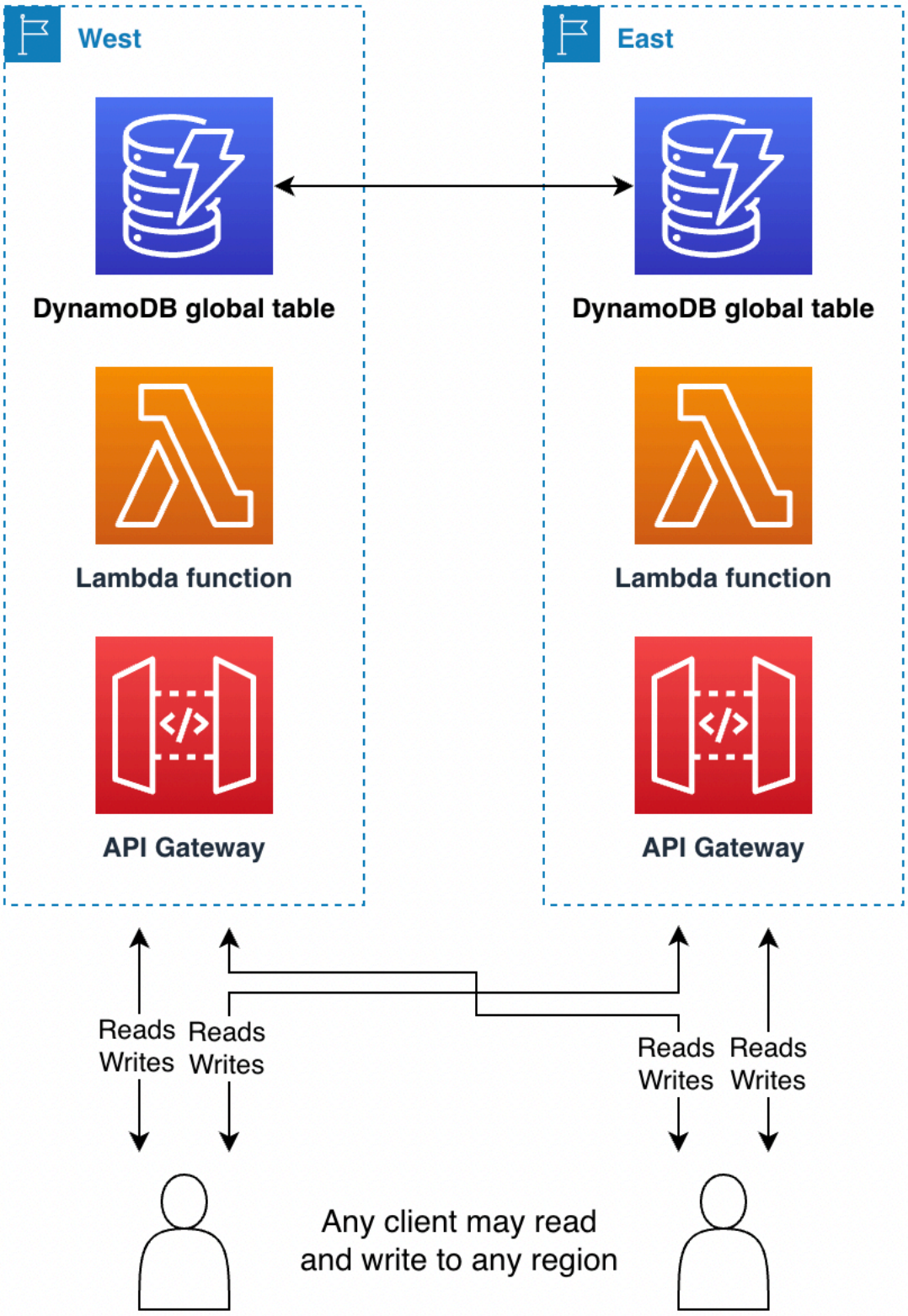
Ada tiga kategorisasi utama pola penulisan terkelola:

- Mode tulis ke Wilayah mana pun (tidak ada primer)
- Mode tulis ke satu Wilayah (primer tunggal)
- Mode penulisan ke Wilayah Anda (primer campuran)

Anda harus mempertimbangkan pola penulisan yang sesuai dengan kasus penggunaan Anda. Pilihan ini memengaruhi cara Anda merutekan permintaan, mengevakuasi suatu Wilayah, dan menangani pemulihan bencana. Praktik terbaik secara keseluruhan dapat berbeda tergantung pada mode tulis aplikasi Anda.

Mode tulis ke Wilayah mana pun (tidak ada primer)

Mode tulis ke Wilayah mana pun sepenuhnya aktif-aktif dan tidak memberlakukan batasan tempat penulisan dapat terjadi. Wilayah mana pun dapat menerima penulisan kapan saja. Ini adalah mode yang paling sederhana. Mode ini hanya dapat digunakan dengan beberapa jenis aplikasi. Ini cocok jika semua penulis idempoten, dan oleh karena itu dapat diulang dengan aman sehingga operasi tulis secara bersamaan atau berulang di seluruh Wilayah tidak menimbulkan konflik. Misalnya, saat pengguna memperbarui data kontak mereka. Mode ini juga berfungsi dengan baik untuk kasus khusus idempoten, set data khusus tambahan yang semua penulisannya merupakan sisipan unik dengan kunci primer deterministik. Terakhir, mode ini cocok jika risiko penulisan yang bertentangan dapat diterima.



Mode tulis ke Wilayah mana pun adalah arsitektur yang paling mudah untuk diimplementasikan. Perutean lebih mudah karena Wilayah mana pun dapat menjadi target penulisan kapan saja. Failover

lebih mudah, karena setiap penulisan terbaru dapat diputar ulang beberapa kali ke Wilayah sekunder mana pun. Jika memungkinkan, Anda harus merancang mode tulis ini.

Misalnya, layanan streaming video sering kali menggunakan tabel global untuk melacak bookmark, ulasan, bendera status tontonan, dan sebagainya. Deployment ini dapat menggunakan mode tulis ke Wilayah mana pun selama setiap tulis bersifat idempoten dan nilai benar berikutnya untuk suatu item tidak bergantung pada nilainya saat ini. Hal ini berlaku untuk pembaruan pengguna yang menetapkan status baru pengguna secara langsung, seperti mengatur kode waktu terbaru, menetapkan ulasan baru, atau menyetel status tontonan baru. Jika permintaan tulis pengguna dirutekan ke Wilayah lain, operasi tulis terakhir akan tetap ada dan status global akan diselesaikan sesuai dengan penetapan terakhir. Operasi baca dalam mode ini pada akhirnya akan menjadi konsisten, setelah tertunda oleh nilai `ReplicationLatency` terbaru.

Contoh lain, sebuah perusahaan jasa keuangan menggunakan tabel global sebagai bagian dari sistem untuk menyimpan penghitungan pembelian kartu debit untuk setiap pelanggan, untuk menghitung hadiah cash-back pelanggan tersebut. Transaksi baru mengalir dari seluruh dunia dan masuk ke berbagai Wilayah. Untuk desain mereka saat ini yang tidak memanfaatkan tabel global, mereka menggunakan satu item Running Balance per pelanggan. Tindakan pelanggan memperbarui saldo dengan ekspresi ADD, yang tidak idempoten karena nilai baru yang benar bergantung pada nilai saat ini. Ini berarti saldo menjadi tidak sinkron jika ada dua operasi tulis ke saldo yang sama pada waktu yang hampir bersamaan di Wilayah berbeda.

Perusahaan yang sama ini dapat menerapkan mode tulis ke Wilayah mana pun melalui desain ulang yang cermat dengan tabel global DynamoDB. Desain baru ini dapat mengikuti model "streaming peristiwa" - yang pada dasarnya merupakan buku besar dengan alur kerja tambah-saja. Setiap tindakan pelanggan menambahkan item baru ke koleksi item yang dikelola untuk pelanggan tersebut. Koleksi item adalah kumpulan item yang memiliki kunci primer yang sama, tetapi kunci urutannya berbeda. Setiap tindakan tulis yang menambahkan tindakan pelanggan adalah sisipan idempoten, menggunakan ID pelanggan sebagai kunci partisi dan ID transaksi sebagai kunci pengurutan. Desain ini membuat penghitungan saldo menjadi lebih rumit, karena memerlukan `Query` untuk menarik item yang diikuti oleh beberapa perhitungan sisi klien. Namun, keuntungannya adalah membuat semua penulisan menjadi idempoten, sehingga memberikan penyederhanaan perutean dan failover yang signifikan. Untuk mengetahui informasi selengkapnya, lihat Minta perutean dengan tabel global DynamoDB.

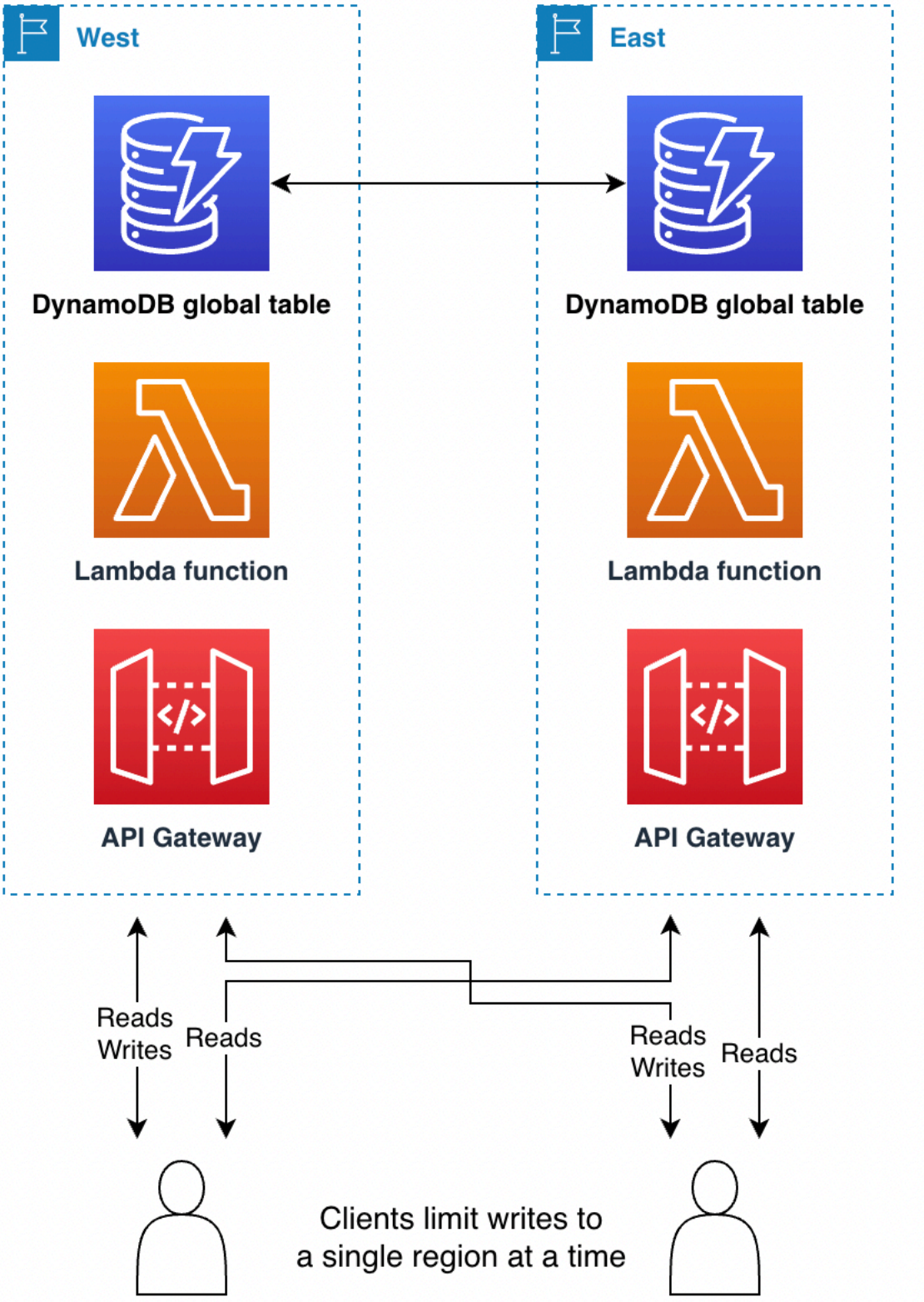
Contoh ketiga, katakanlah ada pelanggan yang menempatkan iklan online. Mereka telah memutuskan risiko rendah kehilangan data akan dapat diterima untuk menghasilkan penyederhanaan desain mode tulis Wilayah mana pun. Saat menayangkan iklan, mereka hanya

memiliki waktu beberapa milidetik untuk mengambil metadata yang cukup guna menentukan iklan yang akan ditampilkan, lalu mencatat tayangan iklan sehingga iklan yang sama tidak terulang kembali kepada pengguna tersebut. Dengan tabel global, mereka bisa mendapatkan pembacaan latensi rendah untuk pengguna akhir di seluruh dunia dan penulisan latensi rendah. Mereka dapat mencatat semua tayangan iklan untuk pengguna dalam satu item, dan menampilkannya sebagai Daftar yang terus bertambah. Mereka dapat menggunakan satu item alih-alih menambahkan ke koleksi item, karena dengan cara ini mereka dapat menghapus tayangan iklan lama yang merupakan bagian dari setiap penulisan tanpa membayar biaya penghapusan. Operasi tulis ini TIDAK idempoten, jadi jika pengguna akhir yang sama melihat iklan ditayangkan di beberapa Wilayah pada waktu yang kurang lebih bersamaan, ada kemungkinan satu penulisan tayangan iklan dapat menimpa yang lain. Untuk penempatan iklan online, risiko bahwa pengguna terkadang melihat iklan berulang-ulang sepadan dengan desain yang lebih sederhana dan efisien ini.

Primer tunggal (“Tulis ke satu Wilayah”)

Mode tulis ke satu Wilayah bersifat aktif-pasif dan merutekan semua penulisan tabel ke satu wilayah aktif. Perhatikan bahwa DynamoDB tidak memiliki gagasan tentang satu wilayah aktif; perutean aplikasi di luar DynamoDB mengatur ini. Mode tulis ke satu Wilayah menghindari konflik penulisan dengan memastikan penulisan hanya mengalir ke satu Wilayah pada satu waktu. Mode tulis ini berguna ketika Anda ingin menggunakan ekspresi atau transaksi bersyarat, karena keduanya tidak akan berfungsi kecuali Anda mengetahui bahwa Anda bertindak berdasarkan data terbaru. Jadi, menggunakan ekspresi dan transaksi bersyarat memerlukan pengiriman semua permintaan tulis ke satu Wilayah dengan data terbaru.

Bacaan akhir konsisten dapat dilakukan di Wilayah replika mana pun untuk mendapatkan latensi yang lebih rendah. Bacaan sangat konsisten harus ditujukan ke satu wilayah utama.


West
East
DynamoDB global table
DynamoDB global table
Lambda function
Lambda function
API Gateway
API Gateway
Reads
Writes
Reads
Reads
Writes
Reads
Clients limit writes to
a single region at a time

Terkadang perlu untuk mengubah Wilayah aktif sebagai respons terhadap kegagalan Regional, untuk membantu data. Mengevakuasi Wilayah dengan tabel global DynamoDBadalah salah satu contoh kasus penggunaan ini. Beberapa pelanggan akan mengubah Wilayah yang aktif saat ini dengan jadwal reguler, seperti deployment "follow-the-sun". Hal ini menempatkan Wilayah aktif di dekat geografi dengan aktivitas terbanyak, sehingga memberikan latensi baca dan tulis terendah. Hal ini juga memiliki keuntungan tambahan dengan memanggil jalur kode yang mengubah Wilayah setiap hari, memastikan jalur tersebut telah diuji dengan baik sebelum pemulihan bencana apa pun.

Wilayah pasif dapat mempertahankan serangkaian infrastruktur yang diperkecil di sekitar DynamoDB yang akan dibangun hanya jika wilayah tersebut menjadi wilayah aktif. Untuk diskusi yang lebih mendalam tentang desain pilot light dan warm standby lihat Disaster Recovery (DR) Architecture on AWS, Bagian III: Pilot Light and Warm Standby.

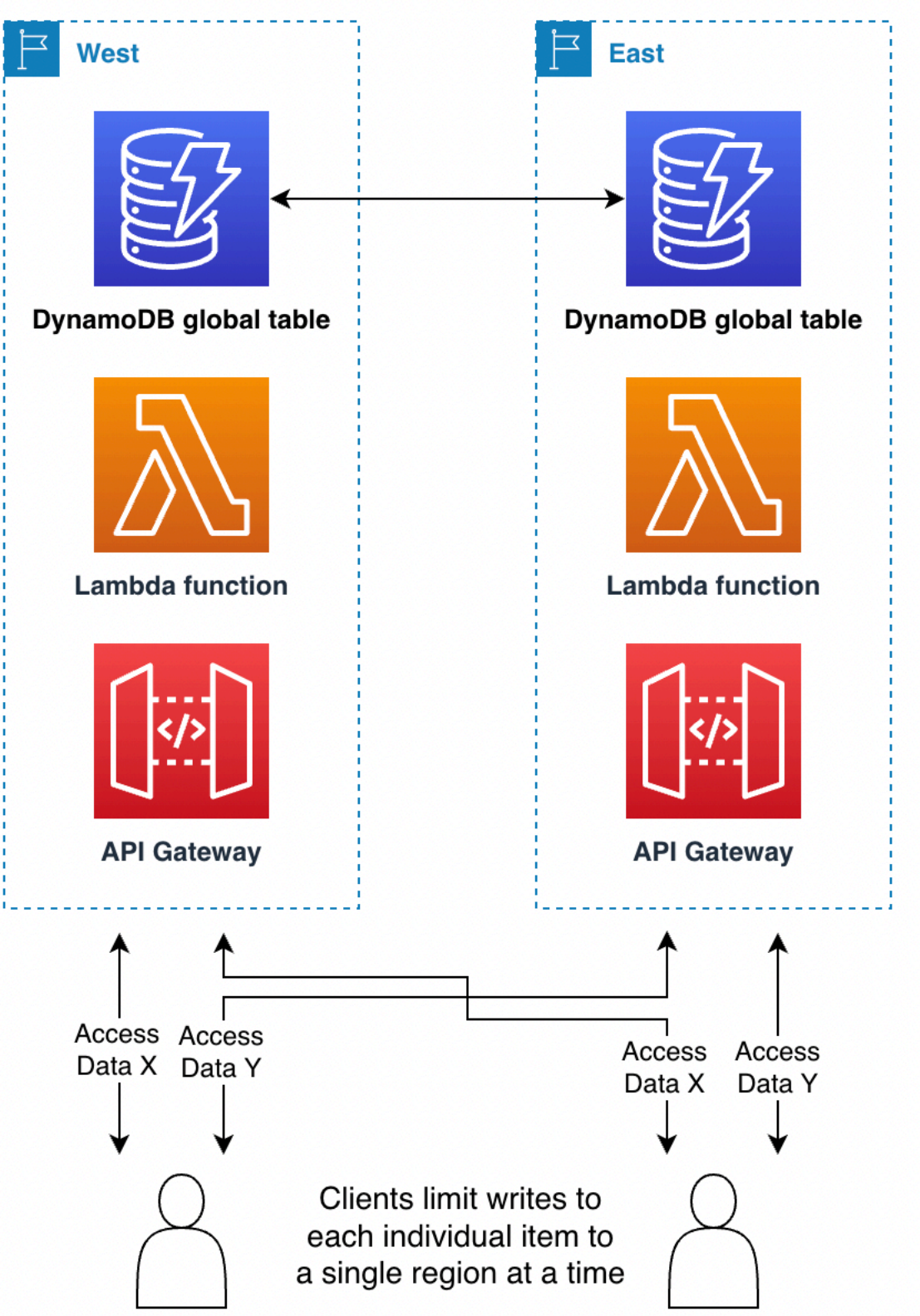
Menggunakan mode tulis ke satu Wilayah berfungsi dengan baik saat memanfaatkan tabel global untuk pembacaan terdistribusi global dengan latensi rendah. Misalnya, sebuah perusahaan media sosial besar memiliki jutaan pengguna dan miliaran postingan. Setiap pengguna ditetapkan ke suatu Wilayah pada saat pembuatan akun, yang ditempatkan dekat dengan lokasi mereka secara geografis. Semua data mereka berada di tabel non-global itu. Perusahaan menggunakan tabel global terpisah untuk menyimpan pemetaan pengguna ke wilayah asal mereka, menggunakan mode tulis ke satu Wilayah. Ini menyimpan salinan hanya-baca di seluruh dunia untuk membantu menemukan lokasi setiap data pengguna secara langsung dengan latensi tambahan minimum. Pembaruan jarang terjadi (hanya ketika memindahkan Wilayah asal pengguna dari satu Wilayah ke Wilayah lain) dan selalu melalui satu Wilayah untuk penulisan, guna menghindari kemungkinan konflik penulisan.

Contoh lainnya, perhatikan pelanggan jasa keuangan yang menerapkan penghitungan cash-back harian. Mereka menggunakan mode tulis ke Wilayah mana pun untuk menghitung saldo tetapi menggunakan mode tulis ke satu Wilayah untuk melacak pembayaran cash-back yang sebenarnya. Jika mereka ingin memberikan hadiah kepada pelanggan sebesar 1 sen untuk setiap $10 yang dibelanjakan dalam sehari, mereka perlu melakukan `Query` untuk semua transaksi dari hari sebelumnya, menghitung total pembelanjaan, menulis keputusan cash-back ke tabel baru, menghapus kumpulan item yang dikueri untuk menandainya sebagai digunakan, dan menggantinya dengan item tunggal yang menyimpan jumlah sisa yang harus dimasukkan ke dalam penghitungan hari berikutnya. Pekerjaan ini memerlukan transaksi, sehingga akan lebih baik dengan mode tulis ke satu Wilayah. Suatu aplikasi dapat menggabungkan mode penulisan, bahkan pada tabel yang sama, selama beban kerja tidak memiliki peluang untuk tumpang tindih.

Primer campuran (“Tulis ke Wilayah Anda”)

Mode tulis ke Wilayah Anda menetapkan subset data yang berbeda ke Wilayah yang berbeda dan hanya mengizinkan operasi tulis pada item melalui wilayah asalnya. Mode ini aktif-pasif tetapi menetapkan Wilayah aktif berdasarkan item. Setiap Wilayah adalah yang utama untuk kumpulan datanya yang tidak tumpang tindih, dan penulisan harus dijaga untuk memastikan lokalitas yang tepat.

Mode ini mirip dengan mode tulis ke satu Wilayah, hanya saja mode ini memungkinkan penulisan dengan latensi lebih rendah, karena data yang terkait dengan setiap pengguna akhir dapat ditempatkan di jaringan yang lebih dekat dengan pengguna tersebut. Mode ini juga menyebarkan infrastruktur sekitar secara lebih merata di antara Wilayah dan memerlukan lebih sedikit pekerjaan untuk membangun infrastruktur selama skenario failover, karena sebagian infrastruktur di semua wilayah sudah aktif.



Menentukan wilayah asal untuk item dapat dilakukan dengan berbagai cara:

- Intrinsik: Beberapa aspek data memperjelas Wilayah tempat data tersebut berada, seperti kunci partisinya. Misalnya, pelanggan dan semua data tentang pelanggan tersebut akan ditandai dalam data pelanggan sebagai berasal dari wilayah tertentu. Teknik ini dijelaskan dalam Menggunakan penyematan Wilayah untuk menetapkan Wilayah asal item dalam tabel global Amazon DynamoDB
- Negosiasi: Wilayah asal setiap kumpulan data dinegosiasikan dengan cara eksternal, seperti dengan layanan global terpisah yang mempertahankan penetapan. Penetapan tersebut mungkin memiliki jangka waktu terbatas dan setelah itu harus dinegosiasikan ulang.
- Berorientasi tabel: Daripada hanya mereplikasi satu tabel global, buatlah tabel global sebanyak jumlah Wilayah yang direplikasi. Nama setiap tabel menunjukkan Wilayah asalnya. Dalam operasi standar, semua data ditulis ke Wilayah asal sementara Wilayah lain menyimpan salinan hanya-baca. Selama failover, Wilayah lain untuk sementara waktu akan menerapkan tugas penulisan untuk tabel tersebut.

Misalnya, anggaplah Anda bekerja di perusahaan game. Anda memerlukan pembacaan dan penulisan dengan latensi rendah untuk semua gamer di seluruh dunia. Anda dapat menempatkan setiap gamer di Wilayah terdekat mereka. Wilayah itu mengambil semua bacaan dan tulisan mereka, memastikan selalu ada read-after-write konsistensi yang kuat. Namun, jika gamer tersebut melakukan perjalanan atau Wilayah asal mereka mengalami gangguan, salinan lengkap data mereka akan tersedia di Wilayah alternatif. Jadi, gamer dapat ditetapkan ke Wilayah asal yang berbeda sesuai kebutuhan.

Contoh lainnya, katakanlah Anda bekerja di perusahaan konferensi video. Setiap metadata panggilan konferensi ditetapkan ke Wilayah tertentu. Pemanggil dapat menggunakan Wilayah yang paling dekat dengan mereka untuk mendapatkan latensi terendah. Jika terjadi gangguan Wilayah, penggunaan tabel global memungkinkan pemulihan cepat karena sistem dapat memindahkan pemrosesan panggilan ke Wilayah lain yang sudah terdapat salinan data yang direplikasi.

Minta perutean dengan tabel global DynamoDB

Mungkin bagian paling rumit dari deployment tabel global adalah mengelola perutean permintaan. Permintaan harus dikirim dari pengguna akhir ke Wilayah yang dipilih terlebih dahulu, lalu dirutekan dengan cara tertentu. Permintaan menemukan beberapa tumpukan layanan di Wilayah itu, termasuk lapisan komputasi yang mungkin terdiri dari penyeimbang beban yang didukung oleh AWS Lambda fungsi, wadah, atau node Amazon Elastic Compute Cloud (Amazon EC2), dan mungkin layanan lain termasuk database lain. Lapisan komputasi tersebut berkomunikasi dengan DynamoDB yang dilakukan menggunakan titik akhir lokal untuk Wilayah tersebut. Data dalam tabel global direplikasi

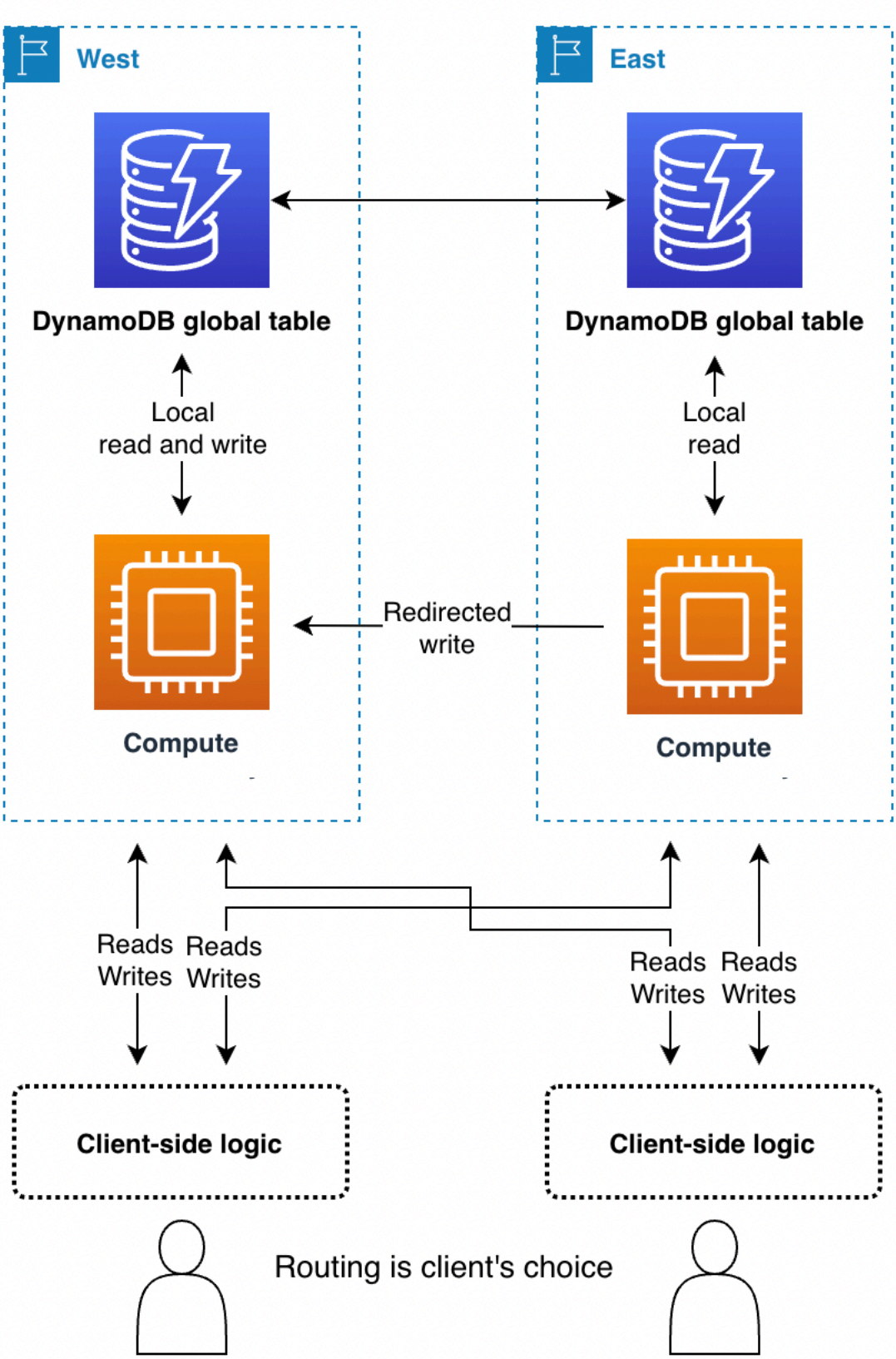
ke semua Wilayah lain yang berpartisipasi, dan setiap Wilayah memiliki tumpukan layanan serupa di sekitar tabel DynamoDB-nya.

Tabel global menyediakan salinan lokal dari data yang sama untuk setiap tumpukan di berbagai Wilayah. Sebaiknya Anda merancang tumpukan tunggal dalam satu Wilayah dan mengantisipasi melakukan panggilan jarak jauh ke titik akhir DynamoDB Wilayah sekunder jika tabel DynamoDB lokal mengalami masalah. Ini bukan praktik terbaik. Latensi yang terkait dengan lintas Wilayah mungkin 100 kali lebih tinggi dibandingkan akses lokal. back-and-forthSerangkaian 5 permintaan mungkin membutuhkan milidetik saat dilakukan secara lokal tetapi detik saat melintasi dunia. Sebaiknya rutekan pengguna akhir ke Wilayah lain untuk diproses. Untuk memastikan ketahanan, Anda memerlukan replikasi di beberapa Wilayah, dengan replikasi lapisan komputasi serta lapisan data.

Ada banyak teknik alternatif untuk merutekan permintaan pengguna akhir ke suatu Wilayah untuk diproses. Pilihan optimal bergantung pada mode tulis dan pertimbangan failover Anda. Bagian ini membahas empat opsi: berbasis klien, lapisan komputasi, Route 53, dan Global Accelerator.

Perutean permintaan berbasis klien

Dengan routing permintaan berbasis klien, klien pengguna akhir seperti aplikasi, halaman web dengan klien JavaScript lain akan melacak titik akhir aplikasi yang valid. Hal tersebut akan menjadi titik akhir aplikasi seperti Amazon API Gateway, bukan titik akhir DynamoDB literal. Klien pengguna akhir menggunakan logika tertanam sendiri untuk memilih Wilayah mana yang akan berkomunikasi. Pengguna akhir tersebut dapat memilih berdasarkan pemilihan acak, latensi terendah yang diamati, pengukuran bandwidth tertinggi yang diamati, atau pemeriksaan kesehatan yang dilakukan secara lokal.


West
East
DynamoDB global table
DynamoDB global table
Local
read and write
Local
read
Redirected
write
Compute
Compute
Reads
Writes
Reads
Writes
Reads
Writes
Reads
Writes
Client-side logic
Client-side logic
Routing is client's choice

Keuntungan perutean permintaan berbasis klien adalah dapat adaptif terhadap hal-hal seperti kondisi lalu lintas internet publik di dunia nyata untuk berpindah Wilayah jika terjadi penurunan performa. Klien harus mengetahui semua titik akhir potensial, tetapi peluncuran titik akhir Regional baru bukanlah hal yang sering terjadi.

Dengan mode tulis ke Wilayah mana pun, klien dapat memilih titik akhir pilihannya secara sepihak. Jika aksesnya ke satu Wilayah terganggu, klien dapat merutekan ke titik akhir lain.

Dengan mode tulis ke satu Wilayah, klien memerlukan mekanisme untuk merutekan penulisannya ke wilayah yang sedang aktif. Hal ini dapat berupa pengujian empiris terhadap wilayah mana yang saat ini menerima penulisan (mencatat setiap penolakan penulisan dan kembali ke alternatif) atau rumit seperti memanggil koordinator global untuk mengkueri status aplikasi saat ini (mungkin dibangun pada kontrol perutean Pengendali Pemulihan Aplikasi (ARC) Route 53 yang menyediakan sistem berbasis kuorum 5 wilayah untuk mempertahankan status global untuk kebutuhan semacam ini). Klien dapat memutuskan apakah pembacaan dapat dikirim ke Wilayah mana pun untuk konsistensi akhir atau harus dirutekan ke wilayah aktif untuk konsistensi yang kuat. Untuk informasi selengkapnya, lihat Cara kerja Route 53.

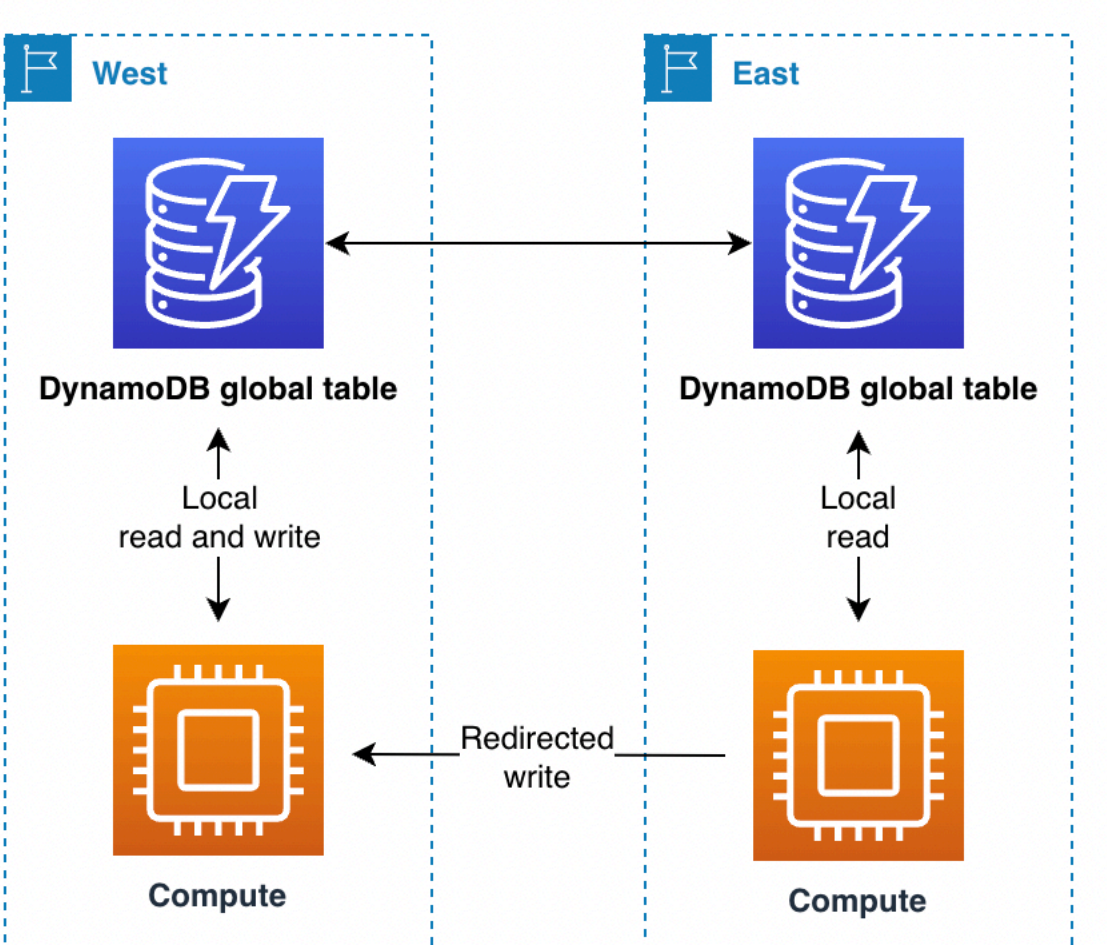
Dengan mode tulis ke Wilayah Anda, klien perlu menentukan wilayah asal kumpulan data yang digunakannya. Misalnya, jika klien berhubungan dengan akun pengguna dan setiap akun pengguna ditempatkan di suatu Wilayah, klien dapat meminta titik akhir yang sesuai dari sistem login global.

Misalnya, perusahaan jasa keuangan yang membantu pengguna mengelola keuangan bisnisnya melalui web dapat menggunakan tabel global dengan mode tulis ke Wilayah Anda. Setiap pengguna harus masuk ke layanan pusat. Layanan tersebut mengembalikan kredensial dan titik akhir untuk Wilayah tempat kredensial tersebut akan berfungsi. Kredensialnya valid untuk waktu yang singkat. Setelah itu, halaman web otomatis menegosiasikan aktivitas masuk baru, yang memberikan peluang untuk mengalihkan aktivitas pengguna ke Wilayah baru.

Perutean permintaan lapisan komputasi

Dengan perutean permintaan lapisan komputasi, kode yang berjalan di lapisan komputasi memutuskan apakah kode akan memproses permintaan secara lokal, atau meneruskannya ke salinannya sendiri yang berjalan di Wilayah lain. Saat Anda menggunakan mode tulis ke satu Wilayah, lapisan komputasi dapat mendeteksi bahwa itu bukan wilayah yang aktif dan memungkinkan operasi baca lokal sekaligus meneruskan semua operasi tulis ke Wilayah lain. Kode lapisan komputasi ini harus mengetahui topologi data dan aturan perutean, serta menerapkannya secara andal berdasarkan pengaturan terbaru yang menentukan Wilayah yang aktif untuk data tertentu. Tumpukan perangkat lunak luar di dalam Wilayah tidak harus mengetahui bagaimana permintaan

baca dan tulis dirutekan oleh layanan mikro. Dalam desain yang kuat, Wilayah penerima memvalidasi apakah Wilayah tersebut merupakan wilayah primer saat ini untuk operasi tulis. Jika tidak, maka akan muncul kesalahan yang menunjukkan bahwa status global perlu diperbaiki. Wilayah penerima mungkin juga melakukan buffering pada operasi tulis untuk sementara waktu jika Wilayah utama sedang dalam proses perubahan. Dalam semua kasus, tumpukan komputasi di suatu Wilayah hanya menulis ke titik akhir DynamoDB lokalnya, tetapi tumpukan komputasi tersebut mungkin berkomunikasi satu sama lain.

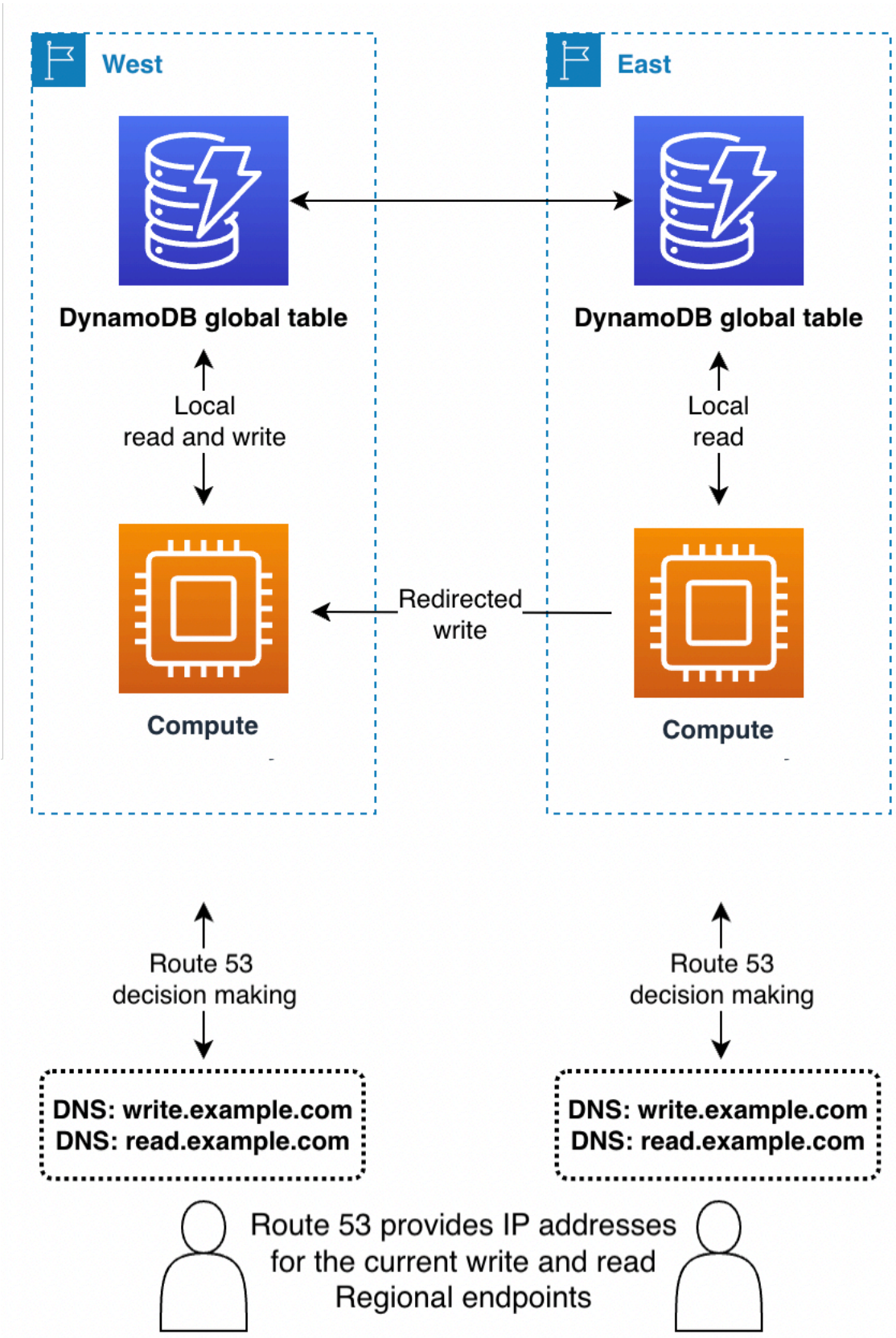


Dalam skenario ini, katakanlah perusahaan jasa keuangan menggunakan model follow-the-sun Single Primary. Mereka menggunakan sistem dan pustaka untuk proses perutean ini. Sistem mereka secara keseluruhan mempertahankan keadaan global, mirip dengan kontrol AWS routing ARC. Mereka menggunakan tabel global untuk melacak Wilayah yang merupakan Wilayah utama, dan kapan peralihan utama berikutnya dijadwalkan. Semua operasi baca dan tulis melalui pustaka, yang berkoordinasi dengan sistemnya. Pustaka memungkinkan operasi baca dilakukan secara lokal dengan latensi rendah. Untuk operasi tulis, aplikasi memeriksa apakah Wilayah lokal adalah Wilayah utama saat ini. Jika ya, operasi tulis selesai secara langsung. Jika tidak, pustaka meneruskan tugas tulis ke pustaka yang ada di Wilayah utama saat ini. Pustaka penerima tersebut mengonfirmasi

bahwa ia juga menganggap dirinya sebagai Wilayah utama dan menimbulkan kesalahan jika bukan, yang menunjukkan penundaan penyebaran dengan keadaan global. Pendekatan ini memberikan manfaat validasi dengan tidak menulis langsung ke titik akhir DynamoDB jarak jauh.

Perutean permintaan Route 53

Amazon Application Recovery Controller (ARC) adalah teknologi Domain Name Service (DNS). Dengan Route 53, klien meminta titik akhir dengan mencari nama domain DNS yang dikenal, dan Route 53 mengembalikan alamat IP yang sesuai dengan titik akhir regional yang dianggap paling tepat. Route 53 memiliki daftar kebijakan perutean yang digunakan untuk menentukan wilayah yang sesuai. Route 53 juga dapat melakukan perutean failover untuk merutekan lalu lintas dari wilayah yang gagal dalam pemeriksaan kondisi.


West
East
DynamoDB global table
DynamoDB global table
Local
read and write
Local
read
Redirected
write
Compute
Compute
Route 53
decision making
Route 53
decision making
DNS: write.example.com
DNS: read.example.com
DNS: write.example.com
DNS: read.example.com
Route 53 provides IP addresses
for the current write and read
Regional endpoints

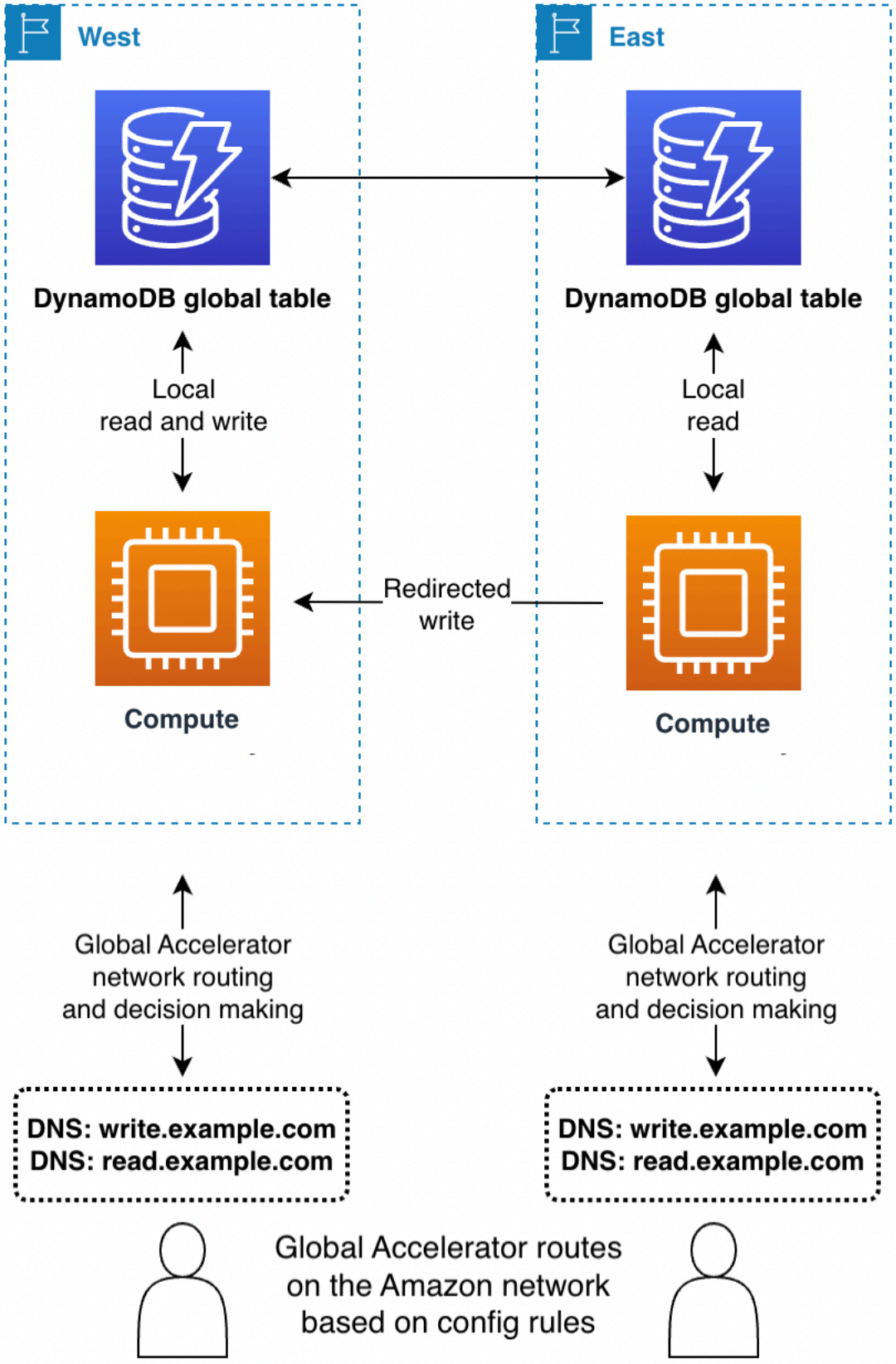
- Dengan mode tulis ke Wilayah mana pun, atau jika digabungkan dengan perutean permintaan lapisan komputasi di backend, Route 53 dapat diberi akses penuh untuk mengembalikan Wilayah berdasarkan aturan internal kompleks seperti Wilayah dalam kedekatan jaringan terdekat, atau kedekatan geografis terdekat, atau pilihan lainnya.
- Dengan mode tulis ke satu Wilayah, Route 53 dapat dikonfigurasi untuk mengembalikan wilayah yang saat ini aktif (menggunakan Route 53 ARC).

> **Note**
>
> Klien menyimpan alamat IP dalam respons dari Route 53 selama waktu yang ditunjukkan oleh pengaturan waktu untuk tayang (TTL) pada nama domain. TTL yang lebih panjang memperpanjang sasaran waktu pemulihan (RTO) bagi semua klien untuk mengenali titik akhir baru. Nilai 60 detik adalah tipikal untuk penggunaan failover. Tidak semua perangkat lunak mematuhi masa berlaku TTL DNS dengan sempurna.

- Dengan mode tulis ke Wilayah Anda, sebaiknya hindari Route 53 kecuali Anda juga menggunakan perutean permintaan lapisan komputasi.

Perutean permintaan Global Accelerator

Klien menggunakan AWS Global Accelerator untuk mencari nama domain yang dikenal di Route 53. Namun, alih-alih mendapatkan kembali alamat IP yang sesuai dengan titik akhir regional, klien menerima alamat IP statis anycast yang merutekan ke lokasi AWS tepi terdekat. Mulai dari lokasi tepi itu, semua lalu lintas dirutekan di AWS jaringan pribadi dan ke beberapa titik akhir (seperti penyeimbang beban atau API Gateway) di Wilayah yang dipilih oleh aturan perutean yang dipertahankan dalam Global Accelerator. Dibandingkan dengan perutean berdasarkan aturan Route 53, perutean permintaan Global Accelerator memiliki latensi yang lebih rendah karena mengurangi jumlah lalu lintas di internet publik. Selain itu, karena Global Accelerator tidak bergantung pada masa berlaku TTL DNS untuk mengubah aturan perutean, perutean dapat disesuaikan dengan lebih cepat.


West
East
DynamoDB global table
DynamoDB global table
Local
read and write
Local
read
Redirected
write
Compute
Compute
Global Accelerator
network routing
and decision making
Global Accelerator
network routing
and decision making
DNS: write.example.com
DNS: read.example.com
DNS: write.example.com
DNS: read.example.com
Global Accelerator routes
on the Amazon network
based on config rules

- Dengan mode tulis ke Wilayah mana pun, atau jika dikombinasikan dengan perutean permintaan lapisan komputasi di back-end, Global Accelerator dapat bekerja dengan lancar. Klien terhubung ke lokasi edge terdekat dan tidak perlu khawatir dengan Wilayah mana yang menerima permintaan.
- Dengan tulis ke satu Wilayah, aturan perutean Global Accelerator harus mengirimkan permintaan ke Wilayah yang sedang aktif. Anda dapat menggunakan pemeriksaan kondisi yang secara artifisial melaporkan kegagalan pada Wilayah yang tidak diperhitungkan sebagai Wilayah aktif oleh sistem global Anda. Seperti halnya DNS, nama domain DNS alternatif dapat digunakan untuk merutekan permintaan baca jika permintaan tersebut dapat berasal dari Wilayah mana pun.
- Dengan mode tulis ke Wilayah Anda, sebaiknya hindari Global Accelerator kecuali Anda juga menggunakan perutean permintaan lapisan komputasi.

Mengevakuasi Wilayah dengan tabel global DynamoDB

Evakuasi suatu Wilayah adalah proses memigrasikan aktivitas baca dan tulis keluar dari Wilayah tersebut. Ini paling sering merupakan aktivitas tulis, dan terkadang aktivitas baca.

Mengevakuasi Wilayah aktif

Anda dapat memutuskan untuk mengevakuasi Wilayah aktif karena sejumlah alasan. Evakuasi dapat menjadi bagian dari aktivitas bisnis biasa seperti jika Anda menggunakan mode follow-the-sun tulis ke satu Wilayah. Evakuasi juga bisa disebabkan oleh keputusan bisnis untuk mengubah Wilayah yang sedang aktif, sebagai respons terhadap kegagalan tumpukan perangkat lunak di luar DynamoDB, atau karena Anda menghadapi masalah umum seperti latensi yang lebih tinggi dari biasanya di dalam Wilayah.

Dengan mode tulis ke Wilayah mana pun, mengevakuasi Wilayah aktif sangatlah mudah. Anda dapat merutekan lalu lintas ke Wilayah alternatif melalui sistem perutean apa pun, dan membiarkan operasi tulis yang telah terjadi di Wilayah yang dievakuasi direplikasi seperti biasa.

Dengan mode tulis ke satu Wilayah dan tulis ke Wilayah Anda, Anda harus memastikan semua penulisan ke Wilayah aktif telah direkam sepenuhnya, diproses streaming, dan disebarkan secara global sebelum memulai penulisan di Wilayah aktif yang baru. Hal ini diperlukan untuk memastikan bahwa penulisan di masa mendatang bertentangan dengan versi data terbaru.

Katakanlah Wilayah A aktif dan Wilayah B pasif (baik untuk tabel lengkap atau untuk item yang ditempatkan di Wilayah A). Mekanisme umum untuk melakukan evakuasi adalah dengan menjeda

operasi tulis ke A, menunggu cukup lama hingga operasi tersebut disebarkan sepenuhnya ke B, memperbarui tumpukan arsitektur untuk mengenali B sebagai aktif, lalu melanjutkan operasi tulis ke B. Tidak ada metrik yang menunjukkan kepastian mutlak bahwa Wilayah A telah sepenuhnya mereplikasi datanya ke Wilayah B. Jika Wilayah A sehat, menjeda operasi tulis ke Wilayah A dan menunggu 10 kali nilai maksimum terbaru metrik `ReplicationLatency` biasanya sudah cukup untuk menentukan bahwa replikasi tersebut selesai. Jika Wilayah A tidak sehat dan menunjukkan area lain mengalami peningkatan latensi, Anda dapat memilih kelipatan waktu tunggu yang lebih besar.

Mengevakuasi Wilayah offline

Ada kasus khusus yang perlu dipertimbangkan: bagaimana jika Wilayah A offline sepenuhnya tanpa pemberitahuan? Hal ini sangat kecil kemungkinannya, tetapi tetap perlu dipertimbangkan. Jika hal ini terjadi, operasi tulis apa pun di Wilayah A yang belum disebarkan akan ditahan dan disebarkan setelah Wilayah A kembali online. Operasi tulis tidak hilang, tetapi penyebarannya tertunda tanpa batas waktu.

Cara melanjutkan peristiwa ini merupakan keputusan aplikasi. Untuk kelangsungan bisnis, operasi tulis mungkin perlu diteruskan ke Wilayah B utama yang baru. Namun, jika item di Wilayah B menerima pembaruan sementara ada penyebaran operasi tulis yang tertunda untuk item tersebut dari Wilayah A, penyebaran tersebut akan disembunyikan di model penulis terakhir menang. Pembaruan apa pun di Wilayah B mungkin menyembunyikan permintaan tulis yang masuk.

Dengan mode tulis ke Wilayah mana pun, pembacaan dan penulisan dapat dilanjutkan di Wilayah B, dengan keyakinan bahwa item di Wilayah A pada akhirnya akan disebarkan ke Wilayah B dan mengenali potensi item yang hilang hingga Wilayah A kembali online. Jika memungkinkan, Anda harus mempertimbangkan untuk memutar ulang lalu lintas tulis terbaru (misalnya, menggunakan sumber peristiwa hulu) untuk mengisi celah dari operasi tulis yang mungkin hilang dan membiarkan resolusi konflik penulis terakhir menang menekan penyebaran akhir dari operasi tulis yang masuk.

Dengan mode tulis lainnya, Anda harus mempertimbangkan sejauh mana pekerjaan dapat dilanjutkan dengan sedikit out-of-date pandangan dunia. Beberapa operasi tulis berdurasi pendek, seperti yang dilacak oleh `ReplicationLatency`, akan hilang hingga Wilayah A kembali online. Apakah bisnis bisa maju? Dalam beberapa kasus penggunaan, bisa, tetapi pada kasus lainnya mungkin tidak bisa, tanpa mekanisme mitigasi tambahan.

Misalnya, katakanlah Anda perlu mempertahankan saldo kredit yang tersedia tanpa gangguan bahkan setelah kegagalan Wilayah. Anda dapat membagi saldo menjadi dua item berbeda, satu

ditempatkan di Wilayah A dan satu lagi di Wilayah B, masing-masing dimulai dengan setengah saldo yang tersedia. Ini akan menggunakan mode tulis ke Wilayah Anda. Pembaruan transaksional yang diproses di setiap Wilayah akan dituliskan pada salinan lokal saldo. Jika Wilayah A sepenuhnya offline, pekerjaan masih dapat dilanjutkan dengan pemrosesan transaksi di Wilayah B, dan operasi tulis akan terbatas pada bagian saldo yang disimpan di Wilayah B. Memisahkan saldo seperti ini menimbulkan kerumitan ketika saldo hampir habis atau kredit harus diseimbangkan kembali, tetapi hal ini memberikan satu contoh pemulihan bisnis yang aman bahkan dengan operasi tulis tertunda yang tidak pasti.

Contoh lain, misalkan Anda mengambil data formulir web. Anda dapat menggunakan Optimistic Concurrency Control (OCC) untuk menetapkan versi ke item data dan menyematkan versi terbaru ke formulir web sebagai bidang tersembunyi. Pada setiap pengiriman, operasi tulis berhasil hanya jika versi dalam basis data masih cocok dengan versi yang digunakan untuk membuat formulir. Jika versinya tidak cocok, formulir web bisa disegarkan (atau digabungkan secara hati-hati) berdasarkan versi saat ini dalam basis data, dan pengguna bisa melanjutkan lagi. Model OCC biasanya melindungi terhadap penimpaan klien lain dan pembuatan versi data yang baru, tetapi model ini juga dapat membantu selama failover ketika klien mungkin menemukan versi data yang lebih lama.

Misalkan Anda menggunakan timestamp sebagai versinya. Katakanlah formulir pertama kali dibuat pada Wilayah A pada pukul 12.00 tetapi (setelah failover) mencoba menulis ke Wilayah B dan mengetahui bahwa versi terbaru dalam basis data adalah 11.59. Dalam skenario ini, klien dapat menunggu versi 12.00 untuk disebarkan ke Wilayah B lalu menulis di atas versi tersebut, atau membangun pada 11.59 dan membuat versi 12.01 baru (yang, setelah ditulis, akan menyembunyikan versi yang masuk setelah Wilayah A pulih).

Contoh terakhir, perusahaan jasa keuangan menyimpan data tentang akun pelanggan dan transaksi keuangan mereka dalam basis data DynamoDB. Jika terjadi pemadaman total Wilayah A, mereka ingin memastikan bahwa aktivitas tulis apa pun yang terkait dengan akun mereka tersedia sepenuhnya di Wilayah B, atau ingin mengarantina akun mereka sebagai diketahui sebagian hingga Wilayah A kembali online. Alih-alih menghentikan semua bisnis, mereka memutuskan untuk menghentikan bisnis untuk sementara waktu, hanya pada sebagian kecil akun yang mereka anggap memiliki transaksi yang tidak disebarkan. Untuk mencapai hal ini, mereka menggunakan Wilayah ketiga, yang akan kita sebut Wilayah C. Sebelum memproses operasi tulis apa pun di Wilayah A, mereka menempatkan ringkasan singkat operasi yang tertunda tersebut (misalnya, jumlah transaksi baru untuk sebuah akun) di Wilayah C. Ringkasan ini cukup bagi Wilayah B untuk menentukan apakah pandangannya benar-benar mutakhir. Tindakan ini mengunci akun secara efektif sejak penulisan di Wilayah C hingga Wilayah A menerima operasi tulis dan Wilayah B menerimanya. Data

di Wilayah C tidak digunakan kecuali sebagai bagian dari proses failover, setelah itu Wilayah B dapat memeriksa datanya dengan Wilayah C untuk mengetahui apakah ada akun yang kedaluwarsa. Akun tersebut akan ditandai sebagai dikarantina hingga pemulihan Wilayah A menyebarkan sebagian data ke Wilayah B.

Jika Wilayah C gagal, Wilayah D baru dapat dibuka untuk digunakan. Data di Wilayah C sangat sementara, dan setelah beberapa menit Wilayah D akan memiliki up-to-date catatan yang cukup tentang operasi penulisan dalam penerbangan untuk sepenuhnya berguna. Jika Wilayah B gagal, Wilayah A dapat terus menerima permintaan tulis yang bekerja sama dengan Wilayah C. Perusahaan ini bersedia menerima penulisan dengan latensi yang lebih tinggi (ke dua Wilayah: C dan kemudian A) dan beruntung memiliki model data yang status akunnya dapat dirangkum secara ringkas.

Perencanaan kapasitas throughput untuk tabel global DynamoDB

Migrasi lalu lintas dari satu Wilayah ke Wilayah lain memerlukan pertimbangan cermat terhadap pengaturan tabel DynamoDB terkait kapasitas.

Beberapa pertimbangan dalam mengelola kapasitas tulis:

- Tabel global harus berada dalam mode sesuai permintaan atau disediakan dengan penskalaan otomatis yang diaktifkan.
- Jika disediakan dengan penskalaan otomatis, pengaturan tulis (pemanfaatan minimum, maksimum, dan target) direplikasi di seluruh Wilayah. Meskipun pengaturan penskalaan otomatis disinkronkan, kapasitas tulis aktual yang disediakan mungkin mengambang secara independen di antara Wilayah.
- Salah satu alasan Anda mungkin melihat perbedaan kapasitas tulis yang disediakan adalah karena fitur TTL. Saat mengaktifkan TTL di DynamoDB, Anda dapat menentukan nama atribut yang nilainya menunjukkan waktu kedaluwarsa item, dalam format waktu Unix dalam hitungan detik. Setelah itu, DynamoDB dapat menghapus item tanpa menimbulkan biaya tulis. Dengan tabel global, Anda dapat mengonfigurasi TTL di Wilayah mana pun, dan pengaturannya otomatis direplikasi ke Wilayah lain yang terkait dengan tabel global. Ketika suatu item memenuhi syarat untuk penghapusan melalui aturan TTL, pekerjaan tersebut dapat dilakukan di Wilayah mana pun. Operasi penghapusan dilakukan tanpa menggunakan unit tulis pada tabel sumber, tetapi tabel replika akan mendapatkan penulisan yang direplikasi untuk operasi penghapusan tersebut dan akan dikenakan biaya unit tulis yang direplikasi.
- Jika Anda menggunakan penskalaan otomatis, pastikan pengaturan kapasitas tulis maksimum yang disediakan cukup tinggi untuk menangani semua operasi tulis serta semua potensi operasi

penghapusan TTL. Penskalaan otomatis menyesuaikan setiap Wilayah sesuai dengan konsumsi tulisnya. Tabel sesuai permintaan tidak memiliki pengaturan kapasitas tulis maksimum yang disediakan, tetapi batas throughput tulis maksimum tingkat tabel menentukan kapasitas tulis berkelanjutan maksimum yang diperbolehkan oleh tabel sesuai permintaan. Batas defaultnya adalah 40.000, tetapi dapat disesuaikan. Sebaiknya Anda mengaturnya cukup tinggi untuk menangani semua operasi tulis (termasuk operasi tulis TTL) yang mungkin diperlukan tabel sesuai permintaan. Nilai ini harus sama di seluruh Wilayah yang berpartisipasi saat Anda menyiapkan tabel global.

Beberapa pertimbangan dalam mengelola kapasitas baca:

- Pengaturan manajemen kapasitas baca boleh berbeda di antara Wilayah karena diasumsikan bahwa Wilayah yang berbeda mungkin memiliki pola baca yang independen. Saat Anda pertama kali menambahkan replika global ke tabel, kapasitas Wilayah sumber disebarkan. Setelah pembuatan, Anda dapat menyesuaikan pengaturan kapasitas baca, yang tidak ditransfer ke sisi lain.
- Saat Anda menggunakan penskalaan otomatis DynamoDB, pastikan pengaturan kapasitas baca maksimum yang disediakan cukup tinggi untuk menangani semua operasi baca di seluruh Wilayah. Selama operasi standar, kapasitas baca mungkin akan disebarkan di seluruh Wilayah, tetapi tabel selama failover harus mampu beradaptasi secara otomatis dengan peningkatan beban kerja baca. Tabel sesuai permintaan tidak memiliki pengaturan kapasitas baca maksimum yang disediakan, tetapi batas throughput baca maksimum tingkat tabel menentukan kapasitas baca berkelanjutan maksimum yang diperbolehkan oleh tabel sesuai permintaan. Batas defaultnya adalah 40.000, tetapi dapat disesuaikan. Sebaiknya Anda mengaturnya cukup tinggi untuk menangani semua operasi baca yang mungkin diperlukan tabel jika semua operasi baca dirutekan ke Wilayah tunggal ini.
- Jika tabel di satu Wilayah biasanya tidak menerima lalu lintas baca tetapi mungkin harus menyerap sejumlah besar lalu lintas baca setelah failover, Anda dapat meningkatkan kapasitas baca tabel yang disediakan, menunggu tabel selesai diperbarui, lalu menyediakan tabelnya lagi. Anda dapat membiarkan tabel dalam mode yang disediakan atau mengalihkannya ke mode sesuai permintaan. Ini akan menghangatkan tabel untuk menerima tingkat lalu lintas baca yang lebih tinggi.

ARC memiliki pemeriksaan kesiapan yang dapat berguna untuk mengonfirmasi bahwa DynamoDB Regions memiliki pengaturan tabel dan kuota akun yang serupa, apakah Anda menggunakan Route 53 untuk merutekan permintaan atau tidak. Pemeriksaan kesiapan ini juga dapat membantu menyesuaikan kuota tingkat akun untuk memastikannya cocok.

Daftar periksa persiapan untuk tabel global DynamoDB dan Pertanyaan yang Sering Diajukan

Gunakan daftar periksa berikut untuk keputusan dan tugas saat Anda men-deploy tabel global.

- Tentukan jumlah dan Wilayah yang akan berpartisipasi dalam tabel global.
- Tentukan mode tulis aplikasi Anda. Untuk informasi selengkapnya, lihat Tulis mode dengan tabel global DynamoDB.
- Rencanakan strategi Minta perutean dengan tabel global DynamoDB, berdasarkan mode tulis Anda.
- Tentukan rencana

Evakuasi suatu Wilayah adalah proses memigrasikan aktivitas baca dan tulis keluar dari Wilayah tersebut. Ini paling sering merupakan aktivitas tulis, dan terkadang aktivitas baca.

Mengevakuasi Wilayah aktif

Anda dapat memutuskan untuk mengevakuasi Wilayah aktif karena sejumlah alasan. Evakuasi dapat menjadi bagian dari aktivitas bisnis biasa seperti jika Anda menggunakan mode follow-the-sun tulis ke satu Wilayah. Evakuasi juga bisa disebabkan oleh keputusan bisnis untuk mengubah Wilayah yang sedang aktif, sebagai respons terhadap kegagalan tumpukan perangkat lunak di luar DynamoDB, atau karena Anda menghadapi masalah umum seperti latensi yang lebih tinggi dari biasanya di dalam Wilayah.

Dengan mode tulis ke Wilayah mana pun, mengevakuasi Wilayah aktif sangatlah mudah. Anda dapat merutekan lalu lintas ke Wilayah alternatif melalui sistem perutean apa pun, dan membiarkan operasi tulis yang telah terjadi di Wilayah yang dievakuasi direplikasi seperti biasa.

Dengan mode tulis ke satu Wilayah dan tulis ke Wilayah Anda, Anda harus memastikan semua penulisan ke Wilayah aktif telah direkam sepenuhnya, diproses streaming, dan disebarkan secara global sebelum memulai penulisan di Wilayah aktif yang baru. Hal ini diperlukan untuk memastikan bahwa penulisan di masa mendatang bertentangan dengan versi data terbaru.

Katakanlah Wilayah A aktif dan Wilayah B pasif (baik untuk tabel lengkap atau untuk item yang ditempatkan di Wilayah A). Mekanisme umum untuk melakukan evakuasi adalah dengan menjeda operasi tulis ke A, menunggu cukup lama hingga operasi tersebut disebarkan sepenuhnya ke B, memperbarui tumpukan arsitektur untuk mengenali B sebagai aktif, lalu

melanjutkan operasi tulis ke B. Tidak ada metrik yang menunjukkan kepastian mutlak bahwa Wilayah A telah sepenuhnya mereplikasi datanya ke Wilayah B. Jika Wilayah A sehat, menjeda operasi tulis ke Wilayah A dan menunggu 10 kali nilai maksimum terbaru metrik `ReplicationLatency` biasanya sudah cukup untuk menentukan bahwa replikasi tersebut selesai. Jika Wilayah A tidak sehat dan menunjukkan area lain mengalami peningkatan latensi, Anda dapat memilih kelipatan waktu tunggu yang lebih besar.

Mengevakuasi Wilayah offline

Ada kasus khusus yang perlu dipertimbangkan: bagaimana jika Wilayah A offline sepenuhnya tanpa pemberitahuan? Hal ini sangat kecil kemungkinannya, tetapi tetap perlu dipertimbangkan. Jika hal ini terjadi, operasi tulis apa pun di Wilayah A yang belum disebarkan akan ditahan dan disebarkan setelah Wilayah A kembali online. Operasi tulis tidak hilang, tetapi penyebarannya tertunda tanpa batas waktu.

Cara melanjutkan peristiwa ini merupakan keputusan aplikasi. Untuk kelangsungan bisnis, operasi tulis mungkin perlu diteruskan ke Wilayah B utama yang baru. Namun, jika item di Wilayah B menerima pembaruan sementara ada penyebaran operasi tulis yang tertunda untuk item tersebut dari Wilayah A, penyebaran tersebut akan disembunyikan di model penulis terakhir menang. Pembaruan apa pun di Wilayah B mungkin menyembunyikan permintaan tulis yang masuk.

Dengan mode tulis ke Wilayah mana pun, pembacaan dan penulisan dapat dilanjutkan di Wilayah B, dengan keyakinan bahwa item di Wilayah A pada akhirnya akan disebarkan ke Wilayah B dan mengenali potensi item yang hilang hingga Wilayah A kembali online. Jika memungkinkan, Anda harus mempertimbangkan untuk memutar ulang lalu lintas tulis terbaru (misalnya, menggunakan sumber peristiwa hulu) untuk mengisi celah dari operasi tulis yang mungkin hilang dan membiarkan resolusi konflik penulis terakhir menang menekan penyebaran akhir dari operasi tulis yang masuk.

Dengan mode tulis lainnya, Anda harus mempertimbangkan sejauh mana pekerjaan dapat dilanjutkan dengan sedikit out-of-date pandangan dunia. Beberapa operasi tulis berdurasi pendek, seperti yang dilacak oleh `ReplicationLatency`, akan hilang hingga Wilayah A kembali online. Apakah bisnis bisa maju? Dalam beberapa kasus penggunaan, bisa, tetapi pada kasus lainnya mungkin tidak bisa, tanpa mekanisme mitigasi tambahan.

Misalnya, katakanlah Anda perlu mempertahankan saldo kredit yang tersedia tanpa gangguan bahkan setelah kegagalan Wilayah. Anda dapat membagi saldo menjadi dua item berbeda,

satu ditempatkan di Wilayah A dan satu lagi di Wilayah B, masing-masing dimulai dengan setengah saldo yang tersedia. Ini akan menggunakan mode tulis ke Wilayah Anda. Pembaruan transaksional yang diproses di setiap Wilayah akan dituliskan pada salinan lokal saldo. Jika Wilayah A sepenuhnya offline, pekerjaan masih dapat dilanjutkan dengan pemrosesan transaksi di Wilayah B, dan operasi tulis akan terbatas pada bagian saldo yang disimpan di Wilayah B. Memisahkan saldo seperti ini menimbulkan kerumitan ketika saldo hampir habis atau kredit harus diseimbangkan kembali, tetapi hal ini memberikan satu contoh pemulihan bisnis yang aman bahkan dengan operasi tulis tertunda yang tidak pasti.

Contoh lain, misalkan Anda mengambil data formulir web. Anda dapat menggunakan Optimistic Concurrency Control (OCC) untuk menetapkan versi ke item data dan menyematkan versi terbaru ke formulir web sebagai bidang tersembunyi. Pada setiap pengiriman, operasi tulis berhasil hanya jika versi dalam basis data masih cocok dengan versi yang digunakan untuk membuat formulir. Jika versinya tidak cocok, formulir web bisa disegarkan (atau digabungkan secara hati-hati) berdasarkan versi saat ini dalam basis data, dan pengguna bisa melanjutkan lagi. Model OCC biasanya melindungi terhadap penimpaan klien lain dan pembuatan versi data yang baru, tetapi model ini juga dapat membantu selama failover ketika klien mungkin menemukan versi data yang lebih lama.

Misalkan Anda menggunakan timestamp sebagai versinya. Katakanlah formulir pertama kali dibuat pada Wilayah A pada pukul 12.00 tetapi (setelah failover) mencoba menulis ke Wilayah B dan mengetahui bahwa versi terbaru dalam basis data adalah 11.59. Dalam skenario ini, klien dapat menunggu versi 12.00 untuk disebarkan ke Wilayah B lalu menulis di atas versi tersebut, atau membangun pada 11.59 dan membuat versi 12.01 baru (yang, setelah ditulis, akan menyembunyikan versi yang masuk setelah Wilayah A pulih).

Contoh terakhir, perusahaan jasa keuangan menyimpan data tentang akun pelanggan dan transaksi keuangan mereka dalam basis data DynamoDB. Jika terjadi pemadaman total Wilayah A, mereka ingin memastikan bahwa aktivitas tulis apa pun yang terkait dengan akun mereka tersedia sepenuhnya di Wilayah B, atau ingin mengarantina akun mereka sebagai diketahui sebagian hingga Wilayah A kembali online. Alih-alih menghentikan semua bisnis, mereka memutuskan untuk menghentikan bisnis untuk sementara waktu, hanya pada sebagian kecil akun yang mereka anggap memiliki transaksi yang tidak disebarkan. Untuk mencapai hal ini, mereka menggunakan Wilayah ketiga, yang akan kita sebut Wilayah C. Sebelum memproses operasi tulis apa pun di Wilayah A, mereka menempatkan ringkasan singkat operasi yang tertunda tersebut (misalnya, jumlah transaksi baru untuk sebuah akun) di Wilayah C. Ringkasan ini cukup bagi Wilayah B untuk menentukan apakah pandangannya benar-benar mutakhir.

Tindakan ini mengunci akun secara efektif sejak penulisan di Wilayah C hingga Wilayah A menerima operasi tulis dan Wilayah B menerimanya. Data di Wilayah C tidak digunakan kecuali sebagai bagian dari proses failover, setelah itu Wilayah B dapat memeriksa datanya dengan Wilayah C untuk mengetahui apakah ada akun yang kedaluwarsa. Akun tersebut akan ditandai sebagai dikarantina hingga pemulihan Wilayah A menyebarkan sebagian data ke Wilayah B.
Jika Wilayah C gagal, Wilayah D baru dapat dibuka untuk digunakan. Data di Wilayah C sangat sementara, dan setelah beberapa menit Wilayah D akan memiliki up-to-date catatan yang cukup tentang operasi penulisan dalam penerbangan untuk sepenuhnya berguna. Jika Wilayah B gagal, Wilayah A dapat terus menerima permintaan tulis yang bekerja sama dengan Wilayah C. Perusahaan ini bersedia menerima penulisan dengan latensi yang lebih tinggi (ke dua Wilayah: C dan kemudian A) dan beruntung memiliki model data yang status akunnya dapat dirangkum secara ringkas.

evakuasi Anda, berdasarkan mode tulis dan strategi perutean Anda.

- Tangkap metrik tentang kesehatan, latensi, dan kesalahan di setiap Wilayah. Untuk daftar metrik DynamoDB, lihat AWS posting blog Memantau Amazon DynamoDB untuk Kesadaran Operasional untuk daftar metrik yang harus diamati. Anda juga harus menggunakan synthetic canary (permintaan buatan yang dirancang untuk mendeteksi kegagalan, dinamai menurut nama kenari di tambang batu bara), serta pengamatan langsung terhadap lalu lintas pelanggan. Tidak semua masalah akan muncul di metrik DynamoDB.
- Atur alarm untuk peningkatan `ReplicationLatency` yang berkelanjutan. Peningkatan mungkin menunjukkan kesalahan konfigurasi yang tidak disengaja, yaitu tabel global memiliki pengaturan tulis berbeda di Wilayah berbeda, yang menyebabkan kegagalan permintaan yang direplikasi dan peningkatan latensi. Hal ini juga dapat mengindikasikan adanya gangguan regional. Contoh yang baik adalah menghasilkan peringatan jika rata-rata terkini melebihi 180.000 milidetik. Anda mungkin juga memperhatikan `ReplicationLatency` penurunan ke 0, yang menunjukkan replikasi terhenti.
- Tetapkan pengaturan baca dan tulis maksimum yang memadai untuk setiap tabel global.
- Identifikasi terlebih dahulu alasan untuk mengevakuasi suatu Wilayah. Jika keputusan melibatkan penilaian manusia, dokumentasikan semua pertimbangannya. Pekerjaan ini harus dilakukan dengan hati-hati sebelumnya, bukan di bawah tekanan.
- Simpan runbook untuk setiap tindakan yang harus dilakukan saat Anda mengevakuasi suatu Wilayah. Biasanya pekerjaan yang dilakukan untuk tabel global sangat sedikit, tetapi memindahkan sisa tumpukan dapat menjadi pekerjaan yang rumit.

> **Note**
>
> Praktik terbaiknya adalah hanya mengandalkan operasi bidang data, bukan operasi bidang kontrol karena beberapa operasi bidang kontrol mungkin terdegradasi selama kegagalan wilayah.

Untuk informasi selengkapnya, lihat posting AWS blog Membangun aplikasi tangguh dengan tabel global Amazon DynamoDB: Bagian 4.

- Uji semua aspek runbook secara berkala, termasuk evakuasi Wilayah. Runbook yang belum teruji adalah runbook yang tidak dapat diandalkan.
- Pertimbangkan untuk menggunakan Resilience Hub untuk mengevaluasi ketahanan seluruh aplikasi Anda (termasuk tabel global). Resilience Hub memberikan pandangan komprehensif tentang status ketahanan portofolio aplikasi Anda secara keseluruhan melalui dasbornya.
- Pertimbangkan untuk menggunakan pemeriksaan kesiapan ARC untuk mengevaluasi konfigurasi aplikasi Anda saat ini dan melacak penyimpangan dari praktik terbaik.
- Saat menulis pemeriksaan kondisi untuk digunakan dengan Route 53 atau Global Accelerator, tidak cukup hanya dengan melakukan ping untuk mengetahui bahwa titik akhir DynamoDB sudah aktif. Ping tidak mencakup banyak mode kegagalan seperti kesalahan konfigurasi IAM, masalah deployment kode, kegagalan dalam tumpukan di luar DynamoDB, latensi baca atau tulis yang lebih tinggi dari rata-rata, dan sebagainya. Praktik terbaiknya adalah melakukan serangkaian panggilan yang menggunakan aliran basis data penuh.

Pertanyaan yang Sering Diajukan (FAQ) untuk men-deploy tabel global

Apa saja prinsip yang berguna untuk penggunaan keseluruhan tabel global DynamoDB?

Tabel global DynamoDB memiliki sedikit tombol kontrol, tetapi masih memerlukan sejumlah pertimbangan. Anda harus menentukan mode tulis, model perutean, dan proses evakuasi. Anda harus melengkapi aplikasi Anda di setiap Wilayah dan siap menyesuaikan rute atau melakukan evakuasi untuk menjaga kesehatan global. Sehingga Anda akan mendapatkan kumpulan data yang terdistribusi secara global dengan pembacaan dan penulisan latensi rendah serta perjanjian tingkat layanan 99,999%.

Berapa harga untuk tabel global?

Tulis ke tabel DynamoDB tradisional diberi harga dalam Unit Kapasitas Tulis WCUs (, untuk tabel yang disediakan) atau Unit Permintaan Tulis (, untuk tabel sesuai permintaan). WRUs Jika menulis item 5 KB, Anda dikenakan biaya 5 unit. Tulis ke tabel global diberi harga dalam Unit Kapasitas Tulis Replikasi (rWCUs, untuk tabel yang disediakan) atau Unit Permintaan Tulis Replikasi (rWRUs, untuk tabel sesuai permintaan).

R WCUs dan r WRUs termasuk biaya infrastruktur streaming yang diperlukan untuk mengelola replikasi.

Biaya Unit Tulis yang Direplikasi dikenakan di setiap Wilayah tempat item tersebut ditulis langsung atau ditulis dengan replikasi.

Menulis ke Indeks Sekunder Global (GSI) dianggap sebagai penulisan lokal dan menggunakan Unit Tulis biasa.

Tidak ada Kapasitas Cadangan yang tersedia untuk r WCUs saat ini. Membeli Kapasitas Cadangan mungkin masih bermanfaat untuk tabel dengan unit tulis yang GSIs dikonsumsi.

Bootstrap awal saat menambahkan Wilayah baru ke tabel global dikenakan biaya seperti pemulihan per GB data yang dipulihkan, ditambah biaya transfer data lintas Wilayah.

Wilayah mana yang didukung tabel global?

Tabel Global versi 2019.11.21 (Saat Ini) tersedia di semua Wilayah.

Bagaimana GSIs ditangani dengan tabel global?

Di Tabel Global versi 2019.11.21 (Saat Ini), saat Anda membuat GSI di satu Wilayah, GSI secara otomatis dibuat di Wilayah lain yang berpartisipasi dan diisi ulang secara otomatis.

Bagaimana cara menghentikan replikasi tabel global?

Anda dapat menghapus tabel replika seperti Anda menghapus tabel lainnya. Menghapus tabel global akan menghentikan replikasi ke Wilayah tersebut dan menghapus salinan tabel yang disimpan di Wilayah tersebut. Namun, Anda tidak dapat menghentikan replikasi sambil menyimpan salinan tabel sebagai entitas independen, dan Anda juga tidak dapat menjeda replikasi.

Bagaimana DynamoDB Streams berinteraksi dengan tabel global?

Setiap tabel global menghasilkan aliran independen berdasarkan semua penulisannya, dari mana pun tabel tersebut dimulai. Anda dapat menggunakan aliran DynamoDB di satu Wilayah atau di semua Wilayah (secara independen). Jika ingin memproses operasi tulis lokal tetapi tidak direplikasi,

Anda dapat menambahkan atribut Wilayah Anda sendiri ke setiap item untuk mengidentifikasi Wilayah penulisan. Anda kemudian dapat menggunakan filter peristiwa Lambda untuk memanggil fungsi Lambda hanya untuk operasi tulis di Wilayah lokal. Ini akan membantu dalam operasi penyisipan dan pembaruan, tetapi sayangnya tidak membantu operasi penghapusan.

Bagaimana tabel global menangani transaksi?

Operasi transaksional memberikan jaminan atomisitas, konsistensi, isolasi, dan daya tahan (ACID) hanya di Wilayah tempat penulisan dilakukan pada awalnya. Transaksi tidak didukung di seluruh Wilayah dalam tabel global. Misalnya, jika Anda memiliki tabel global dengan replika di Wilayah AS Timur (Ohio) dan AS Barat (Oregon) serta melakukan operasi `TransactWriteItems` di Wilayah AS Timur (Ohio), Anda mungkin melihat transaksi yang selesai sebagian di Wilayah AS Barat (Oregon) seiring perubahan direplikasi. Perubahan direplikasi ke Wilayah lain hanya setelah diterapkan di Wilayah sumber.

Bagaimana tabel global berinteraksi dengan cache DynamoDB Accelerator (DAX)?

Tabel global melewati DAX dengan memperbarui DynamoDB secara langsung, sehingga DAX tidak mengetahui jika menyimpan data yang sudah usang. Cache DAX disegarkan hanya ketika TTL cache kedaluwarsa.

Apakah tanda pada tabel disebarkan?

Tidak, tanda tidak disebarkan secara otomatis.

Apakah saya harus mencadangkan tabel di semua Wilayah atau cukup satu Wilayah?

Jawabannya tergantung pada tujuan pencadangan. Jika Anda ingin memastikan ketahanan data, DynamoDB sudah menyediakan perlindungan itu. Layanan ini memastikan ketahanan. Jika Anda ingin menyimpan snapshot untuk catatan historis (misalnya, untuk memenuhi persyaratan peraturan), membuat cadangan di satu Wilayah sudah cukup. Anda dapat menyalin cadangan ke Wilayah tambahan dengan menggunakan AWS Backup. Jika Anda ingin memulihkan data yang dihapus atau dimodifikasi secara keliru, gunakan DynamoDB point-in-time recovery (PITR) di satu Wilayah.

Bagaimana cara menerapkan tabel global menggunakan? AWS CloudFormation

CloudFormation merupakan tabel DynamoDB dan tabel global sebagai dua sumber daya terpisah: dan. `AWS::DynamoDB::Table` `AWS::DynamoDB::GlobalTable` Salah satu pendekatannya adalah membuat semua tabel yang berpotensi bersifat global menggunakan konsep `GlobalTable`. Anda kemudian dapat menyimpannya sebagai tabel mandiri pada awalnya, dan menambahkan Wilayah nanti jika diperlukan.

Dalam CloudFormation, setiap tabel global dikendalikan oleh satu tumpukan, dalam satu Wilayah, terlepas dari jumlah replika. Saat Anda menerapkan template Anda, CloudFormation membuat dan memperbarui semua replika sebagai bagian dari operasi tumpukan tunggal. Jangan men-deploy sumber daya AWS::DynamoDB::GlobalTable yang sama di beberapa Wilayah. Tindakan tersebut tidak didukung dan akan mengakibatkan kesalahan. Jika men-deploy templat aplikasi di beberapa Wilayah, Anda dapat menggunakan ketentuan untuk membuat sumber daya `AWS::DynamoDB::GlobalTable` di satu Wilayah. Alternatifnya, Anda dapat memilih untuk menentukan sumber daya `AWS::DynamoDB::GlobalTable` Anda dalam tumpukan yang terpisah dari tumpukan aplikasi Anda, dan memastikan bahwa sumber daya tersebut disebarkan ke satu Wilayah.

Jika Anda memiliki tabel reguler dan ingin mengonversinya menjadi tabel global sambil menjaganya agar tetap dikelola CloudFormation kemudian mengatur kebijakan penghapusan ke Retain, hapus tabel dari tumpukan, ubah tabel menjadi tabel global di konsol, lalu impor tabel global sebagai sumber daya baru ke tumpukan.

Replikasi lintas akun tidak didukung saat ini.

Praktik terbaik untuk mengelola bidang kontrol di DynamoDB

> **Note**
>
> DynamoDB memperkenalkan batas throttling bidang kontrol 2.500 permintaan per detik dengan opsi percobaan ulang. Lihat di bawah ini untuk detail tambahan.

Operasi bidang kontrol DynamoDB memungkinkan Anda mengelola tabel DynamoDB serta objek yang bergantung pada tabel seperti indeks. Untuk informasi selengkapnya tentang operasi ini, lihat Bidang kontrol.

Dalam beberapa situasi, Anda mungkin perlu mengambil tindakan dan menggunakan data yang dikembalikan oleh panggilan bidang kontrol sebagai bagian dari logika bisnis Anda. Misalnya, Anda mungkin perlu mengetahui nilai `ProvisionedThroughput` yang dikembalikan oleh `DescribeTable`. Dalam situasi ini, ikuti praktik terbaik berikut:

- Jangan mengkueri bidang kontrol DynamoDB secara berlebihan.
- Jangan menggabungkan panggilan bidang kontrol dan panggilan bidang data dalam kode yang sama.

- Tangani throttling pada permintaan bidang kontrol dan coba lagi dengan backoff.
- Invokasi dan lacak perubahan pada sumber daya tertentu dari satu klien.
- Daripada mengambil data untuk tabel yang sama beberapa kali dalam interval pendek, simpan data dalam cache untuk diproses.

Praktik terbaik untuk menggunakan operasi data massal di DynamoDB

DynamoDB mendukung operasi batch `BatchWriteItem` seperti menggunakan yang Anda dapat melakukan hingga `PutItem` 25 `DeleteItem` dan permintaan bersama-sama. Namun, `BatchWriteItem` tidak mendukung `UpdateItem` operasi. Ketika datang ke pembaruan massal, perbedaannya terletak pada persyaratan dan sifat pembaruan. Anda dapat menggunakan APIs DynamoDB lain `TransactWriteItems` seperti untuk ukuran batch hingga 100. Ketika lebih banyak item terlibat, Anda dapat menggunakan layanan seperti AWS Glue, Amazon EMR, AWS Step Functions atau menggunakan skrip dan alat khusus seperti DynamoDB-Shell untuk pembaruan massal.

Topik

- Pembaruan batch bersyarat
- Operasi massal yang efisien

Pembaruan batch bersyarat

DynamoDB mendukung operasi batch `BatchWriteItem` seperti menggunakan yang dapat Anda lakukan hingga `PutItem` 25 `DeleteItem` dan permintaan dalam satu batch. Namun, `BatchWriteItem` tidak mendukung `UpdateItem` operasi dan tidak mendukung ekspresi kondisi. Sebagai solusinya, Anda dapat menggunakan DynamoDB lain APIs seperti `TransactWriteItems` untuk ukuran batch hingga 100.

Ketika lebih banyak item terlibat, dan sebagian besar data perlu diubah, Anda dapat menggunakan layanan seperti, Amazon EMR AWS Step Functions , atau menggunakan skrip dan alat khusus seperti AWS Glue DynamoDB-Shell untuk pembaruan massal yang efisien.

Kapan menggunakan pola ini

- DynamoDB-shell tidak didukung untuk kasus penggunaan produksi.

- `TransactWriteItems`— hingga 100 pembaruan individu dengan atau tanpa kondisi, dijalankan sebagai bundel ACID semua atau tidak sama sekali. `TransactWriteItems`panggilan juga dapat diberikan dengan `ClientRequestToken` jika aplikasi Anda memerlukan idempotensi, yang berarti beberapa panggilan identik memiliki efek yang sama dengan satu panggilan tunggal. Ini memastikan Anda tidak melakukan transaksi yang sama beberapa kali dan berakhir dengan status data yang salah.

  Trade-off - Throughput tambahan dikonsumsi. 2 WCUs per 1KB tulis alih-alih standar 1 WGU per 1 KB tulis.
- `BatchExecuteStatement`PartiQL — hingga 25 pembaruan dengan atau tanpa kondisi. `BatchExecuteStatement`selalu mengembalikan respons sukses terhadap permintaan keseluruhan, dan juga mengembalikan daftar respons operasi individu yang menjaga ketertiban.

  Trade-off — Untuk batch yang lebih besar, logika sisi klien tambahan diperlukan untuk mendistribusikan permintaan dalam batch 25. Respons kesalahan individu perlu dipertimbangkan untuk menentukan strategi coba lagi.

Contoh kode

Contoh kode ini menggunakan pustaka boto3, yang merupakan AWS SDK untuk Python. Contoh mengasumsikan Anda telah menginstal boto3 dan dikonfigurasi dengan kredenal yang sesuai AWS .

Asumsikan database inventaris untuk vendor alat listrik yang memiliki beberapa gudang di kota-kota Eropa. Karena ini adalah akhir musim panas, vendor ingin membersihkan kipas meja untuk memberi ruang bagi stok lainnya. Vendor ingin memberikan diskon harga untuk semua penggemar meja yang dipasok dari gudang di Italia tetapi hanya jika mereka memiliki stok cadangan 20 kipas meja. Tabel DynamoDB disebut inventaris, ia memiliki skema kunci sku kunci partisi yang merupakan pengidentifikasi unik untuk setiap produk dan gudang kunci Sort yang merupakan pengidentifikasi untuk gudang.

Kode Python berikut menunjukkan cara melakukan pembaruan batch bersyarat ini menggunakan panggilan API. `BatchExecuteStatement`

```
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'])
```

Eksekusi menghasilkan output di bawah ini pada data sampel:

```
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'}}]
```

Karena ini adalah operasi terbatas untuk sistem internal, persyaratan idempotensi belum dipertimbangkan. Dimungkinkan untuk menempatkan pagar tambahan seperti pembaruan harga harus melalui hanya jika harga lebih besar dari 35 dan kurang dari 40 untuk membuat pembaruan lebih kuat.

Atau, kami dapat melakukan operasi pembaruan batch yang sama menggunakan `TransactWriteItems` jika persyaratan idempotensi dan ACID yang lebih ketat. Namun, penting untuk diingat bahwa semua operasi dalam bundel transaksi berjalan atau seluruh bundel gagal.

Mari kita asumsikan kasus di mana ada gelombang panas di Italia dan permintaan untuk penggemar meja telah meningkat tajam. Vendor ingin meningkatkan biaya kipas meja mereka keluar dari setiap gudang di Italia sebesar 20 Euro tetapi badan pengawas hanya mengizinkan kenaikan biaya ini jika biaya saat ini kurang dari 70 Euro di seluruh inventaris mereka. Sangat penting bahwa harga diperbarui di seluruh inventaris sekaligus dan hanya sekali dan hanya jika biayanya kurang dari 70 Euro di masing-masing gudang mereka.

Kode Python berikut menunjukkan cara melakukan pembaruan batch ini menggunakan `TransactWriteItems` panggilan API.

```
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'])
```

Eksekusi menghasilkan output di bawah ini pada data sampel:

```
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'}}]
```

Ada beberapa pendekatan untuk melakukan pembaruan batch di DynamoDB. Pendekatan yang sesuai tergantung pada faktor-faktor seperti persyaratan and/or idempotensi ACID, jumlah item yang akan diperbarui, dan keakraban. APIs

Operasi massal yang efisien

Kapan menggunakan pola ini

Pola-pola ini berguna untuk secara efisien melakukan pembaruan massal pada item DynamoDB.

- DynamoDB-shell tidak didukung untuk kasus penggunaan produksi.
- `TransactWriteItems`— hingga 100 pembaruan individu dengan atau tanpa kondisi, dijalankan sebagai bundel ACID semua atau tidak sama sekali

  Trade-off — Throughput tambahan dikonsumsi, 2 WCUs per 1 KB tulis.
- `BatchExecuteStatement`PartiQL - hingga 25 pembaruan dengan atau tanpa kondisi

Trade-off — Logika tambahan diperlukan untuk mendistribusikan permintaan dalam batch 25.

- AWS Step Functions — operasi massal terbatas tarif untuk pengembang yang akrab dengannya. AWS Lambda

  Trade-off — Waktu eksekusi berbanding terbalik dengan rate-limit. Dibatasi oleh batas waktu fungsi Lambda maksimum. Fungsionalitas mensyaratkan bahwa perubahan data yang terjadi antara baca dan tulis dapat ditimpa. Untuk info selengkapnya, lihat Mengisi ulang atribut Amazon DynamoDB Time to Live menggunakan Amazon EMR: Bagian 2.
- AWS Glue dan Amazon EMR — operasi massal terbatas tarif dengan paralelisme terkelola. Untuk aplikasi atau pembaruan yang tidak sensitif terhadap waktu, opsi ini dapat berjalan di latar belakang hanya menghabiskan sebagian kecil throughput. Kedua layanan menggunakan emr-dynamodb-connector untuk melakukan operasi DynamoDB. Layanan ini melakukan pembacaan besar diikuti dengan penulisan besar item yang diperbarui dengan opsi untuk membatasi nilai.

  Trade-off — Waktu eksekusi berbanding terbalik dengan rate-limit. Fungsionalitas mencakup bahwa perubahan data yang terjadi antara baca dan tulis dapat ditimpa. Anda tidak dapat membaca dari Global Secondary Indexes (GSIs). Lihat, Mengisi ulang atribut Amazon DynamoDB Time to Live menggunakan Amazon EMR: Bagian 2.
- DynamoDB Shell — operasi massal terbatas kecepatan menggunakan kueri mirip SQL. Anda dapat membaca dari GSIs untuk efisiensi yang lebih baik.

  Trade-off — Waktu eksekusi berbanding terbalik dengan rate-limit. Lihat Menilai operasi massal terbatas di DynamoDB Shell.

Menggunakan pola

Pembaruan massal dapat memiliki implikasi biaya yang signifikan terutama jika Anda menggunakan mode throughput sesuai permintaan. Ada trade-off antara kecepatan dan biaya jika Anda menggunakan mode throughput yang disediakan. Mengatur parameter batas laju dengan sangat ketat dapat menyebabkan waktu pemrosesan yang sangat besar. Anda dapat secara kasar menentukan kecepatan pembaruan menggunakan ukuran item rata-rata dan batas tarif.

Atau, Anda dapat menentukan jumlah throughput yang diperlukan untuk proses berdasarkan durasi yang diharapkan dari proses pembaruan dan ukuran item rata-rata. Referensi blog yang dibagikan dengan setiap pola memberikan rincian tentang strategi, implementasi, dan batasan penggunaan pola. Untuk informasi selengkapnya, lihat Pemrosesan massal hemat biaya dengan Amazon DynamoDB.

Ada beberapa pendekatan untuk melakukan pembaruan massal terhadap tabel DynamoDB langsung. Pendekatan yang sesuai tergantung pada faktor-faktor seperti persyaratan and/or idempotensi ACID, jumlah item yang akan diperbarui dan keakraban. APIs Penting untuk mempertimbangkan biaya versus waktu trade-off, sebagian besar pendekatan yang dibahas di atas memberikan opsi untuk membatasi tingkat throughput yang digunakan oleh pekerjaan pembaruan massal.

Praktik terbaik untuk menerapkan kontrol versi di DynamoDB

Dalam sistem terdistribusi seperti DynamoDB, kontrol versi item menggunakan penguncian optimis mencegah pembaruan yang bertentangan. Dengan melacak versi item dan menggunakan penulisan bersyarat, aplikasi dapat mengelola modifikasi bersamaan, memastikan integritas data di seluruh lingkungan konkurensi tinggi.

Optimistic locking adalah strategi yang digunakan untuk memastikan bahwa modifikasi data diterapkan dengan benar tanpa konflik. Alih-alih mengunci data saat dibaca (seperti dalam penguncian pesimis), penguncian optimis memeriksa apakah data telah berubah sebelum menulisnya kembali. Di DynamoDB, ini dicapai melalui bentuk kontrol versi, di mana setiap item menyertakan pengidentifikasi yang bertambah dengan setiap pembaruan. Saat memperbarui item, operasi hanya akan berhasil jika pengidentifikasi itu cocok dengan yang diharapkan oleh aplikasi Anda.

Kapan menggunakan pola ini

Pola ini berguna dalam skenario berikut:

- Beberapa pengguna atau proses dapat mencoba memperbarui item yang sama secara bersamaan.
- Memastikan integritas dan konsistensi data adalah yang terpenting.
- Ada kebutuhan untuk menghindari overhead dan kompleksitas mengelola kunci terdistribusi.

Contohnya termasuk:

- Aplikasi e-commerce di mana tingkat inventaris sering diperbarui.
- Platform kolaboratif tempat beberapa pengguna mengedit data yang sama.
- Sistem keuangan di mana catatan transaksi harus tetap konsisten.

Pengorbanan

Sementara penguncian optimis dan pemeriksaan bersyarat memberikan integritas data yang kuat, mereka datang dengan pengorbanan berikut:

Konflik konkurensi

Dalam lingkungan konkurensi tinggi, kemungkinan konflik meningkat, berpotensi menyebabkan percobaan ulang dan biaya penulisan yang lebih tinggi.

Kompleksitas implementasi

Menambahkan kontrol versi ke item dan menangani pemeriksaan bersyarat dapat menambah kompleksitas logika aplikasi.

Overhead penyimpanan tambahan

Menyimpan nomor versi untuk setiap item sedikit meningkatkan persyaratan penyimpanan.

Desain pola

Untuk mengimplementasikan pola ini, skema DynamoDB harus menyertakan atribut version untuk setiap item. Berikut adalah desain skema sederhana:

- Kunci partisi - Pengidentifikasi unik untuk setiap item (mis. `ItemId`).
- Atribut:
  - `ItemId`— Pengidentifikasi unik untuk item tersebut.
  - `Version`— Bilangan bulat yang mewakili nomor versi item.
  - `QuantityLeft`— Inventaris item yang tersisa.

Ketika item pertama kali dibuat, `Version` atribut diatur ke 1. Dengan setiap pembaruan, nomor versi bertambah 1.

VersionControl

Primary key Partition key: ItemID	Attributes	
Bananas	Version	QuantityLeft
	1	10
Apples	Version	QuantityLeft
	1	5
Oranges	Version	QuantityLeft
	1	7

Menggunakan pola

Untuk menerapkan pola ini, ikuti langkah-langkah berikut dalam alur aplikasi Anda:

1. Baca versi item saat ini.

   Ambil item saat ini dari DynamoDB dan baca nomor versinya.

   ```
   def get_document(item_id):
       response = table.get_item(Key={'ItemID': item_id})
       return response['Item']

   document = get_document('Bananas')
   current_version = document['Version']
   ```

2. Tingkatkan nomor versi dalam logika aplikasi Anda. Ini akan menjadi versi yang diharapkan untuk pembaruan.

   ```
   new_version = current_version + 1
   ```

3. Mencoba memperbarui item menggunakan ekspresi bersyarat untuk memastikan nomor versi cocok.

   ```
   def update_document(item_id, qty_bought, current_version):
       try:
           response = table.update_item(
               Key={'ItemID': item_id},
               UpdateExpression="set #qty = :qty, Version = :v",
   ```

```
            ConditionExpression="Version = :expected_v",
            ExpressionAttributeNames={
                '#qty': 'QuantityLeft'
            },
            ExpressionAttributeValues={
                ':qty': qty_bought,
                ':v': current_version + 1,
                ':expected_v': current_version
            },
            ReturnValues="UPDATED_NEW"
        )
        return response
    except ClientError as e:
        if e.response['Error']['Code'] == 'ConditionalCheckFailedException':
            print("Update failed due to version conflict.")
        else:
            print("Unexpected error: %s" % e)
        return None

update_document('Bananas', 2, new_version)
```

Jika pembaruan berhasil, item QuantityLeft untuk akan dikurangi 2.

VersionControl

Primary key Partition key: ItemID	Attributes	
Bananas	Version	QuantityLeft
	2	8
Apples	Version	QuantityLeft
	1	5
Oranges	Version	QuantityLeft
	1	7

4. Menangani konflik jika terjadi.

   Jika terjadi konflik (misalnya, proses lain telah memperbarui item sejak Anda terakhir membacanya), tangani konflik dengan tepat, seperti dengan mencoba kembali operasi atau memperingatkan pengguna.

Ini akan membutuhkan pembacaan tambahan item untuk setiap percobaan ulang, jadi batasi jumlah total percobaan ulang yang Anda izinkan sebelum benar-benar gagal dalam loop permintaan.

```
def update_document_with_retry(item_id, new_data, retries=3):
    for attempt in range(retries):
        document = get_document(item_id)
        current_version = document['Version']

        result = update_document(item_id, qty_bought, current_version)

        if result is not None:
            print("Update succeeded.")
            return result
        else:
            print(f"Retrying update... ({attempt + 1}/{retries})")

    print("Update failed after maximum retries.")
    return None

update_document_with_retry('Bananas', 2)
```

Menerapkan kontrol versi item menggunakan DynamoDB dengan penguncian optimis dan pemeriksaan bersyarat adalah pola yang kuat untuk memastikan integritas data dalam aplikasi terdistribusi. Meskipun memperkenalkan beberapa kompleksitas dan potensi pengorbanan kinerja, itu sangat berharga dalam skenario yang membutuhkan kontrol konkurensi yang kuat. Dengan merancang skema secara hati-hati dan menerapkan pemeriksaan yang diperlukan dalam logika aplikasi Anda, Anda dapat secara efektif mengelola pembaruan bersamaan dan menjaga konsistensi data.

Panduan dan strategi tambahan untuk cara menerapkan kontrol versi data DynamoDB Anda dapat ditemukan di AWS Blog Database.

Praktik terbaik untuk memahami laporan AWS penagihan dan penggunaan Anda di DynamoDB

Dokumen ini menjelaskan kode `UsageType` penagihan untuk biaya yang terkait dengan DynamoDB.

AWS menyediakan laporan biaya dan penggunaan (CUR) yang berisi data untuk layanan yang digunakan. Anda dapat menggunakan AWS Cost and Usage Report untuk mempublikasikan laporan penagihan ke Amazon S3 dalam format CSV. Saat menyiapkan CUR, Anda dapat memilih untuk memecah periode waktu berdasarkan jam, hari, atau bulan, dan Anda dapat memilih apakah Anda ingin memecah penggunaan berdasarkan ID sumber daya atau tidak. Untuk detail lebih lanjut tentang menghasilkan CUR, silakan lihat Membuat Laporan Biaya dan Penggunaan

Dalam ekspor CSV, Anda akan menemukan atribut yang relevan terdaftar untuk setiap baris. Berikut ini adalah contoh atribut yang dapat disertakan:

- lineitem/UsageStartDate: Tanggal dan waktu mulai untuk item baris di UTC, inklusif.
- lineitem/UsageEndDate: Tanggal dan waktu akhir untuk item baris yang sesuai di UTC, eksklusif.
- lineitem/: ProductCode Untuk DynamoDB ini akan menjadi “DB” AmazonDynamo
- lineitem/UsageType: Kode deskripsi khusus untuk jenis penggunaan, sebagaimana disebutkan dalam dokumen ini
- LineItem/Operation: Nama yang memberikan konteks muatan seperti nama operasi yang dikenakan biaya (opsional).
- lineitem/ResourceId: Pengidentifikasi untuk sumber daya yang dikeluarkan penggunaan. Tersedia jika CUR menyertakan rincian berdasarkan ID sumber daya.
- lineitem/UsageAmount: Jumlah penggunaan yang terjadi selama periode waktu yang ditentukan.
- lineitem/UnblendedCost: Biaya penggunaan ini.
- lineitem/LineItemDescription: Deskripsi tekstual dari item baris.

Untuk informasi selengkapnya tentang kamus data CUR, lihat Laporan Biaya dan Penggunaan (CUR) 2.0. Perhatikan bahwa nama yang tepat bervariasi tergantung pada konteksnya.

A `UsageType` adalah string dengan nilai seperti`ReadCapacityUnit-Hrs`,, `USW2-ReadRequestUnitsEU-WriteCapacityUnit-Hrs`, atau`USE1-TimedPITRStorage-ByteHrs`. Setiap jenis penggunaan dimulai dengan awalan Region opsional. Jika tidak ada, itu menunjukkan Wilayah us-east-1. Jika ada, tabel di bawah ini memetakan kode Region penagihan pendek ke kode Region konvensional dan nama.

Misalnya, penggunaan bernama `USW2-ReadRequestUnits` menunjukkan unit permintaan baca yang dikonsumsi di us-west-2.

Kode Wilayah Penagihan	Kode Wilayah	Nama wilayah
AFS1	af-south-1	Afrika (Cape Town)
APE1	ap-east-1	Asia Pasifik (Hong Kong)
APN1	ap-northeast-1	Asia Pasifik (Tokyo)
APN2	ap-northeast-2	Asia Pasifik (Seoul)
APN3	ap-northeast-3	Asia Pasifik (Osaka)
APS1	ap-southeast-1	Asia Pasifik (Singapura)
APS2	ap-southeast-2	Asia Pasifik (Sydney)
APS3	ap-south-1	Asia Pasifik (Mumbai)
APS4	ap-southeast-3	Asia Pasifik (Jakarta)
APS5	ap-south-2	Asia Pasifik (Hyderabad)
APS6	ap-southeast-4	Asia Pasifik (Melbourne)
CAN1	ca-central-1	Kanada (Pusat)
EU	eu-west-1	Eropa (Irlandia)
EUC1	eu-central-1	Eropa (Frankfurt)
EUC2	eu-central-2	Eropa (Zürich)
EUN1	eu-north-1	Eropa (Stockholm)
EUS1	eu-south-1	Eropa (Milan)
EUS2	eu-south-2	Eropa (Spanyol)
EUW1	eu-west-1	Eropa (Irlandia)
EUW2	eu-west-2	Eropa (London)

Kode Wilayah Penagihan	Kode Wilayah	Nama wilayah
EUW3	eu-west-3	Eropa (Paris)
ILC1	Il-sentral-1	Israel (Tel Aviv)
MEC1	me-central-1	Timur Tengah (UEA)
MES1	me-south-1	Timur Tengah (Bahrain)
SAE1	sa-east-1	Amerika Selatan (Sao Paulo)
USE1 (default)	us-east-1	AS Timur (Virginia Utara)
USE2	us-east-2	AS Timur (Ohio)
UGE1	us-gov-east-1	Pemerintah AS Timur
UGW1	us-gov-west-1	Pemerintah AS Barat
USW1	us-west-1	AS Barat (California Utara)
USW2	us-west-2	AS Barat (Oregon)

Pada bagian berikut, kita menggunakan `REG-UsageType` pola ketika akan melalui biaya untuk DynamoDB, di mana REG menentukan wilayah di mana penggunaan terjadi dan UsageType adalah kode untuk jenis biaya. Misalnya jika Anda melihat item baris `USW1- ReadCapacityUnit-Hrs` di file CSV Anda, itu berarti penggunaan dilakukan di AS-West-1 untuk kapasitas baca yang disediakan. Dalam hal ini daftar akan mengatakan`REG-ReadCapacityUnit-Hrs`.

Topik

- Kapasitas Throughput
- Pengaliran
- Penyimpanan
- Pencadangan dan Pemulihan
- Transfer Data
- CloudWatch Wawasan Kontributor
- DynamoDB Accelerator (DAX)

Kapasitas Throughput

Kapasitas yang Disediakan Membaca dan Menulis

Saat Anda membuat tabel DynamoDB dalam mode kapasitas yang disediakan, Anda menentukan kapasitas baca dan tulis yang Anda harapkan dibutuhkan aplikasi Anda. Jenis penggunaan tergantung pada kelas tabel Anda (Standard atau Standard-Infrequent Access). Anda menyediakan membaca dan menulis berdasarkan tingkat konsumsi per detik, tetapi biaya tersebut diberi harga per jam berdasarkan kapasitas yang disediakan.

UsageType	Unit	Granularitas	Deskripsi
REG- -Jam ReadCapacityUnit	RCU-jam	Jam	Biaya untuk pembacaan dalam mode kapasitas yang disediakan menggunakan kelas tabel Standar.
REG-IA-ReadCapacityUnit-Hrs	RCU-jam	Jam	Biaya untuk pembacaan dalam mode kapasitas yang disediakan menggunakan kelas tabel IA standar.
REG- -Jam WriteCapacityUnit	Jam WCU	Jam	Biaya untuk menulis dalam mode kapasitas yang disediakan menggunakan kelas tabel Standar.
REG-IA-WriteCapacityUnit-Hrs	WCU-jam	Jam	Biaya untuk menulis dalam mode kapasitas yang disediakan

UsageType	Unit	Granularitas	Deskripsi
			menggunakan kelas tabel IA standar.

Kapasitas Cadangan Membaca dan Menulis

Dengan kapasitas yang dicadangkan, Anda membayar biaya satu kali di muka dan berkomitmen pada tingkat penggunaan minimum yang disediakan selama jangka waktu tertentu. Kapasitas cadangan ditagih dengan tarif per jam diskon. Kapasitas apa pun yang Anda sediakan melebihi kapasitas cadangan Anda akan ditagih sesuai tarif kapasitas standar yang disediakan. Kapasitas cadangan tersedia untuk unit kapasitas baca dan tulis satu wilayah, yang disediakan (RCU dan WCU) pada tabel DynamoDB yang menggunakan kelas tabel standar. Kapasitas cadangan 1 tahun dan 3 tahun ditagih menggunakan yang sama. SKUs

UsageType	Unit	Granularitas	Deskripsi
REG-:dynamodb.read HeavyUsage	RCU-jam	Di muka lalu bulanan	Biaya untuk kapasitas cadangan berbunyi: biaya di muka satu kali dan biaya bulanan pada awal setiap bulan yang mencakup semua jam RCU yang didiskon selama sebulan. Akan memiliki item baris tanpa biaya yang REG-ReadC apacityUnit-Hrs cocok.
REG-:dynamodb.write HeavyUsage	WCU-jam	Di muka lalu bulanan	Biaya untuk kapasitas cadangan menulis: biaya di muka satu kali dan biaya bulanan pada awal setiap

UsageType	Unit	Granularitas	Deskripsi
			bulan yang mencakup semua jam WCU yang didiskon selama sebulan. Akan memiliki item baris tanpa biaya yang REG-WriteCapacityUnit-Hrs cocok.

Kapasitas Sesuai Permintaan Membaca dan Menulis

Saat Anda membuat tabel DynamoDB dalam mode kapasitas sesuai permintaan, Anda hanya membayar untuk membaca dan menulis yang dilakukan aplikasi Anda. Harga untuk permintaan baca dan tulis tergantung pada kelas tabel Anda.

UsageType	Unit	Granularitas	Deskripsi
REG- ReadRequestUnits	RRUs	Unit	Biaya untuk membaca dalam mode kapasitas sesuai permintaan dengan kelas tabel Standar.
REG-IA- ReadRequestUnits	RRUs	Unit	Biaya untuk membaca dalam mode kapasitas sesuai permintaan dengan kelas tabel IA standar.
REG- WriteRequestUnits	WRUs	Unit	Biaya untuk menulis dalam mode kapasitas sesuai permintaan dengan kelas tabel Standar.

UsageType	Unit	Granularitas	Deskripsi
REG-IA- WriteRequ estUnits	WRUs	Unit	Biaya untuk menulis dalam mode kapasitas sesuai permintaan dengan kelas tabel IA standar.

Tabel Global Membaca dan Menulis

DynamoDB mengenakan biaya untuk penggunaan tabel global berdasarkan sumber daya yang digunakan pada setiap tabel replika. Untuk tabel global yang disediakan, permintaan tulis untuk tabel global diukur dalam replikasi WCUs (RWCu) alih-alih standar WCUs dan penulisan ke indeks sekunder global dalam tabel global diukur. WCUs Untuk tabel global sesuai permintaan, permintaan tulis diukur dalam replikasi WRUs (RwRU), bukan standar. WRUs Jumlah r WCUs atau r yang WRUs dikonsumsi untuk replikasi tergantung pada versi tabel global yang Anda gunakan. Harga tergantung pada kelas meja Anda.

Menulis ke indeks sekunder global (GSIs) ditagih menggunakan unit tulis standar (WCUs dan WRUs). Permintaan baca dan penyimpanan data ditagih identik dengan tabel satu wilayah.

Jika Anda menambahkan replika tabel untuk membuat atau memperluas tabel global di Wilayah baru, DynamoDB mengenakan biaya untuk pemulihan tabel di Wilayah tambahan per gigabyte data yang dipulihkan. Data yang Dipulihkan dibebankan sebagai REG-RestoreDataSize-Bytes. Silakan merujuk ke Backup dan restore untuk DynamoDB untuk detailnya. Replikasi Lintas Wilayah dan menambahkan replika ke tabel yang berisi data juga dikenakan biaya untuk transfer data keluar.

Saat memilih mode kapasitas sesuai permintaan untuk tabel global DynamoDB, Anda hanya membayar sumber daya yang digunakan aplikasi pada setiap tabel replika.

UsageType	Unit	Granularitas	Deskripsi
REG- -Jam ReplWrite CapacityUnit	RWCU-jam	Jam	Tabel global, disediakan, Kelas tabel standar.

UsageType	Unit	Granularitas	Deskripsi
REG-IA- -Jam ReplWriteCapacityUnit	RWCU-jam	Jam	Tabel global, disediakan, kelas tabel IA standar.
REG- ReplWrite RequestUnits	RwRU	Unit	Tabel global, sesuai permintaan, Kelas tabel standar.
REG-IA- ReplWrite RequestUnits	RwRU	Unit	Tabel global, sesuai permintaan, kelas tabel Standar-IA

Pengaliran

DynamoDB memiliki dua teknologi streaming, DynamoDB Streams dan Kinesis. Masing-masing memiliki harga terpisah.

DynamoDB Streams mengenakan biaya untuk membaca data dalam unit permintaan baca. Setiap panggilan `GetRecords` API ditagih sebagai permintaan baca aliran. Anda tidak dikenakan biaya untuk panggilan `GetRecords` API yang dipanggil oleh AWS Lambda sebagai bagian dari pemicu DynamoDB atau oleh tabel global DynamoDB sebagai bagian dari replikasi.

UsageType	Unit	Granularitas	Deskripsi
Reg-aliran-RequestsCount	Hitungan	Unit	Baca unit permintaan untuk DynamoDB Streams.

Amazon Kinesis Data Streams mengisi daya dalam unit pengambilan data perubahan. DynamoDB membebankan satu unit pengambilan data perubahan untuk setiap penulisan (hingga 1 KB). Untuk item yang lebih besar dari 1 KB, diperlukan unit pengambilan data perubahan tambahan. Anda hanya membayar untuk penulisan yang dilakukan aplikasi Anda tanpa harus mengelola kapasitas throughput di atas meja.

UsageType	Unit	Granularitas	Deskripsi
REG- ChangeDataCaptureUnits - Kinesis	Unit CDC	Unit	Ubah unit pengambilan data untuk Kinesis Data Streams.

Penyimpanan

DynamoDB mengukur ukuran data yang dapat ditagih dengan menambahkan ukuran byte mentah data Anda ditambah overhead penyimpanan per item yang bergantung pada fitur yang telah Anda aktifkan.

> **Note**
>
> Nilai penggunaan penyimpanan di CUR akan lebih tinggi dibandingkan dengan nilai penyimpanan saat menggunakan`DescribeTable`, karena `DescribeTable` tidak termasuk overhead penyimpanan per item.

Penyimpanan dihitung setiap jam tetapi dihargai setiap bulan seperti yang dihitung dari rata-rata biaya per jam.

Meskipun penyimpanan `UsageType` digunakan `ByteHrs` sebagai akhiran, penggunaan penyimpanan di CUR diukur dalam GB dan dihargai oleh GB-bulan.

UsageType	Unit	Granularitas	Deskripsi
REG- - TimedStorage ByteHrs	GB	Bulan	Jumlah penyimpanan yang digunakan oleh tabel dan indeks DynamoDB Anda, untuk tabel dengan kelas tabel Standar.
REG-IA- - TimedStorage ByteHrs	GB	Bulan	Jumlah penyimpanan yang digunakan oleh tabel dan indeks

UsageType	Unit	Granularitas	Deskripsi
			DynamoDB Anda, untuk tabel dengan kelas tabel Standard-IA.

Pencadangan dan Pemulihan

DynamoDB menawarkan dua jenis backup: Point In Time Recovery (PITR) backup dan on- demand backup. Pengguna juga dapat memulihkan dari cadangan tersebut ke tabel DynamoDB. Biaya di bawah ini mengacu pada pencadangan dan pemulihan.

Biaya penyimpanan cadangan dikeluarkan pada awal bulan dengan penyesuaian yang dilakukan sepanjang bulan karena cadangan ditambahkan atau dihapus. Lihat blog Memahami Amazon DynamoDB On-Demand Backup and Billing untuk informasi selengkapnya

UsageType	Unit	Granularitas	Deskripsi
REG- - TimedBackupStorage ByteHrs	GB	Bulan	Penyimpanan yang dikonsumsi oleh cadangan sesuai permintaan dari tabel DynamoDB dan Indeks Sekunder Lokal Anda.
Jangka waktu PITRStorage - ByteHrs	GB	Bulan	Penyimpanan yang digunakan oleh backup point-in-time recovery (PITR). DynamoDB memantau ukuran tabel berkemampuan PITR Anda secara terus menerus sepanjang bulan

UsageType	Unit	Granularitas	Deskripsi
			untuk menentukan biaya cadangan dan tagihan penyimpan an Anda selama PITR diaktifkan.
REG- RestoreDa taSize -Byte	GB	Size	Ukuran total data yang dipulihkan (termasuk data tabel, indeks sekunder lokal, dan indeks sekunder global) diukur dalam GB dari cadangan DynamoDB.

AWS Backup

AWS Backup adalah layanan pencadangan yang dikelola sepenuhnya yang memudahkan untuk memusatkan dan mengotomatiskan cadangan data di seluruh AWS layanan di cloud maupun di tempat. AWS Backup dikenakan biaya untuk penyimpanan (penyimpanan hangat atau dingin), aktivitas restorasi, dan transfer data lintas wilayah. `UsageType`Biaya berikut muncul di bawah “AWS Backup” dan ProductCode bukan “AmazonDynamoDB”.

UsageType	Unit	Granularitas	Deskripsi
REG- WarmStorage - ByteHrs -DynamoDB	GB	Bulan	Penyimpanan yang digunakan oleh backup DynamoDB dikelola AWS Backup oleh sepanjang bulan, diukur dalam GB-bulan.

UsageType	Unit	Granularitas	Deskripsi
REG- CrossRegion - WarmBytes - DynamoDB	GB	Size	Data yang ditransfer ke AWS Wilayah yang berbeda baik dalam akun yang sama atau ke AWS akun yang berbeda. Biaya transfer Lintas Wilayah terjadi saat menyalin cadangan dari satu Wilayah ke Wilayah lain. Tagihan selalu ditagih ke akun tempat data ditransfer.
Reg-Restore--DynamoDB WarmBytes	GB	Size	Ukuran total data yang dipulihkan dari penyimpanan hangat, diukur dalam GB.
REG- ColdStorage - ByteHrs -DynamoDB	GB	Bulan	Cold storage yang digunakan oleh DynamoDB backup dikelola AWS Backup oleh sepanjang bulan, diukur dalam GB-bulan.
Reg-Restore- - DynamoDB ColdBytes	GB	Bulan	Ukuran total data yang dipulihkan dari cold storage, diukur dalam GB.

Ekspor dan Impor

Anda dapat mengekspor data dari DynamoDB ke Amazon S3 atau mengimpor data dari Amazon S3 ke tabel DynamoDB baru.

Meskipun `UsageType` penggunaan `Bytes` sebagai akhiran, penggunaan ekspor dan impor di CUR diukur dan diberi harga dalam GB.

UsageType	Unit	Granularitas	Deskripsi
REG- ExportDataSize -Byte	GB	Size	Biaya untuk mengekspor data ke S3. DynamoDB mengenakan biaya untuk data yang Anda ekspor berdasarkan ukuran tabel dasar DynamoDB (data tabel dan indeks sekunder lokal) pada titik waktu yang ditentukan saat ekspor dibuat.
REG- ImportDataSize -Byte	GB	Size	Biaya untuk mengimpor data dari S3. Ukuran dihitung berdasarkan ukuran objek yang tidak terkompresi dari data dalam Amazon S3. Tidak ada biaya tambahan untuk mengimpor ke tabel dengan GSIs.

UsageType	Unit	Granularitas	Deskripsi
REG- IncrementalExportDataSize - Byte	GB	Size	Biaya untuk ukuran data yang diproses dari cadangan berkelanjutan untuk menghasilkan ekspor tambahan.

Transfer Data

Aktivitas transfer data mungkin muncul terkait dengan layanan DynamoDB. DynamoDB tidak mengenakan biaya untuk transfer data masuk, dan tidak mengenakan biaya untuk data yang ditransfer antara DynamoDB dan layanan AWS lain dalam Wilayah yang AWS sama (dengan kata lain, $0,00 per GB). Data yang ditransfer di seluruh AWS Wilayah (seperti antara DynamoDB di Wilayah AS Timur [Virginia N.] dan EC2 Amazon di Wilayah Uni Eropa [Irlandia]) dikenakan biaya pada kedua sisi transfer.

UsageType	Unit	Granularitas	Deskripsi
REG- DataTransfer - Dalam Byte	GB	Unit	Data ditransfer ke DynamoDB dari internet.
REG- DataTransfer - Out-Bytes	GB	Unit	Data ditransfer keluar dari DynamoDB ke internet.

CloudWatch Wawasan Kontributor

CloudWatch Contributor Insights for DynamoDB adalah alat diagnostik untuk mengidentifikasi kunci yang paling sering diakses dan dibatasi di tabel DynamoDB Anda. `UsageType`Biaya berikut muncul di bawah “AmazonCloudWatch” dan ProductCode bukan “AmazonDynamoDB”.

UsageType	Unit	Granularitas	Deskripsi
REG-CW: ContributorEventsManaged	Acara diproses	Unit	Jumlah peristiwa DynamoDB diproses. Misalnya untuk tabel dengan CloudWatch Contributor Insights diaktifkan, kapan saja item dibaca atau ditulis, itu dihitung sebagai satu peristiwa. Jika tabel memiliki kunci pengurutan, itu menghasilkan biaya untuk dua peristiwa.
REG-CW: ContributorRulesManaged	Hitungan aturan	Bulan	DynamoDB membuat aturan untuk mengidentifikasi item yang paling banyak diakses dan kunci yang paling dibatasi saat Anda mengaktifkan Wawasan Kontributor Cloud Watch. Biaya ini dikenakan untuk aturan yang ditambahkan untuk setiap entitas (tabel dan GSIs) yang dikonfigurasi untuk mencatat wawasan CloudWatch kontributor.

DynamoDB Accelerator (DAX)

DynamoDB Accelerator (DAX) ditagih berdasarkan jam berdasarkan jenis instance yang dipilih untuk layanan. Biaya di bawah ini mengacu pada instans DynamoDB Accelerator yang disediakan. `UsageType`Biaya berikut muncul di bawah “AmazonDAX” ProductCode daripada “AmazonDynamoDB”.

UsageType	Unit	Granularitas	Deskripsi
REG NodeUsage -:dax- <INSTANCE TYPE>	Jam nod	Jam	Penggunaan per jam dari jenis instance tertentu. Harga adalah per node-jam yang dikonsumsi, dari saat node diluncurkan hingga dihentikan. Setiap jam simpul sebagian yang dikonsumsi akan ditagih sebagai satu jam penuh. Biaya DAX untuk setiap node dalam cluster DAX. Jika Anda memiliki klaster dengan beberapa node, Anda akan melihat beberapa item baris dalam laporan penagihan Anda.

Jenis instance akan menjadi salah satu nilai dari daftar berikut. Untuk detail tentang jenis node, lihatSimpul.

- r3.2xlarge, r4.8xlarge, atau r5.8xlarge
- r3.4xlarge, r4.large, atau r5.large

- r3.8xlarge, r4.xlarge, atau r5.xlarge
- r3.2xlarge, r5.12xlarge, atau t2.medium
- r3.4xlarge, r4.large, atau r5.large
- r3.xlarge, r5.16xlarge, atau t2.small
- r4.16xlarge, r5.24xlarge, atau t3.medium
- r4.2xlarge, r5.2xlarge, atau t3.small
- r4.4xlarge atau r5.4xlarge

Migrasi tabel DynamoDB dari satu akun ke akun lainnya

Anda dapat memigrasikan tabel Amazon DynamoDB dari satu akun ke akun lainnya untuk menerapkan strategi multi-akun atau strategi pencadangan. Anda juga dapat melakukannya untuk alasan pengujian, debugging, atau kepatuhan. Kasus penggunaan umum adalah menyalin tabel DynamoDB di seluruh lingkungan produksi, pementasan, pengujian, dan pengembangan di mana setiap lingkungan menggunakan akun yang berbeda. AWS

DynamoDB menawarkan dua opsi untuk memigrasikan tabel dari AWS satu akun ke akun lainnya:

- AWS Backup untuk Pencadangan dan Pemulihan Lintas Akun: AWS Backup adalah layanan pencadangan yang dikelola sepenuhnya yang memungkinkan Anda mengelola cadangan secara terpusat di beberapa layanan. AWS Dengan fungsi pencadangan dan pemulihan lintas akun, Anda dapat mencadangkan tabel DynamoDB di satu akun dan mengembalikan cadangan ke akun lain di Organisasi yang sama. AWS
- DynamoDB Ekspor dan Impor ke Amazon S3: Menggunakan fitur Ekspor dan Impor DynamoDB ke Amazon S3 memungkinkan Anda melakukan ekspor penuh ke bucket Amazon S3 dan kemudian mengimpor data tersebut ke tabel baru di akun lain. AWS Pendekatan ini cocok ketika Anda perlu bermigrasi antar akun yang bukan bagian dari AWS Organisasi yang sama atau jika Anda tidak ingin menggunakannya AWS Backup.

> **Note**
>
> Impor dari Amazon S3 tidak mendukung tabel dengan Local Secondary Indexes (LSIs), tetapi mendukung Global Secondary Indexes (). GSIs Untuk informasi lebih lanjut tentang LSIs dan GSIs, lihatMeningkatkan akses data dengan indeks sekunder di DynamoDB.

Topik

- Migrasi tabel menggunakan AWS Backup pencadangan dan pemulihan lintas akun
- Migrasikan tabel menggunakan ekspor ke S3 dan impor dari S3

Migrasi tabel menggunakan AWS Backup pencadangan dan pemulihan lintas akun

Prasyarat

- AWS Akun sumber dan target harus dimiliki oleh organisasi yang sama dalam layanan AWS Organizations
- Izin AWS Identity and Access Management (IAM) yang valid untuk membuat dan menggunakan vault AWS Backup

Untuk informasi selengkapnya tentang menyiapkan cadangan lintas akun, lihat Membuat salinan cadangan di seluruh akun. AWS

Informasi harga

AWS biaya untuk cadangan (berdasarkan ukuran tabel), setiap penyalinan data antar AWS Wilayah (berdasarkan jumlah data), untuk pemulihan (berdasarkan jumlah data), dan untuk biaya penyimpanan yang sedang berlangsung. Untuk menghindari tagihan yang sedang berlangsung, Anda dapat menghapus cadangan jika Anda tidak membutuhkannya setelah pemulihan.

Untuk informasi selengkapnya tentang harga, silakan lihat harga AWS Backup.

Langkah 1: Aktifkan fitur-fitur canggih untuk DynamoDB dan cadangan lintas akun

1. Di AWS akun sumber dan target, akses Konsol AWS Manajemen dan buka konsol AWS Cadangan.
2. Pilih opsi Pengaturan.
3. Di bawah Fitur lanjutan untuk cadangan Amazon DynamoDB, konfirmasikan bahwa fitur lanjutan diaktifkan. Jika tidak, pilih Aktifkan.
4. Di bawah Manajemen lintas akun, untuk pencadangan lintas akun, pilih Aktifkan.

Langkah 2: Buat brankas cadangan di akun sumber dan akun target

1. Di AWS akun sumber, buka konsol AWS Cadangan.
2. Pilih Brankas Cadangan.
3. Pilih Buat brankas Cadangan.
4. Salin dan simpan Nama Sumber Daya Amazon (ARN) dari brankas cadangan yang dibuat dan akun target. AWS
5. Anda akan memerlukan vault cadangan sumber dan target saat menyalin cadangan tabel DynamoDB antar akun. ARNs

Langkah 3: Buat cadangan tabel DynamoDB di akun sumber

1. Pada halaman Dasbor AWS Cadangan, pilih Buat cadangan sesuai permintaan.
2. Di bagian Pengaturan, pilih DynamoDB sebagai tipe Sumber Daya, lalu pilih nama tabel.
3. Di daftar dropdown Backup vault, pilih brankas cadangan yang Anda buat di akun sumber.
4. Pilih periode Retensi yang diinginkan.
5. Pilih Buat cadangan sesuai permintaan.
6. Pantau status pekerjaan cadangan pada tab Backup Jobs di halaman AWS Backup Jobs.

Langkah 4: Salin cadangan tabel DynamoDB dari akun sumber ke akun target

1. Setelah pekerjaan pencadangan selesai, buka AWS Backup konsol di akun sumber dan pilih Backup vaults.
2. Di bawah Backup, pilih cadangan tabel DynamoDB. Pilih Tindakan dan kemudian Salin.
3. Masukkan AWS Wilayah akun target.
4. Untuk ARN vault eksternal, masukkan ARN dari brankas cadangan yang Anda buat di akun target.
5. Di brankas cadangan akun target, aktifkan akses dari akun sumber untuk memungkinkan menyalin cadangan.

Langkah 5: Kembalikan cadangan tabel DynamoDB di akun target

1. Di AWS akun target, buka AWS Backup konsol dan pilih Backup vaults

2. Di bawah Cadangan, pilih cadangan yang Anda salin dari akun sumber. Pilih Tindakan, lalu Pulihkan.
3. Masukkan nama untuk tabel DynamoDB baru, enkripsi yang akan dimiliki tabel baru ini, kunci yang ingin Anda pulihkan dienkripsi, dan opsi lainnya.
4. Ketika pemulihan selesai, status tabel akan ditampilkan sebagai Aktif.

Migrasikan tabel menggunakan ekspor ke S3 dan impor dari S3

Prasyarat

- Anda harus mengaktifkan Point-in-Time Pemulihan (PITR) untuk tabel Anda untuk melakukan ekspor ke S3. Untuk informasi selengkapnya, lihat Aktifkan point-in-time pemulihan di DynamoDB.
- Izin IAM yang valid untuk melakukan ekspor. Untuk informasi selengkapnya, lihat Meminta ekspor tabel di DynamoDB.
- Izin IAM yang valid cukup untuk melakukan impor. Untuk informasi selengkapnya, lihat Meminta impor tabel di DynamoDB.

Informasi harga

AWS biaya untuk PITR (berdasarkan ukuran tabel dan berapa lama PITR diaktifkan). Jika Anda tidak memerlukan PITR kecuali untuk ekspor, Anda dapat mematikannya setelah ekspor selesai. AWS juga mengenakan biaya untuk permintaan yang dibuat terhadap S3, untuk menyimpan data yang diekspor di S3 dan untuk mengimpor (berdasarkan ukuran data yang diimpor yang tidak terkompresi).

Untuk informasi selengkapnya tentang harga DynamoDB, lihat harga DynamoDB.

> **Note**
>
> Ada batasan ukuran dan jumlah objek saat mengimpor dari S3 ke DynamoDB. Untuk informasi selengkapnya, lihat Kuota impor.

Meminta ekspor tabel ke Amazon S3

1. Masuk ke Konsol AWS Manajemen dan buka konsol DynamoDB.
2. Di panel navigasi di sisi kiri konsol, pilih Ekspor ke S3.

3. Pilih tabel sumber dan bucket S3 tujuan. Masukkan URL bucket akun tujuan menggunakan `s3://bucketname/prefix` format. `/prefix`adalah folder opsional untuk membantu menjaga keranjang tujuan Anda tetap teratur.
4. Pilih Ekspor penuh. Ekspor penuh menghasilkan snapshot tabel lengkap tabel Anda, pada titik waktu yang Anda tentukan.
    a. Pilih Waktu saat ini untuk mengekspor snapshot tabel lengkap terbaru.
    b. Untuk format file yang diekspor, pilih antara DynamoDB JSON dan Amazon Ion. Opsi default adalah DynamoDB JSON.
5. Klik tombol Ekspor untuk memulai ekspor.
6. Ekspor meja kecil harus selesai dalam beberapa menit, tetapi tabel dalam kisaran terabyte dapat memakan waktu lebih dari satu jam.

Meminta impor tabel dari Amazon S3

1. Masuk ke Konsol AWS Manajemen dan buka konsol DynamoDB.
2. Di panel navigasi di sisi kiri konsol, pilih Impor dari S3.
3. Pada halaman yang muncul, pilih Impor dari S3.
4. Masukkan URL sumber Amazon S3. Anda juga dapat menemukannya dengan menggunakan tombol Browse S3. Jalur yang diharapkan adalah format`s3://bucket/prefix/AWSDynamoDB/<XXXXXXXX-XXXXXX>/data/`.
5. Tentukan apakah Anda adalah pemilik bucket S3.
6. Di bawah Impor kompresi file, pilih GZIP agar sesuai dengan ekspor.
7. Di bawah Impor format file, pilih DynamoDB JSON untuk mencocokkan ekspor.
8. Pilih Selanjutnya. Untuk Tentukan detail tabel, pilih opsi untuk tabel baru yang akan dibuat untuk menyimpan data Anda.
9. Pilih Selanjutnya. Untuk Konfigurasi pengaturan tabel, sesuaikan pengaturan tabel tambahan apa pun jika berlaku.
10. Pilih Berikutnya lagi untuk meninjau opsi impor Anda, lalu klik Impor untuk memulai tugas impor. Anda akan melihat tabel baru Anda tercantum di bawah Impor dari S3 dengan status Mengimpor. Anda tidak dapat mengakses tabel Anda selama waktu ini. Impor kecil harus selesai dalam beberapa menit, tetapi tabel dalam kisaran terabyte dapat memakan waktu lebih dari satu jam.
11. Setelah impor selesai, status ditampilkan sebagai Aktif, dan Anda dapat mulai menggunakan tabel.

Menjaga tabel tetap sinkron selama migrasi

Jika Anda dapat menjeda operasi tulis pada tabel sumber selama durasi migrasi, maka sumber dan output harus cocok tepat setelah migrasi. Jika Anda tidak dapat menjeda operasi tulis, tabel target biasanya akan sedikit di belakang sumber setelah migrasi. Untuk menangkap tabel sumber, Anda dapat menggunakan streaming (DynamoDB Streams atau Kinesis Data Streams untuk DynamoDB) untuk memutar ulang penulisan yang terjadi di tabel sumber sejak pencadangan atau ekspor.

Anda harus mulai membaca catatan aliran sebelum stempel waktu saat Anda mengekspor tabel sumber ke S3. Misalnya, jika ekspor ke S3 terjadi pada pukul 14:00 dan impor ke tabel target disimpulkan pada pukul 11:00, Anda harus memulai pembacaan aliran DynamoDB pada pukul 13:58. Opsi streaming untuk mengubah tabel pengambilan data merangkum fitur dari setiap model streaming.

Menggunakan DynamoDB Streams dengan Lambda menawarkan pendekatan yang efisien untuk menyinkronkan data antara tabel DynamoDB sumber dan target. Anda dapat menggunakan fungsi Lambda untuk memutar ulang setiap tulisan di tabel target.

> **Note**
>
> Item disimpan di DynamoDB Streams selama 24 jam, jadi Anda harus merencanakan untuk menyelesaikan pencadangan dan memulihkan atau mengekspor dan mengimpor dalam jendela itu.

Panduan preskriptif untuk mengintegrasikan DAX dengan aplikasi DynamoDB

DynamoDB Accelerator (DAX), adalah layanan caching yang kompatibel dengan DynamoDB yang menyediakan kinerja dalam memori yang cepat untuk aplikasi yang menuntut, seperti aplikasi read-heavy. Menggunakan DAX, Anda dapat mencapai waktu respons dalam mikrodetik untuk mengakses data yang sering diminta. Panduan preskriptif DynamoDB Accelerator ini memberikan wawasan komprehensif dan praktik terbaik untuk mengintegrasikan DAX dengan aplikasi DynamoDB Anda.

Panduan ini memberikan pengetahuan dasar bagi mereka yang baru mengenal DAX atau ingin mengoptimalkan konfigurasi yang ada. Panduan ini mencakup berbagai topik, misalnya, kapan menggunakan DAX dan membuat cluster DAX. Ini juga mencakup contoh praktis dan penjelasan terperinci untuk membantu Anda menerapkan DAX secara efektif dalam proyek Anda. Terakhir,

panduan ini menawarkan strategi lanjutan yang perlu Anda terapkan untuk memaksimalkan kemampuan caching DAX untuk memastikan aplikasi yang cepat dan terukur.

Topik

- Mengevaluasi kesesuaian DAX untuk kasus penggunaan Anda
- Mengkonfigurasi klien DAX Anda
- Mengonfigurasi klaster DAX Anda
- Mengukur kluster DAX Anda
- Menerapkan cluster
- Mengelola operasi klaster
- Memantau DAX

Mengevaluasi kesesuaian DAX untuk kasus penggunaan Anda

Bagian ini menjelaskan kapan dan mengapa menggunakan DAX. Menggunakan panduan ini membantu Anda menentukan apakah mengintegrasikan DAX dengan DynamoDB menguntungkan untuk pola beban kerja aplikasi Anda, persyaratan kinerja, dan kebutuhan konsistensi data. Ini juga mencakup skenario di mana DAX mungkin tidak cocok, misalnya, beban kerja berat tulis dan data yang jarang diakses.

Dalam bagian ini

- Kapan dan mengapa memilih DAX
- Kapan tidak menggunakan DAX

Kapan dan mengapa memilih DAX

Anda dapat mempertimbangkan untuk menambahkan DAX ke tumpukan aplikasi Anda dalam beberapa skenario. Misalnya, gunakan DAX untuk mengurangi latensi keseluruhan permintaan baca terhadap DynamoDB atau untuk meminimalkan pembacaan berulang data yang sama dari tabel. Daftar berikut menyajikan contoh skenario di mana Anda dapat memanfaatkan mengintegrasikan DAX dengan DynamoDB:

- Persyaratan kinerja tinggi
  - Pembacaan latensi rendah — Anda harus mempertimbangkan untuk menggunakan DAX jika aplikasi Anda memerlukan waktu respons dalam mikrodetik untuk pembacaan yang konsisten.

DAX juga dapat secara drastis mengurangi waktu respons untuk mengakses data yang sering dibaca.

- Beban kerja intensif baca
  - Aplikasi baca-berat - Untuk aplikasi dengan read-to-write rasio tinggi, misalnya, 10:1 atau lebih, DAX menghasilkan lebih banyak klik cache dan data yang kurang basi. Ini mengurangi pembacaan terhadap tabel. Untuk menghindari membaca data basi dari cache jika aplikasi Anda berat tulis, pastikan untuk mengatur lebih rendah Menggunakan time to live (TTL) di DynamoDB untuk cache.
  - Caching query umum — Jika aplikasi Anda sering membaca data yang sama, misalnya, produk populer di platform e-commerce, DAX dapat melayani permintaan ini langsung dari cache-nya.
- Pola lalu lintas meledak
  - Penskalaan tabel yang lebih halus — DAX membantu menghaluskan dampak lonjakan lalu lintas mendadak. DAX menyediakan buffer untuk meningkatkan kapasitas tabel DynamoDB dengan anggun, yang mengurangi risiko pelambatan baca.
  - Throughput baca yang lebih tinggi untuk setiap item - DynamoDB mengalokasikan partisi individu untuk setiap item. Namun, partisi mulai membatasi pembacaan item ketika mencapai 3.000 unit kapasitas baca (RCU). DAX memungkinkan Anda menskalakan pembacaan satu item di luar 3.000 RCU.
- Optimalisasi biaya
  - Mengurangi biaya DynamoDB - Membaca dari DAX dapat mengurangi pembacaan yang dikirim ke tabel DynamoDB, yang kemudian dapat berdampak langsung pada biaya. Dengan hit rate cache yang tinggi, pengurangan biaya baca tabel dapat melebihi biaya cluster DAX, yang menghasilkan pengurangan biaya bersih.
- Persyaratan konsistensi data
  - Konsistensi akhirnya - DAX mendukung pembacaan yang konsisten pada akhirnya. Hal ini membuat DAX cocok untuk kasus penggunaan di mana konsistensi langsung tidak penting.
  - Write-through caching — Menulis yang Anda buat terhadap DAX adalah write-through. Setelah DAX mengonfirmasi bahwa item tersebut ditulis ke DynamoDB, ia tetap mempertahankan versi item tersebut di cache item. Mekanisme penulisan ini membantu menjaga konsistensi data yang lebih ketat antara cache dan database, tetapi menggunakan sumber daya cluster DAX tambahan.

Kapan tidak menggunakan DAX

Meskipun DAX kuat, itu tidak cocok untuk semua skenario. Daftar berikut menyajikan contoh skenario di mana mengintegrasikan DAX dengan DynamoDB tidak cocok:

- Beban kerja berat tulis — Keuntungan utama DAX adalah mempercepat pembacaan, tetapi menulis menggunakan lebih banyak sumber daya DAX daripada membaca. Jika aplikasi Anda sebagian besar ditulis berat, manfaat DAX mungkin terbatas.
- Jarang membaca data — Jika aplikasi Anda jarang mengakses data atau berbagai data yang jarang digunakan kembali (data dingin), Anda mungkin akan mengalami penurunan. cache hit ratio Dalam hal ini, overhead pemeliharaan cache mungkin tidak membenarkan peningkatan kinerja.
- Bacaan atau tulis massal - Jika aplikasi Anda melakukan penulisan massal lebih banyak daripada penulisan individual, Anda harus menulis di sekitar DAX. Selain itu, untuk pembacaan massal, Anda harus menjalankan pemindaian tabel penuh langsung terhadap tabel DynamoDB.
- Konsistensi atau persyaratan transaksi yang kuat — DAX meneruskan pembacaan dan TransactGetItemspanggilan yang sangat konsisten ke tabel DynamoDB. Anda harus membuat pembacaan ini di sekitar cluster DAX untuk menghindari penggunaan sumber daya cluster. Item yang dibaca dengan cara ini tidak akan di-cache; oleh karena itu, merutekan item tersebut melalui DAX tidak ada gunanya.
- Aplikasi sederhana dengan persyaratan kinerja sederhana - Untuk aplikasi dengan persyaratan kinerja sederhana dan toleransi untuk latensi DynamoDB langsung, kompleksitas dan biaya penambahan DAX mungkin tidak diperlukan. Dengan sendirinya, DynamoDB menangani throughput tinggi dan memberikan kinerja milidetik satu digit.
- Kebutuhan kueri kompleks di luar akses nilai kunci — DAX dioptimalkan untuk pola akses nilai kunci. Jika aplikasi Anda memerlukan kemampuan kueri yang kompleks dengan pemfilteran yang kompleks, seperti operasi Kueri dan Pemindaian, manfaat caching DAX mungkin terbatas.

  Dalam situasi ini, gunakan Amazon ElastiCache (Redis OSS) sebagai alternatif. ElastiCache (Redis OSS) mendukung struktur data tingkat lanjut, seperti, daftar, set, dan hash. Ini juga menawarkan fitur, seperti pub/sub, indeks geospasial, dan skrip.
- Persyaratan kepatuhan — DAX saat ini tidak menawarkan akreditasi kepatuhan yang sama seperti DynamoDB. Misalnya, DAX belum memperoleh akreditasi SOC.

Mengkonfigurasi klien DAX Anda

Cluster DAX adalah cluster berbasis instance yang dapat diakses menggunakan berbagai DAX. SDKs Setiap SDK memberi pengembang opsi yang dapat dikonfigurasi, seperti RequestTimeout dan koneksi, untuk memenuhi persyaratan aplikasi tertentu.

Saat mengonfigurasi klien DAX Anda, pertimbangan penting adalah skala aplikasi klien Anda—khususnya, rasio instance klien terhadap instans server DAX (yang memiliki maksimum 11). Armada instance klien yang besar dapat menghasilkan banyak koneksi ke instance server DAX, yang berpotensi membuat mereka kewalahan. Panduan ini menguraikan praktik terbaik untuk konfigurasi klien DAX.

Praktik terbaik

1. Instans klien - Menerapkan instance klien tunggal untuk memastikan penggunaan kembali instance di seluruh permintaan. Untuk detail implementasi, lihat the section called “Langkah 4: Jalankan aplikasi sampel”.
2. Waktu tunggu permintaan — Meskipun aplikasi sering memerlukan batas waktu permintaan rendah untuk memastikan latensi minimal untuk sistem hulu, pengaturan batas waktu terlalu rendah dapat menyebabkan masalah. Batas waktu yang rendah dapat memicu koneksi ulang yang sering ke instance server saat server DAX mengalami lonjakan latensi sementara. Ketika batas waktu terjadi, klien DAX menghentikan koneksi node server yang ada dan membuat yang baru. Karena pembentukan koneksi intensif sumber daya, banyak koneksi berturut-turut dapat membebani server DAX. Sebaiknya lakukan hal berikut:
   - Mempertahankan pengaturan batas waktu permintaan default.
   - Jika batas waktu yang lebih rendah diperlukan, terapkan utas aplikasi terpisah dengan nilai batas waktu yang lebih rendah dan sertakan mekanisme coba lagi dengan back-off eksponensial.
3. Batas waktu koneksi — Untuk sebagian besar aplikasi, kami sarankan untuk mempertahankan pengaturan batas waktu koneksi default.
4. Koneksi bersamaan — Tertentu SDKs, seperti JavaV2, memungkinkan penyesuaian koneksi bersamaan ke server DAX. Pertimbangan utama:
   - Instans server DAX dapat menangani hingga 40.000 koneksi bersamaan.
   - Pengaturan default cocok untuk sebagian besar kasus penggunaan.
   - Instans klien besar yang dikombinasikan dengan koneksi bersamaan yang tinggi dapat membebani server.

- Nilai koneksi bersamaan yang lebih rendah mengurangi risiko kelebihan server.
- Contoh perhitungan kinerja:
  - Dengan asumsi latensi permintaan 1ms, setiap koneksi secara teoritis dapat menangani 1.000 permintaan/detik.
  - Untuk cluster 3-node, satu instance klien yang terhubung ke semua node dapat memproses 3.000 permintaan/detik.
  - Dengan 10 koneksi, klien dapat menangani sekitar 30.000 permintaan/detik.

    Rekomendasi - Mulailah dengan pengaturan koneksi bersamaan yang lebih rendah dan validasi melalui pengujian kinerja dengan pola beban kerja produksi yang diharapkan.

Mengonfigurasi klaster DAX Anda

Cluster DAX adalah cluster terkelola, tetapi Anda dapat menyesuaikan konfigurasinya agar sesuai dengan kebutuhan aplikasi Anda. Karena integrasinya yang erat dengan operasi API DynamoDB, Anda harus mempertimbangkan aspek-aspek berikut saat mengintegrasikan aplikasi Anda dengan DAX.

Dalam bagian ini

- Harga DAX
- Cache item dan cache kueri
- Memilih pengaturan TTL untuk cache
- Caching beberapa tabel dengan cluster DAX
- Replikasi data dalam tabel global DAX dan DynamoDB
- Ketersediaan Wilayah DAX
- Perilaku caching DAX

Harga DAX

Biaya cluster tergantung pada jumlah dan ukuran node yang telah disediakannya. Setiap node ditagih untuk setiap jam berjalan di cluster. Untuk informasi selengkapnya, lihat harga Amazon DynamoDB.

Hit cache tidak menimbulkan biaya DynamoDB, tetapi berdampak pada sumber daya cluster DAX. Kesalahan cache menimbulkan biaya baca DynamoDB dan memerlukan sumber daya DAX.

Penulisan menimbulkan biaya penulisan DynamoDB dan memengaruhi sumber daya cluster DAX untuk mem-proxy penulisan.

Cache item dan cache kueri

DAX mempertahankan cache item dan cache kueri. Memahami perbedaan antara cache ini dapat membantu Anda menentukan karakteristik kinerja dan konsistensi yang mereka tawarkan ke aplikasi Anda.

Karakteristik cache	Cache item	Cache kueri
Tujuan	Menyimpan hasil GetItemdan operasi BatchGetItemAPI.	Menyimpan hasil operasi Query and Scan API. Operasi ini dapat mengembalikan beberapa item berdasarkan kondisi kueri alih-alih kunci item tertentu.
Jenis Akses	Menggunakan akses berbasis kunci. Saat aplikasi meminta data menggunakan `GetItem` atau`BatchGetItem` , DAX terlebih dahulu memeriksa cache item menggunak an kunci utama item yang diminta. Jika item di-cache dan belum kedaluwarsa, DAX segera mengembal ikannya tanpa mengakses tabel DynamoDB.	Menggunakan akses berbasis parameter. DAX menyimpan kumpulan hasil `Query` dan operasi `Scan` API. DAX melayani permintaan berikutnya dengan parameter yang sama yang mencakup kondisi kueri yang sama, tabel, indeks, dari cache. Ini secara signifikan mengurangi waktu respons dan konsumsi throughput baca DynamoDB.
Pembatalan Cache	DAX secara otomatis mereplikasi item yang diperbarui ke dalam cache item node di cluster DAX dalam skenario berikut:	Cache kueri lebih sulit untuk dibatalkan daripada cache item. Pembaruan item mungkin tidak langsung dipetakan ke kueri atau

Karakteristik cache	Cache item	Cache kueri
	• Anda menulis pembaruan item melalui cache. • Baca versi item yang diperbarui dari tabel.	pindaian yang di-cache. Anda harus hati-hati menyetel cache kueri TTL untuk menjaga konsistensi data. Menulis melalui DAX atau tabel dasar tidak tercermin dalam cache kueri sampai TTL kedaluwar sa respons cache sebelumny a dan DAX melakukan kueri baru terhadap DynamoDB.
Indeks sekunder global	Karena operasi `GetItem` API tidak didukung pada indeks sekunder lokal atau indeks sekunder global, cache item hanya cache yang dibaca dari tabel dasar.	Cache kueri menyimpan kueri terhadap tabel dan indeks.

Memilih pengaturan TTL untuk cache

TTL menentukan periode penyimpanan data dalam cache sebelum menjadi basi. Setelah periode ini, data secara otomatis di-refresh pada permintaan berikutnya. Memilih pengaturan TTL yang tepat untuk cache DAX Anda melibatkan keseimbangan antara optimalisasi kinerja aplikasi dan konsistensi data. Karena tidak ada pengaturan TTL universal yang berfungsi untuk semua aplikasi, pengaturan TTL optimal bervariasi berdasarkan karakteristik dan persyaratan spesifik aplikasi Anda. Kami menyarankan Anda memulai dengan pengaturan TTL konservatif menggunakan panduan preskriptif ini. Kemudian, sesuaikan pengaturan TTL Anda secara berulang berdasarkan data kinerja dan wawasan aplikasi Anda.

DAX mempertahankan daftar yang paling tidak baru digunakan (LRU) untuk cache item. Daftar LRU melacak kapan item pertama kali ditulis atau terakhir dibaca dari cache. Ketika memori node DAX penuh, DAX mengusir item yang lebih lama meskipun belum kedaluwarsa untuk memberi ruang bagi item baru. Algoritma LRU selalu diaktifkan dan tidak dapat dikonfigurasi pengguna.

Untuk menetapkan durasi TTL yang berfungsi untuk aplikasi Anda, pertimbangkan hal-hal berikut:

Memahami pola akses data Anda

- Beban kerja baca-berat - Untuk aplikasi dengan beban kerja baca-berat dan pembaruan data yang jarang terjadi, tetapkan durasi TTL yang lebih lama untuk mengurangi jumlah cache yang hilang. Durasi TTL yang lebih lama juga mengurangi kebutuhan untuk mengakses tabel DynamoDB yang mendasarinya.
- Beban kerja berat tulis — Untuk aplikasi dengan pembaruan sering yang tidak ditulis melalui DAX, tetapkan durasi TTL yang lebih pendek untuk memastikan cache tetap konsisten dengan database. Durasi TTL yang lebih pendek juga mengurangi risiko menyajikan data basi.

Evaluasi persyaratan kinerja aplikasi Anda

- Sensitivitas latensi — Jika aplikasi Anda memerlukan latensi rendah dibandingkan kesegaran data, gunakan durasi TTL yang lebih lama. Durasi TTL yang lebih lama memaksimalkan klik cache, yang mengurangi latensi baca rata-rata.
- Throughput dan skalabilitas - Durasi TTL yang lebih lama mengurangi beban pada tabel DynamoDB dan meningkatkan throughput dan skalabilitas. Namun, Anda harus menyeimbangkan ini dengan kebutuhan akan up-to-date data.

Menganalisis penggusuran cache dan penggunaan memori

- Batas memori cache - Pantau penggunaan memori cluster DAX Anda. Durasi TTL yang lebih lama dapat menyimpan lebih banyak data dalam cache, yang mungkin mencapai batas memori dan menyebabkan penggusuran berbasis LRU.

Gunakan metrik dan pemantauan untuk menyesuaikan TTL

Tinjau metrik secara teratur, misalnya, klik dan kesalahan cache, dan pemanfaatan CPU dan memori. Sesuaikan pengaturan TTL Anda berdasarkan metrik ini untuk menemukan keseimbangan optimal antara kinerja dan kesegaran data. Jika kesalahan cache tinggi dan pemanfaatan memori rendah, tingkatkan durasi TTL untuk meningkatkan kecepatan hit cache.

Pertimbangkan persyaratan dan kepatuhan bisnis

Kebijakan penyimpanan data mungkin menentukan durasi TTL maksimum yang dapat Anda tetapkan untuk informasi sensitif atau pribadi.

Perilaku cache jika Anda menyetel TTL ke nol

Jika Anda menyetel TTL ke 0, cache item dan cache kueri menyajikan perilaku berikut:

- Cache item — Item dalam cache di-refeshed hanya ketika penggusuran LRU atau operasi write-through terjadi.
- Cache kueri - Respons kueri tidak di-cache.

Caching beberapa tabel dengan cluster DAX

Untuk beban kerja dengan beberapa tabel DynamoDB kecil yang tidak memerlukan cache individual, satu cluster DAX akan menyimpan permintaan untuk tabel ini. Ini memberikan penggunaan DAX yang lebih fleksibel dan efisien, terutama untuk aplikasi yang mengakses beberapa tabel dan memerlukan pembacaan berkinerja tinggi.

Mirip dengan APIs bidang data DynamoDB, permintaan DAX memerlukan nama tabel. Jika Anda menggunakan beberapa tabel dalam cluster DAX yang sama, Anda tidak memerlukan konfigurasi tertentu. Namun, Anda harus memastikan bahwa izin keamanan klaster memungkinkan akses ke semua tabel yang di-cache.

Pertimbangan untuk menggunakan DAX dengan beberapa tabel

Bila Anda menggunakan DAX dengan beberapa tabel DynamoDB, Anda harus mempertimbangkan hal-hal berikut:

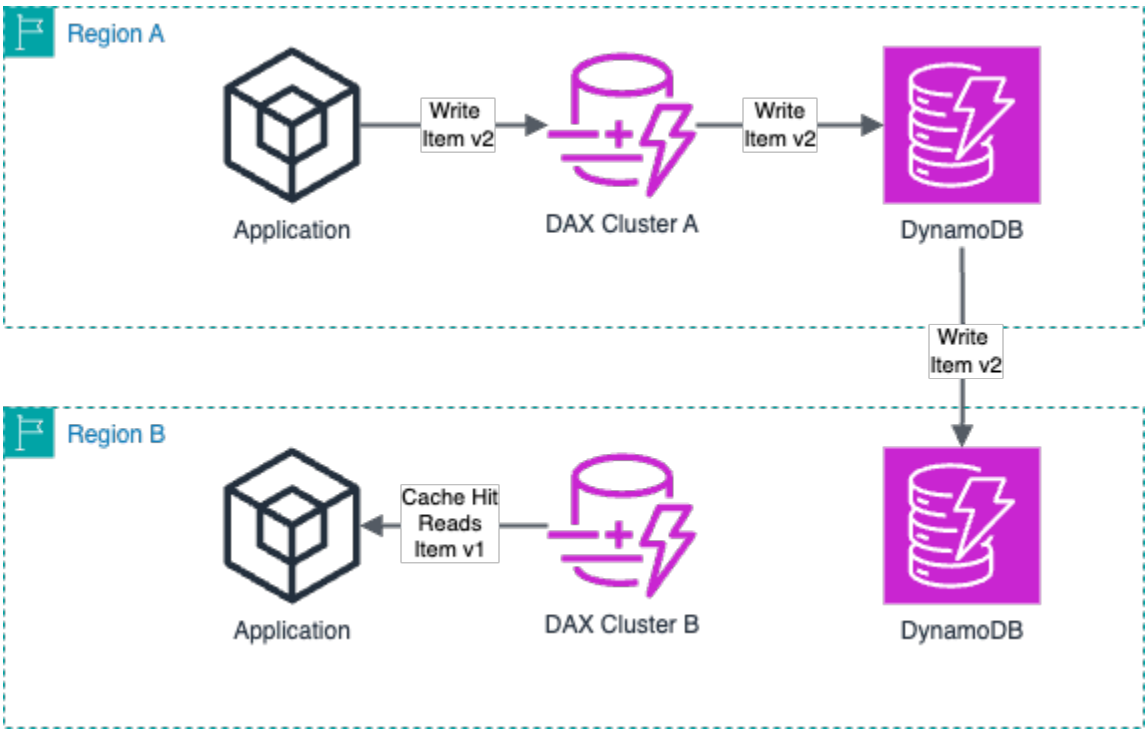
- Manajemen memori — Saat Anda menggunakan DAX dengan beberapa tabel, Anda harus mempertimbangkan ukuran total kumpulan data kerja Anda. Semua tabel dalam kumpulan data Anda akan berbagi ruang memori yang sama dari jenis node yang Anda pilih.
- Alokasi sumber daya - Sumber daya cluster DAX dibagi di antara semua tabel yang di-cache. Namun, tabel lalu lintas tinggi dapat menyebabkan penggusuran data dari tabel kecil tetangga.
- Skala ekonomi — Kelompokkan sumber daya yang lebih kecil ke dalam cluster DAX yang lebih besar untuk rata-rata lalu lintas ke pola yang lebih stabil. Untuk jumlah total sumber daya baca yang dibutuhkan cluster DAX, juga ekonomis untuk memiliki tiga atau lebih node. Ini juga meningkatkan ketersediaan semua tabel cache di cluster.

Replikasi data dalam tabel global DAX dan DynamoDB

DAX adalah layanan berbasis Region, jadi cluster hanya mengetahui lalu lintas di dalamnya. Wilayah AWS Tabel global menulis di sekitar cache ketika mereka mereplikasi data dari Wilayah lain.

Durasi TTL yang lebih lama dapat menyebabkan data basi tetap berada di Wilayah sekunder Anda lebih lama daripada di Wilayah primer. Hal ini dapat mengakibatkan kesalahan cache di cache lokal Wilayah sekunder.

Diagram berikut menunjukkan replikasi data yang terjadi pada tingkat tabel global di wilayah sumber A. Cluster DAX di Wilayah B tidak segera menyadari data yang baru direplikasi dari sumber Wilayah A.



Ketersediaan Wilayah DAX

Tidak semua Wilayah yang mendukung tabel DynamoDB mendukung penerapan kluster DAX. Jika aplikasi Anda memerlukan latensi baca rendah melalui DAX, tinjau dulu daftar Wilayah yang mendukung DAX. Kemudian, pilih Region untuk tabel DynamoDB Anda.

Perilaku caching DAX

DAX melakukan metadata dan caching negatif. Memahami perilaku caching ini akan membantu Anda mengelola metadata atribut item cache dan entri cache negatif secara efektif.

- Caching metadata — Cluster DAX mempertahankan metadata tanpa batas tentang nama atribut item yang di-cache. Metadata ini tetap ada bahkan setelah item kedaluwarsa atau diusir dari cache.

  Seiring waktu, aplikasi yang menggunakan jumlah nama atribut yang tidak terbatas dapat menyebabkan kelelahan memori di cluster DAX. Batasan ini hanya berlaku untuk nama atribut tingkat atas, tetapi tidak untuk nama atribut bersarang. Contoh nama atribut tak terbatas termasuk cap waktu,, UUIDs dan sesi. IDs Meskipun Anda dapat menggunakan stempel waktu dan sesi IDs sebagai nilai atribut, kami sarankan untuk menggunakan nama atribut yang lebih pendek dan lebih dapat diprediksi.
- Caching negatif - Jika terjadi kesalahan cache dan pembacaan dari tabel DynamoDB tidak menghasilkan item yang cocok, DAX menambahkan entri cache negatif di item atau cache kueri masing-masing. Entri ini tetap sampai durasi TTL cache berakhir atau write-through terjadi. DAX terus mengembalikan entri cache negatif ini untuk permintaan future.

  Jika perilaku caching negatif tidak sesuai dengan pola aplikasi Anda, baca tabel DynamoDB secara langsung saat DAX mengembalikan hasil kosong. Kami juga menyarankan Anda mengatur durasi cache TTL yang lebih rendah untuk menghindari hasil kosong yang tahan lama di cache dan meningkatkan konsistensi dengan tabel.

Mengukur kluster DAX Anda

Total kapasitas dan ketersediaan cluster DAX bergantung pada jenis dan jumlah node. Lebih banyak node di cluster meningkatkan kapasitas bacanya, tetapi bukan kapasitas tulis. Jenis node yang lebih besar (hingga r5.8xlarge) dapat menangani lebih banyak penulisan, tetapi terlalu sedikit node dapat memengaruhi ketersediaan ketika kegagalan node terjadi. Untuk informasi selengkapnya tentang ukuran cluster DAX Anda, lihat. Panduan pengukuran klaster DAX

Bagian berikut membahas berbagai aspek ukuran yang harus Anda pertimbangkan untuk menyeimbangkan jenis node dan hitungan untuk membuat cluster yang dapat diskalakan dan hemat biaya.

Dalam bagian ini

- Ketersediaan perencanaan
- Perencanaan throughput tulis
- Merencanakan throughput baca
- Merencanakan ukuran dataset
- Menghitung perkiraan persyaratan kapasitas cluster

- Memperkirakan kapasitas throughput cluster menurut tipe node
- Menskalakan kapasitas tulis di cluster DAX

Ketersediaan perencanaan

Saat mengukur cluster DAX, Anda harus terlebih dahulu fokus pada ketersediaan yang ditargetkan. Ketersediaan layanan berkerumun, seperti DAX, adalah dimensi dari jumlah total node dalam cluster. Karena cluster node tunggal tidak memiliki toleransi untuk kegagalan, ketersediaannya sama dengan satu node. Dalam cluster 10-node, hilangnya satu node memiliki dampak minimal terhadap kapasitas keseluruhan cluster. Kerugian ini tidak berdampak langsung pada ketersediaan karena node yang tersisa masih dapat memenuhi permintaan baca. Untuk melanjutkan penulisan, DAX dengan cepat menominasikan node primer baru.

DAX berbasis VPC. Ini menggunakan grup subnet untuk menentukan Zona Ketersediaan mana yang dapat menjalankan node dan alamat IP mana yang akan digunakan dari subnet. Untuk beban kerja produksi, kami sangat menyarankan Anda menggunakan DAX dengan setidaknya tiga node di Availability Zone yang berbeda. Ini memastikan bahwa cluster memiliki lebih dari satu node tersisa untuk menangani permintaan bahkan jika satu node atau Availability Zone gagal. Sebuah cluster dapat memiliki hingga 11 node, di mana satu adalah node utama dan 10 adalah replika baca.

Perencanaan throughput tulis

Semua cluster DAX memiliki node utama untuk permintaan write-through. Ukuran tipe node untuk cluster menentukan kapasitas penulisannya. Menambahkan replika baca tambahan tidak meningkatkan kapasitas tulis cluster. Oleh karena itu, Anda harus mempertimbangkan kapasitas tulis selama pembuatan cluster karena Anda tidak dapat mengubah jenis node nanti.

Jika aplikasi Anda perlu menulis melalui DAX untuk memperbarui cache item, pertimbangkan peningkatan penggunaan sumber daya klaster untuk memfasilitasi penulisan. Menulis terhadap DAX mengkonsumsi sekitar 25 kali lebih banyak sumber daya daripada pembacaan cache-hit. Ini mungkin memerlukan tipe node yang lebih besar daripada cluster read-only.

Untuk panduan tambahan tentang menentukan apakah write-through atau write-around akan bekerja paling baik untuk aplikasi Anda, lihat. Strategi untuk penulisan

Merencanakan throughput baca

Kapasitas baca cluster DAX bergantung pada rasio hit cache beban kerja Anda. Karena DAX membaca data dari DynamoDB ketika cache miss terjadi, ia mengkonsumsi sekitar 10 kali lebih

banyak sumber daya cluster daripada cache-hit. Untuk meningkatkan klik cache, tingkatkan pengaturan TTL cache untuk menentukan periode penyimpanan item dalam cache. Durasi TTL yang lebih tinggi, bagaimanapun, meningkatkan kemungkinan membaca versi item yang lebih lama kecuali pembaruan ditulis melalui DAX.

Untuk memastikan bahwa cluster memiliki kapasitas baca yang cukup, skala cluster secara horizontal seperti yang disebutkan dalamMenskalakan cluster secara horizontal. Menambahkan lebih banyak node menambahkan replika baca ke kumpulan sumber daya, sementara menghapus node mengurangi kapasitas baca. Saat Anda memilih jumlah node dan ukurannya untuk sebuah cluster, pertimbangkan jumlah minimum dan maksimum kapasitas baca yang diperlukan. Jika Anda tidak dapat menskalakan cluster secara horizontal dengan tipe node yang lebih kecil untuk memenuhi persyaratan baca Anda, gunakan tipe node yang lebih besar.

Merencanakan ukuran dataset

Setiap tipe node yang tersedia memiliki ukuran memori yang ditetapkan untuk DAX untuk menyimpan data cache. Jika tipe node terlalu kecil, kumpulan data kerja yang diminta aplikasi tidak akan muat dalam memori dan mengakibatkan cache hilang. Karena node yang lebih besar mendukung cache yang lebih besar, gunakan tipe node yang lebih besar dari perkiraan kumpulan data yang perlu Anda cache. Cache yang lebih besar juga meningkatkan rasio hit cache.

Anda mungkin mendapatkan pengembalian yang semakin berkurang untuk item caching dengan beberapa pembacaan berulang. Hitung ukuran memori untuk item yang sering diakses dan pastikan cache cukup besar untuk menyimpan kumpulan data tersebut.

Menghitung perkiraan persyaratan kapasitas cluster

Anda dapat memperkirakan total kebutuhan kapasitas beban kerja Anda untuk membantu Anda memilih ukuran dan jumlah node cluster yang sesuai. Untuk melakukan estimasi ini, hitung variabel permintaan normalisasi per detik (Normalized RPS). Variabel ini mewakili total unit kerja yang dibutuhkan aplikasi Anda untuk didukung oleh cluster DAX, termasuk cache hits, cache misses, dan write. Untuk menghitung RPS Normalisasi, gunakan input berikut:

- `ReadRPS_CacheHit`— Menentukan jumlah pembacaan per detik yang menghasilkan hit cache.
- `ReadRPS_CacheMiss`— Menentukan jumlah pembacaan per detik yang mengakibatkan kehilangan cache.
- `WriteRPS`— Menentukan jumlah penulisan per detik yang akan melalui DAX.
- `DaxNodeCount`— Menentukan jumlah node dalam cluster DAX.

- `Size`— Menentukan ukuran item yang ditulis atau dibaca dalam KB dibulatkan ke KB terdekat.
- `10x_ReadMissFactor`- Merupakan nilai 10. Ketika cache miss terjadi, DAX menggunakan sekitar 10 kali lebih banyak sumber daya daripada cache hits.
- `25x_WriteFactor`— Merupakan nilai 25 karena penulisan DAX menggunakan sekitar 25 kali lebih banyak sumber daya daripada klik cache.

Dengan menggunakan rumus berikut, Anda dapat menghitung RPS Normalisasi.

```
Normalized RPS = (ReadRPS_CacheHit * Size) + (ReadRPS_CacheMiss * Size *
 10x_ReadMissFactor) + (WriteRequestRate * 25x_WriteFactor * Size * DaxNodeCount)
```

Misalnya, pertimbangkan nilai input berikut:

- `ReadRPS_CacheHit`= 50.000
- `ReadRPS_CacheMiss`= 1.000
- `ReadMissFactor`= 1
- `Size`= 2 KB
- `WriteRPS`= 100
- `WriteFactor`= 1
- `DaxNodeCount`= 3

Dengan mengganti nilai-nilai ini dalam rumus, Anda dapat menghitung RPS Normalisasi sebagai berikut.

```
Normalized RPS = (50,000 Cache Hits/Sec * 2KB) + (1,000 Cache Misses/Sec * 2KB * 10) +
 (100 Writes/Sec * 25 * 2KB * 3)
```

Dalam contoh ini, nilai yang dihitung dari RPS Normalisasi adalah 135.000. Namun, nilai RPS Normalisasi ini tidak memperhitungkan penggunaan cluster di bawah 100% atau kehilangan node. Kami menyarankan Anda memperhitungkan kapasitas tambahan. Untuk melakukan ini, tentukan yang lebih besar dari dua faktor pengali: pemanfaatan target atau toleransi kehilangan simpul. Kemudian, kalikan RPS Normalisasi dengan faktor yang lebih besar untuk mendapatkan permintaan target per detik (Target RPS).

- Target pemanfaatan

Karena dampak kinerja meningkatkan kesalahan cache, kami tidak menyarankan menjalankan cluster DAX pada pemanfaatan 100%. Idealnya, Anda harus menjaga penggunaan cluster pada atau di bawah 70%. Untuk mencapai ini, kalikan RPS Normalisasi dengan 1,43.

- Toleransi kehilangan simpul

  Jika sebuah node gagal, aplikasi Anda harus dapat mendistribusikan permintaannya di antara node yang tersisa. Untuk memastikan node tetap di bawah pemanfaatan 100%, pilih jenis node yang cukup besar untuk menyerap lalu lintas ekstra sampai node yang gagal kembali online. Untuk cluster dengan node lebih sedikit, setiap node harus mentolerir peningkatan lalu lintas yang lebih besar ketika satu node gagal.

  Jika node primer gagal, DAX secara otomatis gagal ke replika baca dan menetapkannya sebagai primer baru. Jika replika node gagal, node lain dalam cluster DAX masih dapat melayani permintaan sampai node gagal dipulihkan.

  Misalnya, cluster DAX 3-node dengan kegagalan node memerlukan tambahan kapasitas 50% pada dua node yang tersisa. Ini membutuhkan faktor pengalikan 1,5. Sebaliknya, cluster 11-node dengan node gagal membutuhkan tambahan kapasitas 10% pada node yang tersisa atau faktor perkalian 1,1.

Dengan menggunakan rumus berikut, Anda dapat menghitung RPS Target.

```
Target RPS = Normalized RPS * CEILING(TargetUtilization, NodeLossTolerance)
```

Misalnya, untuk menghitung Target RPS, pertimbangkan nilai-nilai berikut:

- `Normalized RPS`= 135.000
- `TargetUtilization`= 1.43

  Karena kami bertujuan untuk pemanfaatan cluster maksimum 70%, kami menetapkan `TargetUtilization` ke 1,43.
- `NodeLossTolerance`= 1.5

  Katakanlah kami menggunakan cluster 3-node, oleh karena itu, kami mengatur `NodeLossTolerance` ke kapasitas 50%.

Dengan mengganti nilai-nilai ini dalam rumus, Anda dapat menghitung Target RPS sebagai berikut.

```
Target RPS = 135,000 * CEILING(1.43, 1.5)
```

Dalam contoh ini, karena nilai `NodeLossTolerance` lebih besar dari`TargetUtilization`, kami menghitung nilai Target RPS dengan`NodeLossTolerance`. Ini memberi kami Target RPS sebesar 202,500, yang merupakan jumlah total kapasitas yang harus didukung oleh cluster DAX. Untuk menentukan jumlah node yang Anda perlukan dalam sebuah cluster, petakan RPS Target ke kolom yang sesuai dalam tabel berikut. Untuk contoh RPS Target 202,500 ini, Anda memerlukan tipe node dax.r5.large dengan tiga node.

Memperkirakan kapasitas throughput cluster menurut tipe node

Dengan menggunakanTarget RPS formula, Anda dapat memperkirakan kapasitas cluster untuk berbagai jenis node. Tabel berikut menunjukkan perkiraan kapasitas untuk cluster dengan tipe node 1, 3, 5, dan 11. Kapasitas ini tidak menggantikan kebutuhan untuk melakukan pengujian beban DAX dengan data dan pola permintaan Anda sendiri. Selain itu, kapasitas ini tidak termasuk instance tipe-t karena kurangnya kapasitas CPU tetap. Unit untuk semua nilai dalam tabel berikut adalah Normalisasi RPS.

Jenis simpul (memori)	1 simpul	3 node	5 node	11 node
dax.r5.24xlarge (768GB)	1M	3M	5M	11M
dax.r5.16xlarge (512GB)	1M	3M	5M	11M
dax.r5.12xlarge (384GB)	1M	3M	5M	11M
dax.r5.8xlarge (256GB)	1M	3M	5M	11M
dax.r5.4xlarge (128GB)	600k	1,8 M	3M	6,6 M
dax.r5.2xlarge (64GB)	300k	900k	1,5 M	3.3M

Jenis simpul (memori)	1 simpul	3 node	5 node	11 node
dax.r5.xlarge (32GB)	150k	450k	750k	1,65M
dax.r5.large (16GB)	75k	225k	375k	825k

Karena batas maksimum 1 juta NPS (operasi jaringan per detik) untuk setiap node, node tipe dax.r5.8xlarge atau lebih besar tidak memberikan kontribusi kapasitas cluster tambahan. Jenis node yang lebih besar dari 8xlarge mungkin tidak berkontribusi pada total kapasitas throughput cluster. Namun, jenis node tersebut dapat membantu untuk menyimpan kumpulan data kerja yang lebih besar dalam memori.

Menskalakan kapasitas tulis di cluster DAX

Setiap penulisan ke DAX menggunakan 25 permintaan yang dinormalisasi pada setiap node. Karena ada batas 1 juta RPS untuk setiap node, cluster DAX dibatasi hingga 40.000 penulisan per detik, tidak memperhitungkan penggunaan baca.

Jika kasus penggunaan Anda memerlukan lebih dari 40.000 penulisan per detik dalam cache, Anda harus menggunakan cluster DAX terpisah dan memisahkan tulisan di antara mereka. Mirip dengan DynamoDB, Anda dapat hash kunci partisi untuk data yang Anda tulis ke cache. Kemudian, gunakan modulus untuk menentukan pecahan mana untuk menulis data.

Contoh berikut menghitung hash dari string input. Kemudian menghitung modulus nilai hash dengan 10.

```
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}.")
```

Menerapkan cluster

Membuat cluster DAX baru memerlukan konfigurasi di luar yang diperlukan untuk DynamoDB. Konfigurasi ini terutama untuk jaringan karena DAX didasarkan pada Amazon VPC. Ini memberi Anda kendali penuh atas lingkungan jaringan virtual Anda, termasuk penempatan sumber daya, konektivitas, dan keamanan. Bagian ini menyajikan praktik terbaik untuk pengaturan yang diperlukan selama pembuatan cluster.

Untuk informasi tentang memilih node cluster, lihatMengukur kluster DAX Anda.

Dalam bagian ini

- Konfigurasikan jaringan
- Konfigurasikan keamanan
- Grup parameter
- Periode pemeliharaan

Konfigurasikan jaringan

DAX menggunakan grup subnet untuk menentukan Availability Zone mana yang dapat menjalankan node dan alamat IP mana yang akan digunakan dari subnet. Untuk meminimalkan latensi antara aplikasi dan DAX, subnet dan Availability Zone untuk server aplikasi Anda dan cluster DAX harus sama.

Kami menyarankan Anda menyebarkan node DAX di beberapa Availability Zone. Opsi default Alokasi otomatis melakukan ini untuk Anda.

Untuk praktik terbaik tentang menyiapkan VPC, lihat Memulai Amazon VPC di Panduan Pengguna Amazon VPC.

Konfigurasikan keamanan

Bagian ini membahas langkah-langkah keamanan yang harus Anda terapkan untuk aplikasi Anda yang menggunakan DAX. Bagian ini juga membahas secara singkat dukungan yang disertakan DAX untuk enkripsi data.

IAM

DAX dan DynamoDB memiliki mekanisme kontrol akses yang terpisah. DAX memerlukan peran IAM untuk mengakses tabel DynamoDB Anda. Peran ini harus mengikuti prinsip hak istimewa terkecil dan memberikan akses hanya ke tabel tertentu dan operasi DynamoDB, seperti dan. GetItemPutItem Untuk informasi selengkapnya tentang mekanisme kontrol akses yang disediakan oleh DAX, lihatKontrol akses DAX.

Enkripsi

Anda mengonfigurasi enkripsi saat istirahat dan enkripsi saat transit saat membuat cluster DAX. Ini diaktifkan secara default. Kami menyarankan Anda menyimpan pengaturan enkripsi default kecuali persyaratan bisnis mencegahnya. Untuk informasi selengkapnya, lihat Enkripsi DAX saat diam dan Enkripsi bergerak DAX.

Grup parameter

DAX menerapkan satu set konfigurasi pada setiap node dalam cluster yang disebut grup parameter. Anda dapat mengubah konfigurasi ini setelah membuat cluster.

Grup parameter DAX menyimpan pengaturan TTL untuk cache item dan cache kueri. Secara default, durasi TTL adalah 5 menit. Anda dapat mengganti durasi TTL ke nilai integer yang lebih besar dari atau sama dengan 1 milidetik.

Anda tidak dapat mengubah grup parameter saat instans DAX yang sedang berjalan menggunakannya. Anda dapat mengubah nilai grup parameter selama downtime cluster DAX.

Periode pemeliharaan

Untuk memungkinkan peningkatan dan tambalan perangkat lunak sesekali ke node Anda, jendela pemeliharaan mingguan dikonfigurasi untuk klaster DAX. Selama jendela ini, DAX melakukan pembaruan bergulir ke node. Cluster dengan lebih dari satu node tidak kehilangan ketersediaan cluster selama pembaruan ini, tetapi telah mengurangi kapasitas cluster hingga node kembali. Jika organisasi Anda memiliki waktu penggunaan rendah yang dapat diprediksi, pertimbangkan untuk mengatur jendela pemeliharaan secara manual hingga saat ini.

Mengelola operasi klaster

DAX menangani pemeliharaan dan kesehatan cluster untuk Anda. Namun, Anda perlu memberikan input operasional untuk menskalakan cluster secara horizontal atau vertikal agar sesuai dengan pola

penggunaan Anda. Bagian ini menjelaskan proses yang disarankan untuk menskalakan klaster DAX Anda.

Dalam bagian ini

- Menskalakan cluster secara horizontal
- Menskalakan cluster secara vertikal

Menskalakan cluster secara horizontal

Penskalaan cluster DAX melibatkan penyesuaian kapasitasnya untuk memenuhi permintaan throughput. Penyesuaian ini dilakukan dengan menambah atau mengurangi jumlah node (replika) di cluster saat sedang berjalan. Proses ini, yang dikenal sebagai penskalaan horizontal, membantu mendistribusikan beban kerja di lebih banyak node atau mengkonsolidasikan ke lebih sedikit node saat permintaan rendah.

Anda dapat secara horizontal menskalakan cluster DAX Anda masuk dan keluar menggunakan `increase-replication-factor` perintah `decrease-replication-factor` atau di. AWS CLI

Meningkatkan faktor replikasi (skala keluar)

Meningkatkan faktor replikasi cluster DAX menambahkan lebih banyak node ke cluster. Contoh berikut menunjukkan penggunaan `increase-replication-factor` perintah.

```
aws dax increase-replication-factor \
    --cluster-name yourClusterName  \
    --new-replication-factor desiredReplicationFactor
```

- Dalam perintah ini, `cluster-name` argumen menentukan nama cluster Anda. Misalnya, *`yourClusterName`*.
- `new-replication-factor`Argumen menentukan jumlah total node untuk ditambahkan dalam cluster setelah penskalaan. Ini termasuk node primer dan node replika. Misalnya, jika cluster Anda saat ini memiliki 3 node dan Anda ingin menambahkan 2 node lagi, tetapkan nilainya `new-replication-factor` menjadi 5.

Kurangi faktor replikasi (skala dalam)

Mengurangi faktor replikasi cluster DAX menghapus node dari cluster. Menghapus node dapat membantu mengurangi biaya selama periode permintaan rendah. Contoh berikut menunjukkan penggunaan `decrease-replication-factor` perintah.

```
aws dax decrease-replication-factor \
    --cluster-name yourClusterName  \
    --new-replication-factor desiredReplicationFactor
```

- Dalam perintah ini, `cluster-name` argumen menentukan nama cluster Anda. Misalnya, *yourClusterName*.
- `new-replication-factor`Argumen menentukan pengurangan jumlah node di cluster Anda setelah penskalaan. Angka ini harus lebih rendah dari faktor replikasi saat ini dan harus menyertakan simpul utama. Misalnya, jika cluster Anda memiliki 5 node dan Anda ingin menghapus 2 node, atur nilainya `new-replication-factor` menjadi 3.

Pertimbangan penskalaan horisontal

Pertimbangkan hal berikut ketika Anda merencanakan penskalaan horizontal:

- Node primer — Cluster DAX mencakup simpul utama. Faktor replikasi termasuk simpul utama ini. Misalnya, faktor replikasi 3 berarti satu simpul utama dan dua node replika.
- Ketersediaan — Menambahkan atau menghapus node DAX mengubah ketersediaan klaster dan toleransi kesalahan. Lebih banyak node dapat meningkatkan ketersediaan, tetapi mereka juga meningkatkan biaya.
- Migrasi data — Saat Anda meningkatkan faktor replikasi, DAX secara otomatis menangani distribusi data di seluruh kumpulan node baru. Ketika node baru mulai melayani lalu lintas, cache-nya sudah dihangatkan. Namun, selama proses ini, mungkin ada dampak sementara pada kinerja selama migrasi data.

Pastikan Anda memantau klaster DAX Anda dengan cermat selama dan setelah proses penskalaan untuk memastikan klaster tersebut berkinerja seperti yang diharapkan dan membuat penyesuaian lebih lanjut seperlunya.

Menskalakan cluster secara vertikal

Untuk menskalakan ukuran node dari cluster yang ada secara vertikal, Anda perlu membuat klaster baru dan memigrasikan lalu lintas aplikasi ke cluster baru. Migrasi ke cluster baru dengan node

berbeda melibatkan beberapa langkah untuk memastikan transisi yang mulus dengan dampak minimal pada kinerja dan ketersediaan aplikasi Anda.

Untuk membuat klaster baru untuk menskalakan ukuran node Anda secara vertikal, pertimbangkan hal-hal berikut:

- Akses penyiapan Anda saat ini — Tinjau metrik klaster DAX Anda saat ini untuk menentukan ukuran dan kuantitas node baru yang Anda butuhkan. Gunakan informasi ini sebagai masukan untuk menentukan ukuran cluster Anda. Untuk informasi, lihat Mengukur kluster DAX Anda.
- Siapkan cluster DAX baru — Buat cluster DAX baru dengan tipe dan kuantitas node yang Anda tentukan. Anda dapat menggunakan pengaturan konfigurasi yang ada dari grup parameter Anda, kecuali Anda perlu melakukan penyesuaian.
- Sinkronisasi data — Karena DAX adalah lapisan caching untuk DynamoDB, Anda tidak perlu memigrasi data secara langsung. Namun, cluster DAX baru tidak akan memiliki kumpulan data kerja Anda di memori sampai Anda mengirim lalu lintas ke sana.
- Perbarui konfigurasi aplikasi — Perbarui konfigurasi aplikasi Anda untuk menunjuk ke titik akhir cluster DAX yang baru. Anda mungkin perlu mengubah kode atau memperbarui variabel lingkungan, tergantung pada konfigurasi aplikasi Anda.

  Untuk mengurangi dampak saat Anda beralih ke klaster baru, kirim lalu lintas kenari ke klaster baru dari sebagian kecil armada aplikasi Anda. Anda dapat melakukan ini dengan perlahan meluncurkan pembaruan aplikasi atau dengan menggunakan entri DNS routing berbasis berat di depan titik akhir DAX Anda.
- Pantau dan optimalkan — Setelah Anda beralih ke cluster DAX baru, pantau metrik dan log kinerjanya dengan cermat untuk masalah apa pun. Bersiaplah untuk menyesuaikan jumlah node berdasarkan pola beban kerja yang diperbarui.

  Sampai cluster baru menyimpan dataset kerja Anda dengan benar, Anda akan melihat tingkat kehilangan cache dan latensi yang lebih tinggi.
- Menonaktifkan klaster lama — Jika Anda yakin klaster baru berkinerja seperti yang diharapkan, nonaktifkan cluster DAX lama dengan aman untuk menghindari biaya yang tidak perlu.

Memantau DAX

Anda dapat memantau metrik utama, misalnya rasio hit cache, untuk memastikan kinerja cluster DAX yang optimal, mendiagnosis masalah, dan menentukan kapan Anda perlu menskalakan klaster. Memeriksa metrik utama secara teratur membantu Anda mempertahankan kinerja, stabilitas, dan

efisiensi biaya dengan menskalakan klaster agar sesuai dengan persyaratan beban kerja Anda. Untuk informasi selengkapnya tentang pemantauan DAX, lihatPemantauan produksi.

Daftar berikut menyajikan beberapa metrik utama yang harus Anda pantau:

- Cache hit ratio - Menunjukkan seberapa efektif DAX melayani data cache, mengurangi kebutuhan untuk mengakses tabel DynamoDB yang mendasarinya. Beberapa kesalahan cache untuk cluster menunjukkan efisiensi caching yang baik. Tetapi beberapa klik cache menunjukkan bahwa Anda mungkin perlu meninjau kembali pengaturan TTL caching atau beban kerja tidak cocok untuk caching.

  Gunakan Amazon CloudWatch untuk menghitung rasio hit cache cluster DAX Anda. Bandingkan`ItemCacheHits`,`ItemCacheMisses`,`QueryCacheHits`, dan `QueryCacheMisses` metrik untuk mendapatkan rasio ini. Rumus berikut menunjukkan bagaimana rasio hit cache dihitung. Untuk menghitung rasio menggunakan rumus ini, bagilah cache hits Anda dengan jumlah cache hits dan misses Anda.

  ```
  Cache hit ratio = Cache hits / (Cache hits + Cache misses)
  ```

  Rasio hit cache adalah angka antara 0 dan 1, yang direpresentasikan sebagai persentase. Persentase yang lebih tinggi menunjukkan pemanfaatan cache keseluruhan yang lebih baik.
- ErrorRequestCount— Jumlah permintaan yang mengakibatkan kesalahan pengguna yang dilaporkan oleh node atau cluster. `ErrorRequestCount`termasuk permintaan yang dibatasi oleh node atau cluster. Memantau kesalahan pengguna dapat membantu Anda mengidentifikasi kesalahan konfigurasi penskalaan atau item/partition pola panas dalam aplikasi Anda.
- Latensi operasi — Memantau latensi operasi baca dan tulis ke dan dari cluster DAX dapat membantu Anda dalam mengidentifikasi kemacetan kinerja. Peningkatan latensi mungkin menunjukkan masalah dengan konfigurasi cluster DAX, jaringan, atau kebutuhan untuk menskalakan.
- Konsumsi jaringan - Awasi `NetworkBytesIn` dan `NetworkBytesOut` metrik untuk memantau lalu lintas jaringan cluster DAX Anda. Peningkatan throughput jaringan yang tidak terduga dapat berarti lebih banyak permintaan klien atau pola kueri yang tidak efisien yang menyebabkan lebih banyak data ditransfer.

  Memantau konsumsi jaringan membantu Anda mengelola biaya untuk cluster DAX Anda. Ini juga memastikan jaringan tidak menjadi hambatan untuk kinerja cluster.

- Tingkat penggusuran - Menunjukkan seberapa sering item dihapus dari cache Anda untuk memberi ruang bagi item baru. Jika tingkat penggusuran meningkat dari waktu ke waktu, cache Anda mungkin terlalu kecil atau strategi caching Anda tidak efektif.

  Pantau `EvictedSize` metrik CloudWatch untuk menentukan apakah ukuran cache Anda memadai untuk beban kerja Anda. Jika ukuran total yang diusir terus bertambah, Anda mungkin perlu meningkatkan klaster DAX Anda untuk mengakomodasi cache yang lebih besar.
- Pemanfaatan CPU — Mengacu pada persentase pemanfaatan CPU dari node atau cluster. Ini adalah metrik penting untuk memantau database atau sistem caching apa pun. Pemanfaatan CPU yang tinggi dapat berarti cluster DAX Anda mungkin kelebihan beban dan perlu penskalaan untuk menangani peningkatan permintaan.

  Pantau `CPUUtilization` metrik untuk cluster DAX Anda. Jika pemanfaatan CPU Anda secara konsisten mendekati atau melebihi 70-80%, pertimbangkan untuk meningkatkan kluster DAX Anda seperti yang dijelaskan di bagian berikut.

  Jika jumlah permintaan yang dikirim ke DAX melebihi kapasitas node, DAX membatasi tingkat penerimaan permintaan tambahan. Ia melakukan ini dengan mengembalikan a ThrottlingException. DAX terus mengevaluasi pemanfaatan CPU klaster Anda untuk menentukan volume permintaan yang dapat diproses sambil mempertahankan status klaster yang sehat.

  Anda dapat memantau `ThrottledRequestCount` metrik yang dipublikasikan DAX. CloudWatch Jika melihat pengecualian ini secara rutin, Anda harus mempertimbangkan untuk menaikkan skala klaster.

Menskalakan cluster DAX Anda menggunakan data pemantauan

Anda dapat menentukan apakah Anda perlu meningkatkan atau menurunkan klaster DAX Anda dengan memantau metrik kinerjanya.

- Tingkatkan atau perkecil — Jika cluster DAX Anda memiliki pemanfaatan CPU yang tinggi, klik cache rendah (setelah mengoptimalkan strategi caching), atau latensi operasi yang tinggi, Anda harus meningkatkan skala klaster Anda. Menambahkan lebih banyak node, juga disebut scaling out, dapat membantu mendistribusikan beban secara lebih merata. Untuk beban kerja dengan peningkatan penulisan per detik, Anda mungkin perlu memilih node yang lebih kuat (meningkatkan skala).
- Turunkan skala — Jika Anda secara konsisten melihat pemanfaatan CPU yang rendah dan latensi operasi di bawah ambang batas Anda, Anda mungkin memiliki sumber daya yang disediakan

secara berlebihan. Dalam kasus seperti itu, kurangi node untuk mengurangi biaya. Anda dapat mengurangi jumlah node menjadi 1 selama periode pemanfaatan rendah, tetapi Anda tidak dapat mematikan cluster sepenuhnya.

Menggunakan DynamoDB dengan layanan lain AWS

Amazon DynamoDB terintegrasi dengan layanan AWS lain, memungkinkan Anda mengotomatiskan tugas berulang atau membangun aplikasi yang menjangkau beberapa layanan.

Topik

- Mengonfigurasi AWS kredensional menggunakan Amazon Cognito untuk DynamoDB
- Mengintegrasikan dengan Amazon Redshift
- Memproses Data DynamoDB Dengan Apache Hive di Amazon EMR
- Mengintegrasikan DynamoDB dengan Amazon S3
- Integrasi DynamoDB nol-ETL dengan Amazon Lakehouse SageMaker
- Integrasi DynamoDB Zero-ETL dengan Amazon Service OpenSearch
- Mengintegrasikan DynamoDB dengan Amazon EventBridge
- Mengintegrasikan DynamoDB dengan Amazon Managed Streaming untuk Apache Kafka
- Praktik terbaik integrasi dengan DynamoDB

Mengonfigurasi AWS kredensional menggunakan Amazon Cognito untuk DynamoDB

Cara yang disarankan untuk mendapatkan AWS kredensi untuk aplikasi web dan seluler Anda adalah dengan menggunakan Amazon Cognito. Amazon Cognito membantu Anda menghindari hardcoding AWS kredensional Anda pada file Anda. Ini menggunakan peran AWS Identity and Access Management (IAM) untuk menghasilkan kredensi sementara untuk pengguna yang diautentikasi dan tidak diautentikasi aplikasi Anda.

Misalnya, untuk mengonfigurasi JavaScript file Anda agar menggunakan peran Amazon Cognito yang tidak diautentikasi untuk mengakses layanan web Amazon DynamoDB, lakukan hal berikut.

Cara mengonfigurasi kredensial untuk diintegrasikan dengan Amazon Cognito

1. Buat kolam identitas Amazon Cognito yang mengizinkan identitas yang tidak terautentikasi.

```
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. Salin kebijakan berikut ini pada file bernama `myCognitoPolicy.json`. Ganti ID kumpulan identitas (*`us-west-2:12345678-1ab2-123a-1234-a12345ab12`*) dengan ID Anda sendiri `IdentityPoolId` yang diperoleh pada langkah sebelumnya.

```
{
  "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. Buat peran IAM yang mengasumsikan kebijakan sebelumnya. Dengan cara ini, Amazon Cognito menjadi entitas tepercaya yang dapat mengasumsikan peran `Cognito_DynamoPoolUnauth`.

```
aws iam create-role --role-name Cognito_DynamoPoolUnauth \
--assume-role-policy-document file://PathToFile/myCognitoPolicy.json --output json
```

4. Berikan peran `Cognito_DynamoPoolUnauth` akses penuh ke DynamoDB dengan melampirkan kebijakan yang dikelola (`AmazonDynamoDBFullAccess`).

```
aws iam attach-role-policy --policy-arn arn:aws:iam::aws:policy/
AmazonDynamoDBFullAccess \
--role-name Cognito_DynamoPoolUnauth
```

> **Note**
>
> Selain itu, Anda dapat memberikan akses ketat ke DynamoDB. Untuk informasi selengkapnya, lihat Menggunakan syarat kebijakan IAM untuk kontrol akses ketat.

5. Dapatkan dan salin peran IAM Amazon Resource Name (ARN).

```
aws iam get-role --role-name Cognito_DynamoPoolUnauth --output json
```

6. Tambahkan peran `Cognito_DynamoPoolUnauth` ke kolam identitas `DynamoPool`. Format untuk menentukannya adalah `KeyName=string`; `KeyName` adalah `unauthenticated`, dan string adalah peran ARN yang diperoleh pada langkah sebelumnya.

```
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. Tentukan kredensial Amazon Cognito pada file Anda. Modifikasi `IdentityPoolId` dan `RoleArn` dengan sesuai.

```
AWS.config.credentials = new AWS.CognitoIdentityCredentials({
IdentityPoolId: "us-west-2:12345678-1ab2-123a-1234-a12345ab12",
RoleArn: "arn:aws:iam::123456789012:role/Cognito_DynamoPoolUnauth"
});
```

Anda sekarang dapat menjalankan JavaScript program Anda terhadap layanan web DynamoDB menggunakan kredensi Amazon Cognito. Untuk informasi selengkapnya, lihat Pengaturan kredensial di browser web pada Panduan Memulai AWS SDK untuk JavaScript .

Mengintegrasikan dengan Amazon Redshift

Amazon Redshift adalah layanan gudang data berskala petabyte yang cepat, dikelola sepenuhnya, yang membuatnya sederhana dan hemat biaya untuk menganalisis semua data Anda secara efisien menggunakan alat intelijen bisnis yang ada.

DynamoDB dan Amazon Redshift dapat digunakan bersama untuk mengatasi kebutuhan penyimpanan dan pemrosesan data yang berbeda dalam suatu aplikasi atau ekosistem data.

Lihat topik di bawah ini untuk topik lebih rinci tentang cara mengintegrasikan DynamoDB dengan Amazon Redshift.

Topik

- Integrasi DynamoDB Zero-ETL dengan Amazon Redshift
- Memuat data dari DynamoDB ke Amazon Redshift dengan perintah COPY

Integrasi DynamoDB Zero-ETL dengan Amazon Redshift

Integrasi Amazon DynamoDB Zero-ETL dengan Amazon Redshift memungkinkan analisis tanpa batas pada data DynamoDB tanpa pengkodean apa pun. Fitur yang dikelola sepenuhnya ini secara otomatis mereplikasi tabel DynamoDB ke dalam database Amazon Redshift sehingga pengguna dapat menjalankan kueri dan analitik SQL pada data DynamoDB mereka tanpa harus menyiapkan proses ETL yang kompleks. Integrasi bekerja dengan mereplikasi data dari tabel DynamoDB ke database Amazon Redshift.

Untuk mengatur integrasi, cukup tentukan tabel DynamoDB sebagai sumber dan database Amazon Redshift sebagai target. Pada aktivasi, integrasi mengekspor tabel DynamoDB lengkap untuk mengisi database Amazon Redshift. Waktu yang dibutuhkan untuk proses awal ini untuk menyelesaikan tergantung pada ukuran tabel DynamoDB. Integrasi nol-ETL kemudian secara bertahap mereplikasi pembaruan dari DynamoDB ke Amazon Redshift setiap 15-30 menit menggunakan ekspor tambahan DynamoDB. Ini berarti data DynamoDB yang direplikasi di Amazon Redshift disimpan secara otomatis. up-to-date

Setelah dikonfigurasi, pengguna dapat menganalisis data DynamoDB di Amazon Redshift menggunakan klien dan alat SQL standar, tanpa memengaruhi kinerja tabel DynamoDB. Dengan menghilangkan ETL yang rumit, integrasi nol-ETL ini menyediakan cara cepat dan mudah untuk membuka wawasan dari DynamoDB melalui analitik Amazon Redshift dan kemampuan pembelajaran mesin.

Topik

- Prasyarat sebelum membuat integrasi DynamoDB Zero-ETL dengan Amazon Redshift
- Keterbatasan saat menggunakan integrasi DynamoDB Zero-ETL dengan Amazon Redshift
- Membuat integrasi DynamoDB Zero-ETL dengan Amazon Redshift
- Melihat integrasi DynamoDB Zero-ETL dengan Amazon Redshift
- Menghapus integrasi DynamoDB Zero-ETL dengan Amazon Redshift

Prasyarat sebelum membuat integrasi DynamoDB Zero-ETL dengan Amazon Redshift

1. Anda harus memiliki tabel DynamoDB sumber dan menargetkan klaster Amazon Redshift yang dibuat sebelum membuat integrasi. Informasi ini tercakup dalam Langkah 1: Mengkonfigurasi tabel DynamoDB sumber danLangkah 2: Membuat gudang data Amazon Redshift.
2. Integrasi nol-ETL antara Amazon DynamoDB dan Amazon Redshift memerlukan tabel DynamoDB sumber Anda untuk mengaktifkan pemulihan (PITR). Point-in-time
3. Untuk kebijakan berbasis sumber daya, jika Anda membuat integrasi di mana tabel DynamoDB dan gudang data Amazon Redshift berada di akun yang sama, Anda dapat menggunakan opsi Perbaiki untuk saya selama langkah buat integrasi untuk menerapkan kebijakan sumber daya yang diperlukan secara otomatis ke DynamoDB dan Amazon Redshift.

   Jika Anda membuat integrasi di mana tabel DynamoDB dan gudang data Amazon Redshift berada di akun yang AWS berbeda, Anda harus menerapkan kebijakan sumber daya berikut pada tabel DynamoDB Anda.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "Statement that allows Amazon Redshift service to DescribeTable
 and ExportTable",
            "Effect": "Allow",
            "Principal": {
                "Service": "redshift.amazonaws.com"
            },
            "Action": [
                "dynamodb:ExportTableToPointInTime",
                "dynamodb:DescribeTable"
            ],
            "Resource": "*",
```

```
            "Condition": {
                "StringEquals": {
                    "aws:SourceAccount": "<account>"
                },
                "ArnEquals": {
                    "aws:SourceArn":
 "arn:aws:redshift:<region>:<account>:integration:*"
                }
            }
        },
        {
            "Sid": "Statement that allows Amazon Redshift service to see all exports
 performed on the table",
            "Effect": "Allow",
            "Principal": {
                "Service": "redshift.amazonaws.com"
            },
            "Action": "dynamodb:DescribeExport",
            "Resource": "arn:aws:dynamodb:<region>:<account>:table/<table-name>/
export/*",
            "Condition": {
                "StringEquals": {
                    "aws:SourceAccount": "<account>"
                },
                "ArnEquals": {
                    "aws:SourceArn":
 "arn:aws:redshift:<region>:<account>:integration:*"
                }
            }
        }
    ]
}
```

Anda mungkin juga perlu mengonfigurasi kebijakan sumber daya di gudang data Amazon Redshift Anda. Untuk informasi selengkapnya, lihat Mengonfigurasi otorisasi menggunakan Amazon Redshift API.

4. Untuk kebijakan berbasis identitas:

   a. Pengguna yang membuat integrasi memerlukan kebijakan berbasis identitas yang mengotorisasi tindakan berikut:`GetResourcePolicy`,, dan. `PutResourcePolicy` `UpdateContinuousBackups`

> **Note**
>
> Contoh kebijakan berikut akan menunjukkan sumber daya sebagai`arn:aws:redshift{-serverless}`. Ini adalah contoh untuk menunjukkan bahwa arn dapat berupa `arn:aws:redshift` atau `arn:aws:redshift-serverless` bergantung pada apakah namespace Anda adalah cluster Amazon Redshift atau namespace Tanpa Server Amazon Redshift.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "dynamodb:ListTables"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "dynamodb:GetResourcePolicy",
                "dynamodb:PutResourcePolicy",
                "dynamodb:UpdateContinuousBackups"
            ],
            "Resource": [
                "arn:aws:dynamodb:<region>:<account>:table/<table-name>"
            ]
        },
        {
            "Sid": "AllowRedshiftDescribeIntegration",
            "Effect": "Allow",
            "Action": [
                "redshift:DescribeIntegrations"
            ],
            "Resource": "*"
        },
        {
            "Sid": "AllowRedshiftCreateIntegration",
            "Effect": "Allow",
```

```
            "Action": "redshift:CreateIntegration",
            "Resource": "arn:aws:redshift:<region>:<account>:integration:*"
        },
        {
            "Sid": "AllowRedshiftModifyDeleteIntegration",
            "Effect": "Allow",
            "Action": [
                "redshift:ModifyIntegration",
                "redshift:DeleteIntegration"
            ],
            "Resource": "arn:aws:redshift:<region>:<account>:integration:<uuid>"
        },
        {
            "Sid": "AllowRedshiftCreateInboundIntegration",
            "Effect": "Allow",
            "Action": "redshift:CreateInboundIntegration",
            "Resource":
            // The Amazon Resource Name (arn) for a Redshift provisioned cluster
 and a Redshift Serverless namespace have different formats.
            // Choose the one that applies to you:
            "arn:aws:redshift:<region>:<account>:namespace:<uuid>"
            "arn:aws:redshift-serverless:<region>:<account>:namespace/<uuid>"
        }
    ]
}
```

b. Pengguna yang bertanggung jawab untuk mengonfigurasi namespace Amazon Redshift tujuan memerlukan kebijakan berbasis identitas yang mengizinkan tindakan berikut:,, dan. `PutResourcePolicy` `DeleteResourcePolicy` `GetResourcePolicy`

```
{
  "Statement": [
    # This statement authorizes the user to change, view or remove resource
 policies on a specific namespace
    {
       "Effect": "Allow",
       "Action": [
           "redshift:PutResourcePolicy",
           "redshift:DeleteResourcePolicy",
           "redshift:GetResourcePolicy"
       ],
       "Resource": [
```

```
            "arn:aws:redshift{-serverless}:<region>:<account>:namespace/
ExampleNamespace"
        ]
    },
    # This statement authorizes the user to view integrations connected to any
 target namespaces in the account
    {
        "Effect": "Allow",
        "Action": [
            "redshift:DescribeInboundIntegrations"
        ],
        "Resource": [
            "arn:aws:redshift{-serverless}:<region>:<account>:namespace/*"
        ]
    }
  ],
  "Version": "2012-10-17"
}
```

5. Izin kunci enkripsi

   Jika tabel DynamoDB sumber dienkripsi menggunakan kunci AWS KMS terkelola pelanggan, Anda harus menambahkan kebijakan berikut pada kunci KMS Anda. Kebijakan ini memungkinkan Amazon Redshift untuk dapat mengekspor data dari tabel terenkripsi menggunakan kunci KMS Anda.

```
{
    "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:*"
```

```
        }
    }
}
```

Anda juga dapat mengikuti langkah-langkah Memulai integrasi Nol-ETL di panduan pengelolaan Amazon Redshift untuk mengonfigurasi izin namespace Amazon Redshift.

Keterbatasan saat menggunakan integrasi DynamoDB Zero-ETL dengan Amazon Redshift

Keterbatasan umum berikut berlaku untuk rilis integrasi ini saat ini. Keterbatasan ini dapat berubah dalam rilis berikutnya.

> **Note**
>
> Selain batasan di bawah ini, tinjau juga pertimbangan umum saat menggunakan integrasi nol-ETL lihat Pertimbangan saat menggunakan integrasi nol-ETL dengan Amazon Redshift di Panduan Manajemen Pergeseran Merah Amazon.

- Tabel DynamoDB dan cluster Amazon Redshift harus berada di Wilayah yang sama.
- Tabel DynamoDB sumber harus dienkripsi dengan kunci milik Amazon atau yang dikelola Pelanggan. AWS KMS Enkripsi terkelola Amazon tidak didukung untuk tabel DynamoDB sumber.

Membuat integrasi DynamoDB Zero-ETL dengan Amazon Redshift

Sebelum membuat integrasi nol-ETL, Anda harus terlebih dahulu mengatur tabel DynamoDB sumber Anda dan kemudian menargetkan gudang data Amazon Redshift.

Langkah 1: Mengkonfigurasi tabel DynamoDB sumber

Untuk membuat integrasi nol-ETL dengan Amazon Redshift, Anda perlu mengaktifkan point-in-time pemulihan (PITR) di meja Anda. Jika Anda tidak mengaktifkan PITR, konsol dapat memperbaikinya untuk Anda selama proses pengaturan integrasi. Untuk detail tentang cara mengaktifkan PITR, lihat Point-in-time pemulihan.

Langkah 2: Membuat gudang data Amazon Redshift

Jika Anda belum memiliki gudang data Amazon Redshift, Anda dapat membuatnya. Untuk membuat workgroup Amazon Redshift Tanpa Server, lihat Membuat workgroup dengan namespace. Untuk membuat klaster Amazon Redshift, lihat Membuat klaster.

Workgroup atau cluster Amazon Redshift target harus mengaktifkan parameter enable_case_sensitive_identifier agar integrasi berhasil. Untuk informasi selengkapnya tentang mengaktifkan sensitivitas huruf besar, lihat Mengaktifkan sensitivitas huruf besar untuk gudang data Anda di panduan pengelolaan Amazon Redshift.

Setelah grup kerja Amazon Redshift atau pengaturan cluster selesai, Anda perlu mengonfigurasi gudang data Anda. Lihat Integrasi nol-ETL di Panduan Manajemen Pergeseran Merah Amazon untuk informasi selengkapnya.

Langkah 3: Membuat integrasi DynamoDB nol-ETL

Sebelum Anda membuat integrasi nol-ETL, pastikan untuk menyelesaikan tugas di bagian berjudul. Prasyarat sebelum membuat integrasi DynamoDB Zero-ETL dengan Amazon Redshift Membuat integrasi antara DynamoDB dan Amazon Redshift adalah proses dua langkah. Pertama buat integrasi dari DynamoDB, lalu lampirkan database Amazon Redshift ke integrasi yang baru dibuat ini.

Buat integrasi nol-ETL

1. Masuk ke AWS Management Console dan buka konsol Amazon DynamoDB di. https://console.aws.amazon.com/dynamodbv2
2. Di panel navigasi, pilih Integrasi.
3. Pilih Buat integrasi Nol-ETL dan pilih Amazon Redshift.
4. Ini akan membawa Anda ke konsol Amazon Redshift. Untuk melanjutkan prosedur, lihat bagian DynamoDB di Buat integrasi nol-ETL untuk DynamoDB.

Melihat integrasi DynamoDB Zero-ETL dengan Amazon Redshift

Anda dapat melihat detail integrasi nol-ETL untuk melihat informasi konfigurasi dan status saat ini.

Untuk melihat detail integrasi nol-ETL di konsol Amazon DynamoDB:

1. Masuk ke AWS Management Console dan buka konsol Amazon DynamoDB di. https://console.aws.amazon.com/dynamodbv2

2. Di konsol DynamoDB, pilih Integrasi.
3. Di panel integrasi nol-ETL, pilih integrasi nol-ETL yang ingin Anda lihat.

Untuk melihat detail integrasi nol-ETL di konsol Amazon Redshift:

1. Masuk ke Konsol AWS Manajemen dan buka konsol Amazon Redshift di. https://console.aws.amazon.com/redshiftv2
2. Ikuti langkah-langkah di Melihat integrasi Nol-ETL.

> **Note**
>
> Status yang mungkin dari integrasi nol-ETL dengan Amazon Redshift tercantum dalam Melihat integrasi Nol-ETL di Panduan Manajemen Pergeseran Merah Amazon.

Menghapus integrasi DynamoDB Zero-ETL dengan Amazon Redshift

Saat Anda menghapus integrasi nol-ETL, data Anda tidak dihapus dari DynamoDB atau Amazon Redshift, tetapi DynamoDB berhenti mengirim data dari tabel sumber Anda ke target Amazon Redshift.

Untuk menghapus integrasi nol-ETL

1. Masuk ke AWS Management Console dan buka konsol Amazon DynamoDB di. https://console.aws.amazon.com/dynamodbv2
2. Di konsol DynamoDB, pilih Integrasi.
3. Di panel integrasi nol-ETL, pilih integrasi nol-ETL yang ingin Anda hapus.
4. Pilih Kelola. Ini akan membawa Anda ke halaman detail integrasi.
5. Untuk mengonfirmasi penghapusan, pilih Hapus.

Memuat data dari DynamoDB ke Amazon Redshift dengan perintah COPY

Amazon Redshift bekerja dengan Amazon DynamoDB dengan kemampuan intelijen bisnis canggih dan antarmuka berbasis SQL yang kuat. Ketika menyalin data dari tabel DynamoDB ke Amazon

Redshift, Anda dapat melakukan kueri analisis data yang kompleks pada data tersebut, termasuk penggabungan dengan tabel lain di klaster Amazon Redshift Anda.

Dalam hal throughput yang disediakan, operasi penyalinan dari tabel DynamoDB dihitung berdasarkan kapasitas baca tabel tersebut. Setelah data disalin, kueri SQL Anda di Amazon Redshift tidak memengaruhi DynamoDB dengan cara apa pun. Hal ini karena kueri Anda bertindak berdasarkan salinan data dari DynamoDB, bukan berdasarkan DynamoDB itu sendiri.

Sebelum Anda dapat memuat data dari tabel DynamoDB, Anda harus terlebih dahulu membuat tabel Amazon Redshift untuk dijadikan sebagai tujuan data. Ingatlah bahwa Anda menyalin data dari lingkungan NoSQL ke lingkungan SQL, dan ada aturan tertentu di satu lingkungan yang tidak berlaku di lingkungan lainnya. Berikut beberapa perbedaan yang perlu dipertimbangkan:

- Nama tabel DynamoDB dapat berisi hingga 255 karakter, termasuk '.' (titik) dan '-' (tanda hubung), dan peka huruf besar-kecil. Nama tabel Amazon Redshift dibatasi hingga 127 karakter, tidak boleh berisi titik atau garis, dan tidak peka huruf besar-kecil. Selain itu, nama tabel tidak boleh bertentangan dengan kata-kata cadangan Amazon Redshift.
- DynamoDB tidak mendukung konsep SQL NULL. Anda perlu menentukan bagaimana Amazon Redshift menafsirkan nilai atribut kosong atau kosong di DynamoDB, memperlakukannya sebagai atau sebagai bidang kosong. NULLs
- Jenis data DynamoDB tidak berhubungan langsung dengan jenis data Amazon Redshift. Anda perlu memastikan bahwa setiap kolom dalam tabel Amazon Redshift memiliki jenis dan ukuran data yang benar untuk mengakomodasi data dari DynamoDB.

Berikut ini contoh perintah COPY dari Amazon Redshift SQL:

```
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;
```

Dalam contoh ini, tabel sumber di DynamoDB adalah `my-favorite-movies-table`. Tabel target di Amazon Redshift adalah `favoritemovies`. Klausul `readratio 50` mengatur persentase throughput yang disediakan yang dikonsumsi; dalam hal ini, perintah COPY akan menggunakan tidak lebih dari 50 persen unit kapasitas baca yang disediakan untuk `my-favorite-movies-table`. Kami sangat menyarankan untuk mengatur rasio ini ke nilai yang lebih kecil dari rata-rata throughput yang disediakan yang tidak digunakan.

Untuk instruksi mendetail tentang memuat data dari DynamoDB ke Amazon Redshift, lihat bagian berikut dalam Panduan Pengembang Database Amazon Redshift:

- Memuat data dari tabel DynamoDB
- Perintah COPY
- Contoh COPY

Memproses Data DynamoDB Dengan Apache Hive di Amazon EMR

Amazon DynamoDB terintegrasi dengan Apache Hive, aplikasi gudang data yang berjalan di Amazon EMR. Hive dapat membaca dan menulis data dalam tabel DynamoDB, sehingga Anda dapat:

- Mengkueri data DynamoDB langsung menggunakan bahasa seperti SQL (HiveQL).
- Menyalin data dari tabel DynamoDB ke bucket Amazon S3, dan sebaliknya.
- Menyalin data dari tabel DynamoDB ke Hadoop Distributed File System (HDFS), dan sebaliknya.
- Melakukan operasi gabungan pada tabel DynamoDB.

Topik

- Gambaran Umum
- Tutorial: Menggunakan Amazon DynamoDB dan Apache Hive
- Membuat tabel eksternal di Hive
- Memproses pernyataan HiveQL
- Mengkueri data di DynamoDB
- Menyalin data ke dan dari Amazon DynamoDB
- Penyempurnaan performa

Gambaran Umum

Amazon EMR adalah layanan yang mempermudah proses data dalam jumlah besar dengan cepat dan hemat biaya. Untuk menggunakan Amazon EMR, Anda meluncurkan klaster EC2 instans Amazon terkelola yang menjalankan kerangka kerja open source Hadoop. Hadoop adalah aplikasi

terdistribusi yang mengimplementasikan MapReduce algoritma, di mana tugas dipetakan ke beberapa node di cluster. Setiap simpul memproses pekerjaan yang ditentukan secara paralel dengan simpul lain. Terakhir, output dikurangi pada simpul tunggal, sehingga menghasilkan hasil akhir.

Anda dapat memilih untuk meluncurkan klaster Amazon EMR Anda sehingga menjadi persisten atau transien:

- Klaster persisten berjalan sampai Anda menonaktifkannya. Klaster persisten ideal untuk analisis data, gudang data, atau penggunaan interaktif lainnya.
- Klaster transien berjalan cukup lama untuk memproses alur kerja, lalu akan tidak aktif secara otomatis. Klaster transien ideal untuk tugas-tugas pemrosesan berkala, seperti menjalankan skrip.

Untuk informasi tentang arsitektur dan administrasi Amazon EMR, lihat Panduan Manajemen Amazon EMR.

Saat meluncurkan kluster EMR Amazon, Anda menentukan nomor awal dan jenis instans Amazon EC2. Anda juga menentukan aplikasi terdistribusi lainnya (selain Hadoop itu sendiri) yang ingin Anda jalankan di klaster. Aplikasi ini termasuk Hue, Mahout, Pig, Spark, dan banyak lagi.

Untuk informasi tentang aplikasi untuk Amazon EMR, lihat Panduan Rilis Amazon EMR.

Tergantung pada konfigurasi klaster, Anda mungkin memiliki satu atau lebih jenis simpul berikut:

- Leader node — Mengelola cluster, mengoordinasikan distribusi MapReduce executable dan subset dari data mentah, ke inti dan kelompok instance tugas. Ini juga melacak status setiap tugas yang dilakukan dan memantau kondisi grup instans. Hanya ada satu simpul pemimpin dalam klaster.
- Node inti — Menjalankan MapReduce tugas dan menyimpan data menggunakan Hadoop Distributed File System (HDFS).
- Node tugas (opsional) - Menjalankan MapReduce tugas.

Tutorial: Menggunakan Amazon DynamoDB dan Apache Hive

Dalam tutorial ini, Anda akan meluncurkan klaster Amazon EMR, lalu menggunakan Apache Hive untuk memproses data yang disimpan dalam tabel DynamoDB.

Hive adalah aplikasi gudang data untuk Hadoop yang memungkinkan Anda memproses dan menganalisis data dari berbagai sumber. Hive menyediakan bahasa seperti SQL, HiveQL, yang

memungkinkan Anda menggunakan data yang disimpan secara lokal di klaster Amazon EMR atau sumber data eksternal (seperti Amazon DynamoDB).

Untuk informasi selengkapnya, lihat tutorial Hive.

Topik

- Sebelum Anda mulai
- Langkah 1: Buat EC2 key pair Amazon
- Langkah 2: Luncurkan klaster Amazon EMR
- Langkah 3: Hubungkan ke simpul Pemimpin
- Langkah 4: Muat data ke HDFS
- Langkah 5: Salin data ke DynamoDB
- Langkah 6: Kueri data dalam tabel DynamoDB
- Langkah 7: (Opsional) hapus

Sebelum Anda mulai

Dalam tutorial ini, Anda akan memerlukan berikut:

- Sebuah AWS akun. Jika belum memilikinya, lihat Mendaftar untuk AWS.
- Klien SSH (Secure Shell). Anda akan menggunakan klien SSH untuk terhubung ke simpul pemimpin klaster Amazon EMR dan menjalankan perintah interaktif. Klien SSH tersedia secara default pada sebagian besar instalasi Linux, Unix, dan Mac OS X. Pengguna Windows dapat mengunduh dan menginstal klien PuTTY, yang memiliki dukungan SSH.

Langkah selanjutnya

Langkah 1: Buat EC2 key pair Amazon

Langkah 1: Buat EC2 key pair Amazon

Pada langkah ini, Anda akan membuat EC2 key pair Amazon yang perlu Anda sambungkan ke node pemimpin Amazon EMR dan menjalankan perintah Hive.

1. Masuk ke AWS Management Console dan buka EC2 konsol Amazon di https://console.aws.amazon.com/ec2/.

2. Pilih wilayah (misalnya, `US West (Oregon)`). Ini harus berupa wilayah yang sama dengan tempat tabel DynamoDB Anda berada.
3. Di panel navigasi, pilih pasangan kunci.
4. Pilih Buat pasangan kunci.
5. Di Nama pasangan kunci, ketik nama untuk pasangan kunci Anda (misalnya, `mykeypair`), lalu pilih Buat.
6. Unduh file kunci privat. Nama file akan diakhiri dengan `.pem` (seperti `mykeypair.pem`). Simpan file kunci privat di tempat yang aman. Anda akan membutuhkannya untuk mengakses klaster Amazon EMR yang Anda luncurkan dengan pasangan kunci ini.

   > **Important**
   >
   > Jika kehilangan pasangan kunci, Anda tidak dapat terhubung ke simpul pemimpin klaster Amazon EMR Anda.

   Untuk informasi selengkapnya tentang pasangan kunci, lihat Pasangan EC2 Kunci Amazon di Panduan EC2 Pengguna Amazon.

Langkah selanjutnya

Langkah 2: Luncurkan klaster Amazon EMR

Langkah 2: Luncurkan klaster Amazon EMR

Pada langkah ini, Anda akan mengonfigurasi dan meluncurkan klaster Amazon EMR. Hive dan handler penyimpanan untuk DynamoDB akan terinstal di klaster.

1. Buka konsol EMR Amazon di https://console.aws.amazon.com /emr.
2. Pilih Buat Klaster.
3. Di halaman Buat Klaster - Opsi Cepat, lakukan tindakan berikut:

   a. Di Nama klaster, ketik nama untuk klaster Anda (misalnya: `My EMR cluster`).

   b. Di EC2 key pair, pilih key pair yang Anda buat sebelumnya.

   Biarkan pengaturan lainnya tetap default.

4. Pilih Buat klaster.

Peluncuran klaster Anda akan memakan waktu beberapa menit. Anda dapat menggunakan halaman Detail Klaster di konsol Amazon EMR untuk memantau kemajuannya.

Ketika status klaster berubah menjadi `Waiting`, klaster siap untuk digunakan.

File log klaster dan Amazon S3

klaster Amazon EMR menghasilkan file log yang berisi informasi tentang status klaster dan informasi debug. Pengaturan default untuk Buat Klaster - Opsi Cepat termasuk menyiapkan pencatatan log Amazon EMR.

Jika belum ada, itu AWS Management Console membuat ember Amazon S3. Nama bucket adalah`aws-logs-`*`account-id`*`-`*`region`*, di mana *`account-id`* nomor AWS akun Anda dan *`region`* merupakan wilayah tempat Anda meluncurkan cluster (misalnya,`aws-logs-123456789012-us-west-2`).

> **Note**
>
> Anda dapat menggunakan konsol Amazon S3 untuk melihat file log. Untuk informasi selengkapnya, lihat Melihat File Log di Panduan Manajemen Amazon EMR.

Anda dapat menggunakan bucket ini untuk tujuan selain pencatatan log. Misalnya, Anda dapat menggunakan bucket sebagai lokasi untuk menyimpan skrip Hive atau sebagai tujuan ketika mengekspor data dari Amazon DynamoDB ke Amazon S3.

Langkah berikutnya

Langkah 3: Hubungkan ke simpul Pemimpin

Langkah 3: Hubungkan ke simpul Pemimpin

Ketika status klaster Amazon EMR Anda berubah menjadi `Waiting`, Anda dapat menghubungkan ke simpul pemimpin menggunakan SSH dan melakukan operasi baris perintah.

1. Di konsol Amazon EMR, pilih nama klaster Anda untuk melihat statusnya.
2. Di halaman Detail Klaster, temukan bidang DNS publik pemimpin. Ini adalah nama DNS publik untuk simpul pemimpin klaster Amazon EMR Anda.

3. Di sebelah kanan nama DNS, pilih tautan SSH.
4. Ikuti petunjuk di Menghubungkan ke Simpul Pemimpin Menggunakan SSH.

   Tergantung pada sistem operasi Anda, pilih tab Windows atau tab Mac/Linux, lalu ikuti petunjuk untuk menghubungkan ke simpul pemimpin.

Setelah terhubung ke simpul pemimpin menggunakan SSH atau PuTTY, Anda akan melihat prompt perintah yang mirip dengan berikut ini:

```
[hadoop@ip-192-0-2-0 ~]$
```

Langkah berikutnya

Langkah 4: Muat data ke HDFS

Langkah 4: Muat data ke HDFS

Pada langkah ini, Anda akan menyalin file data ke Hadoop Distributed File System (HDFS), lalu membuat tabel Hive eksternal yang dipetakan ke file data.

Mengunduh data sampel

1. Unduh arsip data sampel (`features.zip`):

   ```
   wget https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/samples/
   features.zip
   ```

2. Ekstrak file `features.txt` dari arsip:

   ```
   unzip features.zip
   ```

3. Lihat beberapa baris pertama dari file `features.txt`:

   ```
   head features.txt
   ```

   Hasilnya akan tampak mirip dengan ini:

   ```
   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
```

`features.txt`File tersebut berisi subset data dari Dewan Amerika Serikat tentang Nama Geografis (http://geonames.usgs. gov/domestic/download_data.htm). Bidang di setiap baris mewakili berikut ini:

- ID Fitur (pengidentifikasi unik)
- Nama
- Kelas (danau; hutan; aliran; dan sebagainya)
- Negara Bagian
- Garis lintang (derajat)
- Garis bujur (derajat)
- Tinggi (dalam kaki)

4. Di prompt perintah, masukkan perintah berikut:

```
hive
```

Prompt perintah berubah menjadi ini: `hive>`

5. Masukkan pernyataan HiveQL berikut untuk membuat tabel Hive asli:

```
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. Masukkan pernyataan HiveQL berikut untuk memuat tabel dengan data:

```
LOAD DATA
LOCAL
INPATH './features.txt'
OVERWRITE
INTO TABLE hive_features;
```

7. Anda sekarang memiliki tabel Hive asli yang diisi dengan data dari file `features.txt`. Untuk memverifikasi, masukkan pernyataan HiveQL berikut:

```
SELECT state_alpha, COUNT(*)
FROM hive_features
GROUP BY state_alpha;
```

   Output akan menunjukkan daftar negara bagian dan jumlah fitur geografis di masing-masing tempat.

Langkah berikutnya

Langkah 5: Salin data ke DynamoDB

Langkah 5: Salin data ke DynamoDB

Pada langkah ini, Anda akan menyalin data dari tabel Hive (`hive_features`) ke tabel baru di DynamoDB.

1. Buka konsol DynamoDB di. https://console.aws.amazon.com/dynamodb/
2. Pilih Buat Tabel.
3. Di halaman Buat tabel DynamoDB, lakukan tindakan berikut:
   a. Di Tabel, masukkan **`Features`**.
   b. Untuk Kunci primer, di bidang Kunci partisi, masukkan **`Id`**. Atur jenis data ke Number.

      Kosongkan Gunakan Pengaturan Default. Untuk Kapasitas yang Disediakan, masukkan berikut ini:

      - Unit Kapasitas Baca—`10`
      - Unit Kapasitas Tulis—`10`

Pilih Buat.

4. Di prompt Hive, masukkan pernyataan HiveQL berikut:

```
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
);
```

Anda sekarang telah membuat pemetaan antara Hive dan tabel Fitur di DynamoDB.

5. Masukkan pernyataan HiveQL berikut untuk mengimpor data ke DynamoDB:

```
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 akan mengirimkan MapReduce pekerjaan, yang akan diproses oleh cluster EMR Amazon Anda. Proses ini akan memakan waktu beberapa menit untuk menyelesaikan tugas.

6. Verifikasi bahwa data telah dimuat ke DynamoDB:

   a. Di panel navigasi konsol DynamoDB, pilih Tabel.

   b. Pilih tabel Fitur, lalu pilih tab Item untuk melihat data.

Langkah berikutnya

[Langkah 6: Kueri data dalam tabel DynamoDB]

Langkah 6: Kueri data dalam tabel DynamoDB

Pada langkah ini, Anda akan menggunakan HiveQL untuk mengkueri tabel Fitur di DynamoDB. Coba kueri Hive berikut:

1. Semua jenis fitur (`feature_class`) dalam urutan abjad:

```
SELECT DISTINCT feature_class
FROM ddb_features
ORDER BY feature_class;
```

2. Semua danau yang dimulai dengan huruf "M":

```
SELECT feature_name, state_alpha
FROM ddb_features
WHERE feature_class = 'Lake'
AND feature_name LIKE 'M%'
ORDER BY feature_name;
```

3. Negara dengan setidaknya tiga fitur yang lebih tinggi dari satu mil (5.280 kaki):

```
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;
```

Langkah berikutnya

[Langkah 7: (Opsional) hapus]

Langkah 7: (Opsional) hapus

Setelah menyelesaikan tutorial, Anda dapat terus membaca bagian ini untuk mempelajari selengkapnya tentang penggunaan data DynamoDB di Amazon EMR. Sebaiknya klaster Amazon EMR Anda tetap aktif dan berjalan saat Anda mempelajari selengkapnya.

Jika tidak memerlukan klaster lagi, Anda dapat mengakhirinya dan menghapus sumber daya yang terkait. Ini akan membantu Anda menghindari biaya untuk sumber daya yang tidak Anda butuhkan.

1. Mengakhiri klaster Amazon EMR:
   a. Buka konsol EMR Amazon di https://console.aws.amazon.com /emr.
   b. Pilih klaster Amazon EMR, pilih Akhiri, lalu konfirmasikan.
2. Menghapus tabel Fitur di DynamoDB:
   a. Buka konsol DynamoDB di. https://console.aws.amazon.com/dynamodb/
   b. Di panel navigasi, pilih Tabel.
   c. Pilih tabel Fitur. Dari menu Tindakan, pilih Hapus Tabel.
3. Menghapus bucket Amazon S3 yang berisi file log Amazon EMR:
   a. Buka konsol Amazon S3 di. https://console.aws.amazon.com/s3/
   b. Dari daftar ember, pilih`aws-logs- accountID-region`, di mana *accountID* nomor AWS akun Anda dan *region* merupakan wilayah tempat Anda meluncurkan cluster.
   c. Dari menu Tindakan, pilih Hapus.

Membuat tabel eksternal di Hive

Di Tutorial: Menggunakan Amazon DynamoDB dan Apache Hive, Anda membuat tabel Hive eksternal yang dipetakan ke tabel DynamoDB. Ketika Anda mengeluarkan pernyataan HiveQL terhadap tabel eksternal, operasi baca dan tulis diteruskan ke tabel DynamoDB.

Anda dapat mempertimbangkan tabel eksternal sebagai pengarah ke sumber data yang dikelola dan disimpan di tempat lain. Dalam hal ini, sumber data yang mendasarinya adalah tabel DynamoDB. (Tabel harus sudah ada. Anda tidak dapat membuat, memperbarui, atau menghapus tabel DynamoDB dari dalam Hive.) Anda menggunakan pernyataan `CREATE EXTERNAL TABLE` untuk membuat tabel eksternal. Setelah itu, Anda dapat menggunakan HiveQL untuk menggunakan data di DynamoDB, seolah-olah data disimpan secara lokal di dalam Hive.

> **Note**
>
> Anda dapat menggunakan pernyataan `INSERT` untuk memasukkan data ke dalam tabel eksternal dan pernyataan `SELECT` untuk memilih data dari tabel eksternal. Namun, Anda

tidak dapat menggunakan pernyataan UPDATE atau DELETE untuk memanipulasi data dalam tabel tersebut.

Jika tidak membutuhkan tabel eksternal lagi, Anda dapat menghapusnya menggunakan pernyataan DROP TABLE. Dalam kasus ini, DROP TABLE hanya menghapus tabel eksternal di Hive. Tabel DynamoDB yang mendasarinya atau datanya tidak akan terpengaruh.

Topik

- MEMBUAT sintaks TABEL EKSTERNAL
- Pemetaan jenis data

MEMBUAT sintaks TABEL EKSTERNAL

Berikut ini menunjukkan sintaks HiveQL untuk membuat tabel Hive eksternal yang dipetakan ke tabel DynamoDB:

```
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..."
);
```

Baris 1 adalah awal dari pernyataan CREATE EXTERNAL TABLE, Anda menyediakan nama untuk tabel Hive (hive_table) yang ingin Anda buat.

Baris 2 menentukan kolom dan jenis data untuk hive_table. Anda perlu menentukan kolom dan jenis data yang sesuai dengan atribut di tabel DynamoDB.

Baris 3 adalah klausul STORED BY, Anda menentukan kelas yang menangani manajemen data antara Hive dan tabel DynamoDB. Untuk DynamoDB, STORED BY harus diatur ke 'org.apache.hadoop.hive.dynamodb.DynamoDBStorageHandler'.

Baris 4 adalah awal dari klausul TBLPROPERTIES, Anda menentukan parameter berikut untuk DynamoDBStorageHandler:

- `dynamodb.table.name`—nama tabel DynamoDB.
- `dynamodb.column.mapping`—pasang nama kolom di tabel Hive dan atributnya yang sesuai di tabel DynamoDB. Setiap pasangan memiliki format hive_column_name:dynamodb_attribute_name, dan pasangan dipisahkan dengan koma.

Perhatikan hal-hal berikut:

- Nama tabel Hive tidak harus sama dengan nama tabel DynamoDB.
- Nama kolom tabel Hive tidak harus sama dengan nama kolom di tabel DynamoDB.
- Tabel yang ditentukan oleh `dynamodb.table.name` harus ada di DynamoDB.
- Untuk `dynamodb.column.mapping`:
  - Anda harus memetakan atribut skema kunci untuk tabel DynamoDB. Ini termasuk kunci partisi dan kunci urutan (jika ada).
  - Anda tidak harus memetakan atribut non-kunci tabel DynamoDB. Namun, Anda tidak akan melihat data apa pun dari atribut tersebut saat Anda mengkueri tabel Hive.
  - Jika jenis data dari kolom tabel Hive dan atribut DynamoDB tidak kompatibel, Anda akan melihat `NULL` dalam kolom tersebut saat Anda mengkueri tabel Hive.

> **Note**
>
> Pernyataan `CREATE EXTERNAL TABLE` tidak melakukan validasi apa pun pada klausul `TBLPROPERTIES`. Nilai-nilai yang Anda berikan untuk `dynamodb.table.name` dan `dynamodb.column.mapping` hanya dievaluasi oleh kelas `DynamoDBStorageHandler` Anda mencoba mengakses tabel.

Pemetaan jenis data

Tabel berikut menunjukkan jenis data DynamoDB dan jenis data Hive yang kompatibel:

Jenis Data DynamoDB	Jenis Data Hive
String	`STRING`
Angka	`BIGINT` atau `DOUBLE`

Jenis Data DynamoDB	Jenis Data Hive
Biner	`BINARY`
Set String	`ARRAY<STRING>`
Set Angka	`ARRAY<BIGINT>` atau `ARRAY<DOUBLE>`
Set Biner	`ARRAY<BINARY>`

> **Note**
>
> Jenis data DynamoDB berikut tidak didukung oleh kelas `DynamoDBStorageHandler`, sehingga tidak dapat digunakan dengan `dynamodb.column.mapping`:
>
> - Peta
> - Daftar
> - Boolean
> - Null
>
> Namun, jika perlu menggunakan jenis data ini, Anda dapat membuat satu entitas yang disebut `item` yang mewakili seluruh item DynamoDB sebagai peta string untuk kunci dan nilai di peta. Untuk informasi selengkapnya, silakan lihat Menyalin data tanpa pemetaan kolom

Jika ingin memetakan atribut DynamoDB jenis Angka, Anda harus memilih jenis Hive yang sesuai:

- Jenis `BIGINT` Hive adalah untuk angka bulat bertanda 8-byte. Itu sama dengan jenis data `long` di Java.
- Jenis `DOUBLE` Hive adalah untuk angka floating-point presisi ganda 8-bit. Itu sama dengan jenis `double` di Java.

Jika Anda memiliki data numerik yang disimpan di DynamoDB yang memiliki presisi lebih tinggi dibandingkan jenis data Hive yang Anda pilih, maka mengakses data DynamoDB dapat menyebabkan hilangnya presisi.

Jika Anda mengekspor data jenis Biner dari DynamoDB ke (Amazon S3) atau HDFS, data disimpan sebagai string yang dienkode Base64. Jika mengimpor data dari Amazon S3 atau HDFS ke jenis Biner DynamoDB, Anda harus memastikan data dienkode sebagai string Base64.

Memproses pernyataan HiveQL

Hive adalah aplikasi yang berjalan di Hadoop, yang merupakan kerangka kerja berorientasi batch untuk menjalankan pekerjaan. MapReduce Saat Anda mengeluarkan pernyataan HiveQL, Hive menentukan apakah pernyataan HiveQL dapat segera mengembalikan hasilnya atau apakah harus mengirimkan pekerjaan. MapReduce

Misalnya, pertimbangkan tabel ddb_features (dari Tutorial: Menggunakan Amazon DynamoDB dan Apache Hive). Kueri Hive berikut mencetak singkatan negara dan jumlah pertemuan puncak di masing-masing negara:

```
SELECT state_alpha, count(*)
FROM ddb_features
WHERE feature_class = 'Summit'
GROUP BY state_alpha;
```

Hive tidak segera menampilkan hasil. Sebaliknya, ia mengirimkan MapReduce pekerjaan, yang diproses oleh kerangka Hadoop. Hive akan menunggu sampai tugas selesai sebelum menunjukkan hasil dari kueri:

```
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)
```

Pemantauan dan pembatalan tugas

Saat Hive meluncurkan tugas Hadoop, Hive akan mencetak output dari tugas itu. Status penyelesaian tugas diperbarui saat tugas berlangsung. Dalam beberapa kasus, status mungkin tidak diperbarui untuk waktu yang lama. (Hal ini dapat terjadi ketika Anda mengkueri tabel DynamoDB besar yang memiliki pengaturan kapasitas baca dengan ketersediaan rendah.)

Jika harus membatalkan tugas sebelum selesai, Anda dapat mengetik **`Ctrl+C`** kapan saja.

Mengkueri data di DynamoDB

Contoh berikut menunjukkan beberapa cara yang dapat Anda gunakan dalam menggunakan HiveQL untuk mengkueri data yang disimpan di DynamoDB.

Contoh-contoh ini mengacu pada tabel ddb_features di tutorial (Langkah 5: Salin data ke DynamoDB).

Topik

- Menggunakan fungsi agregat
- Menggunakan klausul GROUP BY dan HAVING
- Menggabungkan dua tabel DynamoDB
- Tabel Gabungan dari berbagai sumber

Menggunakan fungsi agregat

HiveQL menyediakan fungsi bawaan untuk meringkas nilai-nilai data. Misalnya, Anda dapat menggunakan fungsi `MAX` untuk menemukan nilai terbesar untuk kolom tertentu. Contoh berikut mengembalikan elevasi fitur tertinggi di negara bagian Colorado.

```
SELECT MAX(elev_in_ft)
FROM ddb_features
WHERE state_alpha = 'CO';
```

Menggunakan klausul GROUP BY dan HAVING

Anda dapat menggunakan klausul `GROUP BY` untuk mengumpulkan data di beberapa catatan. Klausul ini sering kali digunakan dengan fungsi agregat seperti `SUM`, `COUNT`, `MIN`, atau `MAX`. Anda juga dapat menggunakan klausul `HAVING` untuk membuang hasil yang tidak memenuhi kriteria tertentu.

Contoh berikut menghasilkan daftar elevasi tertinggi dari negara-negara yang memiliki lebih dari lima fitur di tabel ddb_features.

```
SELECT state_alpha, max(elev_in_ft)
FROM ddb_features
GROUP BY state_alpha
HAVING count(*) >= 5;
```

Menggabungkan dua tabel DynamoDB

Contoh berikut memetakan tabel Hive lain (east_coast_states) ke tabel di DynamoDB. Pernyataan `SELECT` adalah gabungan di dua tabel ini. Gabungan tersebut dikomputasi pada klaster dan dikembalikan. Gabungan ini tidak terjadi di DynamoDB.

Pertimbangkan tabel DynamoDB EastCoastStates bernama yang berisi data berikut:

```
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
```

Mari kita asumsikan tabel tersedia sebagai tabel eksternal Hive bernama east_coast_states:

```
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");
```

Gabungan berikut menampilkan negara-negara di Pantai Timur Amerika Serikat yang memiliki setidaknya tiga fitur:

```
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;
```

Tabel Gabungan dari berbagai sumber

Di contoh berikut, s3_east_coast_states adalah tabel Hive yang terkait dengan file CSV yang disimpan di Amazon S3. Tabel ddb_features terkait dengan data dalam DynamoDB. Contoh berikut menggabungkan dua tabel ini, menampilkan fitur geografis dari negara-negara yang namanya dimulai dengan "New."

```
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%';
```

Menyalin data ke dan dari Amazon DynamoDB

Di Tutorial: Menggunakan Amazon DynamoDB dan Apache Hive, Anda menyalin data dari tabel Hive asli ke tabel DynamoDB eksternal, lalu mengkueri tabel DynamoDB eksternal. Tabel tersebut

eksternal karena berada di luar Hive. Meskipun Anda menghapus tabel Hive yang dipetakan ke sana, tabel di DynamoDB tidak terpengaruh.

Hive adalah solusi yang sangat baik untuk menyalin data antara tabel DynamoDB, bucket Amazon S3, tabel Hive asli, dan Hadoop Distributed File System (HDFS). Bagian ini menyediakan contoh operasi tersebut.

Topik

- Menyalin data antara DynamoDB dan tabel Hive asli
- Menyalin data antara DynamoDB dan Amazon S3
- Menyalin data antara DynamoDB dan HDFS
- Menggunakan kompresi data
- Membaca data karakter UTF-8 yang tidak dapat dicetak

Menyalin data antara DynamoDB dan tabel Hive asli

Jika memiliki data dalam tabel DynamoDB, Anda dapat menyalin data tersebut ke tabel Hive asli. Tindakan ini akan menghasilkan snapshot data tersebut, saat Anda menyalinnya.

Anda dapat melakukan tindakan ini jika perlu melakukan banyak kueri HiveQL, tetapi tidak ingin menggunakan kapasitas throughput yang disediakan dari DynamoDB. Karena data dalam tabel Hive asli adalah salinan data dari DynamoDB, dan bukan data “langsung”, kueri Anda seharusnya tidak mengharapkan data tersebut. up-to-date

> **Note**
>
> Contoh di bagian ini ditulis dengan asumsi Anda mengikuti langkah-langkah di Tutorial: Menggunakan Amazon DynamoDB dan Apache Hive dan memiliki tabel eksternal di DynamoDB bernama ddb_features.

Example Dari DynamoDB ke tabel Hive asli

Anda dapat membuat tabel Hive asli dan mengisinya dengan data dari ddb_features, seperti ini:

```
CREATE TABLE features_snapshot AS
SELECT * FROM ddb_features;
```

Anda kemudian dapat menyegarkan data kapan saja:

```
INSERT OVERWRITE TABLE features_snapshot
SELECT * FROM ddb_features;
```

Dalam contoh ini, subkueri `SELECT * FROM ddb_features` akan mengambil semua data dari ddb_features. Jika hanya ingin menyalin subset data, Anda dapat menggunakan klausul `WHERE` dalam subkueri.

Contoh berikut membuat tabel Hive asli, yang hanya berisi beberapa atribut untuk danau dan puncak:

```
CREATE TABLE lakes_and_summits AS
SELECT feature_name, feature_class, state_alpha
FROM ddb_features
WHERE feature_class IN ('Lake','Summit');
```

Example Dari tabel Hive asli ke DynamoDB

Gunakan pernyataan HiveQL berikut untuk menyalin data dari tabel Hive asli ke ddb_features:

```
INSERT OVERWRITE TABLE ddb_features
SELECT * FROM features_snapshot;
```

Menyalin data antara DynamoDB dan Amazon S3

Jika memiliki data dalam tabel DynamoDB, Anda dapat menggunakan Hive untuk menyalin data tersebut ke bucket Amazon S3.

Anda dapat melakukannya jika ingin membuat arsip data dalam tabel DynamoDB Anda. Misalnya, anggaplah lingkungan pengujian Anda mengharuskan Anda menggunakan kumpulan data pengujian dasar di DynamoDB. Anda dapat menyalin data dasar ke bucket Amazon S3, lalu menjalankan pengujian. Setelah itu, Anda dapat mengatur ulang lingkungan pengujian dengan memulihkan data dasar dari bucket Amazon S3 ke DynamoDB.

Jika Anda bekerja menggunakan Tutorial: Menggunakan Amazon DynamoDB dan Apache Hive, maka Anda sudah memiliki bucket Amazon S3 yang berisi log Amazon EMR. Anda dapat menggunakan bucket ini untuk contoh di bagian ini, jika Anda tahu jalur root untuk bucket:

1. Buka konsol EMR Amazon di https://console.aws.amazon.com /emr.

2. Untuk Nama, pilih klaster Anda.
3. URI tercantum di URI Log di bagian Detail Konfigurasi.
4. Membuat catatan tentang jalur root bucket. Konvensi penamaan adalah:

   `s3://aws-logs-`*`accountID`*`-`*`region`*

   di *`accountID`* mana ID AWS akun Anda dan wilayah adalah AWS wilayah untuk bucket.

> **Note**
>
> Untuk contoh ini, kita akan menggunakan subpath dalam bucket, seperti dalam contoh ini: `s3://aws-logs-123456789012-us-west-2/hive-test`

Prosedur berikut ditulis dengan asumsi Anda mengikuti langkah-langkah di tutorial dan memiliki tabel eksternal di DynamoDB bernama ddb_features.

Topik

- Menyalin data menggunakan format default Hive
- Menyalin data dengan format yang ditentukan pengguna
- Menyalin data tanpa pemetaan kolom
- Melihat Data di Amazon S3

Menyalin data menggunakan format default Hive

Example Dari DynamoDB ke Amazon S3

Gunakan pernyataan `INSERT OVERWRITE` untuk menulis langsung ke Amazon S3.

```
INSERT OVERWRITE DIRECTORY 's3://aws-logs-123456789012-us-west-2/hive-test'
SELECT * FROM ddb_features;
```

File data di Amazon S3 terlihat seperti ini:

```
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
```

Setiap bidang dipisahkan oleh karakter SOH (awal mulai, 0x01). Di file, SOH muncul sebagai **^A**.

Example Dari Amazon S3 ke DynamoDB

1. Buat tabel eksternal yang menunjuk ke data yang belum diformat di Amazon S3.

```
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. Salin data ke DynamoDB.

```
INSERT OVERWRITE TABLE ddb_features
SELECT * FROM s3_features_unformatted;
```

Menyalin data dengan format yang ditentukan pengguna

Jika ingin menentukan karakter pemisah bidang Anda sendiri, Anda dapat membuat tabel eksternal yang dipetakan ke bucket Amazon S3. Anda dapat menggunakan teknik ini untuk membuat file data dengan nilai yang dipisahkan oleh koma (CSV).

Example Dari DynamoDB ke Amazon S3

1. Buat tabel eksternal Hive yang dipetakan ke Amazon S3. Jika Anda melakukannya, pastikan jenis data konsisten dengan jenis data yang ada di tabel eksternal DynamoDB.

```
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. Salin data dari DynamoDB.

```
INSERT OVERWRITE TABLE s3_features_csv
SELECT * FROM ddb_features;
```

File data di Amazon S3 terlihat seperti ini:

```
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 Dari Amazon S3 ke DynamoDB

Dengan pernyataan HiveQL tunggal, Anda dapat mengisi tabel DynamoDB menggunakan data dari Amazon S3:

```
INSERT OVERWRITE TABLE ddb_features
SELECT * FROM s3_features_csv;
```

Menyalin data tanpa pemetaan kolom

Anda dapat menyalin data dari DynamoDB dalam format mentah dan menuliskannya ke Amazon S3 tanpa menentukan jenis data atau pemetaan kolom. Anda dapat menggunakan metode ini untuk membuat arsip data DynamoDB dan menyimpannya di Amazon S3.

Example Dari DynamoDB ke Amazon S3

1. Buat tabel eksternal yang terkait dengan tabel DynamoDB Anda. (Tidak ada `dynamodb.column.mapping` dalam pernyataan HiveQL ini.)

```
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. Buat tabel eksternal lain yang terkait dengan bucket Amazon S3 Anda.

```
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. Salin data dari DynamoDB ke Amazon S3.

```
INSERT OVERWRITE TABLE s3_features_no_mapping
SELECT * FROM ddb_features_no_mapping;
```

File data di Amazon S3 terlihat seperti ini:

```
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
```

Setiap bidang diawali dengan karakter STX (awal teks, 0x02) dan diakhiri dengan karakter ETX (akhir teks, 0x03). Dalam file tersebut, STX muncul sebagai **^B** dan ETX muncul sebagai **^C**.

Example Dari Amazon S3 ke DynamoDB

Dengan pernyataan HiveQL tunggal, Anda dapat mengisi tabel DynamoDB menggunakan data dari Amazon S3:

```
INSERT OVERWRITE TABLE ddb_features_no_mapping
```

```
SELECT * FROM s3_features_no_mapping;
```

Melihat Data di Amazon S3

Jika menggunakan SSH untuk menghubungkan ke simpul pemimpin, Anda dapat menggunakan AWS Command Line Interface (AWS CLI) untuk mengakses data yang ditulis Hive ke Amazon S3.

Langkah-langkah berikut ditulis dengan asumsi Anda telah menyalin data dari DynamoDB ke Amazon S3 menggunakan salah satu prosedur di bagian ini.

1. Jika saat ini Anda berada di prompt perintah Hive, keluarlah ke prompt perintah Linux.

   ```
   hive> exit;
   ```

2. Cantumkan konten direktori hive-test di bucket Amazon S3 Anda. (Di sinilah Hive menyalin data dari DynamoDB.)

   ```
   aws s3 ls s3://aws-logs-123456789012-us-west-2/hive-test/
   ```

   Responsnya akan tampak mirip dengan ini:

   ```
   2016-11-01 23:19:54 81983 000000_0
   ```

   Nama file (000000_0) dihasilkan sistem.

3. (Opsional) Anda dapat menyalin file data dari Amazon S3 ke sistem file lokal di simpul pemimpin. Setelah melakukannya, Anda dapat menggunakan utilitas baris perintah Linux standar untuk menggunakan data dalam file.

   ```
   aws s3 cp s3://aws-logs-123456789012-us-west-2/hive-test/000000_0 .
   ```

   Responsnya akan tampak mirip dengan ini:

   ```
   download: s3://aws-logs-123456789012-us-west-2/hive-test/000000_0
   to ./000000_0
   ```

   > **Note**
   >
   > Kapasitas sistem file lokal di simpul pemimpin terbatas. Jangan gunakan perintah ini dengan file yang lebih besar dari ruang yang tersedia di sistem file lokal.

Menyalin data antara DynamoDB dan HDFS

Jika memiliki data di tabel DynamoDB, Anda dapat menggunakan Hive untuk menyalin data ke Hadoop Distributed File System (HDFS).

Anda dapat melakukan ini jika Anda menjalankan MapReduce pekerjaan yang membutuhkan data dari DynamoDB. Jika Anda menyalin data dari DynamoDB ke HDFS, Hadoop dapat memprosesnya, menggunakan semua simpul yang tersedia di klaster Amazon EMR secara paralel. Ketika MapReduce pekerjaan selesai, Anda kemudian dapat menulis hasil dari HDFS ke DDB.

Dalam contoh berikut, Hive akan membaca dari dan menulis ke direktori HDFS berikut: `/user/hadoop/hive-test`

> **Note**
>
> Contoh di bagian ini ditulis dengan asumsi Anda mengikuti langkah-langkah di Tutorial: Menggunakan Amazon DynamoDB dan Apache Hive dan memiliki tabel eksternal di DynamoDB bernama ddb_features.

Topik

- Menyalin data menggunakan format default Hive
- Menyalin data dengan format yang ditentukan pengguna
- Menyalin data tanpa pemetaan kolom
- Mengakses data di HDFS

Menyalin data menggunakan format default Hive

Example Dari DynamoDB ke HDFS

Gunakan pernyataan `INSERT OVERWRITE` untuk menulis langsung ke HDFS.

```
INSERT OVERWRITE DIRECTORY 'hdfs:///user/hadoop/hive-test'
SELECT * FROM ddb_features;
```

File data di HDFS tampak seperti ini:

```
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
```

Setiap bidang dipisahkan oleh karakter SOH (awal mulai, 0x01). Di file, SOH muncul sebagai **^A**.

Example Dari HDFS ke DynamoDB

1. Buat tabel eksternal yang memetakan data yang tidak diformat di HDFS.

```
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. Salin data ke DynamoDB.

```
INSERT OVERWRITE TABLE ddb_features
SELECT * FROM hdfs_features_unformatted;
```

Menyalin data dengan format yang ditentukan pengguna

Jika ingin menggunakan karakter pemisah bidang lain, Anda dapat membuat tabel eksternal yang dipetakan ke direktori HDFS. Anda dapat menggunakan teknik ini untuk membuat file data dengan nilai yang dipisahkan oleh koma (CSV).

Example Dari DynamoDB ke HDFS

1. Buat tabel eksternal Hive yang dipetakan ke HDFS. Jika Anda melakukannya, pastikan jenis data konsisten dengan jenis data yang ada di tabel eksternal DynamoDB.

```
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. Salin data dari DynamoDB.

```
INSERT OVERWRITE TABLE hdfs_features_csv
SELECT * FROM ddb_features;
```

File data di HDFS tampak seperti ini:

```
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 Dari HDFS ke DynamoDB

Dengan pernyataan HiveQL tunggal, Anda dapat mengisi tabel DynamoDB menggunakan data dari HDFS:

```
INSERT OVERWRITE TABLE ddb_features
SELECT * FROM hdfs_features_csv;
```

Menyalin data tanpa pemetaan kolom

Anda dapat menyalin data dari DynamoDB dalam format mentah dan menuliskannya ke HDFS tanpa menentukan jenis data atau pemetaan kolom. Anda dapat menggunakan metode ini untuk membuat arsip data DynamoDB dan menyimpannya di HDFS.

> **Note**
>
> Jika tabel DynamoDB Anda berisi atribut jenis Peta, Daftar, Boolean, atau Null, maka ini adalah satu-satunya cara Anda dapat menggunakan Hive untuk menyalin data dari DynamoDB ke HDFS.

Example Dari DynamoDB ke HDFS

1. Buat tabel eksternal yang terkait dengan tabel DynamoDB Anda. (Tidak ada `dynamodb.column.mapping` dalam pernyataan HiveQL ini.)

```
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. Buat tabel eksternal lain yang terkait dengan direktori HDFS Anda.

```
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. Salin data dari DynamoDB ke HDFS.

```
INSERT OVERWRITE TABLE hdfs_features_no_mapping
SELECT * FROM ddb_features_no_mapping;
```

File data di HDFS tampak seperti ini:

```
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
```

Setiap bidang diawali dengan karakter STX (awal teks, 0x02) dan diakhiri dengan karakter ETX (akhir teks, 0x03). Dalam file tersebut, STX muncul sebagai **^B** dan ETX muncul sebagai **^C**.

Example Dari HDFS ke DynamoDB

Dengan pernyataan HiveQL tunggal, Anda dapat mengisi tabel DynamoDB menggunakan data dari HDFS:

```
INSERT OVERWRITE TABLE ddb_features_no_mapping
SELECT * FROM hdfs_features_no_mapping;
```

Mengakses data di HDFS

HDFS adalah sistem file terdistribusi, dapat diakses oleh semua simpul dalam klaster Amazon EMR. Jika menggunakan SSH untuk menghubungkan ke simpul pemimpin, Anda dapat menggunakan alat baris perintah untuk mengakses data yang ditulis Hive ke HDFS.

HDFS tidak sama dengan sistem file lokal di simpul pemimpin. Anda tidak dapat menggunakan file dan direktori di HDFS menggunakan perintah Linux standar (seperti `cat`, `cp`, `mv`, atau `rm`). Sebaliknya, Anda melakukan tugas-tugas ini menggunakan perintah `hadoop fs`.

Langkah-langkah berikut ditulis dengan asumsi Anda telah menyalin data dari DynamoDB ke HDFS menggunakan salah satu prosedur di bagian ini.

1. Jika saat ini Anda berada di prompt perintah Hive, keluarlah ke prompt perintah Linux.

   ```
   hive> exit;
   ```

2. Daftar isi direktori the /user/hadoop/hive -test di HDFS. (Di sinilah Hive menyalin data dari DynamoDB.)

   ```
   hadoop fs -ls /user/hadoop/hive-test
   ```

   Responsnya akan tampak mirip dengan ini:

   ```
   Found 1 items
   -rw-r--r-- 1 hadoop hadoop 29504 2016-06-08 23:40 /user/hadoop/hive-test/000000_0
   ```

   Nama file (000000_0) dihasilkan sistem.

3. Lihat konten file:

   ```
   hadoop fs -cat /user/hadoop/hive-test/000000_0
   ```

> **Note**
>
> Dalam contoh ini, file relatif kecil (sekitar 29 KB). Hati-hati saat Anda menggunakan perintah ini dengan file yang sangat besar atau berisi karakter yang tidak dapat dicetak.

4. (Opsional) Anda dapat menyalin file data dari HDFS ke sistem file lokal di simpul pemimpin. Setelah melakukannya, Anda dapat menggunakan utilitas baris perintah Linux standar untuk menggunakan data dalam file.

```
hadoop fs -get /user/hadoop/hive-test/000000_0
```

Perintah ini tidak akan menimpa file.

> **Note**
>
> Kapasitas sistem file lokal di simpul pemimpin terbatas. Jangan gunakan perintah ini dengan file yang lebih besar dari ruang yang tersedia di sistem file lokal.

Menggunakan kompresi data

Saat Anda menggunakan Hive untuk menyalin data di antara sumber data yang berbeda, Anda dapat meminta kompresi on-the-fly data. Hive menyediakan beberapa codec kompresi. Anda dapat memilih salah satu selama sesi Hive Anda. Jika Anda melakukannya, data dikompres dalam format yang ditentukan.

Contoh berikut memampatkan data menggunakan algoritma Lempel-Ziv-Oberhumer (LZO).

```
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;
```

File yang dihasilkan di Amazon S3 akan memiliki nama yang dihasilkan sistem dengan `.lzo` di bagian akhirnya (misalnya, `8d436957-57ba-4af7-840c-96c2fc7bb6f5-000000.lzo`).

Codec kompresi yang tersedia adalah:

- `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`

Membaca data karakter UTF-8 yang tidak dapat dicetak

Untuk membaca dan menulis data karakter UTF-8 yang tidak dapat dicetak, Anda dapat menggunakan klausul `STORED AS SEQUENCEFILE` saat membuat tabel Hive. A SequenceFile adalah format file biner Hadoop. Anda perlu menggunakan Hadoop untuk membaca file ini. Contoh berikut menunjukkan cara mengekspor data dari DynamoDB ke Amazon S3. Anda dapat menggunakan fungsionalitas ini untuk menangani karakter berenkode UTF-8 yang tidak dapat dicetak.

```
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;
```

Penyempurnaan performa

Saat membuat tabel eksternal Hive yang dipetakan ke tabel DynamoDB, Anda tidak menggunakan kapasitas baca atau tulis dari DynamoDB. Namun, aktivitas baca dan tulis pada tabel Hive (seperti `INSERT` atau `SELECT`) diterjemahkan langsung menjadi operasi baca dan tulis pada tabel DynamoDB yang mendasarinya.

Apache Hive di Amazon EMR mengimplementasikan logikanya sendiri untuk menyeimbangkan beban I/O pada tabel DynamoDB dan berupaya meminimalkan kemungkinan melebihi throughput

tabel yang disediakan. Di akhir setiap kueri Hive, Amazon EMR mengembalikan metrik runtime, termasuk frekuensi throughput yang Anda sediakan terlampaui. Anda dapat menggunakan informasi ini, bersama dengan CloudWatch metrik pada tabel DynamoDB Anda, untuk meningkatkan kinerja dalam permintaan berikutnya.

Konsol Amazon EMR menyediakan alat pemantauan dasar untuk klaster Anda. Untuk informasi selengkapnya, lihat Melihat dan Memantau klaster di Panduan Manajemen Amazon EMR.

Anda juga dapat memantau performa klaster dan Hadoop menggunakan alat berbasis web, seperti Hue, Ganglia, dan antarmuka web Hadoop. Untuk informasi selengkapnya, lihat Melihat Antarmuka Web yang Di-hosting di Klaster Amazon EMR di Panduan Manajemen Amazon EMR.

Bagian ini menjelaskan langkah-langkah yang dapat Anda ambil untuk operasi Hive penyempurnaan performa pada tabel DynamoDB eksternal.

Topik

- Throughput yang disediakan DynamoDB
- Menyesuaikan pemeta
- Topik tambahan

Throughput yang disediakan DynamoDB

Saat Anda mengeluarkan pernyataan HiveQL terhadap tabel DynamoDB eksternal, kelas `DynamoDBStorageHandler` membuat permintaan DynamoDB API level rendah yang sesuai, yang menggunakan throughput yang disediakan. Jika kapasitas baca atau tulis pada tabel DynamoDB tidak memadai, permintaan akan mengalami throttling, sehingga performa HiveQL menjadi lambat. Oleh karena itu, Anda harus memastikan bahwa tabel tersebut memiliki kapasitas throughput yang memadai.

Misalnya, anggaplah bahwa Anda telah menetapkan 100 unit kapasitas baca untuk tabel DynamoDB Anda. Ini akan memungkinkan Anda membaca 409.600 byte per detik (100 × 4 KB ukuran unit kapasitas baca). Sekarang, anggaplah bahwa tabel berisi 20 GB data (21.474.836.480 byte) dan Anda ingin menggunakan pernyataan `SELECT` untuk memilih semua data menggunakan HiveQL. Anda dapat memperkirakan durasi yang diperlukan untuk menjalankan kueri seperti ini:

21.474.836.480 / 409.600 = 52.429 detik = 14,56 jam

Dalam skenario ini, tabel DynamoDB adalah kemacetan. Menambahkan lebih banyak simpul Amazon EMR tidak akan membantu, karena throughput Hive dibatasi hanya 409.600 byte per detik. Satu-

satunya cara untuk mengurangi waktu yang dibutuhkan untuk pernyataan SELECT adalah dengan meningkatkan kapasitas baca tabel DynamoDB yang disediakan.

Anda dapat melakukan penghitungan serupa untuk memperkirakan durasi yang diperlukan untuk memuat data secara massal ke dalam tabel eksternal Hive yang dipetakan ke tabel DynamoDB. Tentukan jumlah total unit kapasitas tulis yang dibutuhkan per item (kurang dari 1KB = 1, 1-2KB = 2, dll), dan kalikan dengan jumlah item yang akan dimuat. Ini akan menghasilkan jumlah unit kapasitas tulis yang diperlukan. Bagi angka tersebut dengan jumlah alokasi unit kapasitas tulis per detik. Ini akan menghasilkan jumlah detik yang dibutuhkan untuk memuat tabel.

Anda harus secara teratur memantau CloudWatch metrik untuk tabel Anda. Untuk gambaran umum singkat di konsol DynamoDB, pilih tabel Anda lalu pilih tab Metrik. Dari sini, Anda dapat melihat unit kapasitas baca dan tulis yang dikonsumsi dan permintaan baca dan tulis yang mengalami throttling.

Kapasitas baca

Amazon EMR mengelola beban permintaan terhadap tabel DynamoDB Anda, sesuai dengan pengaturan throughput yang ditetapkan tabel ini. Namun, jika Anda melihat sejumlah besar pesan `ProvisionedThroughputExceeded` dalam output tugas, Anda dapat menyesuaikan tingkat baca default. Untuk melakukannya, Anda dapat memodifikasi variabel konfigurasi `dynamodb.throughput.read.percent`. Anda dapat menggunakan perintah SET untuk mengatur variabel ini pada prompt perintah Hive:

```
SET dynamodb.throughput.read.percent=1.0;
```

Variabel ini hanya berlaku untuk sesi Hive saat ini. Jika Anda keluar dari Hive dan kembali lagi setelah beberapa saat, `dynamodb.throughput.read.percent` akan kembali ke nilai default.

Nilai `dynamodb.throughput.read.percent` bisa antara `0.1` dan `1.5`, secara inklusif. `0.5` mewakili tingkat baca default, yang berarti bahwa Hive akan mencoba untuk menggunakan setengah dari kapasitas baca tabel. Jika Anda meningkatkan nilai di atas `0.5`, Hive akan meningkatkan tingkat permintaan; menurunkan nilai di bawah `0.5` akan menurunkan tingkat permintaan baca. (Tingkat baca sebenarnya akan bervariasi, tergantung pada faktor-faktor seperti apakah ada distribusi kunci seragam dalam tabel DynamoDB.)

Jika Anda memperhatikan bahwa Hive sering menghabiskan kapasitas baca tabel yang disediakan, atau jika permintaan baca Anda terlalu dibatasi, cobalah mengurangi `dynamodb.throughput.read.percent` di bawah `0.5`. Jika Anda memiliki kapasitas baca yang

memadai dalam tabel dan ingin operasi HiveQL lebih responsif, Anda dapat mengatur nilai di atas `0.5`.

Kapasitas tulis

Amazon EMR mengelola beban permintaan terhadap tabel DynamoDB Anda, sesuai dengan pengaturan throughput yang ditetapkan tabel ini. Namun, jika Anda melihat sejumlah besar pesan `ProvisionedThroughputExceeded` dalam output tugas, Anda dapat menyesuaikan tingkat tulis default. Untuk melakukannya, Anda dapat memodifikasi variabel konfigurasi `dynamodb.throughput.write.percent`. Anda dapat menggunakan perintah `SET` untuk mengatur variabel ini pada prompt perintah Hive:

```
SET dynamodb.throughput.write.percent=1.0;
```

Variabel ini hanya berlaku untuk sesi Hive saat ini. Jika Anda keluar dari Hive dan kembali lagi setelah beberapa saat, `dynamodb.throughput.write.percent` akan kembali ke nilai default.

Nilai `dynamodb.throughput.write.percent` bisa antara `0.1` dan `1.5`, secara inklusif. `0.5` mewakili tingkat tulis default, yang berarti bahwa Hive akan mencoba untuk menggunakan setengah dari kapasitas tulis tabel. Jika Anda meningkatkan nilai di atas `0.5`, Hive akan meningkatkan tingkat permintaan; menurunkan nilai di bawah `0.5` akan menurunkan tingkat permintaan tulis. (Tingkat tulis sebenarnya akan bervariasi, tergantung pada faktor-faktor seperti apakah ada distribusi kunci seragam dalam tabel DynamoDB.)

Jika Anda memperhatikan bahwa Hive sering menghabiskan kapasitas tulis tabel yang disediakan, atau jika permintaan tulis Anda terlalu dibatasi, cobalah mengurangi `dynamodb.throughput.write.percent` di bawah `0.5`. Jika Anda memiliki kapasitas yang memadai dalam tabel dan ingin operasi HiveQL lebih responsif, Anda dapat mengatur nilai di atas `0.5`.

Ketika Anda menulis data ke DynamoDB menggunakan Hive, pastikan jumlah unit kapasitas tulis lebih besar dari jumlah pemeta di klaster. Sebagai contoh, pertimbangkan sebuah klaster Amazon EMR yang terdiri dari 10 simpul m1.xlarge. Jenis simpul m1.xlarge menyediakan 8 tugas pemeta, sehingga klaster akan memiliki total 80 pemeta (10 × 8). Jika tabel DynamoDB Anda memiliki kurang dari 80 unit kapasitas tulis, maka operasi tulis Hive mungkin akan menggunakan semua throughput tulis untuk tabel itu.

Untuk menentukan jumlah pemeta untuk jenis simpul Amazon EMR, lihat Konfigurasi Tugas di Panduan Developer Amazon EMR.

Untuk informasi selengkapnya tentang pemeta, lihat Menyesuaikan pemeta.

Menyesuaikan pemeta

Ketika Hive meluncurkan tugas Hadoop, tugas tersebut diproses oleh satu atau beberapa tugas pemeta. Dengan asumsi bahwa tabel DynamoDB Anda memiliki kapasitas throughput yang memadai, Anda dapat mengubah jumlah pembuat peta di klaster, sehingga berpotensi meningkatkan performa.

> **Note**
>
> Jumlah tugas pemeta yang digunakan dalam tugas Hadoop dipengaruhi oleh pembagian input, yaitu Hadoop membagi data menjadi beberapa blok logis. Jika Hadoop tidak melakukan pemisahan input yang cukup, maka operasi tulis Anda mungkin tidak dapat menggunakan semua throughput tulis yang tersedia di tabel DynamoDB.

Meningkatkan jumlah pemeta

Setiap pemeta dalam Amazon EMR memiliki tingkat baca maksimum 1 MiB per detik. Jumlah pemeta di klaster tergantung pada ukuran simpul di klaster Anda. (Untuk informasi tentang ukuran simpul dan jumlah pemeta per simpul, lihat Konfigurasi Tugas di Panduan Developer Amazon EMR.)

Jika tabel DynamoDB Anda memiliki kapasitas throughput yang memadai untuk baca, Anda dapat mencoba meningkatkan jumlah pemeta dengan melakukan salah satu dari berikut ini:

- Tingkatkan ukuran simpul dalam klaster. Sebagai contoh, jika klaster Anda menggunakan simpul m1.large (tiga pemeta per simpul), Anda dapat mencoba meningkatkan ke simpul m1.xlarge (delapan pemeta per simpul).
- Tingkatkan jumlah simpul dalam klaster. Misalnya, jika Anda memiliki klaster tiga simpul untuk simpul m1.xlarge, Anda memiliki total 24 pemeta yang tersedia. Jika Anda menggandakan ukuran klaster, dengan jenis simpul yang sama, Anda akan memiliki 48 pemeta.

Anda dapat menggunakan AWS Management Console untuk mengelola ukuran atau jumlah node di cluster Anda. (Anda mungkin perlu memulai ulang klaster untuk menerapkan perubahan ini.)

Cara lain untuk meningkatkan jumlah pemeta adalah dengan mengubah konfigurasi parameter Hadoop `mapred.tasktracker.map.tasks.maximum`. (Ini adalah parameter Hadoop, bukan

parameter Hive. Anda tidak dapat mengubahnya secara interaktif dari prompt perintah). Jika meningkatkan nilai `mapred.tasktracker.map.tasks.maximum`, Anda dapat meningkatkan jumlah pemeta tanpa meningkatkan ukuran atau jumlah simpul. Namun, simpul klaster kemungkinan bisa kehabisan memori jika Anda menetapkan nilai terlalu tinggi.

Anda menetapkan nilai untuk `mapred.tasktracker.map.tasks.maximum` sebagai tindakan bootstrap ketika Anda pertama kali meluncurkan klaster Amazon EMR. Untuk informasi selengkapnya, lihat (Opsional) Membuat Tindakan Bootstrap untuk Menginstal Perangkat Lunak Tambahan di Panduan Pengelolaan Amazon EMR.

Menurunkan jumlah pemeta

Jika Anda menggunakan pernyataan `SELECT` untuk memilih data dari tabel Hive eksternal yang dipetakan ke DynamoDB, tugas Hadoop dapat menggunakan tugas sebanyak yang diperlukan, hingga jumlah maksimum pemeta di klaster. Dalam skenario ini, ada kemungkinan bahwa kueri Hive yang berjalan lama dapat menggunakan semua kapasitas baca tabel DynamoDB yang disediakan, sehingga berdampak negatif pada pengguna lain.

Anda dapat menggunakan parameter `dynamodb.max.map.tasks` untuk menetapkan batas atas tugas peta:

```
SET dynamodb.max.map.tasks=1
```

Nilai ini harus lebih besar atau sama dengan 1. Saat Hive memproses kueri Anda, tugas Hadoop yang dihasilkan tidak akan menggunakan lebih dari `dynamodb.max.map.tasks` saat membaca dari tabel DynamoDB.

Topik tambahan

Berikut ini adalah beberapa cara untuk menyetel aplikasi yang menggunakan Hive untuk mengakses DynamoDB.

Durasi percobaan ulang

Secara default, Hive akan menjalankan kembali tugas Hadoop jika sama sekali belum mengembalikan hasil apa pun dari DynamoDB dalam waktu dua menit. Anda dapat menyesuaikan interval ini dengan mengubah parameter `dynamodb.retry.duration`:

```
SET dynamodb.retry.duration=2;
```

Nilainya harus berupa angka bulat selain nol, yang mewakili jumlah menit dalam interval percobaan ulang. Default untuk `dynamodb.retry.duration` adalah 2 (menit).

Permintaan data paralel

Beberapa permintaan data, baik dari lebih dari satu pengguna atau lebih dari satu aplikasi ke satu tabel dapat menguras throughput baca yang disediakan dan memperlambat performa.

Durasi proses

Konsistensi data di DynamoDB bergantung pada urutan operasi baca dan tulis pada setiap simpul. Saat kueri Hive sedang berjalan, aplikasi lain mungkin memuat data baru ke dalam tabel DynamoDB atau memodifikasi atau menghapus data yang ada. Dalam kasus ini, hasil kueri Hive mungkin tidak mencerminkan perubahan yang dilakukan pada data saat kueri sedang berjalan.

Waktu permintaan

Menjadwalkan kueri Hive yang mengakses tabel DynamoDB ketika permintaan pada tabel DynamoDB rendah akan meningkatkan performa. Misalnya, jika sebagian besar pengguna aplikasi Anda tinggal di San Francisco, Anda dapat memilih untuk mengekspor data harian pada pukul 04.00 PST ketika sebagian besar pengguna tertidur dan tidak memperbarui catatan di basis data DynamoDB Anda.

Mengintegrasikan DynamoDB dengan Amazon S3

Kemampuan impor dan ekspor Amazon DynamoDB memberikan cara sederhana dan efisien untuk memindahkan data antara tabel Amazon S3 dan DynamoDB tanpa menulis kode apa pun.

Fitur impor dan ekspor DynamoDB membantu Anda memindahkan, mengubah, dan menyalin akun tabel DynamoDB. Anda dapat mengimpor dari sumber S3, dan Anda dapat mengekspor data tabel DynamoDB ke Amazon S3 dan menggunakan layanan AWS seperti Athena, SageMaker Amazon AI, dan untuk menganalisis data Anda serta mengekstrak wawasan yang dapat ditindaklanjuti. AWS Lake Formation Anda juga dapat mengimpor data langsung ke tabel DynamoDB baru untuk membangun aplikasi baru dengan performa satu digit milidetik dalam skala besar, memfasilitasi berbagi data antara tabel dan akun, serta menyederhanakan rencana pemulihan bencana dan kelangsungan bisnis Anda.

Topik

- Impor data DynamoDB dari Amazon S3: cara kerjanya
- Ekspor data DynamoDB ke Amazon S3: cara kerjanya

Impor data DynamoDB dari Amazon S3: cara kerjanya

Untuk mengimpor data ke DynamoDB, data Anda harus berada dalam bucket Amazon S3 dalam format CSV, DynamoDB JSON, atau Amazon Ion. Data dapat dikompresi dalam format ZSTD atau GZIP, atau dapat langsung diimpor dalam bentuk tidak terkompresi. Sumber data dapat berupa satu objek Amazon S3 atau beberapa objek Amazon S3 yang menggunakan prefiks yang sama.

Data Anda akan diimpor ke tabel DynamoDB baru, yang akan dibuat saat Anda memulai permintaan impor. Anda dapat membuat tabel ini dengan indeks sekunder, lalu mengkueri dan memperbarui data Anda di seluruh indeks primer dan sekunder segera setelah impor selesai. Anda juga dapat menambahkan replika tabel global setelah impor selesai.

> **Note**
>
> Selama proses impor Amazon S3, DynamoDB membuat tabel target baru yang akan diimpor. Impor ke tabel yang ada saat ini tidak didukung oleh fitur ini.

Impor dari Amazon S3 tidak menggunakan kapasitas tulis pada tabel baru, sehingga Anda tidak perlu menyediakan kapasitas tambahan apa pun untuk mengimpor data ke DynamoDB. Harga impor data didasarkan pada ukuran data sumber yang tidak terkompresi di Amazon S3, yang diproses sebagai hasil impor. Item yang diproses namun gagal dimuat ke dalam tabel karena format atau ketidakkonsistenan lainnya dalam data sumber juga ditagih sebagai bagian dari proses impor. Lihat harga Amazon DynamoDB untuk detailnya.

Anda dapat mengimpor data dari bucket Amazon S3 yang dimiliki oleh akun berbeda jika Anda memiliki izin yang benar untuk membaca dari bucket tertentu. Tabel baru mungkin juga berada di Wilayah yang berbeda dari bucket Amazon S3 sumber. Untuk informasi selengkapnya, lihat Pengaturan dan izin Amazon Simple Storage Service.

Waktu impor berhubungan langsung dengan karakteristik data Anda di Amazon S3. Hal ini mencakup ukuran data, format data, skema kompresi, keseragaman distribusi data, jumlah objek Amazon S3, dan variabel terkait lainnya. Secara khusus, kumpulan data dengan kunci yang terdistribusi secara seragam akan lebih cepat diimpor dibandingkan kumpulan data yang miring. Misalnya, jika kunci indeks sekunder Anda menggunakan bulan dalam setahun untuk mempartisi, dan semua data Anda

berasal dari bulan Desember, maka mengimpor data ini mungkin memerlukan waktu yang jauh lebih lama.

Atribut yang terkait dengan kunci diharapkan unik pada tabel dasar. Jika ada kunci yang tidak unik, impor akan menimpa item terkait hingga hanya penimpaan terakhir yang tersisa. Misalnya, jika kunci primer adalah bulan dan beberapa item disetel ke bulan September, setiap item baru akan menimpa item yang ditulis sebelumnya dan hanya satu item dengan kunci primer "bulan" yang diatur ke bulan September yang akan tersisa. Dalam kasus tersebut, jumlah item yang diproses dalam deskripsi tabel impor tidak akan cocok dengan jumlah item dalam tabel target.

AWS CloudTrail mencatat semua tindakan konsol dan API untuk impor tabel. Untuk informasi selengkapnya, lihat Pencatatan log operasi DynamoDB menggunakan AWS CloudTrail.

Video berikut adalah pengantar untuk mengimpor langsung dari Amazon S3 ke DynamoDB.

Mengimpor dari Amazon S3

Topik

- Meminta impor tabel di DynamoDB
- Format impor Amazon S3 untuk DynamoDB
- Kuota format impor dan validasi
- Praktik terbaik untuk mengimpor dari Amazon S3 ke DynamoDB

Meminta impor tabel di DynamoDB

Impor DynamoDB memungkinkan Anda mengimpor data dari bucket Amazon S3 ke tabel DynamoDB baru. Anda dapat meminta impor tabel menggunakan konsol DynamoDB, CLI CloudFormation, atau DynamoDB API.

Jika Anda ingin menggunakan AWS CLI, Anda harus mengkonfigurasinya terlebih dahulu. Untuk informasi selengkapnya, lihat Mengakses DynamoDB.

> **Note**
>
> - Fitur Tabel Impor berinteraksi dengan beberapa AWS Layanan berbeda seperti Amazon CloudWatch S3 dan. Sebelum memulai impor, pastikan bahwa pengguna atau peran yang memanggil impor APIs memiliki izin untuk semua layanan dan sumber daya yang bergantung pada fitur tersebut.

- Jangan memodifikasi objek Amazon S3 saat impor sedang berlangsung, karena ini dapat menyebabkan operasi gagal atau dibatalkan.

Untuk informasi selengkapnya tentang kesalahan dan pemecahan masalah, lihat Kuota format impor dan validasi

Topik

- Menyiapkan izin Peran IAM
- Meminta impor menggunakan AWS Management Console
- Mendapatkan detail tentang impor sebelumnya di AWS Management Console
- Meminta impor menggunakan AWS CLI
- Mendapatkan detail tentang impor sebelumnya di AWS CLI

Menyiapkan izin Peran IAM

Anda dapat mengimpor data dari bucket Amazon S3 mana pun yang izin bacanya Anda miliki. Bucket sumber tidak harus berada di Wilayah yang sama atau memiliki pemilik yang sama dengan tabel sumber. AWS Identity and Access Management (IAM) Anda harus menyertakan tindakan yang relevan pada bucket Amazon S3 sumber, dan izin yang CloudWatch diperlukan untuk memberikan informasi debugging. Contoh kebijakan ditunjukkan di bawah ini.

```
{
  "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": "*"
    }
  ]
}
```

Izin Amazon S3

Saat memulai impor pada sumber bucket Amazon S3 yang dimiliki oleh akun lain, pastikan bahwa peran atau pengguna memiliki akses ke objek Amazon S3. Anda dapat memeriksanya dengan menjalankan perintah `GetObject` Amazon S3 dan menggunakan kredensial. Saat menggunakan API, parameter pemilik bucket Amazon S3 defaultnya adalah ID akun pengguna saat ini. Untuk impor lintas akun, pastikan parameter ini diisi dengan benar dengan ID akun pemilik bucket. Kode berikut adalah contoh kebijakan bucket Amazon S3 di akun sumber.

```
{
    "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

Saat membuat tabel baru untuk impor, jika Anda memilih enkripsi pada kunci istirahat yang tidak dimiliki oleh DynamoDB maka Anda harus memberikan AWS KMS izin yang diperlukan untuk mengoperasikan tabel DynamoDB yang dienkripsi dengan kunci yang dikelola pelanggan. Untuk informasi selengkapnya, lihat Mengotorisasi penggunaan AWS KMS kunci Anda. Jika objek Amazon S3 dienkripsi dengan enkripsi sisi server KMS (SSE-KMS), pastikan bahwa peran atau pengguna yang memulai impor memiliki akses untuk mendekripsi menggunakan kunci. AWS KMS Fitur ini tidak mendukung kunci enkripsi yang disediakan pelanggan (SSE-C) yang mengenkripsi objek Amazon S3.

CloudWatch izin

Peran atau pengguna yang memulai impor perlu membuat dan mengelola izin untuk grup log dan log stream yang terkait dengan impor.

Meminta impor menggunakan AWS Management Console

Contoh berikut menunjukkan cara menggunakan konsol DynamoDB untuk mengimpor data yang ada ke tabel baru bernama `MusicCollection`.

Untuk meminta impor tabel

1. Masuk ke AWS Management Console dan buka konsol DynamoDB di. https://console.aws.amazon.com/dynamodb/
2. Di panel navigasi di sisi kiri konsol, pilih Impor dari S3.
3. Pada halaman yang muncul, pilih Impor dari S3.
4. Pilih Impor dari S3.

5. Di URL Sumber S3, masukkan URL sumber Amazon S3.

   Jika Anda memiliki bucket sumber, pilih Browse S3 untuk mencarinya. Atau, masukkan URL bucket dalam format berikut —`s3://bucket/prefix`. `prefix`Ini adalah awalan kunci Amazon S3. Ini adalah nama objek Amazon S3 yang ingin Anda impor atau awalan key yang dibagikan oleh semua objek Amazon S3 yang ingin Anda impor.

   > **Note**
   >
   > Anda tidak dapat menggunakan awalan yang sama dengan permintaan ekspor DynamoDB Anda. Fitur ekspor membuat struktur folder dan file manifes untuk semua ekspor. Jika Anda menggunakan jalur Amazon S3 yang sama, itu akan menghasilkan kesalahan.
   > Sebagai gantinya, arahkan impor ke folder, yang berisi data dari ekspor tertentu. Format jalur yang benar dalam hal ini `XXXXXXXX-XXXXXX` adalah`s3://bucket/prefix/AWSDynamoDB/<XXXXXXXX-XXXXXX>/Data/`, di mana ID ekspor. Anda dapat menemukan ID ekspor di ARN ekspor, yang memiliki format berikut —. `arn:aws:dynamodb:<Region>:<AccountID>:table/<TableName>/export/<XXXXXXXX-XXXXXX>` Misalnya, `arn:aws:dynamodb:`*`us-east-1`*`:`*`123456789012`*`:table/`*`ProductCatalog`*`/export/`*`01234567890123-a1b2c3d4`*.

6. Tentukan apakah Anda adalah pemilik bucket S3. Jika bucket sumber dimiliki oleh akun lain, pilih Akun yang berbeda AWS . Lalu masukkan ID akun pemilik bucket.
7. Di bagian Impor kompresi file, pilih Tanpa kompresi, GZIP, atau ZSTD yang sesuai.
8. Pilih format file Impor yang sesuai. Pilihannya adalah DynamoDB JSON, Amazon Ion, atau CSV. Jika Anda memilih CSV, Anda akan memiliki dua opsi tambahan: header CSV dan karakter pembatas CSV.

   Untuk header CSV, pilih apakah header akan diambil dari baris pertama file atau disesuaikan. Jika Anda memilih Sesuaikan header Anda, Anda dapat menentukan nilai header yang ingin Anda gunakan untuk mengimpor. Header CSV yang ditentukan dengan metode ini peka huruf besar-kecil dan diharapkan berisi kunci tabel target.

   Untuk Karakter pembatas CSV, Anda mengatur karakter yang akan memisahkan item. Koma dipilih secara default. Jika Anda memilih Karakter pembatas khusus, pembatas harus sesuai dengan pola ekspresi reguler: `[,;:|\t ]`.

9. Pilih tombol Berikutnya dan pilih opsi untuk tabel baru yang akan dibuat untuk menyimpan data Anda.

> **Note**
>
> Kunci Primer dan Kunci Urutan harus cocok dengan atribut dalam file, atau impor akan gagal. Atributnya peka huruf besar-kecil.

10. Pilih Berikutnya lagi untuk meninjau opsi impor Anda, lalu klik Impor untuk memulai tugas impor. Pertama-tama Anda akan melihat tabel baru Anda tercantum di “Tabel” dengan status “Membuat”. Saat ini tabel tersebut tidak dapat diakses.
11. Setelah impor selesai, statusnya akan ditampilkan sebagai "Aktif" dan Anda dapat mulai menggunakan tabel tersebut.

Mendapatkan detail tentang impor sebelumnya di AWS Management Console

Anda dapat menemukan informasi tentang tugas impor yang pernah Anda jalankan sebelumnya dengan mengklik Impor dari S3 di bar samping navigasi, lalu memilih tab Impor. Panel impor berisi daftar semua impor yang Anda buat dalam 90 hari terakhir. Memilih ARN tugas yang tercantum di tab Impor akan mengambil informasi tentang impor tersebut, termasuk pengaturan konfigurasi lanjutan yang Anda pilih.

Meminta impor menggunakan AWS CLI

Contoh berikut mengimpor data berformat CSV dari bucket S3 yang disebut bucket dengan prefiks dari prefiks ke tabel baru yang disebut tabel target.

```
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**
>
> Jika Anda memilih untuk mengenkripsi impor menggunakan kunci yang dilindungi oleh AWS Key Management Service (AWS KMS), kunci harus berada di Wilayah yang sama dengan bucket Amazon S3 tujuan.

Mendapatkan detail tentang impor sebelumnya di AWS CLI

Anda dapat menemukan informasi tentang tugas impor yang pernah Anda jalankan sebelumnya dengan menggunakan perintah `list-imports`. Perintah ini mengembalikan daftar semua impor yang Anda buat dalam 90 hari terakhir. Perhatikan bahwa meskipun metadata tugas impor akan kedaluwarsa setelah 90 hari dan tugas yang lebih lama dari itu tidak lagi ditemukan dalam daftar ini, DynamoDB tidak menghapus objek apa pun di bucket Amazon S3 Anda atau tabel yang dibuat selama impor.

```
aws dynamodb list-imports
```

Untuk mengambil informasi mendetail tentang tugas impor tertentu, termasuk pengaturan konfigurasi lanjutan, gunakan perintah `describe-import`.

```
aws dynamodb describe-import \
    --import-arn arn:aws:dynamodb:us-east-1:123456789012:table/ProductCatalog/exp
```

Format impor Amazon S3 untuk DynamoDB

DynamoDB dapat mengimpor data dalam tiga format: CSV, DynamoDB JSON, dan Amazon Ion.

Topik

- CSV
- DynamoDB Json
- Amazon Ion

CSV

File dalam format CSV terdiri dari beberapa item yang dibatasi oleh baris baru. Secara default, DynamoDB menafsirkan baris pertama dari file impor sebagai header dan mengharapkan kolom akan dibatasi oleh koma. Anda juga dapat menentukan header yang akan diterapkan, asalkan sesuai

dengan jumlah kolom dalam file. Jika Anda mendefinisikan header secara eksplisit, baris pertama file akan diimpor sebagai nilai.

> **Note**
>
> Saat mengimpor dari file CSV, semua kolom selain rentang hash dan kunci tabel dasar serta indeks sekunder Anda diimpor sebagai string DynamoDB.

Menghindari tanda kutip ganda

Setiap karakter tanda kutip ganda yang ada di file CSV harus dihindari. Jika tidak dihindari, seperti pada contoh berikut ini, impor akan gagal:

```
id,value
"123",Women's Full Lenth Dress
```

Impor yang sama ini akan berhasil jika tanda kutip dihindari dengan menambah dua rangkaian tanda kutip ganda:

```
id,value
"""123""",Women's Full Lenth Dress
```

Setelah teks dihindari dan diimpor dengan benar, teks tersebut akan muncul seperti di file CSV asli:

```
id,value
"123",Women's Full Lenth Dress
```

DynamoDB Json

File dalam format DynamoDB JSON dapat terdiri dari beberapa objek Item. Setiap objek individual berada dalam format JSON marshall standar DynamoDB, dan baris baru digunakan sebagai pembatas item. Sebagai fitur tambahan, ekspor dari titik waktu tertentu didukung sebagai sumber impor secara default.

> **Note**
>
> Baris baru digunakan sebagai pembatas item untuk file dalam format DynamoDB JSON dan tidak boleh digunakan dalam objek item.

```
[{
    "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"
        }
    }
}]
```

> **Note**
>
> Baris baru digunakan sebagai pembatas item untuk file dalam format DynamoDB JSON dan tidak boleh digunakan dalam objek item.

```
[{
    "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 adalah format serialisasi data hierarkis yang kaya jenis, dapat dijelaskan sendiri, yang dibuat untuk mengatasi tantangan perkembangan pesat, pemisahan, dan efisiensi yang dihadapi setiap hari saat merekayasa arsitektur berorientasi layanan berskala besar.

Saat Anda mengimpor data dalam format Ion, jenis data Ion dipetakan ke jenis data DynamoDB di tabel DynamoDB baru.

S. Tidak.	Konversi jenis data Ion ke 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`

Item dalam file Ion dibatasi oleh baris baru. Setiap baris dimulai dengan penanda versi Ion, diikuti dengan item dalam format Ion.

> **Note**
>
> Dalam contoh berikut, kami telah memformat item dari file berformat ion pada beberapa baris untuk meningkatkan keterbacaan.

```
$ion_1_0
[
  {
    Item:{
      Authors:$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"
    }
  }
]
```

Kuota format impor dan validasi

Kuota impor

Impor DynamoDB dari Amazon S3 dapat mendukung hingga 50 tugas impor bersamaan dengan total ukuran objek sumber impor sebesar 15 TB sekaligus di wilayah us-east-1, us-west-2, dan eu-west-1. Di seluruh wilayah lainnya, didukung hingga 50 tugas impor bersamaan dengan ukuran total 1 TB. Setiap pekerjaan impor dapat mengambil hingga 50.000 objek Amazon S3 di semua wilayah. Kuota default ini berlaku untuk setiap akun. Jika Anda merasa perlu merevisi kuota ini, silakan hubungi tim akun Anda, dan ini akan dipertimbangkan atas dasar. case-by-case Untuk detail selengkapnya tentang batasan DynamoDB, lihat Kuota Layanan.

Kesalahan validasi

Selama proses impor, DynamoDB mungkin mengalami kesalahan saat mengurai data Anda. Untuk setiap kesalahan, DynamoDB memancarkan CloudWatch log dan menyimpan hitungan jumlah total kesalahan yang ditemui. Jika objek Amazon S3 itu sendiri salah format atau jika isinya tidak dapat membentuk item DynamoDB, kami dapat melewatkan pemrosesan bagian objek yang tersisa.

> **Note**
>
> Jika sumber data Amazon S3 memiliki beberapa item yang berbagi kunci yang sama, item tersebut akan ditimpa hingga tersisa satu item. Hal ini dapat terlihat seolah-olah 1 item diimpor dan item lainnya diabaikan. Item duplikat akan ditimpa secara acak, tidak dihitung sebagai kesalahan, dan tidak dipancarkan ke log. CloudWatch
> Setelah impor selesai, Anda dapat melihat jumlah total item yang diimpor, jumlah total kesalahan, dan jumlah total item yang diproses. Untuk pemecahan masalah selengkapnya, Anda juga dapat memeriksa ukuran total item yang diimpor dan ukuran total data yang diproses.

Ada tiga kategori kesalahan impor: kesalahan validasi API, kesalahan validasi data, dan kesalahan konfigurasi.

Kesalahan validasi API

Kesalahan validasi API adalah kesalahan tingkat item dari API sinkronisasi. Penyebab umumnya adalah masalah izin, tidak adanya parameter yang diperlukan, dan kegagalan validasi parameter. Detail tentang alasan kegagalan panggilan API terdapat dalam pengecualian yang diberikan oleh permintaan `ImportTable`.

Kesalahan validasi data

Kesalahan validasi data dapat terjadi pada tingkat item atau tingkat file. Selama impor, item divalidasi berdasarkan aturan DynamoDB sebelum diimpor ke tabel target. Ketika item gagal validasi dan tidak diimpor, tugas impor akan melewati item tersebut dan melanjutkan ke item berikutnya. Di akhir pekerjaan, status impor disetel ke GAGAL dengan FailureCode, ItemValidationError dan FailureMessage “Beberapa item gagal memeriksa validasi dan tidak diimpor. Silakan periksa log CloudWatch kesalahan untuk lebih jelasnya.”

Penyebab umum kesalahan validasi data mencakup objek tidak dapat diurai, format objek salah (input menentukan DYNAMODB_JSON namun objek tidak ada di DYNAMODB_JSON), dan skema tidak cocok dengan kunci tabel sumber yang ditentukan.

Kesalahan konfigurasi

Kesalahan konfigurasi biasanya merupakan kesalahan alur kerja karena validasi izin. Alur kerja Impor memeriksa beberapa izin setelah menerima permintaan. Jika ada masalah saat memanggil dependensi yang diperlukan seperti Amazon S3 CloudWatch atau proses menandai status impor sebagai GAGAL. `failureCode` dan `failureMessage` menunjukkan alasan kegagalan. Jika berlaku, pesan kegagalan juga berisi id permintaan yang dapat Anda gunakan untuk menyelidiki alasan kegagalan CloudTrail.

Kesalahan konfigurasi umum termasuk memiliki URL yang salah untuk bucket Amazon S3, dan tidak memiliki izin untuk mengakses bucket Amazon S3, Log CloudWatch , AWS KMS dan kunci yang digunakan untuk mendekripsi objek Amazon S3. Untuk informasi lebih lanjut lihat Menggunakan dan kunci data.

Memvalidasi objek sumber Amazon S3

Untuk memvalidasi objek S3 sumber, lakukan langkah-langkah berikut.

1. Validasi format data dan jenis kompresi

   - Pastikan semua objek Amazon S3 yang cocok dengan prefiks yang ditentukan memiliki format yang sama (DYNAMODB_JSON, DYNAMODB_ION, CSV)
   - Pastikan semua objek Amazon S3 yang cocok dengan prefiks yang ditentukan dikompresi dengan cara yang sama (GZIP, ZSTD, NONE)

> **Note**
>
> Objek Amazon S3 tidak perlu memiliki ekstensi yang sesuai (.csv/.json/.ion/.gz/.zstd dll) karena format input yang ditentukan dalam panggilan diutamakan. ImportTable

2. Validasi bahwa data impor sesuai dengan skema tabel yang diinginkan

   - Pastikan setiap item dalam data sumber memiliki kunci primer. Kunci urutan bersifat opsional untuk impor.
   - Pastikan jenis atribut yang terkait dengan kunci primer dan kunci urutan apa pun cocok dengan jenis atribut dalam Tabel dan skema GSI, seperti yang ditentukan dalam parameter pembuatan tabel

Pemecahan Masalah

CloudWatch log

Untuk pekerjaan Impor yang gagal, pesan kesalahan terperinci akan diposting ke CloudWatch log. Untuk mengakses log ini, pertama-tama ambil ImportArn dari output dan deskripsikan impor menggunakan perintah ini:

```
aws dynamodb describe-import --import-arn arn:aws:dynamodb:us-east-1:ACCOUNT:table/
target-table/import/01658528578619-c4d4e311
}
```

Contoh output:

```
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."
    }
}
```

Ambil grup log dan id impor dari respons di atas dan gunakan untuk mengambil log kesalahan. ID impor adalah elemen jalur terakhir dari bidang `ImportArn`. Nama grup log adalah `/aws-dynamodb/imports`. Nama log stream kesalahan adalah `import-id/error`. Untuk contoh ini, akan menjadi `01658528578619-c4d4e311/error`.

Tidak ada pk kunci dalam item

Jika objek S3 sumber tidak berisi kunci primer yang disediakan sebagai parameter, impor akan gagal. Misalnya, ketika Anda menentukan kunci primer untuk impor sebagai nama kolom “pk”.

```
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"}'
```

Kolom “pk” hilang dari objek sumber `import-test.csv` yang memiliki konten berikut:

```
title,artist,year_of_release
The Dark Side of the Moon,Pink Floyd,1973
```

Impor ini akan gagal karena kesalahan validasi item karena kunci primer yang hilang di sumber data.

Contoh log CloudWatch kesalahan:

```
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"
}
```

Log kesalahan ini menunjukkan bahwa “Satu atau lebih nilai parameter tidak valid: Kunci pk dalam item tidak ada”. Karena tugas impor ini gagal, tabel “tabel target” sekarang ada dan kosong karena tidak ada item yang diimpor. Item pertama diproses dan objek gagal Validasi Item.

Untuk memperbaiki masalah ini, pertama-tama hapus “tabel target” jika tidak diperlukan lagi. Kemudian gunakan nama kolom kunci primer yang ada di objek sumber, atau perbarui data sumber menjadi:

```
pk,title,artist,year_of_release
Albums::Rock::Classic::1973::AlbumId::ALB25,The Dark Side of the Moon,Pink Floyd,1973
```

Tabel target ada

Saat Anda memulai tugas impor dan menerima respons sebagai berikut:

```
An error occurred (ResourceInUseException) when calling the ImportTable operation:
 Table already exists: target-table
```

Untuk memperbaiki kesalahan ini, Anda harus memilih nama tabel yang belum ada dan mencoba mengimpor lagi.

Bucket yang ditentukan tidak ada

Jika bucket sumber tidak ada, impor akan gagal dan mencatat detail pesan kesalahan CloudWatch.

Contoh menjelaskan impor:

```
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)"
}
}
```


`FailureCode` adalah `S3NoSuchBucket`, dengan `FailureMessag` berisi detail seperti id permintaan dan layanan yang memunculkan kesalahan. Karena kesalahan diketahui sebelum data diimpor ke tabel, tabel DynamoDB baru tidak dibuat. Dalam beberapa kasus, ketika kesalahan ini terjadi setelah impor data dimulai, tabel dengan data yang diimpor sebagian akan dipertahankan.

Untuk memperbaiki kesalahan ini, pastikan bucket Amazon S3 sumber ada, lalu mulai ulang proses impor.

Praktik terbaik untuk mengimpor dari Amazon S3 ke DynamoDB

Berikut ini adalah praktik terbaik untuk mengimpor data dari Amazon S3 ke DynamoDB.

Tetap di bawah batas 50.000 objek S3

Setiap pekerjaan impor mendukung maksimum 50.000 objek S3. Jika dataset Anda berisi lebih dari 50.000 objek, pertimbangkan untuk mengkonsolidasikannya menjadi objek yang lebih besar.

Hindari objek S3 yang terlalu besar

Objek S3 diimpor secara paralel. Memiliki banyak objek S3 berukuran sedang memungkinkan eksekusi paralel tanpa overhead yang berlebihan. Untuk item di bawah 1 KB, pertimbangkan untuk menempatkan 4.000.000 item ke dalam setiap objek S3. Jika Anda memiliki ukuran rata-rata item yang lebih besar, tempatkan item yang lebih sedikit secara proporsional ke dalam setiap objek S3.

Mengacak data yang diurutkan

Jika objek S3 menyimpan data dalam urutan yang diurutkan, objek tersebut dapat membuat partisi penerapan panas. Ini adalah situasi di mana satu partisi menerima semua aktivitas, lalu partisi berikutnya setelah itu, dan seterusnya. Data dalam urutan yang diurutkan didefinisikan sebagai item secara berurutan dalam objek S3 yang akan ditulis ke partisi target yang sama selama impor. Salah satu situasi umum ketika data diurutkan adalah file CSV di mana item diurutkan berdasarkan kunci partisi sehingga item berulang berbagi kunci partisi yang sama.

Untuk menghindari partisi penerapan panas, kami sarankan Anda mengacak urutan dalam kasus ini. Hal ini dapat meningkatkan performa dengan menyebarkan operasi tulis. Untuk informasi selengkapnya, lihat Mendistribusikan aktivitas menulis secara efisien selama pengunggahan data di DynamoDB.

Kompres data untuk menjaga ukuran total objek S3 di bawah batas Regional

Dalam proses impor dari S3, terdapat batasan jumlah total ukuran data objek S3 yang akan diimpor. Batasannya adalah 15 TB di Wilayah us-east-1, us-west-2, dan eu-west-1, dan 1 TB di semua Wilayah lainnya. Batasnya didasarkan pada ukuran objek S3 mentah.

Kompresi memungkinkan lebih banyak data mentah sesuai dengan batasnya. Jika kompresi saja tidak cukup untuk memenuhi impor dalam batas tersebut, Anda juga dapat menghubungi AWS Premium Support untuk peningkat kuota.

Waspadai bagaimana ukuran item memengaruhi performa

Jika ukuran item rata-rata Anda sangat kecil (di bawah 200 byte), proses impor mungkin memakan waktu sedikit lebih lama dibandingkan ukuran item yang lebih besar.

Pertimbangkan untuk mengimpor tanpa Indeks Sekunder Global

Durasi tugas impor mungkin tergantung pada keberadaan satu atau beberapa indeks sekunder global (GSIs). Jika Anda berencana membuat indeks dengan kunci partisi yang memiliki kardinalitas rendah,

Anda mungkin melihat impor yang lebih cepat jika Anda menunda pembuatan indeks hingga tugas impor selesai (daripada memasukkannya ke dalam tugas impor).

> **Note**
>
> Membuat GSI selama impor tidak dikenakan biaya tulis (membuat GSI setelah impor akan dikenakan).

Ekspor data DynamoDB ke Amazon S3: cara kerjanya

Ekspor DynamoDB ke S3 adalah solusi yang dikelola sepenuhnya untuk mengekspor data DynamoDB Anda ke bucket Amazon S3 dalam skala besar. Menggunakan ekspor DynamoDB ke S3, Anda dapat mengekspor data dari tabel Amazon DynamoDB kapan saja dalam jendela pemulihan (PITR) ke point-in-time bucket Amazon S3. Anda perlu mengaktifkan PITR di tabel Anda untuk menggunakan fungsionalitas ekspor. Fitur ini memungkinkan Anda untuk melakukan analitik dan kueri kompleks pada data Anda menggunakan AWS layanan lain seperti Athena AWS Glue,, Amazon AI, SageMaker Amazon EMR, dan. AWS Lake Formation

Ekspor DynamoDB ke S3 memungkinkan Anda mengekspor data lengkap dan tambahan dari tabel DynamoDB Anda. Ekspor bersifat asinkron, mereka tidak mengkonsumsi unit kapasitas baca (RCUs) dan tidak berdampak pada kinerja dan ketersediaan tabel. Format file ekspor yang didukung adalah format DynamoDB JSON dan Amazon Ion. Anda juga dapat mengekspor data ke bucket S3 yang dimiliki oleh AWS akun lain dan ke AWS wilayah lain. Data Anda selalu dienkripsi end-to-end.

Ekspor penuh DynamoDB dikenakan biaya berdasarkan ukuran tabel DynamoDB (data tabel dan indeks sekunder lokal) pada saat ekspor dilakukan. Ekspor tambahan DynamoDB dikenakan biaya berdasarkan ukuran data yang diproses dari pencadangan berkelanjutan Anda selama periode waktu yang diekspor. Ekspor tambahan memiliki biaya minimum 10MB. Biaya tambahan berlaku untuk menyimpan data yang diekspor di Amazon S3 dan untuk permintaan PUT yang dibuat terhadap bucket Amazon S3 Anda. Untuk informasi selengkapnya tentang tagihan ini, lihat harga Amazon DynamoDB dan harga Amazon S3.

Untuk spesifik tentang kuota layanan, lihat. Ekspor tabel ke Amazon S3

Topik

- Meminta ekspor tabel di DynamoDB
- Format output ekspor tabel DynamoDB

Meminta ekspor tabel di DynamoDB

Ekspor tabel DynamoDB memungkinkan Anda mengekspor data tabel ke bucket Amazon S3, memungkinkan Anda melakukan analitik dan kueri kompleks pada data Anda menggunakan layanan lain AWS seperti Athena, Amazon AI, Amazon EMR AWS Glue, dan. SageMaker AWS Lake Formation Anda dapat meminta ekspor tabel menggunakan AWS Management Console, AWS CLI, atau DynamoDB API.

> **Note**
>
> Pemohon membayar bucket Amazon S3 tidak didukung.

DynamoDB mendukung ekspor penuh dan ekspor tambahan:

- Dengan ekspor penuh, Anda dapat mengekspor snapshot lengkap tabel Anda dari titik waktu mana pun dalam jendela point-in-time pemulihan (PITR) ke bucket Amazon S3 Anda.
- Dengan ekspor tambahan, Anda dapat mengekspor data dari tabel DynamoDB yang diubah, diperbarui, atau dihapus antara periode waktu tertentu, dalam jendela PITR, ke bucket Amazon S3 Anda.

Topik

- Prasyarat
- Meminta ekspor menggunakan AWS Management Console
- Mendapatkan detail tentang ekspor masa lalu di AWS Management Console
- Meminta ekspor menggunakan AWS CLI
- Mendapatkan detail tentang ekspor masa lalu di AWS CLI
- Meminta ekspor menggunakan AWS SDK
- Mendapatkan detail tentang ekspor sebelumnya menggunakan SDK AWS

Prasyarat

Mengaktifkan PITR

Untuk menggunakan fitur ekspor ke S3, Anda harus mengaktifkan PITR di meja Anda. Untuk detail tentang cara mengaktifkan PITR, lihat oint-in-timePemulihan P. Jika Anda meminta

ekspor untuk tabel yang tidak mengaktifkan PITR, permintaan Anda akan gagal dengan pesan pengecualian: “Terjadi kesalahan (PointInTimeRecoveryUnavailableException) saat memanggil `ExportTableToPointInTime` operasi: Pemulihan titik waktu tidak diaktifkan untuk tabel 'my-dynamodb-table”. Anda hanya dapat meminta dan mengekspor dari titik waktu yang ada dalam PITR `RecoveryPeriodInDays` yang dikonfigurasi.

Menyiapkan izin S3

Anda dapat mengekspor data tabel Anda ke setiap bucket Amazon S3 yang izinnya Anda miliki untuk menulis. Bucket tujuan tidak harus berada di AWS Wilayah yang sama atau memiliki pemilik yang sama dengan pemilik tabel sumber. Kebijakan AWS Identity and Access Management (IAM) Anda harus memungkinkan Anda untuk dapat melakukan tindakan S3 (`s3:AbortMultipartUpload`,`s3:PutObject`, dan`s3:PutObjectAcl`) dan tindakan ekspor DynamoDB (). `dynamodb:ExportTableToPointInTime` Berikut adalah contoh kebijakan sampel yang akan memberikan izin kepada pengguna Anda untuk melakukan ekspor ke bucket S3.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "AllowDynamoDBExportAction",
            "Effect": "Allow",
            "Action": "dynamodb:ExportTableToPointInTime",
            "Resource": "arn:aws:dynamodb:us-east-1:111122223333:table/my-table"
        },
        {
            "Sid": "amzn-s3-demo-bucket-AllowWrites",
            "Effect": "Allow",
            "Action": [
                "s3:AbortMultipartUpload",
                "s3:PutObject",
                "s3:PutObjectAcl"
            ],
            "Resource": "arn:aws:s3:::your-bucket/*"
        }
    ]
}
```

Jika Anda perlu menulis ke bucket Amazon S3 yang ada di akun lain atau Anda tidak memiliki izin untuk menulis, pemilik bucket Amazon S3 harus menambahkan kebijakan bucket untuk

memungkinkan Anda mengekspor dari DynamoDB ke bucket tersebut. Berikut adalah contoh kebijakan pada bucket Amazon S3 target.

```
{
    "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/*"
        }
    ]
}
```

Mencabut izin ini saat ekspor sedang berlangsung akan mengakibatkan sebagian file.

> **Note**
>
> Jika tabel atau bucket tujuan ekspor dienkripsi dengan kunci yang dikelola pelanggan, kebijakan kunci KMS tersebut harus memberikan izin kepada DynamoDB untuk menggunakannya. Izin ini diberikan melalui Pengguna/Peran IAM yang memicu tugas ekspor. Untuk informasi selengkapnya tentang enkripsi termasuk praktik terbaik, lihat Bagaimana DynamoDB menggunakan AWS KMS dan Menggunakan kunci KMS kustom.

Meminta ekspor menggunakan AWS Management Console

Contoh berikut menunjukkan cara menggunakan konsol DynamoDB untuk mengekspor tabel yang sudah ada bernama `MusicCollection`.

> **Note**
>
> Prosedur ini mengasumsikan bahwa Anda telah mengaktifkan point-in-time pemulihan. Untuk mengaktifkannya untuk `MusicCollection` tabel, pada tab Ikhtisar tabel, di bagian Rincian tabel, pilih Aktifkan untuk oint-in-timepemulihan P.

Untuk meminta ekspor tabel

1. Masuk ke AWS Management Console dan buka konsol DynamoDB di. https://console.aws.amazon.com/dynamodb/
2. Di panel navigasi di sisi kiri konsol, pilih Ekspor ke S3.
3. Pilih tombol Ekspor ke S3.
4. Pilih tabel sumber dan bucket S3 tujuan. Jika bucket tujuan adalah milik akun Anda, Anda dapat menggunakan tombol Jelajahi S3 untuk menemukannya. Jika tidak, masukkan URL bucket menggunakan format `s3://`**`bucketname/prefix`** `format.`, **`prefix`** adalah folder opsional untuk membantu mengatur keranjang tujuan Anda.
5. Pilih Ekspor penuh atau Ekspor tambahan. Ekspor penuh menghasilkan cuplikan tabel lengkap dari tabel Anda seperti pada titik waktu yang Anda tentukan. Ekspor tambahan menghasilkan perubahan yang dibuat pada tabel Anda selama periode ekspor yang ditentukan. Output Anda dipadatkan sehingga hanya berisi status akhir item dari periode ekspor. Item hanya akan muncul satu kali dalam ekspor meskipun memiliki beberapa pembaruan dalam periode ekspor yang sama.

   Full export

   1. Pilih titik waktu di mana Anda ingin mengekspor snapshot tabel lengkap. Ini bisa terjadi kapan saja dalam jendela PITR. Atau, Anda dapat memilih Waktu saat ini untuk mengekspor snapshot terbaru.
   2. Untuk Format file yang diekspor, pilih antara DynamoDB JSON dan Amazon Ion. Secara default, tabel Anda akan diekspor dalam format DynamoDB JSON dari waktu pemulihan terbaru di jendela pemulihan titik waktu dan dienkripsi menggunakan kunci Amazon S3 (SSE-S3). Anda dapat mengubah pengaturan ekspor ini jika perlu.

> **Note**
>
> Jika Anda memilih untuk mengenkripsi ekspor menggunakan kunci yang dilindungi oleh AWS Key Management Service (AWS KMS), kunci harus berada di Region yang sama dengan bucket S3 tujuan.

Incremental export

1. Pilih Periode ekspor yang ingin Anda ekspor data tambahannya. Pilih waktu mulai dalam jendela PITR. Durasi periode ekspor minimal harus 15 menit dan tidak lebih dari 24 jam. Waktu mulai periode ekspor bersifat inklusif dan waktu berakhirnya bersifat eksklusif.
2. Pilih antara Mode absolut atau Mode relatif.
   a. Mode absolut akan mengekspor data tambahan untuk jangka waktu yang Anda tentukan.
   b. Mode relatif akan mengekspor data tambahan untuk periode ekspor yang relatif terhadap waktu pengiriman tugas ekspor Anda.
3. Untuk Format file yang diekspor, pilih antara DynamoDB JSON dan Amazon Ion. Secara default, tabel Anda akan diekspor dalam format DynamoDB JSON dari waktu pemulihan terbaru di jendela pemulihan titik waktu dan dienkripsi menggunakan kunci Amazon S3 (SSE-S3). Anda dapat mengubah pengaturan ekspor ini jika perlu.

   > **Note**
   >
   > Jika Anda memilih untuk mengenkripsi ekspor menggunakan kunci yang dilindungi oleh AWS Key Management Service (AWS KMS), kunci harus berada di Region yang sama dengan bucket S3 tujuan.

4. Untuk Jenis tampilan ekspor, pilih Gambar baru dan lama atau Gambar baru saja. Gambar baru memberikan status terkini item. Gambar lama menunjukkan status item tepat sebelum “tanggal dan waktu mulai” yang ditentukan. Pengaturan defaultnya adalah Gambar baru dan lama. Untuk informasi selengkapnya tentang gambar baru dan gambar lama, lihat Output ekspor inkremental.

6. Pilih Ekspor untuk memulai.

Data yang diekspor tidak konsisten secara transaksional. Operasi transaksi Anda dapat terpecah antara dua output ekspor. Subset item dapat dimodifikasi oleh operasi transaksi yang tercermin dalam ekspor, sementara subset modifikasi lain dalam transaksi yang sama tidak tercermin dalam permintaan ekspor yang sama. Namun, ekspor pada akhirnya konsisten. Jika transaksi terkoyak selama ekspor, Anda akan memiliki sisa transaksi di ekspor berikutnya yang berdekatan, tanpa duplikat. Jangka waktu yang digunakan untuk ekspor didasarkan pada jam sistem internal dan dapat bervariasi menurut satu menit jam lokal aplikasi Anda.

Mendapatkan detail tentang ekspor masa lalu di AWS Management Console

Anda dapat menemukan informasi tentang tugas ekspor yang pernah Anda jalankan di masa lalu dengan memilih bagian Ekspor ke S3 di bilah sisi navigasi. Bagian ini berisi daftar semua ekspor yang Anda buat dalam 90 hari terakhir. Pilih ARN tugas yang tercantum di tab Ekspor untuk mengambil informasi tentang ekspor itu, termasuk pengaturan konfigurasi lanjutan yang Anda pilih. Perhatikan bahwa meskipun metadata tugas ekspor akan kedaluwarsa setelah 90 hari dan tugas yang lebih lama dari itu tidak lagi ditemukan dalam daftar ini, objek di bucket S3 Anda tetap ada selama kebijakan bucketnya mengizinkan. DynamoDB tidak pernah menghapus objek apa pun yang dibuatnya di bucket S3 Anda selama ekspor.

Meminta ekspor menggunakan AWS CLI

Contoh berikut menunjukkan cara menggunakan AWS CLI untuk mengekspor tabel yang ada bernama `MusicCollection` ke bucket S3 yang dipanggil`ddb-export-musiccollection`.

> **Note**
>
> Prosedur ini mengasumsikan bahwa Anda telah mengaktifkan point-in-time pemulihan. Untuk mengaktifkannya untuk tabel `MusicCollection`, jalankan perintah berikut.
>
> ```
> aws dynamodb update-continuous-backups \
>     --table-name MusicCollection \
>     --point-in-time-recovery-specification PointInTimeRecoveryEnabled=True
> ```

Full export

Perintah berikut mengekspor `MusicCollection` ke bucket S3 yang disebut `ddb-export-musiccollection-9012345678` dengan prefiks `2020-Nov`. Data tabel akan diekspor dalam

format DynamoDB JSON dari waktu tertentu dalam jendela pemulihan titik waktu dan dienkripsi menggunakan kunci Amazon S3 (SSE-S3).

> **Note**
>
> Jika meminta ekspor tabel lintas akun, pastikan untuk menyertakan opsi `--s3-bucket-owner`.

```
aws dynamodb export-table-to-point-in-time \
  --table-arn arn:aws:dynamodb:us-west-2:123456789012: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
```

Incremental export

Perintah berikut melakukan ekspor tambahan dengan menyediakan `--export-type` dan `--incremental-export-specification` baru. Gantikan nilai Anda sendiri dengan apa pun yang dicetak miring. Waktu ditentukan sebagai detik sejak zaman.

```
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
```

> **Note**
>
> Jika Anda memilih untuk mengenkripsi ekspor menggunakan kunci yang dilindungi oleh AWS Key Management Service (AWS KMS), kunci harus berada di Region yang sama dengan bucket S3 tujuan.

Mendapatkan detail tentang ekspor masa lalu di AWS CLI

Anda dapat menemukan informasi tentang permintaan ekspor yang pernah Anda jalankan sebelumnya dengan menggunakan perintah `list-exports`. Perintah ini mengembalikan daftar semua ekspor yang Anda buat dalam 90 hari terakhir. Perhatikan bahwa meskipun metadata tugas ekspor akan kedaluwarsa setelah 90 hari dan tugas yang lebih lama dari itu tidak lagi dikembalikan oleh perintah `list-exports`, objek dalam bucket S3 Anda tetap ada selama kebijakan bucketnya mengizinkan. DynamoDB tidak pernah menghapus objek apa pun yang dibuatnya di bucket S3 Anda selama ekspor.

Ekspor memiliki status `PENDING` hingga berhasil atau gagal. Jika mereka berhasil, statusnya berubah menjadi`COMPLETED`. Jika gagal, statusnya berubah `FAILED` dengan a `failure_message` dan`failure_reason`.

Dalam contoh berikut, kami menggunakan parameter `table-arn` opsional untuk mencantumkan ekspor tabel tertentu saja.

```
aws dynamodb list-exports \
    --table-arn arn:aws:dynamodb:us-east-1:123456789012:table/ProductCatalog
```

Untuk mengambil informasi mendetail tentang tugas ekspor tertentu, termasuk pengaturan konfigurasi lanjutan, gunakan perintah `describe-export`.

```
aws dynamodb describe-export \
    --export-arn arn:aws:dynamodb:us-east-1:123456789012:table/ProductCatalog/
export/01234567890123-a1b2c3d4
```

Meminta ekspor menggunakan AWS SDK

Gunakan cuplikan kode ini untuk meminta ekspor tabel menggunakan AWS SDK pilihan Anda.

Python

Ekspor penuh

```
import boto3
from datetime import datetime

# remove endpoint_url for real use
```

```
client = boto3.client('dynamodb')

# https://boto3.amazonaws.com/v1/documentation/api/latest/reference/services/
dynamodb/client/export_table_to_point_in_time.html
client.export_table_to_point_in_time(
    TableArn='arn:aws:dynamodb:us-east-1:0123456789:table/TABLE',
    ExportTime=datetime(2023, 9, 20, 12, 0, 0),
    S3Bucket='bucket',
    S3Prefix='prefix',
    S3SseAlgorithm='AES256',
    ExportFormat='DYNAMODB_JSON'
)
```

Ekspor tambahan

```
import boto3
from datetime import datetime

client = boto3.client('dynamodb')

client.export_table_to_point_in_time(
    TableArn='arn:aws:dynamodb:us-east-1:0123456789: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'
)
```

Mendapatkan detail tentang ekspor sebelumnya menggunakan SDK AWS

Gunakan cuplikan kode ini untuk mendapatkan detail tentang ekspor tabel sebelumnya menggunakan AWS SDK pilihan Anda.

Python

Daftar

```
import boto3

client = boto3.client('dynamodb')

# https://boto3.amazonaws.com/v1/documentation/api/latest/reference/services/
dynamodb/client/list_exports.html

print(
  client.list_exports(
     TableArn='arn:aws:dynamodb:us-east-1:0123456789:table/TABLE',
  )
)
```

Jelaskan

```
import boto3

client = boto3.client('dynamodb')

# https://boto3.amazonaws.com/v1/documentation/api/latest/reference/services/
dynamodb/client/describe_export.html

print(
  client.describe_export(
     ExportArn='arn:aws:dynamodb:us-east-1:0123456789:table/TABLE/
export/01695353076000-06e2188f',
  )['ExportDescription']
)
```

Format output ekspor tabel DynamoDB

Sebuah ekspor tabel DynamoDB mencakup file manifes selain file yang berisi data tabel Anda. File ini semua disimpan di bucket Amazon S3 yang Anda tentukan di permintaan ekspor. Bagian berikut menjelaskan format dan isi dari setiap objek output.

Output ekspor penuh

File manifes

DynamoDB membuat dua file manifes, bersama dengan file checksum, dalam bucket S3 yang ditentukan untuk setiap permintaan ekspor.

```
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
```

Anda memilih **`export-prefix`** saat meminta ekspor tabel. Ini membantu Anda menjaga kerapian file di bucket S3 tujuan. **`ExportId`** adalah token unik yang dihasilkan oleh layanan untuk memastikan bahwa beberapa ekspor ke bucket S3 dan `export-prefix` ekspor yang sama tidak saling menimpa.

Ekspor akan membuat setidaknya 1 file per partisi. Untuk partisi yang kosong, permintaan ekspor Anda akan membuat file kosong. Semua item di setiap file berasal dari keyspace hash partisi tertentu.

> **Note**
>
> DynamoDB juga membuat sebuah file kosong bernama `_started` dalam direktori yang sama sebagai file manifes. File ini memverifikasi bahwa bucket tujuan dapat ditulis dan bahwa ekspor telah dimulai. File ini bisa dihapus dengan aman.

Manifes ringkasan

File `manifest-summary.json` berisi informasi ringkasan tentang tugas ekspor. Ini memungkinkan Anda mengetahui file data mana di folder data bersama yang terkait dengan ekspor ini. Formatnya adalah sebagai berikut:

```
{
   "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"
}
```

Manifes file

File `manifest-files.json` berisi informasi tentang file yang berisi data tabel Anda yang diekspor. File ada dalam format baris JSON, jadi baris baru digunakan sebagai pembatas item. Pada contoh berikut, perincian satu file data dari manifes file diformat pada beberapa baris untuk keterbacaan.

```
{
"itemCount": 8,
   "md5Checksum": "sQMSpEILNgoQmarvDFonGQ==",
   "etag": "af83d6f217c19b8b0fff8023d8ca4716-1",
   "dataFileS3Key": "AWSDynamoDB/01693685827463-2d8752fd/data/asdl123dasas.json.gz"
}
```

File data

DynamoDB dapat mengekspor data tabel Anda dalam dua format: DynamoDB JSON dan Amazon Ion. Terlepas dari format yang Anda pilih, data Anda akan ditulis ke beberapa file terkompresi yang dinamai berdasarkan kunci. File-file ini juga tercantum dalam `manifest-files.json` file.

Struktur direktori bucket Amazon S3 Anda setelah ekspor penuh akan berisi semua file manifes dan file data Anda di bawah folder 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

Sebuah ekspor tabel dalam format DynamoDB JSON terdiri dari beberapa objek `Item`. Setiap objek individual berada dalam format JSON marshalled standar DynamoDB.

Saat membuat parser kustom untuk data ekspor DynamoDB JSON, formatnya adalah garis JSON. Ini berarti bahwa baris baru digunakan sebagai pembatas item. Banyak AWS layanan, seperti Athena dan AWS Glue, akan mengurai format ini secara otomatis.

Dalam contoh berikut, satu item dari ekspor JSON DynamoDB telah diformat pada beberapa baris demi keterbacaan.

```
{
    "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 adalah format serialisasi data hierarkis yang kaya jenis, dapat dijelaskan sendiri, yang dibuat untuk mengatasi tantangan perkembangan pesat, pemisahan, dan efisiensi yang dihadapi setiap hari saat merekayasa arsitektur berorientasi layanan berskala besar. DynamoDB mendukung ekspor data tabel dalam format teks Ion, yang merupakan superset dari JSON.

Saat Anda mengekspor tabel ke format Ion, jenis data DynamoDB yang digunakan dalam tabel dipetakan ke jenis data Ion. Set DynamoDB menggunakan anotasi jenis Ion untuk memperjelas jenis data yang digunakan dalam tabel sumber.

Tabel berikut mencantumkan pemetaan tipe data DynamoDB ke tipe data ion:

Jenis data DynamoDB	Perwakilan Ion
String (S)	string
Boolean (BOOL)	bool
Angka (N)	desimal
Biner (B)	blob
Set (SS, NS, BS)	daftar (dengan anotasi jenis $dynamodb_SS, $dynamodb_NS, or $dynamodb_BS)
Daftar	daftar
Peta	struct

Item dalam ekspor Ion dibatasi oleh baris baru. Setiap baris dimulai dengan penanda versi Ion, diikuti dengan item dalam format Ion. Dalam contoh berikut, item dari ekspor Ion telah diformat dalam beberapa baris demi keterbacaan.

```
$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"
    }
}
```

Output ekspor inkremental

File manifes

DynamoDB membuat dua file manifes, bersama dengan file checksum, dalam bucket S3 yang ditentukan untuk setiap permintaan ekspor.

```
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
```

Anda memilih **`export-prefix`** saat meminta ekspor tabel. Ini membantu Anda menjaga kerapian file di bucket S3 tujuan. **`ExportId`** adalah token unik yang dihasilkan oleh layanan untuk memastikan bahwa beberapa ekspor ke bucket S3 dan `export-prefix` ekspor yang sama tidak saling menimpa.

Ekspor akan membuat setidaknya 1 file per partisi. Untuk partisi yang kosong, permintaan ekspor Anda akan membuat file kosong. Semua item di setiap file berasal dari keyspace hash partisi tertentu.

> **Note**
>
> DynamoDB juga membuat sebuah file kosong bernama `_started` dalam direktori yang sama sebagai file manifes. File ini memverifikasi bahwa bucket tujuan dapat ditulis dan bahwa ekspor telah dimulai. File ini bisa dihapus dengan aman.

Manifes ringkasan

File `manifest-summary.json` berisi informasi ringkasan tentang tugas ekspor. Ini memungkinkan Anda mengetahui file data mana di folder data bersama yang terkait dengan ekspor ini. Formatnya adalah sebagai berikut:

```
{
 "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"
}
```

Manifes file

File `manifest-files.json` berisi informasi tentang file yang berisi data tabel Anda yang diekspor. File ada dalam format baris JSON, jadi baris baru digunakan sebagai pembatas item. Pada contoh berikut, perincian satu file data dari manifes file diformat pada beberapa baris untuk keterbacaan.

```
{
"itemCount": 8,
   "md5Checksum": "sQMSpEILNgoQmarvDFonGQ==",
   "etag": "af83d6f217c19b8b0fff8023d8ca4716-1",
   "dataFileS3Key": "AWSDynamoDB/data/sgad6417s6vss4p7owp0471bcq.json.gz"
}
```

File data

DynamoDB dapat mengekspor data tabel Anda dalam dua format: DynamoDB JSON dan Amazon Ion. Terlepas dari format yang Anda pilih, data Anda akan ditulis ke beberapa file terkompresi yang dinamai berdasarkan kunci. File-file ini juga tercantum dalam `manifest-files.json` file.

File data untuk ekspor inkremental semuanya ada di folder data umum di bucket S3 Anda. File manifes Anda berada di bawah folder ID ekspor Anda.

```
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
    #   ...
```

Saat Anda mengekspor file, output setiap item menyertakan stempel waktu yang menunjukkan kapan item tersebut diperbarui di tabel Anda dan struktur data yang menunjukkan apakah itu adalah operasi `insert`, `update`, atau `delete`. Stempel waktu didasarkan pada jam sistem internal dan dapat bervariasi dari jam aplikasi Anda. Untuk ekspor tambahan, Anda dapat memilih antara dua jenis tampilan ekspor untuk struktur keluaran Anda: gambar baru dan lama atau hanya gambar baru.

- Gambar baru memberikan status terkini item
- Gambar lama menunjukkan status item tepat sebelum tanggal dan waktu mulai yang ditentukan

Jenis tampilan dapat membantu jika Anda ingin melihat bagaimana item diubah dalam periode ekspor. Ini juga dapat berguna untuk memperbarui sistem hilir Anda secara efisien, terutama jika sistem hilir tersebut memiliki kunci partisi yang tidak sama dengan kunci partisi DynamoDB Anda.

Anda dapat menyimpulkan apakah item dalam output ekspor tambahan Anda adalah `insert`, `update`, atau `delete` dengan melihat struktur output. Struktur ekspor inkremental dan operasi yang sesuai dirangkum dalam tabel di bawah ini untuk kedua jenis tampilan ekspor.

Operasi	Hanya gambar baru	Gambar baru dan lama
Sisipkan	Tombol + gambar baru	Tombol + gambar baru
Perbarui	Tombol + gambar baru	Tombol+gambar baru+gambar lama
Hapus	Kunci	Tombol + gambar lama
Sisipkan + hapus	Tidak ada output	Tidak ada output

DynamoDB Json

Ekspor tabel dalam format JSON DynamoDB terdiri dari stempel waktu metadata yang menunjukkan waktu penulisan item, diikuti dengan kunci item dan nilainya. Berikut ini menunjukkan contoh output JSON DynamoDB menggunakan jenis tampilan ekspor sebagai gambar Baru dan Lama.

```
// 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 adalah format serialisasi data hierarkis yang kaya jenis, dapat dijelaskan sendiri, yang dibuat untuk mengatasi tantangan perkembangan pesat, pemisahan, dan efisiensi yang dihadapi setiap hari saat merekayasa arsitektur berorientasi layanan berskala besar. DynamoDB mendukung ekspor data tabel dalam format teks Ion, yang merupakan superset dari JSON.

Saat Anda mengekspor tabel ke format Ion, jenis data DynamoDB yang digunakan dalam tabel dipetakan ke jenis data Ion. Set DynamoDB menggunakan anotasi jenis Ion untuk memperjelas jenis data yang digunakan dalam tabel sumber.

Tabel berikut mencantumkan pemetaan tipe data DynamoDB ke tipe data ion:

Jenis data DynamoDB	Perwakilan Ion
String (S)	string
Boolean (BOOL)	bool
Angka (N)	desimal
Biner (B)	blob
Set (SS, NS, BS)	daftar (dengan anotasi jenis $dynamodb_SS, $dynamodb_NS, or $dynamodb_BS)
Daftar	daftar
Peta	struct

Item dalam ekspor Ion dibatasi oleh baris baru. Setiap baris dimulai dengan penanda versi Ion, diikuti dengan item dalam format Ion. Dalam contoh berikut, item dari ekspor Ion telah diformat dalam beberapa baris demi keterbacaan.

```
$ion_1_0 {
    Record:{
        Keys:{
             ISBN:"333-3333333333"
         },
        Metadata:{
            WriteTimestampMicros:1684374845117899.
        },
        OldImage:{
            Authors:$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"
        }
    }
}
```

Integrasi DynamoDB nol-ETL dengan Amazon Lakehouse SageMaker

Integrasi DynamoDB Zero-ETL dengan SageMaker Amazon Lakehouse menghilangkan kebutuhan untuk membangun pipeline pergerakan data kustom dengan secara otomatis mereplikasi data DynamoDB ke Amazon Lakehouse. SageMaker Integrasi tanpa kode ini membantu pelanggan menjalankan beban kerja analitik pada data DynamoDB mereka menggunakan Amazon SageMaker Lakehouse tanpa menghabiskan kapasitas tabel DynamoDB apa pun. Integrasi secara otomatis mengekspor data dari tabel Anda dan menjaga target tetap segar, biasanya dalam 15 hingga 30 menit.

Topik

- Integrasi DynamoDB nol-ETL dengan Amazon Lakehouse SageMaker

Integrasi DynamoDB nol-ETL dengan Amazon Lakehouse SageMaker

Menyiapkan integrasi antara tabel DynamoDB dan SageMaker Amazon Lakehouse memerlukan prasyarat seperti mengonfigurasi peran IAM AWS Glue yang digunakan untuk mengakses data dari sumber dan menulis ke target, dan penggunaan kunci KMS untuk mengenkripsi data di lokasi perantara atau target.

Topik

- Prasyarat sebelum membuat integrasi DynamoDB Zero-ETL dengan Amazon Lakehouse SageMaker
- Membuat integrasi DynamoDB Zero-ETL dengan Amazon Lakehouse SageMaker
- Melihat CloudWatch metrik untuk integrasi

Prasyarat sebelum membuat integrasi DynamoDB Zero-ETL dengan Amazon Lakehouse SageMaker

Untuk mengonfigurasi integrasi nol-ETL dengan sumber DynamoDB, Anda perlu menyiapkan kebijakan Resource-Based Access (RBAC) yang memungkinkan untuk mengakses dan mengekspor data dari tabel DynamoDB. AWS Glue Kebijakan tersebut harus menyertakan izin khusus sepertiExportTableToPointInTime,DescribeTable, dan DescribeExport dengan kondisi yang membatasi akses ke wilayah tertentu Akun AWS . Lihat, Mengonfigurasi sumber Amazon DynamoDB untuk informasi selengkapnya.

Point-in-time pemulihan (PITR) harus diaktifkan untuk tabel, dan Anda dapat menerapkan kebijakan menggunakan AWS CLI perintah. Kebijakan ini dapat disempurnakan lebih lanjut dengan menentukan ARN integrasi penuh untuk kontrol akses yang lebih ketat. Untuk informasi selengkapnya, lihat Prasyarat untuk menyiapkan integrasi nol-ETL.

Membuat integrasi DynamoDB Zero-ETL dengan Amazon Lakehouse SageMaker

Setelah menyelesaikan prasyarat integrasi, Anda dapat membuat, memodifikasi, atau menghapus integrasi nol-ETL dengan mengikuti panduan di bawah ini:

Membuat integrasi

Untuk membuat integrasi

1. Masuk ke AWS Management Console dan buka konsol Amazon DynamoDB di. https://console.aws.amazon.com/dynamodbv2
2. Di panel navigasi, pilih Integrasi.
3. Pilih Buat integrasi Nol-ETL dengan Amazon SageMaker Lakehouse, lalu pilih Berikutnya.
4. Untuk membuat integrasi, lihat Membuat integrasi.
5. Untuk memodifikasi integrasi, lihat Memodifikasi integrasi.
6. Untuk menghapus integrasi, lihat Menghapus integrasi.

7. Untuk menyiapkan integrasi lintas akun, lihat Menyiapkan integrasi lintas akun.

Mengaktifkan pemadatan pada tabel Amazon S3 target

Anda dapat mengaktifkan pemadatan untuk meningkatkan kinerja kueri di Amazon Athena.

Pertama, selesaikan pengaturan prasyarat untuk sumber daya pemadatan, termasuk mengonfigurasi peran IAM yang diperlukan. Lihat dokumentasi Lake Formation untuk langkah-langkah konfigurasi peran IAM yang terperinci. Lihat, Mengoptimalkan tabel untuk pemadatan.

Untuk mengaktifkan pemadatan pada AWS Glue tabel yang dibuat selama integrasi, ikuti proses pengaktifan pemadatan Lake Formation. Ini akan membantu mengoptimalkan kinerja tabel dan efisiensi kueri Anda.

Melihat CloudWatch metrik untuk integrasi

Setelah integrasi selesai, Anda dapat melihat CloudWatch metrik dan EventBridge pemberitahuan yang dihasilkan di akun Anda untuk setiap AWS Glue pekerjaan. Untuk informasi selengkapnya, lihat Memantau integrasi.

Integrasi DynamoDB Zero-ETL dengan Amazon Service OpenSearch

Amazon DynamoDB menawarkan integrasi nol-ETL dengan OpenSearch Amazon Service melalui plugin DynamoDB untuk Ingestion. OpenSearch Amazon OpenSearch Ingestion menawarkan pengalaman tanpa kode yang dikelola sepenuhnya untuk menyerap data ke Layanan Amazon. OpenSearch

Dengan plugin DynamoDB OpenSearch untuk Ingestion, Anda dapat menggunakan satu atau lebih tabel DynamoDB sebagai sumber untuk menelan satu atau lebih indeks Layanan. OpenSearch Anda dapat menelusuri dan mengonfigurasi pipeline OpenSearch Ingestion Anda dengan DynamoDB sebagai sumber dari Integrasi OpenSearch Ingestion atau DynamoDB di. AWS Management Console

- Mulailah dengan OpenSearch Ingestion dengan mengikuti panduan memulai OpenSearch Ingestion.
- Pelajari tentang prasyarat dan semua opsi konfigurasi untuk plugin DynamoDB di plugin DynamoDB untuk dokumentasi Ingestion. OpenSearch

Cara kerjanya

Plugin menggunakan ekspor DynamoDB ke Amazon S3 untuk membuat snapshot awal untuk dimuat. OpenSearch Setelah snapshot dimuat, plugin menggunakan DynamoDB Streams untuk mereplikasi perubahan lebih lanjut dalam waktu dekat. Setiap item diproses sebagai acara di OpenSearch Ingestion dan dapat dimodifikasi dengan plugin prosesor. Anda dapat menghapus atribut atau membuat atribut gabungan dan mengirimkannya ke indeks berbeda melalui rute.

Anda harus mengaktifkan point-in-time pemulihan (PITR) untuk menggunakan ekspor ke Amazon S3. Anda juga harus mengaktifkan DynamoDB Streams (dengan opsi gambar baru & lama dipilih) untuk dapat menggunakannya. Dimungkinkan untuk membuat alur tanpa mengambil snapshot dengan mengecualikan pengaturan ekspor.

Anda juga dapat membuat alur hanya dengan snapshot dan tanpa pembaruan dengan mengecualikan pengaturan aliran. Plugin tidak menggunakan throughput baca atau tulis di meja Anda, sehingga aman digunakan tanpa memengaruhi lalu lintas produksi Anda. Ada batasan jumlah konsumen paralel pada aliran yang harus Anda pertimbangkan sebelum membuat integrasi ini atau lainnya. Untuk pertimbangan lain, lihat the section called “Praktik terbaik integrasi”.

Untuk pipeline sederhana, satu OpenSearch Compute Unit (OCU) dapat memproses sekitar 1 MB per detik penulisan. Hal ini setara dengan sekitar 1000 unit permintaan tulis (WCU). Bergantung pada kompleksitas alur Anda dan faktor lainnya, Anda mungkin mencapai lebih atau kurang dari angka ini.

OpenSearch Ingestion mendukung antrian huruf mati (DLQ) untuk peristiwa yang menyebabkan kesalahan yang tidak dapat dipulihkan. Selain itu, pipeline dapat melanjutkan dari tempat yang ditinggalkannya tanpa campur tangan pengguna meskipun ada gangguan layanan dengan DynamoDB, pipeline, atau Amazon Service. OpenSearch

Jika gangguan berlangsung lebih dari 24 jam, hal ini dapat menyebabkan hilangnya pembaruan. Namun, alur akan terus memproses pembaruan yang masih tersedia saat ketersediaan dipulihkan. Anda perlu melakukan pembuatan indeks baru untuk memperbaiki penyimpangan apa pun karena peristiwa yang dijatuhkan kecuali mereka berada dalam antrean surat mati.

Untuk semua pengaturan dan detail untuk plugin, lihat Dokumentasi plugin OpenSearchDynamoDB Ingestion.

Terintegrasi menciptakan pengalaman melalui konsol

DynamoDB OpenSearch dan Layanan memiliki pengalaman terintegrasi dalam AWS Management Console, yang merampingkan proses memulai. Ketika Anda melalui langkah-langkah ini, layanan

akan secara otomatis memilih cetak biru DynamoDB dan menambahkan informasi DynamoDB yang sesuai untuk Anda.

Untuk membuat integrasi, ikuti panduan Memulai OpenSearch Ingestion. Saat Anda sampai ke Langkah 3: Buat alur, ganti Langkah 1 dan 2 dengan langkah-langkah berikut:

1. Navigasikan ke konsol DynamoDB.
2. Di panel navigasi sebelah kiri, pilih Integrasi.
3. Pilih tabel DynamoDB yang ingin Anda tiru. OpenSearch
4. Pilih Buat.

Dari sini, Anda dapat melanjutkan dengan sisa tutorial.

Langkah selanjutnya

Untuk pemahaman yang lebih baik tentang bagaimana DynamoDB terintegrasi OpenSearch dengan Layanan, lihat berikut ini:

- Memulai dengan Amazon OpenSearch Ingestion
- Konfigurasi dan persyaratan plugin DynamoDB

Menangani perubahan yang menyebabkan gangguan pada indeks Anda

OpenSearch dapat secara dinamis menambahkan atribut baru ke indeks Anda. Namun, setelah template pemetaan Anda disetel untuk kunci tertentu, Anda harus mengambil tindakan tambahan untuk mengubahnya. Selain itu, jika perubahan Anda mengharuskan untuk memproses ulang semua data dalam tabel DynamoDB Anda, Anda harus mengambil langkah-langkah untuk memulai ekspor baru.

> **Note**
>
> Dalam semua opsi ini, Anda mungkin masih mengalami masalah jika tabel DynamoDB Anda memiliki konflik jenis dengan template pemetaan yang Anda tentukan. Pastikan Anda mengaktifkan antrean surat mati (DLQ) (bahkan dalam pengembangan). Ini membuatnya lebih mudah untuk memahami apa yang mungkin salah dengan catatan yang menyebabkan konflik ketika sedang diindeks ke dalam indeks Anda. OpenSearch

Topik

- Cara kerjanya
- Hapus indeks Anda dan setel ulang alur (opsi alur-sentris)
- Buat ulang indeks Anda dan setel ulang alur (opsi indeks-sentris)
- Buat indeks dan sink baru (opsi online)
- Praktik terbaik untuk menghindari dan men-debug konflik jenis

Cara kerjanya

Berikut gambaran umum singkat dari tindakan yang dilakukan saat menangani perubahan yang menyebabkan gangguan pada indeks Anda. Lihat step-by-step prosedur di bagian berikut.

- Hentikan dan mulai alur: Opsi ini mengatur ulang status alur, dan alur akan dimulai ulang dengan ekspor penuh baru. Ini bersifat non-destruktif, sehingga tidak menghapus indeks Anda atau data apa pun di DynamoDB. Jika Anda tidak membuat indeks baru sebelum melakukan ini, Anda mungkin melihat banyak kesalahan dari konflik versi karena ekspor mencoba memasukkan dokumen yang lebih lama dari `_version` saat ini ke dalam indeks. Anda dapat mengabaikan kesalahan ini dengan aman. Anda tidak akan ditagih untuk alur saat alur tersebut dihentikan.
- Perbarui alur: Opsi ini memperbarui konfigurasi dalam alur dengan pendekatan biru/hijau, tanpa kehilangan status apa pun. Jika Anda membuat perubahan signifikan pada alur Anda (seperti menambahkan rute, indeks, atau kunci baru ke indeks yang sudah ada), Anda mungkin perlu melakukan reset penuh pada alur dan membuat ulang indeks Anda. Opsi ini tidak melakukan ekspor penuh.
- Hapus dan buat ulang indeks: Opsi ini menghapus data dan pengaturan pemetaan pada indeks Anda. Anda harus melakukan ini sebelum membuat perubahan yang menyebabkan gangguan pada pemetaan Anda. Ini akan merusak aplikasi apa pun yang mengandalkan indeks hingga indeks dibuat ulang dan disinkronkan. Menghapus indeks tidak memulai ekspor baru. Anda harus menghapus indeks Anda hanya setelah Anda memperbarui alur Anda. Jika tidak, indeks Anda mungkin dibuat ulang sebelum Anda memperbarui pengaturan.

Hapus indeks Anda dan setel ulang alur (opsi alur-sentris)

Metode ini sering menjadi pilihan tercepat jika Anda masih dalam pengembangan. Anda akan menghapus indeks Anda di OpenSearch Layanan, lalu menghentikan dan memulai pipeline Anda untuk memulai ekspor baru semua data Anda. Hal ini memastikan bahwa tidak ada konflik template

pemetaan dengan indeks yang ada, dan tidak ada kehilangan data dari tabel yang diproses tidak lengkap.

1. Hentikan pipeline baik melalui AWS Management Console, atau dengan menggunakan operasi StopPipelineAPI dengan AWS CLI atau SDK.
2. Perbarui konfigurasi alur Anda dengan perubahan baru.
3. Hapus indeks Anda di OpenSearch Layanan, baik melalui panggilan REST API atau OpenSearch Dasbor Anda.
4. Mulai pipeline baik melalui konsol, atau dengan menggunakan operasi `StartPipeline` API dengan AWS CLI atau SDK.

   > **Note**
   >
   > Tindakan ini memulai ekspor penuh baru, yang akan menimbulkan biaya tambahan.

5. Pantau masalah yang tidak terduga karena ekspor baru dihasilkan untuk membuat indeks baru.
6. Konfirmasikan bahwa indeks sesuai dengan harapan Anda di OpenSearch Layanan.

Setelah ekspor selesai dan melanjutkan pembacaan dari aliran, data tabel DynamoDB Anda sekarang akan tersedia dalam indeks.

Buat ulang indeks Anda dan setel ulang alur (opsi indeks-sentris)

Metode ini berfungsi dengan baik jika Anda perlu melakukan banyak iterasi pada desain indeks di OpenSearch Service sebelum melanjutkan pipeline dari DynamoDB. Ini dapat berguna untuk pengembangan ketika Anda ingin mengulangi dengan sangat cepat pada pola pencarian Anda, dan ingin menghindari menunggu ekspor baru selesai di antara setiap iterasi.

1. Hentikan pipeline baik melalui AWS Management Console, atau dengan memanggil operasi StopPipelineAPI dengan AWS CLI atau SDK.
2. Hapus dan buat ulang indeks Anda OpenSearch dengan template pemetaan yang ingin Anda gunakan. Anda dapat memasukkan beberapa data sampel secara manual untuk mengonfirmasi bahwa pencarian Anda berfungsi sebagaimana dimaksud. Jika data sampel Anda mungkin bertentangan dengan data apa pun dari DynamoDB, pastikan untuk menghapusnya sebelum melanjutkan ke langkah berikutnya.

3. Jika Anda memiliki template pengindeksan di pipeline Anda, hapus atau ganti dengan yang sudah Anda buat di OpenSearch Service. Pastikan bahwa nama indeks Anda cocok dengan nama dalam alur.
4. Mulai pipeline baik melalui konsol, atau dengan memanggil operasi `StartPipeline` API dengan AWS CLI atau SDK.

   > **Note**
   >
   > Tindakan ini akan memulai ekspor penuh baru, yang akan menimbulkan biaya tambahan.

5. Pantau masalah yang tidak terduga karena ekspor baru dihasilkan untuk membuat indeks baru.

Setelah ekspor selesai dan melanjutkan pembacaan dari aliran, data tabel DynamoDB Anda sekarang akan tersedia dalam indeks.

Buat indeks dan sink baru (opsi online)

Metode ini berfungsi dengan baik jika Anda perlu memperbarui template pemetaan Anda tetapi saat ini menggunakan indeks Anda dalam produksi. Ini menciptakan indeks baru, yang mana Anda harus memindahkan aplikasi Anda setelah disinkronkan dan divalidasi.

> **Note**
>
> Ini akan membuat konsumen lain di aliran. Ini bisa menjadi masalah jika Anda juga memiliki konsumen lain seperti AWS Lambda atau tabel global. Anda mungkin perlu menjeda pembaruan ke alur yang ada untuk membuat kapasitas memuat indeks baru.

1. Buat alur baru dengan pengaturan baru dan nama indeks yang berbeda.
2. Pantau indeks baru untuk masalah yang tidak terduga.
3. Tukar aplikasi ke indeks baru.
4. Hentikan dan hapus alur lama setelah memvalidasi bahwa semuanya berfungsi dengan benar.

Praktik terbaik untuk menghindari dan men-debug konflik jenis

- Selalu gunakan antrean surat mati (DLQ) untuk memudahkan debug ketika ada konflik jenis.

- Selalu gunakan template indeks dengan pemetaan dan setel `include_keys`. Sementara OpenSearch Layanan secara dinamis memetakan kunci baru, ini dapat menyebabkan masalah dengan perilaku yang tidak terduga (seperti mengharapkan sesuatu menjadi a`GeoPoint`, tetapi dibuat sebagai `string` atau`object`) atau kesalahan (seperti memiliki `number` yang merupakan campuran `long` dan `float` nilai).
- Jika Anda perlu menjaga indeks yang ada tetap berfungsi dalam produksi, Anda juga dapat mengganti salah satu langkah penghapusan indeks sebelumnya hanya dengan mengganti nama indeks Anda di file konfigurasi alur Anda. Ini menciptakan indeks baru. Aplikasi Anda kemudian perlu diperbarui untuk menunjuk ke indeks baru setelah selesai.
- Jika Anda memiliki masalah konversi jenis yang Anda perbaiki dengan prosesor, Anda dapat mengujinya dengan `UpdatePipeline`. Untuk melakukan ini, Anda harus berhenti dan memulai atau memproses antrean surat mati Anda untuk memperbaiki dokumen yang sebelumnya dilewati yang memiliki kesalahan.

Praktik terbaik untuk bekerja dengan integrasi dan Layanan DynamoDB Zero-ETL OpenSearch

DynamoDB memiliki integrasi DynamoDB Zero-ETL dengan Amazon Service. OpenSearch Untuk informasi selengkapnya, lihat plugin DynamoDB untuk Penyerapan dan praktik terbaik khusus OpenSearch untuk Layanan Amazon. OpenSearch

Konfigurasi

- Anda hanya perlu melakukan pencarian pada indeks data. Selalu gunakan templat pemetaan (`template_type: index_template` dan `template_content`) serta `include_keys` untuk mengimplementasikan hal ini.
- Pantau log Anda untuk kesalahan yang terkait dengan konflik tipe. OpenSearch Layanan mengharapkan semua nilai untuk kunci yang diberikan memiliki tipe yang sama. Pengecualian dihasilkan jika ada ketidakcocokan. Jika Anda menemukan salah satu kesalahan ini, Anda dapat menambahkan prosesor untuk mengetahui bahwa kunci yang diberikan selalu memiliki nilai yang sama.
- Secara umum, gunakan nilai metadata `primary_key` untuk nilai `document_id`. Dalam OpenSearch Layanan, ID dokumen setara dengan kunci utama di DynamoDB. Menggunakan kunci primer akan memudahkan dokumen Anda ditemukan dan memastikan bahwa pembaruan secara konsisten direplikasi tanpa konflik.

Anda dapat menggunakan fungsi pembantu `getMetadata` untuk mendapatkan kunci primer Anda (misalnya, `document_id: "${getMetadata('primary_key')}"`). Jika Anda menggunakan kunci primer komposit, fungsi pembantu akan menggabungkannya bersama-sama untuk Anda.

- Secara umum, gunakan nilai metadata `opensearch_action` untuk pengaturan `action`. Ini akan memastikan bahwa pembaruan direplikasi sedemikian rupa sehingga data di OpenSearch Layanan cocok dengan status terbaru di DynamoDB.

  Anda dapat menggunakan fungsi pembantu `getMetadata` untuk mendapatkan kunci primer Anda (misalnya, `action: "${getMetadata('opensearch_action')}"`). Anda juga bisa mendapatkan jenis peristiwa streaming `dynamodb_event_name` untuk kasus penggunaan seperti pemfilteran. Namun, biasanya hal ini tidak digunakan untuk pengaturan `action`.

Observabilitas

- Selalu gunakan antrian huruf mati (DLQ) di OpenSearch wastafel Anda untuk menangani peristiwa yang jatuh. DynamoDB umumnya kurang terstruktur OpenSearch daripada Layanan, dan selalu mungkin terjadi sesuatu yang tidak terduga. Dengan antrean surat mati, Anda dapat memulihkan peristiwa individual, dan bahkan mengotomatiskan proses pemulihan. Hal ini akan membantu agar Anda tidak perlu membangun kembali seluruh indeks Anda.
- Selalu atur peringatan agar penundaan replikasi Anda tidak melebihi jumlah yang diharapkan. Biasanya aman untuk mengasumsikan satu menit tanpa peringatan yang terlalu berisik. Ini dapat bervariasi tergantung pada seberapa runcing lalu lintas tulis Anda dan pengaturan OpenSearch Compute Unit (OCU) Anda di pipeline.

  Jika penundaan replikasi Anda melebihi 24 jam, aliran Anda akan mulai kehilangan peristiwa, dan Anda akan mengalami masalah akurasi kecuali jika Anda melakukan pembangunan ulang indeks Anda dari awal.

Penskalaan

- Gunakan penskalaan otomatis untuk saluran pipa untuk membantu meningkatkan atau menurunkan skala OCUs agar sesuai dengan beban kerja.
- Untuk tabel throughput yang disediakan tanpa penskalaan otomatis, kami sarankan pengaturan OCUs berdasarkan unit kapasitas tulis (WCUs) Anda dibagi 1000. Tetapkan minimum ke 1 OCU

di bawah jumlah tersebut (tetapi setidaknya 1), dan atur maksimum ke setidaknya 1 OCU di atas jumlah tersebut.

- Rumus:

```
OCU_minimum = GREATEST((table_WCU / 1000) - 1, 1)
OCU_maximum = (table_WCU / 1000) + 1
```

- Contoh: Tabel Anda memiliki 25000 yang WCUs disediakan. Pipa Anda OCUs harus diatur dengan minimal 24 (25000/1000 - 1) dan maksimum minimal 26 (25000/1000 + 1).

- Untuk tabel throughput yang disediakan dengan penskalaan otomatis, kami merekomendasikan pengaturan OCUs berdasarkan minimum dan maksimum Anda WCUs, dibagi dengan 1000. Atur minimum ke 1 OCU di bawah minimum dari DynamoDB, dan atur maksimum ke setidaknya 1 OCU di atas maksimum dari DynamoDB.
  - Rumus:

```
OCU_minimum = GREATEST((table_minimum_WCU / 1000) - 1, 1)
OCU_maximum = (table_maximum_WCU / 1000) + 1
```

  - Contoh: Tabel Anda memiliki kebijakan penskalaan otomatis dengan minimum 8000 dan maksimum 14000. Pipa Anda OCUs harus diatur dengan minimal 7 (8000/1000 - 1) dan maksimum 15 (14000/1000 + 1).
- Untuk tabel throughput sesuai permintaan, kami merekomendasikan pengaturan OCUs berdasarkan puncak dan lembah khas Anda untuk unit permintaan tulis per detik. Anda mungkin perlu membuat rata-rata dalam jangka waktu yang lebih lama, tergantung pada agregasi yang tersedia untuk Anda. Atur minimum ke 1 OCU di bawah minimum dari DynamoDB, dan atur maksimum ke setidaknya 1 OCU di atas maksimum dari DynamoDB.
  - Rumus:

```
# 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
```

  - Contoh: Tabel Anda memiliki penurunan rata-rata 300 unit permintaan tulis per detik dan puncak rata-rata 4300. Pipa Anda OCUs harus diatur dengan minimal 1 (300/1000 - 1, tetapi setidaknya 1) dan maksimum 5 (4300/1000 + 1).

- Ikuti praktik terbaik dalam penskalaan indeks OpenSearch Layanan tujuan Anda. Jika penskalaan indeks Anda kurang, ini akan memperlambat konsumsi dari DynamoDB, dan dapat menyebabkan penundaan.

> **Note**
>
> `GREATEST` adalah fungsi SQL yang, jika diberikan sekumpulan argumen, mengembalikan argumen dengan nilai terbesar.

Mengintegrasikan DynamoDB dengan Amazon EventBridge

Amazon DynamoDB menawarkan DynamoDB Streams untuk mengubah pengambilan data, memungkinkan pengambilan perubahan tingkat item dalam tabel DynamoDB. DynamoDB Streams dapat memanggil fungsi Lambda untuk memproses perubahan tersebut, memungkinkan integrasi berbasis peristiwa dengan layanan dan aplikasi lain. DynamoDB Streams juga mendukung penyaringan, yang memungkinkan pemrosesan peristiwa yang efisien dan ditargetkan.

DynamoDB Streams mendukung hingga dua konsumen simultan per shard dan mendukung penyaringan melalui penyaringan acara Lambda sehingga hanya item yang sesuai dengan kriteria tertentu yang diproses. Beberapa pelanggan mungkin memiliki persyaratan untuk mendukung lebih dari dua konsumen. Orang lain mungkin perlu memperkaya peristiwa perubahan sebelum diproses, atau menggunakan pemfilteran dan perutean yang lebih canggih.

Mengintegrasikan DynamoDB EventBridge dengan dapat mendukung persyaratan tersebut.

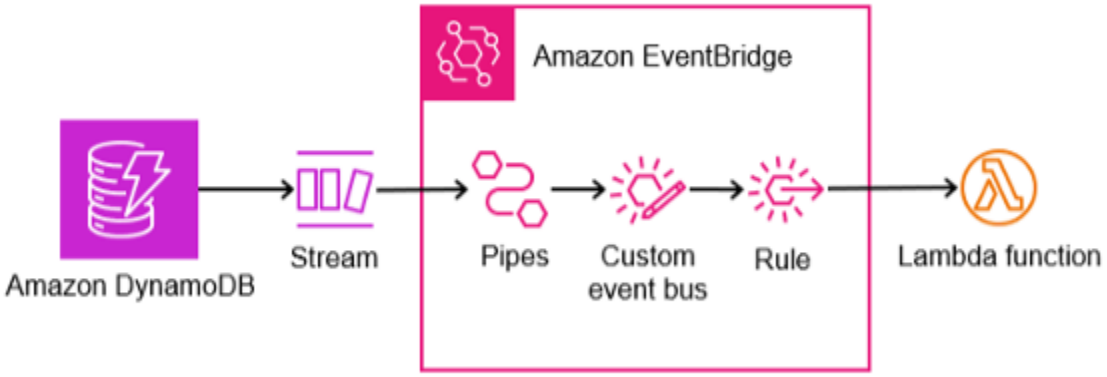
Amazon EventBridge adalah layanan tanpa server yang menggunakan peristiwa untuk menghubungkan komponen aplikasi bersama-sama, sehingga memudahkan Anda untuk membangun aplikasi berbasis peristiwa yang dapat diskalakan. EventBridge menawarkan integrasi asli dengan Amazon DynamoDB EventBridge melalui Pipes, memungkinkan aliran data yang mulus dari DynamoDB ke bus. EventBridge Bus itu kemudian dapat melakukan fan-out ke beberapa aplikasi dan layanan melalui seperangkat aturan dan target.

Topik

- Cara kerjanya
- Membuat integrasi melalui konsol
- Langkah selanjutnya

Cara kerjanya

Integrasi antara DynamoDB EventBridge dan pipa menggunakan DynamoDB Streams untuk menangkap urutan perubahan tingkat item yang diatur waktu dalam tabel DynamoDB. Setiap catatan yang ditangkap dengan cara ini berisi data yang dimodifikasi dalam tabel.



EventBridge Pipa mengkonsumsi peristiwa dari DynamoDB Streams dan merutekan mereka ke target seperti bus (bus EventBridge acara adalah router yang menerima acara dan mengirimkannya ke tujuan, juga disebut target). Pengiriman didasarkan pada aturan mana yang cocok dengan isi acara. Secara opsional, pipa juga mencakup kemampuan untuk memfilter peristiwa tertentu dan melakukan pengayaan pada data peristiwa sebelum mengirimkannya ke target.

Sementara EventBridge mendukung beberapa jenis target, pilihan umum saat menerapkan desain fan-out adalah dengan menggunakan fungsi Lambda sebagai target. Contoh berikut menunjukkan integrasi dengan target fungsi Lambda.

Membuat integrasi melalui konsol

Ikuti langkah-langkah di bawah ini untuk membuat integrasi melalui AWS Management Console.

1. Aktifkan DynamoDB Streams pada tabel sumber dengan mengikuti langkah-langkah di Mengaktifkan aliran bagian panduan pengembang DynamoDB. Jika DynamoDB Streams sudah diaktifkan pada tabel sumber, verifikasi bahwa saat ini ada kurang dari dua konsumen. Konsumen dapat berupa fungsi Lambda, DynamoDB Global Tables, Amazon DynamoDB integrasi Zero-ETL dengan OpenSearch Amazon Service, atau aplikasi yang membaca langsung dari aliran seperti melalui adaptor DynamoDB Streams Kinesis.

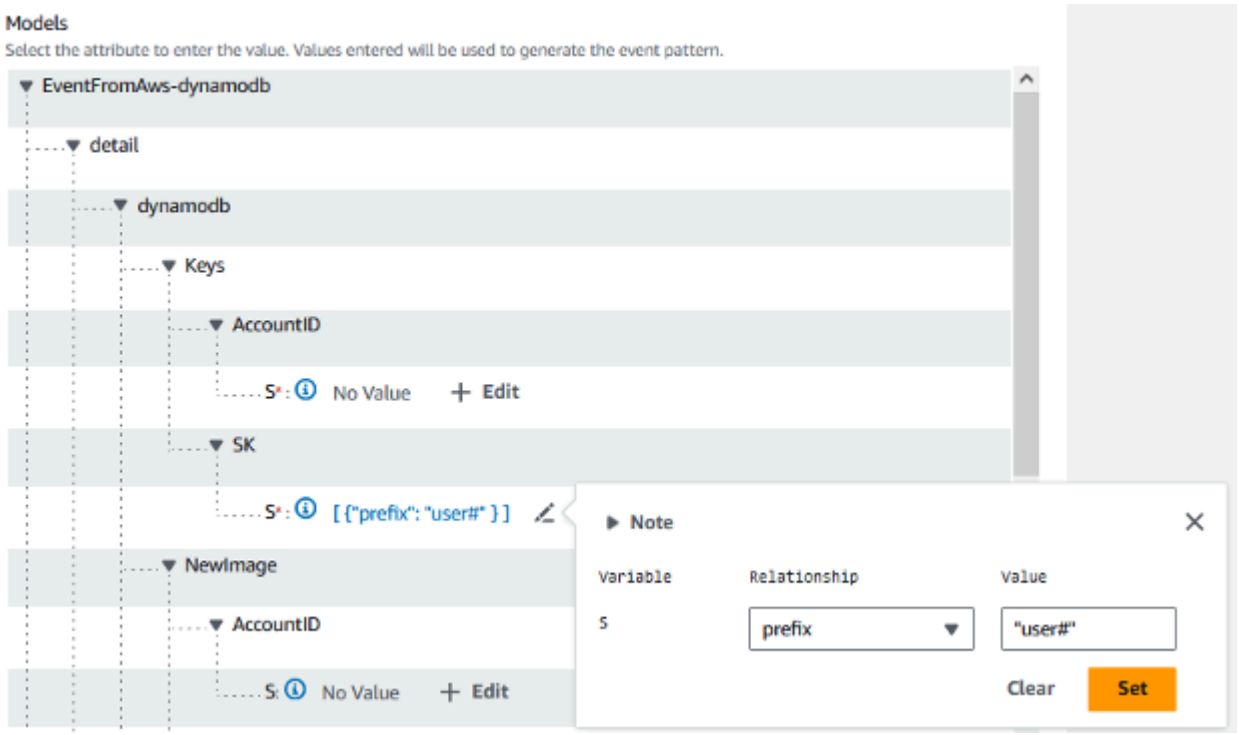
2. Buat bus EventBridge acara dengan mengikuti langkah-langkah di bagian Membuat bus EventBridge acara Amazon di panduan EventBridge pengguna.
   a. Saat membuat bus acara, aktifkan penemuan Skema.
3. Buat EventBridge pipa dengan mengikuti langkah-langkah di bagian Membuat EventBridge pipa Amazon dari panduan EventBridge pengguna.
   a. Saat mengonfigurasi sumber, di bidang Sumber pilih DynamoDB dan di bidang DynamoDB Streams pilih nama aliran tabel sumber.
   b. Saat mengonfigurasi target, di bidang Layanan target pilih bus EventBridge acara dan di bidang Bus acara sebagai target pilih bus acara yang dibuat pada langkah 2.
4. Tulis item contoh ke tabel DynamoDB sumber untuk memicu peristiwa. Ini akan memungkinkan EventBridge untuk menyimpulkan skema dari item contoh. Skema ini dapat digunakan untuk membuat aturan untuk peristiwa routing. Misalnya, jika Anda menerapkan pola desain yang melibatkan atribut overloading, Anda mungkin ingin memicu aturan yang berbeda tergantung pada nilai kunci pengurutan Anda. Detail tentang cara menulis item ke DynamoDB dapat ditemukan di bagian Bekerja dengan item dan atribut dari panduan pengembang DynamoDB.
5. Buat contoh fungsi Lambda Python yang akan digunakan sebagai target dengan mengikuti langkah-langkah dalam Membangun fungsi Lambda dengan bagian Python dari panduan pengembang Lambda. Saat membuat fungsi Anda, Anda dapat menggunakan kode contoh di bawah ini untuk menunjukkan integrasi. Ketika dipanggil, itu akan mencetak `NewImage` dan `OldImage` diterima dengan acara yang dapat dilihat di CloudWatch Log.

```
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. Buat EventBridge aturan yang akan merutekan peristiwa ke fungsi Lambda baru Anda dengan mengikuti langkah-langkah di bagian Buat aturan yang bereaksi terhadap panduan pengguna peristiwa EventBridge .

a. Saat menentukan detail aturan, pilih nama bus acara yang Anda buat di langkah 2 sebagai bus Acara.

b. Saat membuat pola acara, ikuti panduan untuk skema yang ada. Di sini, Anda dapat memilih registri skema yang ditemukan dan skema yang ditemukan untuk acara Anda. Ini memungkinkan Anda mengonfigurasi pola peristiwa khusus untuk kasus penggunaan yang hanya merutekan pesan yang cocok dengan atribut tertentu. Misalnya, jika Anda ingin mencocokkan hanya pada item DynamoDB tempat SK `“user#”` dimulai, Anda akan menggunakan konfigurasi seperti ini.



c. Klik Buat pola acara di JSON setelah Anda selesai mendesain pola terhadap skema Anda. Jika Anda ingin mencocokkan semua peristiwa yang muncul di DynamoDB Streams, gunakan JSON berikut untuk pola acara.

```
{
  "source": ["aws.dynamodb"]
}
```

d. Saat memilih target, ikuti panduan untuk AWS layanan. Di bidang Pilih target, pilih “Fungsi Lambda”. Di bidang Fungsi, pilih fungsi Lambda yang Anda buat di langkah 5.

7. Anda sekarang dapat menghentikan penemuan skema di bus acara Anda dengan mengikuti langkah-langkah di bagian Memulai atau menghentikan penemuan skema di bus acara pada panduan EventBridge pengguna.

8. Tulis item contoh kedua ke tabel DynamoDB sumber untuk memicu peristiwa. Validasi bahwa acara berhasil diproses di setiap langkah.

   a. Lihat CloudWatch metrik PutEventsApproximateSuccessCountuntuk bus acara Anda dengan mengikuti EventBridge bagian Pemantauan Amazon pada panduan EventBridge pengguna.

   b. Lihat log fungsi untuk fungsi Lambda Anda dengan mengikuti bagian Pemantauan dan pemecahan masalah fungsi Lambda dari panduan pengembang Lambda. Jika fungsi Lambda Anda menggunakan kode contoh yang disediakan, Anda akan melihat `NewImage` dan `OldImage` dari DynamoDB Streams dicetak dalam grup log Log. CloudWatch

   c. Lihat metrik jumlah kesalahan dan tingkat keberhasilan (%) untuk fungsi Lambda Anda dengan mengikuti bagian Pemantauan dan pemecahan masalah fungsi Lambda dari panduan pengembang Lambda.

Langkah selanjutnya

Contoh ini menyediakan integrasi dasar dengan fungsi Lambda tunggal sebagai target. Untuk pemahaman yang lebih baik tentang konfigurasi yang lebih kompleks, seperti membuat beberapa aturan, membuat beberapa target, mengintegrasikan dengan layanan lain, dan memperkaya peristiwa, lihat panduan EventBridge pengguna lengkap: Memulai. EventBridge

> **Note**
>
> Waspadai EventBridge kuota apa pun yang mungkin relevan dengan aplikasi Anda. Sementara skala kapasitas DynamoDB Streams dengan tabel Anda, kuota terpisah. EventBridge Kuota umum yang harus diperhatikan dalam aplikasi besar adalah batas throttle Invocations dalam transaksi per detik dan PutEvents batas throttle dalam transaksi per detik. Kuota ini menentukan jumlah pemanggilan yang dapat dikirim ke target dan jumlah acara yang dapat dimasukkan ke dalam bus per detik.

Mengintegrasikan DynamoDB dengan Amazon Managed Streaming untuk Apache Kafka

Amazon Managed Streaming for Apache Kafka (Amazon MSK) memudahkan untuk menelan dan memproses data streaming secara real time dengan layanan Apache Kafka yang dikelola sepenuhnya dan sangat tersedia.

Apache Kafka adalah penyimpanan data terdistribusi yang dioptimalkan untuk menelan dan memproses data streaming secara real-time. Kafka dapat memproses aliran catatan, secara efektif menyimpan aliran catatan dalam urutan pembuatan catatan, dan menerbitkan serta berlangganan aliran catatan.

Karena fitur-fitur ini, Apache Kafka sering digunakan untuk membangun pipeline data streaming real-time. Pipeline data andal memproses dan memindahkan data dari satu sistem ke sistem lain dan dapat menjadi bagian penting dari mengadopsi strategi database yang dibangun khusus dengan memfasilitasi penggunaan beberapa database yang masing-masing mendukung kasus penggunaan yang berbeda.

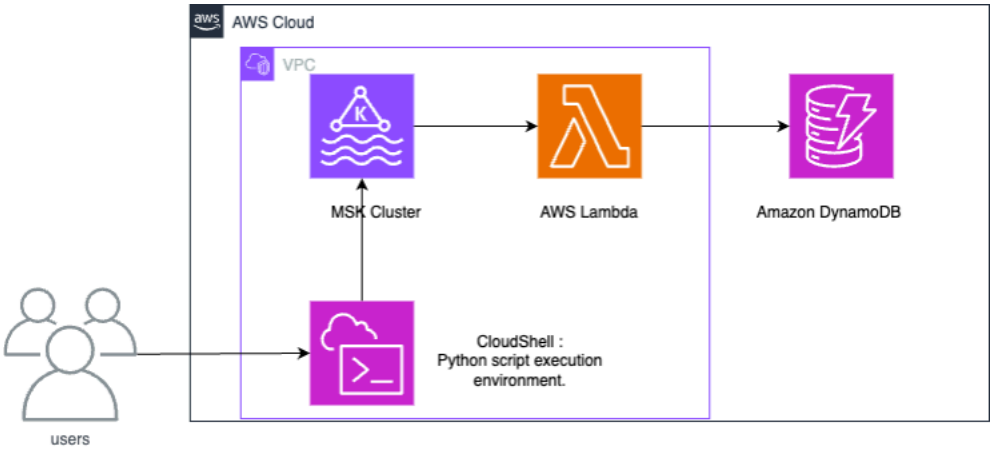
Amazon DynamoDB adalah target umum dalam pipeline data ini untuk mendukung aplikasi yang menggunakan model data nilai kunci atau dokumen dan menginginkan skalabilitas tanpa batas dengan kinerja milidetik satu digit yang konsisten.

Topik

- Cara kerjanya
- Siapkan integrasi antara Amazon MSK dan DynamoDB
- Langkah selanjutnya

Cara kerjanya

Integrasi antara Amazon MSK dan DynamoDB menggunakan fungsi Lambda untuk mengkonsumsi catatan dari Amazon MSK dan menuliskannya ke DynamoDB.



Lambda secara internal melakukan polling untuk pesan baru dari Amazon MSK dan kemudian secara sinkron memanggil fungsi Lambda target. Payload acara fungsi Lambda berisi kumpulan pesan dari

Amazon MSK. Untuk integrasi antara Amazon MSK dan DynamoDB, fungsi Lambda menulis pesan ini ke DynamoDB.

Siapkan integrasi antara Amazon MSK dan DynamoDB

> **Note**
>
> Anda dapat mengunduh sumber daya yang digunakan dalam contoh ini di GitHub repositori berikut.

Langkah-langkah di bawah ini menunjukkan cara menyiapkan contoh integrasi antara Amazon MSK dan Amazon DynamoDB. Contoh ini mewakili data yang dihasilkan oleh perangkat Internet of Things (IoT) dan dicerna ke Amazon MSK. Karena data tertelan ke Amazon MSK, data dapat diintegrasikan dengan layanan analitik atau alat pihak ketiga yang kompatibel dengan Apache Kafka, memungkinkan berbagai kasus penggunaan analitik. Mengintegrasikan DynamoDB juga menyediakan pencarian nilai kunci dari catatan perangkat individual.

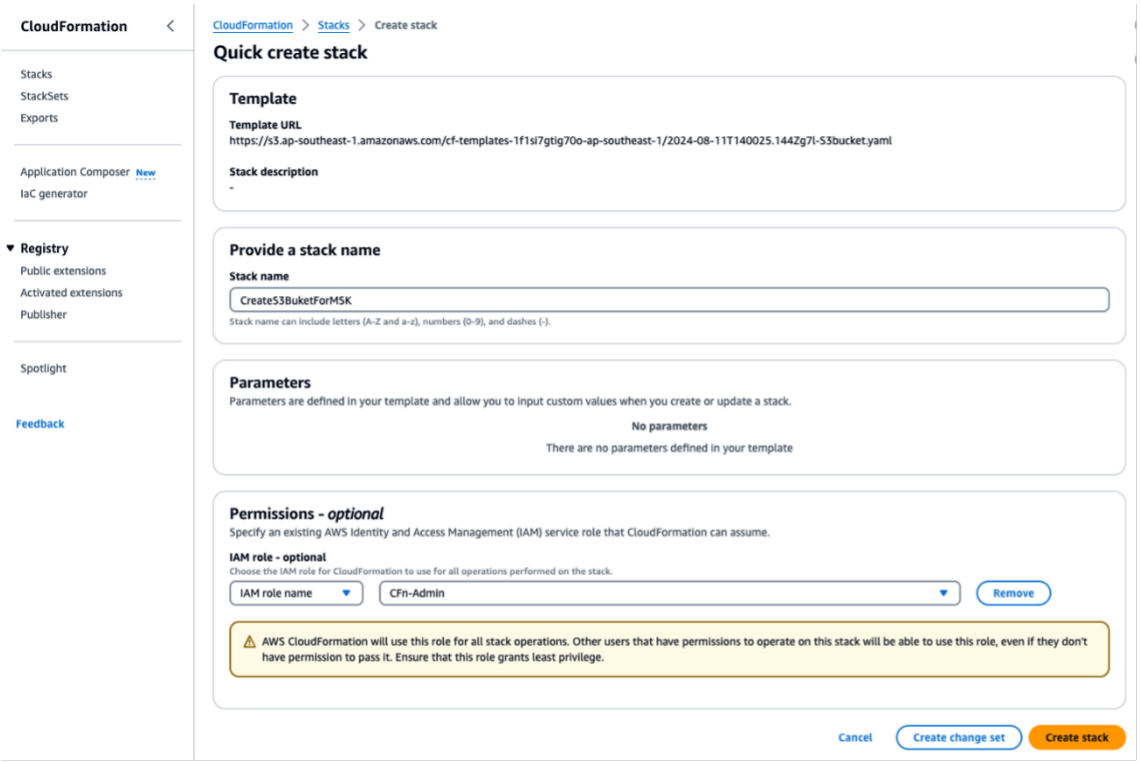
Contoh ini akan menunjukkan bagaimana skrip Python menulis data sensor IoT ke Amazon MSK. Kemudian, fungsi Lambda menulis item dengan kunci partisi "`deviceid`" ke DynamoDB.

CloudFormation Template yang disediakan akan membuat sumber daya berikut: Bucket Amazon S3, VPC Amazon, kluster MSK Amazon, dan untuk menguji operasi AWS CloudShell data.

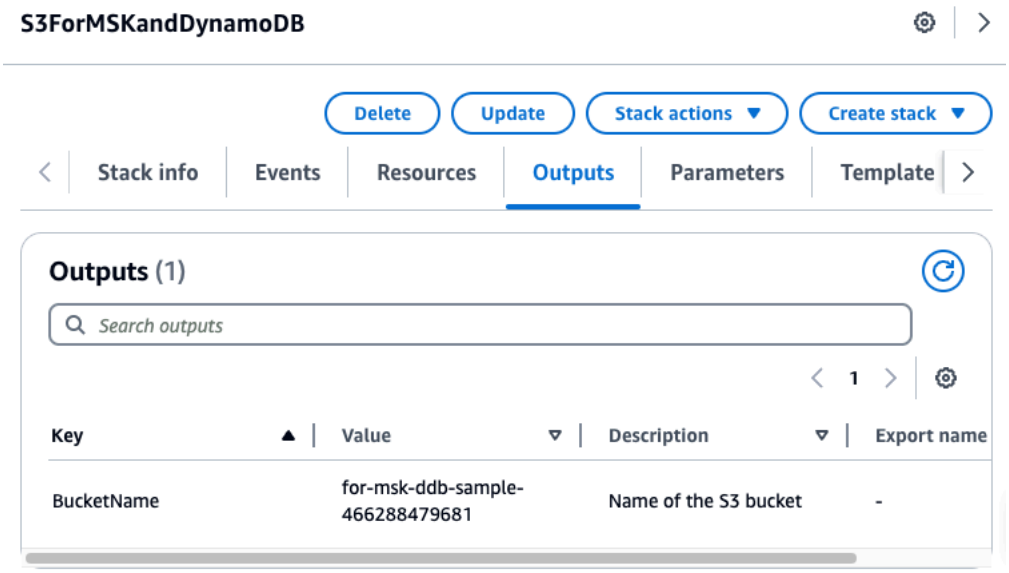
Untuk menghasilkan data pengujian, buat topik MSK Amazon dan kemudian buat tabel DynamoDB. Anda dapat menggunakan Session Manager dari konsol manajemen untuk masuk ke CloudShell sistem operasi dan menjalankan skrip Python.

Setelah menjalankan CloudFormation template, Anda dapat menyelesaikan membangun arsitektur ini dengan melakukan operasi berikut.

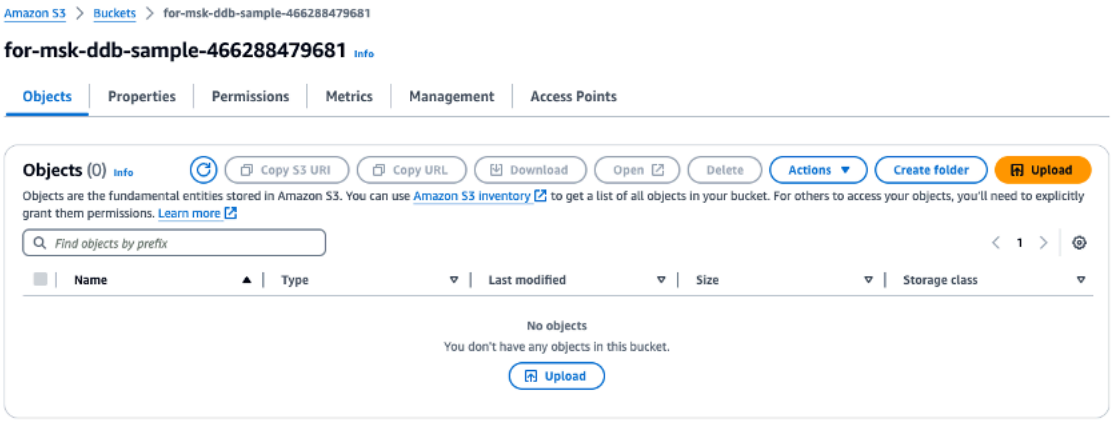
1. Jalankan CloudFormation template `S3bucket.yaml` untuk membuat bucket S3. Untuk skrip atau operasi berikutnya, jalankan di Wilayah yang sama. Masukkan `ForMSKTestS3` sebagai nama CloudFormation tumpukan.



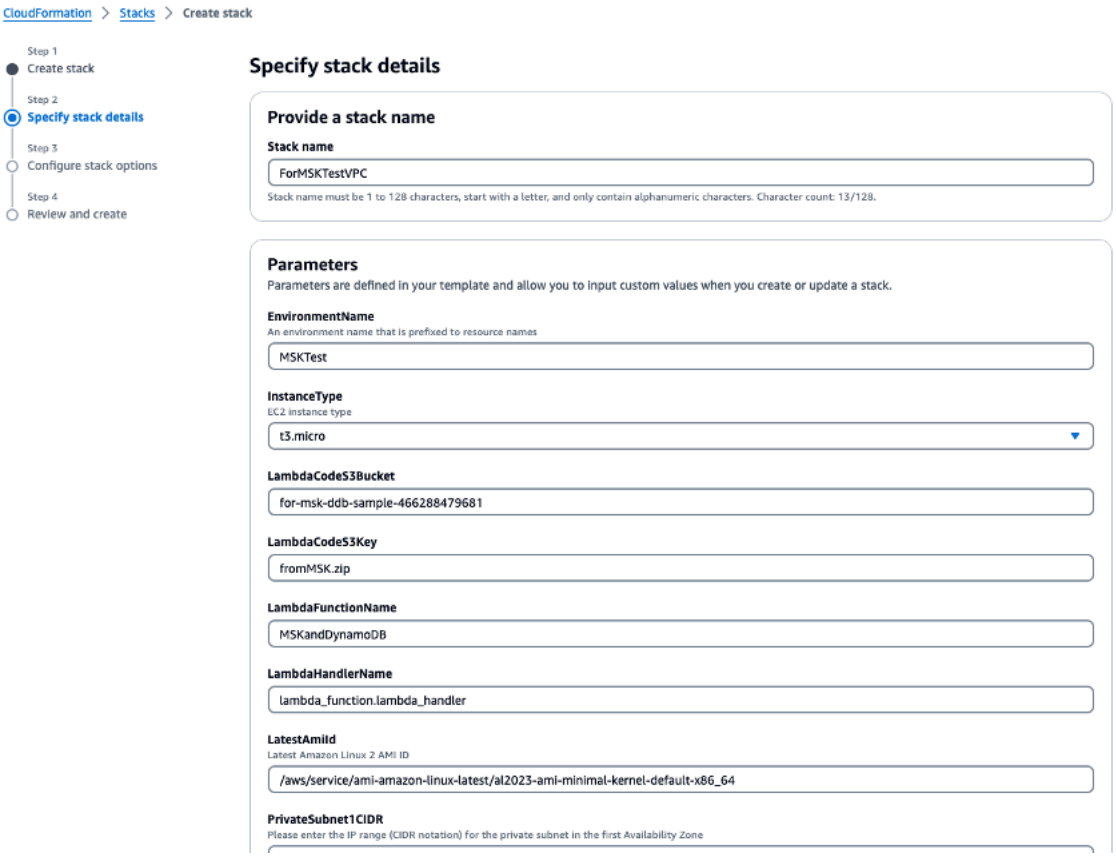
Setelah ini selesai, catat output nama bucket S3 di bawah Output. Anda akan membutuhkan nama di Langkah 3.



2. Unggah file ZIP yang diunduh `fromMSK.zip` ke bucket S3 yang baru saja Anda buat.



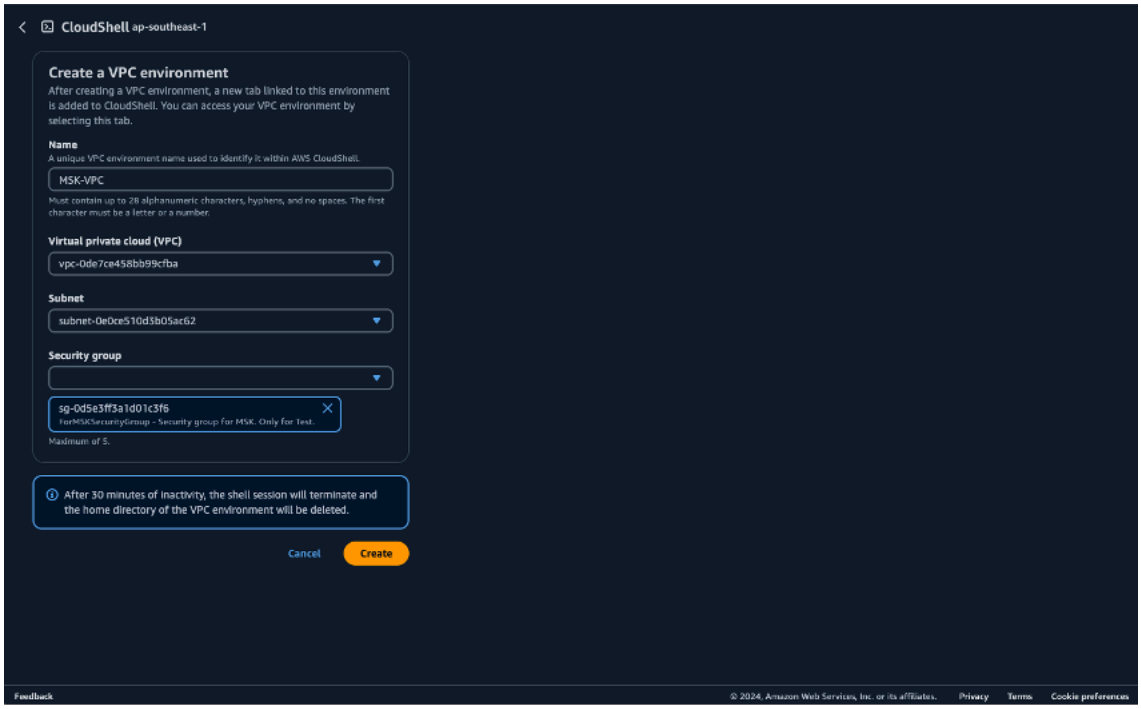
3. Jalankan CloudFormation template `VPC.yaml` untuk membuat fungsi VPC, Amazon MSK cluster, dan Lambda. Pada layar input parameter, masukkan nama bucket S3 yang Anda buat di Langkah 1 di mana ia meminta bucket S3. Atur nama CloudFormation tumpukan ke`ForMSKTestVPC`.



4. Siapkan lingkungan untuk menjalankan skrip Python di. CloudShell Anda dapat menggunakan CloudShell pada AWS Management Console. Untuk informasi selengkapnya tentang penggunaan CloudShell, lihat Memulai dengan AWS CloudShell. Setelah memulai CloudShell,

buat CloudShell yang termasuk dalam VPC yang baru saja Anda buat untuk terhubung ke Amazon MSK Cluster. Buat CloudShell di subnet pribadi. Isi kolom berikut:

1. Nama - dapat diatur ke nama apa pun. Contohnya adalah MSK-VPC
2. VPC - pilih MSKTest
3. Subnet - pilih MSKTest Private Subnet () AZ1
4. SecurityGroup- pilih Untuk MSKSecurity Grup



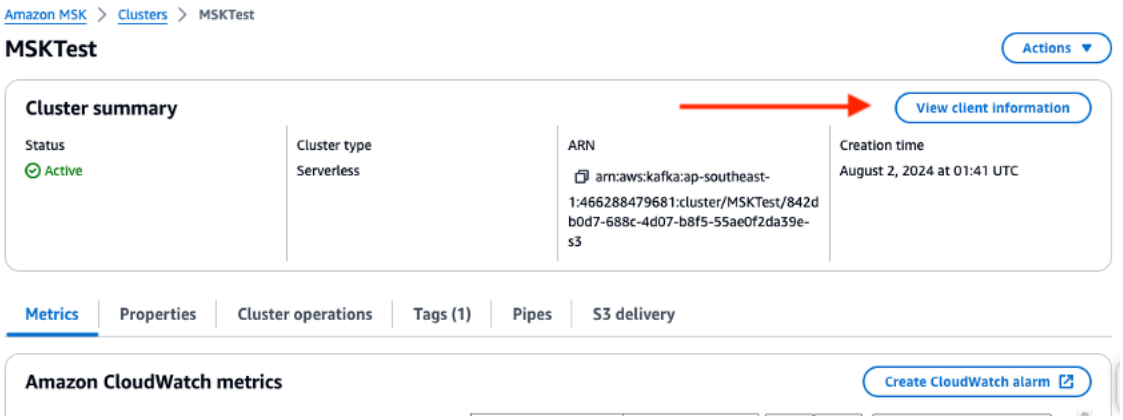
Setelah CloudShell milik Private Subnet dimulai, jalankan perintah berikut:

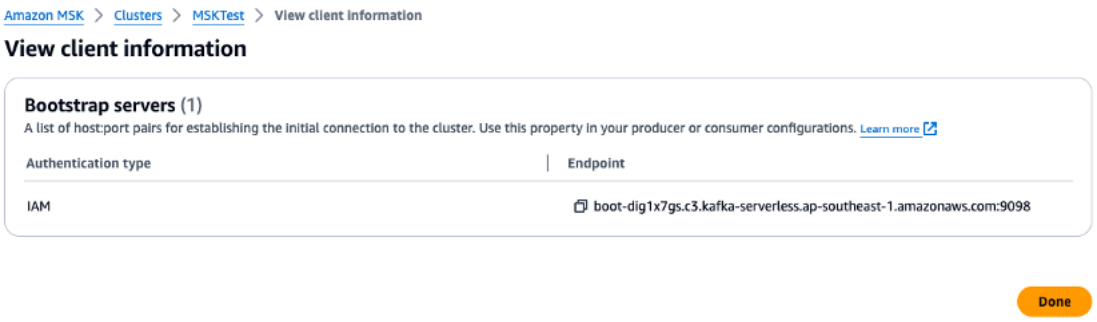
```
pip install boto3 kafka-python aws-msk-iam-sasl-signer-python
```

5. Unduh skrip Python dari bucket S3.

```
aws s3 cp s3://[YOUR-BUCKET-NAME]/pythonScripts.zip ./
unzip pythonScripts.zip
```

6. Periksa konsol manajemen dan atur variabel lingkungan untuk URL broker dan nilai Region dalam skrip Python. Periksa titik akhir broker kluster MSK Amazon di konsol manajemen.





7. Mengatur variabel lingkungan pada CloudShell. Jika Anda menggunakan US West (Oregon):

```
export AWS_REGION="us-west-2"
export MSK_BROKER="boot-YOURMSKCLUSTER.c3.kafka-serverless.ap-
southeast-1.amazonaws.com:9098"
```

8. Jalankan skrip Python berikut.

   Buat topik MSK Amazon:

```
python ./createTopic.py
```

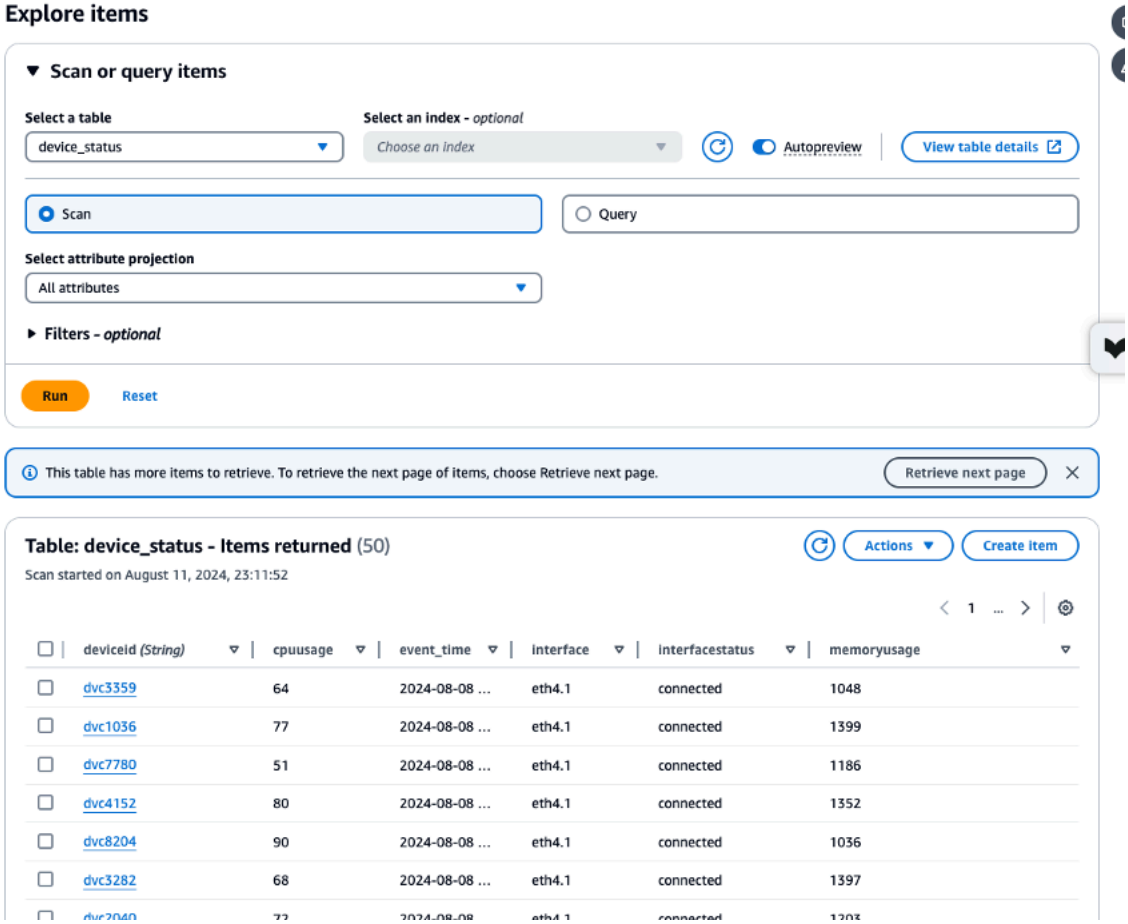
   Buat tabel DynamoDB:

```
python ./createTable.py
```

   Tulis data uji ke topik MSK Amazon:

```
python ./kafkaDataGen.py
```

9. Periksa CloudWatch metrik untuk sumber daya Amazon MSK, Lambda, dan DynamoDB yang dibuat, dan verifikasi data yang disimpan dalam tabel `device_status` menggunakan DynamoDB Data Explorer untuk memastikan semua proses berjalan dengan benar. Jika setiap proses dijalankan tanpa kesalahan, Anda dapat memeriksa apakah data pengujian yang ditulis dari CloudShell ke Amazon MSK juga ditulis ke DynamoDB.



10. Setelah selesai dengan contoh ini, hapus sumber daya yang dibuat dalam tutorial ini. Hapus dua CloudFormation tumpukan: `ForMSKTestS3` dan`ForMSKTestVPC`. Jika penghapusan tumpukan berhasil diselesaikan, semua sumber daya akan dihapus.

Langkah selanjutnya

> **Note**
>
> Jika Anda membuat sumber daya saat mengikuti contoh ini, harap ingat untuk menghapusnya untuk menghindari biaya yang tidak terduga.

Integrasi mengidentifikasi arsitektur yang menghubungkan Amazon MSK dan DynamoDB untuk mengaktifkan data aliran untuk mendukung beban kerja OLTP. Dari sini, pencarian yang lebih kompleks dapat diwujudkan dengan menautkan DynamoDB dengan Service. OpenSearch Pertimbangkan EventBridge untuk mengintegrasikan dengan kebutuhan berbasis peristiwa yang

lebih kompleks, dan ekstensi seperti Amazon Managed Service untuk Apache Flink untuk throughput yang lebih tinggi dan persyaratan latensi yang lebih rendah.

Praktik terbaik integrasi dengan DynamoDB

Saat mengintegrasikan DynamoDB dengan layanan lain, Anda sebaiknya selalu mengikuti praktik terbaik dalam menggunakan setiap layanan individual. Namun, ada beberapa praktik terbaik untuk integrasi tertentu yang sebaiknya Anda pertimbangkan.

Topik

- Membuat snapshot di DynamoDB
- Menangkap perubahan data di DynamoDB

Membuat snapshot di DynamoDB

- Umumnya, kami merekomendasikan penggunaan ekspor ke Amazon S3 untuk membuat snapshot untuk replikasi awal. Cara ini hemat biaya, dan tidak akan menyaingi lalu lintas aplikasi Anda dalam hal throughput. Anda juga dapat mempertimbangkan cadangan dan memulihkan ke tabel baru yang diikuti dengan operasi pemindaian. Hal ini akan menghindari persaingan untuk throughput dengan aplikasi Anda, tetapi secara umum akan jauh lebih hemat biaya daripada ekspor.
- Selalu atur a `StartTime` saat melakukan ekspor. Hal ini memudahkan untuk menentukan dari mana Anda akan memulai pengambilan data perubahan (CDC).
- Saat menggunakan ekspor ke S3, tetapkan tindakan siklus hidup pada bucket S3. Biasanya, tindakan kedaluwarsa yang diatur pada 7 hari sudah aman, tetapi Anda harus mengikuti panduan apa pun yang mungkin dimiliki perusahaan Anda. Meskipun Anda secara eksplisit menghapus item Anda setelah dikonsumsi, tindakan ini dapat membantu menangkap masalah, yang membantu mengurangi biaya yang tidak perlu dan mencegah pelanggaran kebijakan.

Menangkap perubahan data di DynamoDB

- Jika Anda membutuhkan CDC yang mendekati waktu nyata, gunakan DynamoDB Streams atau Amazon Kinesis Data Streams (KDS). Ketika memutuskan layanan mana yang akan Anda gunakan, umumnya pertimbangkan mana yang paling mudah digunakan dengan layanan hilir. Jika Anda perlu menyediakan pemrosesan peristiwa dalam urutan pada tingkat kunci partisi, atau jika Anda memiliki item yang sangat besar, gunakan DynamoDB Streams.

- Jika Anda tidak memerlukan CDC mendekati waktu nyata, Anda dapat menggunakan ekspor ke Amazon S3 dengan ekspor inkremental untuk hanya mengekspor perubahan yang terjadi antara dua titik waktu.

  Jika Anda menggunakan ekspor ke S3 untuk menghasilkan snapshot, hal ini bisa sangat membantu karena Anda dapat menggunakan kode yang serupa untuk memproses ekspor inkremental. Biasanya, ekspor ke S3 sedikit lebih murah dibanding opsi streaming sebelumnya, tetapi biaya biasanya bukan merupakan faktor utama dalam menentukan opsi yang akan digunakan.
- Anda umumnya hanya dapat memiliki dua konsumen simultan dari aliran DynamoDB. Pertimbangkan hal ini saat merencanakan strategi integrasi Anda.
- Jangan gunakan pemindaian untuk mendeteksi perubahan. Hal ini mungkin bekerja dalam skala kecil, tetapi menjadi tidak praktis dengan cepat.

Menggunakan AI generatif dengan DynamoDB

Amazon DynamoDB adalah database tanpa server, NoSQL, yang dikelola sepenuhnya dengan kinerja milidetik satu digit pada skala apa pun. DynamoDB dioptimalkan untuk beban kerja throughput tinggi dan Anda dapat memperluas kemampuannya dengan mengintegrasikan dengan model AI generatif. Dengan menggunakan model AI generatif, Anda dapat bekerja dengan data yang disimpan dalam tabel DynamoDB secara real-time dan membangun aplikasi yang sadar kontekstual dan sangat personal. Anda juga dapat meningkatkan pengalaman pengguna akhir dengan memanfaatkan sepenuhnya data bisnis, pengguna, dan aplikasi Anda untuk menyesuaikan solusi AI generatif Anda.

Untuk informasi selengkapnya tentang gen AI dan solusi yang AWS disediakan untuk membangun aplikasi AI gen, lihat Mengubah bisnis Anda dengan AI generatif.

Topik

- Kasus penggunaan AI generatif untuk DynamoDB
- Blog AI generatif untuk DynamoDB
- Memanfaatkan integrasi DynamoDB Zero-ETL dengan Layanan OpenSearch

Kasus penggunaan AI generatif untuk DynamoDB

DynamoDB banyak digunakan dalam aplikasi percakapan bertenaga AI, seperti chatbots dan call center yang dibangun dengan Foundation Model (FM). Anda dapat mengakses FMs melalui Amazon Bedrock, Amazon SageMaker AI, atau penyedia model lainnya. Aplikasi tersebut biasanya menggunakan DynamoDB untuk meningkatkan personalisasi dan meningkatkan pengalaman pengguna di tiga pola data: data aplikasi, data bisnis, dan data pengguna. Beberapa contoh pola data tersebut adalah sebagai berikut:

- Penyimpanan data aplikasi, seperti riwayat pesan obrolan, melalui integrasi dengan LangChain, LlamaIndex, atau kode khusus. Konteks ini meningkatkan pengalaman pengguna dengan memungkinkan model untuk berkomunikasi bolak-balik dengan pengguna.
- Pembuatan pengalaman pengguna yang disesuaikan dengan memanfaatkan data bisnis, seperti inventaris, harga, dan dokumentasi.
- Aplikasi data pengguna, seperti riwayat web, pesanan sebelumnya, dan preferensi pengguna, untuk memberikan jawaban yang dipersonalisasi.

Misalnya, perusahaan asuransi dapat membangun chatbot menggunakan DynamoDB untuk memberikan akses model AI gen berbasis Retrieval-Augmented Generation (RAG) mereka ke data mendekati waktu nyata. Contoh data tersebut adalah tarif hipotek real-time, harga produk, salinan kontrak yang patuh/standar, riwayat web pengguna, dan preferensi pengguna. Menggabungkan DynamoDB dengan RAG menambahkan informasi mendalam dan terbaru tentang produk asuransi dan data pengguna. Ini memperkaya petunjuk dan jawaban untuk memberi pengguna akhir pengalaman yang akurat, dipersonalisasi, dan mendekati waktu nyata.

Demikian pula, pelanggan industri jasa keuangan menggunakan DynamoDB, basis pengetahuan Amazon Bedrock, dan agen Amazon Bedrock untuk membangun aplikasi AI gen berbasis RAG. Aplikasi ini dapat menggunakan laporan pendapatan sumber terbuka dan transkrip panggilan. Mereka juga dapat menggunakan portofolio khusus pengguna dan riwayat transaksi untuk menghasilkan ringkasan portofolio sesuai permintaan termasuk prospek untuk masa depan.

Blog AI generatif untuk DynamoDB

Artikel berikut menawarkan kasus penggunaan terperinci, praktik terbaik, dan step-by-step panduan untuk membantu Anda memanfaatkan kemampuan DynamoDB dalam membangun aplikasi canggih yang didukung AI.

- Model data Amazon DynamoDB untuk chatbot AI generatif
- Buat chatbot yang dapat diskalakan dan sadar konteks dengan Amazon DynamoDB, Amazon Bedrock, dan LangChain

Memanfaatkan integrasi DynamoDB Zero-ETL dengan Layanan OpenSearch

Anda dapat menggunakan Amazon Bedrock dengan DynamoDB untuk menyediakan akses tanpa server ke model dasar (FMs), seperti Amazon Titan dan model pihak ketiga lainnya. Anda dapat memanfaatkan integrasi nol-ETL dengan Amazon OpenSearch Service untuk mengaktifkan kemampuan pencarian vektor saat membuat aplikasi AI generatif. Generative AI dengan DynamoDB Zero-ETL OpenSearch untuk integrasi dan lokakarya Amazon Bedrock memberi Anda pengalaman langsung dalam menyiapkan integrasi DynamoDB Zero-ETL dengan. OpenSearch Workshop ini melakukan tugas-tugas berikut:

- Membuat pipeline dari tabel DynamoDB Anda ke. OpenSearch

- Membuat Konektor Batuan Dasar Amazon di OpenSearch.
- Kueri Amazon Bedrock memanfaatkan OpenSearch sebagai toko vektor.
- Menggunakan Claude FM di Amazon Bedrock untuk membuat tanggapan tertulis dalam bahasa Inggris sederhana yang menjelaskan hasil pencarian yang dikembalikan oleh. OpenSearch

Workshop ini memungkinkan Anda untuk mengintegrasikan DynamoDB OpenSearch dengan membangun aplikasi AI generatif. Ini juga menunjukkan kemampuan query fleksibel di seluruh mesin database untuk membantu Anda mengintegrasikan DynamoDB dan untuk kasus penggunaan tradisional. OpenSearch Lokakarya ini adalah salah satu dari tujuh modul di Amazon DynamoDB Immersion Day. Anda dapat menjalankan lokakarya ini di mana saja Akun AWS.

Anda juga dapat merujuk ke posting blog berikut tentang cara mengatur integrasi nol-ETL antara DynamoDB dan Layanan. OpenSearch Posting blog ini juga menjelaskan cara mengatur konektor model di OpenSearch Layanan untuk secara otomatis menghasilkan embeddings menggunakan Amazon Bedrock untuk data yang masuk. Pencarian vektor untuk Amazon DynamoDB dengan nol ETL untuk Amazon Service. OpenSearch

Kuota dan kendala untuk Amazon DynamoDB

Topik ini menjelaskan kuota saat ini, sebelumnya disebut sebagai batas, dalam Amazon DynamoDB. Topik ini juga menjelaskan bagaimana Anda dapat melakukan tugas manajemen kuota, misalnya, melihat kuota Anda saat ini dan meminta peningkatan kuota.

Topik

- Melakukan tugas manajemen kuota di DynamoDB
- Meminta peningkatan kuota di DynamoDB
- Kuota di Amazon DynamoDB
- Kendala di Amazon DynamoDB

Melakukan tugas manajemen kuota di DynamoDB

Amazon DynamoDB memiliki beberapa komponen layanan, seperti tabel, aliran, indeks, dan banyak lagi. Saat Anda membuat Akun AWS, ada kuota default (sebelumnya disebut sebagai batas) yang ditetapkan pada komponen ini. Kecuali dinyatakan lain, setiap kuota bersifat khusus per Wilayah. Anda dapat meminta kenaikan untuk beberapa kuota. Setelah kuota sumber daya tercapai, permintaan tambahan untuk membuat sumber daya tersebut gagal dengan pengecualian.

Mengakses kuota DynamoDB

Anda dapat bekerja dengan DynamoDB Service Quotas dengan cara berikut:

- AWS Management Console

  Konsol Service Quotas adalah antarmuka berbasis browser yang dapat Anda gunakan untuk melihat dan mengelola Service Quotas Anda. Anda dapat mengakses Service Quotas dari AWS Management Console halaman mana pun dengan memilihnya di bilah navigasi atas, atau dengan mencari Service Quotas di halaman. AWS Management Console
- AWS Command Line Interface alat

  Saat menggunakan AWS Command Line Interface alat, Anda dapat mengeluarkan perintah di baris perintah sistem Anda untuk melakukan tugas Service Quotas. Alat baris perintah berguna jika Anda ingin membangun skrip yang melakukan AWS tugas.
- AWS SDKs

Anda dapat menggunakan AWS SDKs untuk berbagai bahasa pemrograman dan platform (misalnya, Java, Python, Ruby, .NET, iOS dan Android, dan lain-lain) untuk melakukan tugas Service Quotas.

Jika kuota yang dapat disesuaikan tidak tersedia di konsol Service Quotas, gunakan untuk membuat kasus AWS Support Center Console peningkatan Service Quotas.

Melihat kuota saat ini di konsol

Untuk melihat kuota DynamoDB Anda saat ini menggunakan konsol Service Quotas

1. Buka konsol Service Quotas di https://console.aws.amazon.com/servicequotas/home/services/dynamodb/quotas/
2. Dari bilah navigasi, di bagian atas layar, pilih Wilayah.
3. Konsol menampilkan detail tentang nama Kuota DynamoDB, nilai kuota tingkat akun yang diterapkan, nilai kuota default AWS , Pemanfaatan, dan Penyesuaian kuota di tingkat akun atau tingkat sumber daya.

   Jika nilai kuota atau pemanfaatan yang diterapkan tidak tersedia, konsol akan ditampilkan Tidak tersedia. Anda dapat meminta nilai kuota yang diterapkan melalui Support Center Console.
4. Pilih nama Kuota tertentu untuk melihat halaman Detail, yang menampilkan Deskripsi kuota, kode Kuota, ARN Kuota, Pemanfaatan, nilai kuota tingkat akun Terapan, Penyesuaian, dan nilai kuota default.AWS

   Jika berlaku, halaman Detail juga menampilkan opsi Pemantauan, Alarm, riwayat permintaan, dan Tag kuota apa pun.

Melihat kuota saat ini menggunakan AWS CLI

Untuk melihat nilai default untuk kuota DynamoDB:

- Panggil `ListDefaultServiceQuotas` operasi dengan kode layanan DynamoDB (dynamodb) untuk mengambil nilai default untuk kuota Amazon DynamoDB Service.

```
$ 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
        }......
    ]
}
```

Untuk melihat nilai kuota yang diterapkan:

- Panggil [ListServiceQuotas](#)operasi dengan kode layanan DynamoDB (dynamodb) untuk mengambil semua nilai kuota yang diterapkan baik di tingkat akun, tingkat sumber daya, atau semua level dengan meneruskan,, atau masing-masing sebagai nilai untuk parameter. `ACCOUNT` `RESOURCE` `ALL` `QuotaAppliedAtLevel` Contoh CLI berikut mengambil nilai kuota yang diterapkan di tingkat akun.

```
$ 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"
        }.....

        }
    ]
}
```

Meminta peningkatan kuota di DynamoDB

Anda dapat meminta peningkatan kuota untuk setiap Wilayah menggunakan konsol Service Quotas AWS CLI , atau kasus dukungan. Jika kuota yang dapat disesuaikan tidak tersedia di konsol Service Quotas, gunakan untuk membuat AWS Support Center Console kasus peningkatan kuota layanan.

Dukungan dapat menyetujui, menolak, atau menyetujui sebagian permintaan peningkatan kuota Anda. Kenaikan tidak diberikan segera, dan dapat memakan waktu beberapa hari untuk berlaku.

Untuk meminta peningkatan menggunakan konsol Kuota Layanan

1. Buka konsol Service Quotas di/ https://console.aws.amazon.com/servicequotas/home/services/dynamodb/quotas
2. Dari bilah navigasi, di bagian atas layar, pilih Wilayah.
3. Filter daftar dengan nama sumber daya. Misalnya, masukkan On-Demand untuk menemukan kuota untuk Instans On-Demand.
4. Jika kuota dapat disesuaikan, pilih kuota lalu pilih Minta peningkatan kuota.
5. Untuk Ubah nilai kuota, masukkan nilai kuota baru.
6. Pilih Minta.
7. Untuk melihat permintaan yang tertunda atau baru diselesaikan di konsol, pilih Dasbor dari panel navigasi. Untuk permintaan yang tertunda, pilih status permintaan untuk membuka penerimaan permintaan. Status awal dari permintaan adalah Tertunda. Setelah status berubah menjadi Kuota yang diminta, Anda akan melihat nomor kasus dengan Dukungan. Pilih nomor kasus untuk membuka tiket untuk permintaan Anda.

Untuk informasi selengkapnya, termasuk cara menggunakan AWS CLI atau SDKs meminta peningkatan kuota, lihat Meminta peningkatan kuota dalam Panduan Pengguna Service Quotas.

Kuota di Amazon DynamoDB

Bagian ini menjelaskan kuota saat ini, yang sebelumnya disebut sebagai batas, dalam Amazon DynamoDB. Kecuali ditentukan lain, masing-masing kuota berlaku untuk setiap Wilayah.

Topik

- Membaca/menulis throughput
- Kapasitas Terpesan
- Tabel
- Tabel global
- Indeks sekunder
- Atribut indeks sekunder yang diproyeksikan
- DynamoDB Streams
- Mengimpor dari Amazon S3
- Ekspor tabel ke Amazon S3

- Pencadangan dan pemulihan
- Wawasan Kontributor

Membaca/menulis throughput

Kuota default throughput

AWS menempatkan beberapa kuota default pada throughput yang dapat disediakan dan dikonsumsi akun Anda dalam suatu Wilayah.

Kuota throughput baca tingkat akun dan throughput tulis tingkat akun berlaku di tingkat akun. Kuota tingkat akun ini berlaku untuk jumlah kapasitas throughput yang disediakan untuk semua tabel akun Anda dan indeks sekunder global di Wilayah tertentu. Semua throughput akun yang tersedia dapat disediakan untuk satu tabel atau beberapa tabel. Kuota ini hanya berlaku untuk tabel yang menggunakan mode kapasitas yang disediakan.

Kuota throughput baca tingkat tabel dan throughput tulis tingkat tabel berlaku berbeda pada tabel yang menggunakan mode kapasitas yang disediakan, dan tabel yang menggunakan mode kapasitas sesuai permintaan.

Untuk tabel mode kapasitas yang disediakan dan GSIs, kuota adalah jumlah maksimum unit kapasitas baca dan tulis yang dapat disediakan untuk tabel apa pun atau salah satu miliknya di Wilayah. GSIs Total setiap tabel individu dan semua itu juga GSIs harus tetap di bawah kuota throughput baca dan tulis tingkat akun. Ini merupakan tambahan dari persyaratan bahwa total semua tabel yang disediakan dan mereka GSIs harus tetap di bawah kuota throughput baca dan tulis tingkat akun.

Untuk tabel mode kapasitas sesuai permintaan dan GSIs, kuota tingkat tabel adalah unit kapasitas baca dan tulis maksimum yang tersedia untuk tabel apa pun, atau GSI individu mana pun dalam tabel tersebut. Tidak ada kuota throughput baca dan tulis tingkat akun yang diterapkan pada tabel dalam mode sesuai permintaan.

Berikut ini adalah kuota throughput yang berlaku pada akun Anda, secara default.

> **Note**
>
> Anda dapat meminta sejumlah unit kapasitas baca (RCU) atau unit kapasitas tulis (WCU) untuk tabel DynamoDB Anda melalui peningkatan kuota layanan. Nilai-nilai yang tercantum dalam tabel berikut mewakili kuota default awal. Ini bukan batas maksimum untuk tabel Anda.

Nama kuota throughput	Sesuai Permintaan	Disediakan	Dapat Disesuaikan
Per tabel	40.000 unit permintaan baca dan 40.000 unit permintaan tulis	40.000 unit kapasitas baca dan 40.000 unit kapasitas tulis	Ya
Per akun	Tidak berlaku	80.000 unit kapasitas baca dan 80.000 unit kapasitas tulis	Ya
Throughput minimum untuk setiap tabel atau indeks sekunder global	Tidak berlaku	1 unit kapasitas baca dan 1 unit kapasitas tulis	Ya

Meningkatkan atau mengurangi throughput (untuk tabel yang disediakan)

Meningkatkan throughput yang disediakan

Anda dapat meningkatkan `ReadCapacityUnits` atau `WriteCapacityUnits` sesering yang diperlukan, menggunakan operasi AWS Management Console atau `UpdateTable`. Dalam satu panggilan, Anda dapat meningkatkan throughput yang disediakan untuk sebuah tabel, untuk indeks sekunder global apa pun pada tabel tersebut, atau untuk kombinasi semuanya. Pengaturan baru tidak berlaku sampai operasi `UpdateTable` selesai.

Anda tidak dapat melebihi kuota per akun saat Anda menambahkan kapasitas yang disediakan, dan DynamoDB tidak mengizinkan Anda meningkatkan kapasitas yang disediakan dengan sangat cepat. Selain pembatasan ini, Anda dapat meningkatkan kapasitas yang disediakan untuk tabel Anda sebanyak yang Anda perlukan. Untuk informasi selengkapnya tentang kuota per akun, lihat bagian sebelumnya, Kuota default throughput.

Menurunkan throughput yang disediakan

Untuk setiap tabel dan indeks sekunder global dalam operasi `UpdateTable`, Anda dapat mengurangi `ReadCapacityUnits` atau `WriteCapacityUnits` (atau keduanya). Pengaturan baru tidak berlaku sampai operasi `UpdateTable` selesai.

Terdapat kuota default pada jumlah penurunan kapasitas yang tersedia yang dapat Anda lakukan pada tabel DynamoDB per hari. Satu hari ditentukan berdasarkan Waktu Universal Terkoordinasi (UTC). Pada hari tertentu, Anda dapat memulai dengan melakukan hingga empat penurunan dalam satu jam selama Anda belum melakukan penurunan lainnya pada hari tersebut. Selanjutnya, Anda dapat melakukan satu penurunan tambahan per jam (setiap 60 menit sekali). Ini secara efektif membawa jumlah maksimum penurunan dalam sehari menjadi 27 kali.

> **Important**
>
> Batas penurunan tabel dan indeks sekunder global dipisahkan, sehingga setiap indeks sekunder global untuk tabel tertentu memiliki batas penurunannya sendiri. Namun, jika satu permintaan menurunkan throughput untuk tabel dan indeks sekunder global, permintaan tersebut akan ditolak jika salah satu permintaan tersebut melebihi batas saat ini. Permintaan tidak diproses sebagian.

Example

Dalam 4 jam pertama suatu hari, tabel dengan indeks sekunder global dapat dimodifikasi sebagai berikut:

- Kurangi `WriteCapacityUnits` atau `ReadCapacityUnits` tabel (atau keduanya) sebanyak empat kali.
- Kurangi `WriteCapacityUnits` atau `ReadCapacityUnits` (atau keduanya) dari indeks sekunder global sebanyak empat kali.

Pada akhir hari yang sama, tabel dan throughput indeks sekunder global berpotensi mengalami penurunan masing-masing sebanyak 27 kali lipat.

Kapasitas Terpesan

AWS menempatkan kuota default pada jumlah kapasitas cadangan aktif yang dapat dibeli akun Anda. Batas kuota adalah kombinasi dari kapasitas cadangan untuk unit kapasitas tulis (WCUs) dan unit kapasitas baca (RCUs).

Kuota kapasitas cadangan	Kapasitas terpesan aktif	Dapat Disesuaikan
Per akun	1.000.000 unit kapasitas yang disediakan (_) WCUs RCUs	Ya

Jika Anda mencoba membeli lebih dari 1.000.000 unit kapasitas yang disediakan dalam satu pembelian, Anda akan menerima kesalahan untuk batas kuota layanan ini. Jika Anda memiliki kapasitas terpesan aktif dan mencoba membeli kapasitas terpesan tambahan yang akan menghasilkan lebih dari 1.000.000 unit kapasitas aktif yang disediakan, Anda akan menerima kesalahan untuk batas kuota layanan ini.

Tabel

Ukuran tabel

Tidak ada batas praktis pada ukuran tabel. Tabel tidak dibatasi dalam jumlah item atau jumlah byte.

Jumlah maksimum tabel per akun per wilayah

Untuk AWS akun apa pun, ada kuota awal 2.500 tabel per AWS Wilayah.

Jika Anda membutuhkan lebih dari 2.500 tabel untuk satu akun, silakan hubungi tim AWS akun Anda untuk menjelajahi peningkatan hingga maksimum 10.000 tabel. Untuk lebih dari 10.000, praktik terbaik yang disarankan adalah menyiapkan beberapa akun, yang masing-masing dapat melayani hingga 10.000 tabel.

Tabel global

Kuota default berikut berlaku saat menggunakan tabel global.

Kuota tabel global default	Sesuai Permintaan	Disediakan
Throughput per tabel	40.000 unit permintaan baca dan 40.000 unit permintaan tulis	40.000 unit kapasitas baca dan 40.000 unit kapasitas tulis

Kuota tabel global default	Sesuai Permintaan	Disediakan
Data yang diisi ulang untuk replika baru per akun, per Wilayah, per hari	10 TB	10 TB

> **Note**
>
> Mungkin ada contoh di mana Anda perlu meminta kenaikan batas kuota melalui. AWS Dukungan Jika salah satu dari berikut ini berlaku untuk Anda, silakan lihat https://aws.amazon.com/support:
>
> - Jika Anda menambahkan replika untuk tabel yang dikonfigurasi untuk menggunakan lebih dari 40.000 unit kapasitas tulis (WCU), Anda harus meminta peningkatan kuota layanan untuk penambahan kuota WCU replika Anda.
> - Jika Anda menambahkan replika atau replika ke satu Wilayah tujuan dalam jangka waktu 24 jam dengan total gabungan lebih besar dari 10 TB, Anda harus meminta peningkatan kuota layanan untuk menambah kuota pengisian ulang data replika.
> - Jika Anda mengalami kesalahan seperti berikut:
>   - Tidak dapat membuat replika tabel 'example_table' di wilayah 'example_region_A' karena melebihi batas akun Anda saat ini di wilayah 'example_region_B'.

Indeks sekunder

Anda dapat menentukan hingga 5 indeks sekunder lokal per tabel.

Terdapat kuota default sebanyak 20 indeks sekunder global per tabel.

Atribut indeks sekunder yang diproyeksikan

Anda dapat memproyeksikan hingga 100 atribut yang digabungkan untuk semua indeks sekunder lokal dan global tabel. Kuota ini hanya berlaku untuk atribut proyeksi yang ditentukan pengguna.

Untuk `CreateTable` operasi, jika Anda menentukan `ProjectionType` dari`INCLUDE`, jumlah total atribut yang ditentukan dalam `NonKeyAttributes` penjumlahan di semua indeks sekunder tidak

boleh melebihi 100. Memproyeksikan nama atribut yang sama ke dalam dua indeks yang berbeda dihitung sebagai dua atribut yang berbeda terhadap kuota.

Kuota ini tidak berlaku untuk indeks sekunder dengan `ProjectionType` dari atau. `KEYS_ONLY` `ALL`

DynamoDB Streams

Pembaca serpihan secara bersamaan di DynamoDB Streams

Untuk tabel Single-region yang bukan tabel global, Anda dapat mendesain hingga dua proses simultan untuk dibaca dari pecahan DynamoDB Streams yang sama secara bersamaan. Melebihi batas ini dapat mengakibatkan throttling permintaan. Untuk tabel global, kami menyarankan Anda membatasi jumlah pembaca simultan menjadi satu untuk menghindari throttling permintaan.

Kapasitas tulis maksimum untuk tabel dengan DynamoDB Streams diaktifkan

AWS menempatkan beberapa kuota default pada kapasitas tulis untuk tabel DynamoDB dengan DynamoDB Streams diaktifkan. Kuota default ini hanya berlaku untuk tabel dalam mode kapasitas baca/tulis yang disediakan. jumlah.

- Wilayah AS Timur (Virginia Utara), AS Timur (Ohio), AS Barat (California Utara), AS Barat (Oregon), Amerika Selatan (Sao Paulo), Eropa (Frankfurt), Eropa (Irlandia), Asia Pasifik (Tokyo), Asia Pasifik (Seoul), Asia Pasifik (Singapura), Asia Pasifik (Sydney), Tiongkok (Beijing):
  - Per tabel – 40.000 unit kapasitas tulis
- Semua Wilayah lainnya:
  - Per tabel – 10.000 unit kapasitas tulis

Mengimpor dari Amazon S3

Impor DynamoDB dari Amazon S3 dapat mendukung hingga 50 tugas impor bersamaan dengan total ukuran objek sumber impor sebesar 15 TB sekaligus di wilayah us-east-1, us-west-2, dan eu-west-1. Di seluruh wilayah lainnya, didukung hingga 50 tugas impor bersamaan dengan ukuran total 1 TB. Setiap pekerjaan impor dapat mengambil hingga 50.000 objek Amazon S3 di semua wilayah. Untuk informasi selengkapnya tentang impor dan validasi, lihat kuota format impor dan validasi.

Ekspor tabel ke Amazon S3

Ekspor penuh: hingga 300 tugas ekspor bersamaan, atau hingga total 100TB dari semua ekspor tabel dalam penerbangan, dapat diekspor. Kedua batas ini diperiksa sebelum ekspor dimasukkan ke dalam antrean.

Ekspor tambahan: DynamoDB Incremental Export ke Amazon S3 dapat mendukung hingga 300 pekerjaan ekspor bersamaan atau hingga total 100TB dari semua ekspor tabel dalam penerbangan. Batas jendela periode ekspor adalah minimum 15 menit dan maksimum 24 jam.

Pencadangan dan pemulihan

DynamoDB mendukung hingga 50 pemulihan bersamaan dengan total 50 TB melalui DynamoDB on-demand atau backup berkelanjutan. AWS Backup mendukung hingga 50 pemulihan bersamaan dengan total 25 TB.

Wawasan Kontributor

Saat mengaktifkan Customer Insights di tabel DynamoDB, Anda masih tunduk pada batasan aturan Contributor Insights. Untuk informasi selengkapnya, lihat CloudWatch service quotas.

Kendala di Amazon DynamoDB

Bagian ini menjelaskan batasan saat ini, sebelumnya disebut sebagai batas, dalam Amazon DynamoDB.

Topik

- Mode kapasitas baca/tulis
- Indeks sekunder
- Tombol partisi dan kunci urutan
- Peraturan penamaan
- Jenis Data
- Item
- Atribut
- Parameter ekspresi
- Transaksi DynamoDB
- DynamoDB Streams

- DynamoDB Accelerator (DAX)
- Kendala khusus API
- Enkripsi DynamoDB saat diam

Mode kapasitas baca/tulis

Anda dapat mengganti tabel dari mode sesuai permintaan ke mode kapasitas yang disediakan kapan saja. Saat Anda melakukan beberapa sakelar di antara mode kapasitas, kondisi berikut berlaku:

- Anda dapat mengganti tabel yang baru dibuat dalam mode sesuai permintaan ke mode kapasitas yang disediakan kapan saja. Namun, Anda hanya dapat mengubahnya kembali ke mode sesuai permintaan 24 jam setelah stempel waktu pembuatan tabel.
- Anda dapat mengganti tabel yang ada dalam mode sesuai permintaan ke mode kapasitas yang disediakan kapan saja. Namun, Anda hanya dapat mengubahnya kembali ke mode sesuai permintaan 24 jam setelah stempel waktu terakhir yang menunjukkan peralihan ke sesuai permintaan.

Untuk informasi selengkapnya tentang beralih antara mode kapasitas baca dan tulis, lihatPertimbangan saat mengganti mode kapasitas di DynamoDB.

Ukuran unit kapasitas (untuk tabel yang disediakan)

Satu unit kapasitas baca = satu bacaan sangat konsisten per detik, atau dua bacaan akhir konsisten per detik, untuk sebuah item hingga ukuran 4 KB.

Satu unit kapasitas tulis = satu tulis per detik, untuk item berukuran hingga 1 KB.

Permintaan baca transaksional memerlukan dua unit kapasitas baca untuk melakukan satu pembacaan per detik untuk item hingga 4 KB.

Permintaan tulis transaksional memerlukan dua unit kapasitas tulis untuk melakukan satu penulisan per detik untuk item hingga 1 KB.

Ukuran unit permintaan (untuk tabel sesuai permintaan)

Satu unit kapasitas baca = satu bacaan sangat konsisten per detik, atau dua bacaan akhir konsisten per detik, untuk item hingga ukuran 4 KB.

Satu unit permintaan tulis = satu tulis per detik, untuk item hingga ukuran 1 KB.

Permintaan baca transaksional memerlukan dua unit permintaan baca untuk melakukan satu pembacaan per detik untuk item hingga 4 KB.

Permintaan tulis transaksional memerlukan dua unit permintaan tulis untuk melakukan satu penulisan per detik untuk item hingga 1 KB.

Indeks sekunder

Atribut Indeks Sekunder yang diproyeksikan per tabel

Anda dapat memproyeksikan total hingga 100 atribut ke semua indeks sekunder lokal dan global tabel. Ini hanya berlaku untuk atribut proyeksi yang ditentukan pengguna.

Dalam operasi `CreateTable`, jika Anda menentukan P`ProjectionType` dari `INCLUDE`, jumlah total atribut yang ditentukan dalam `NonKeyAttributes`, yang dijumlahkan di semua indeks sekunder, tidak boleh melebihi 100. Jika Anda memproyeksikan nama atribut yang sama ke dalam dua indeks berbeda, ini dihitung sebagai dua atribut yang berbeda ketika menentukan total.

Batasan ini tidak berlaku untuk indeks sekunder dengan `ProjectionType` `KEYS_ONLY` atau `ALL`.

Tombol partisi dan kunci urutan

Panjang kunci partisi

Panjang minimum nilai kunci partisi adalah 1 byte. Panjang maksimum adalah 2048 byte.

Nilai kunci partisi

Tidak ada batasan praktis untuk jumlah nilai kunci partisi yang berbeda, untuk tabel atau indeks sekunder.

Panjang kunci urutan

Panjang minimum nilai kunci urutan adalah 1 byte. Panjang maksimum adalah 1024 byte.

Nilai kunci urutan

Secara umum, tidak ada batas praktis pada jumlah nilai kunci urutan berbeda per nilai kunci partisi.

Pengecualian ini adalah untuk tabel dengan indeks sekunder. Koleksi item adalah set item yang memiliki nilai atribut kunci partisi yang sama. Dalam indeks sekunder global, kumpulan item tidak

bergantung pada tabel dasar (dan dapat memiliki atribut kunci partisi yang berbeda), tetapi dalam indeks sekunder lokal, tampilan yang diindeks ditempatkan di partisi yang sama dengan item dalam tabel dan berbagi atribut kunci partisi yang sama. Sebagai hasil dari lokalitas ini, ketika tabel memiliki satu atau lebih LSIs, koleksi item tidak dapat didistribusikan ke beberapa partisi.

Untuk tabel dengan satu atau lebih LSIs, koleksi item tidak boleh melebihi ukuran 10GB. Ini mencakup semua item tabel dasar dan semua tampilan LSI yang diproyeksikan yang memiliki nilai atribut kunci partisi yang sama. 10 GB adalah ukuran maksimum sebuah partisi. Untuk informasi selengkapnya, lihat Batas ukuran kumpulan item.

Peraturan penamaan

Nama tabel dan nama Indeks Sekunder

Nama untuk tabel dan indeks sekunder harus memiliki panjang minimal 3 karakter, namun tidak lebih dari 255 karakter. Berikut ini adalah karakter yang diperbolehkan:

- `A-Z`
- `a-z`
- `0-9`
- `_` (garis bawah)
- `-` (tanda hubung)
- `.` (titik)

Nama atribut

Secara umum, panjang nama atribut harus minimal satu karakter, tetapi panjangnya tidak lebih dari 64 KB.

Berikut ini adalah pengecualiannya. Panjang nama atribut berikut tidak boleh lebih dari 255 karakter:

- Nama kunci partisi indeks sekunder.
- Nama kunci urutan indeks sekunder.
- Nama atribut proyeksi yang ditentukan pengguna (hanya berlaku untuk indeks sekunder lokal). Dalam operasi `CreateTable`, jika Anda menentukan `ProjectionType` dari `INCLUDE`, nama atribut dalam parameter `NonKeyAttributes` dibatasi panjangnya. Jenis proyeksi `KEYS_ONLY` dan `ALL` tidak terpengaruh.

Nama atribut ini harus dikodekan menggunakan UTF-8, dan ukuran total setiap nama (setelah pengodean) tidak boleh melebihi 255 byte.

Jenis Data

String

Panjang String dibatasi oleh ukuran item maksimum 400 KB.

String adalah Unicode dengan pengodean biner UTF-8. Karena UTF-8 adalah pengodean lebar variabel, DynamoDB menentukan panjang String menggunakan UTF-8 byte-nya.

Angka

Angka dapat memiliki hingga 38 digit presisi, dan dapat positif, negatif, atau nol.

- Rentang positif: `1E-130` hingga `9.99999999999999999999999999999999999999E+125`
- Rentang negatif: `-9.99999999999999999999999999999999999999E+125` hingga `-1E-130`

DynamoDB menggunakan string JSON untuk mewakili data Angka dalam permintaan dan balasan. Untuk informasi selengkapnya, lihat API tingkat rendah DynamoDB.

Jika presisi angka penting, Anda harus meneruskan angka ke DynamoDB menggunakan string yang Anda konversi dari jenis angka.

Biner

Panjang Biner dibatasi oleh ukuran item maksimum 400 KB.

Aplikasi yang bekerja dengan atribut Binari harus mengodekan data dalam format base64 sebelum mengirimkannya ke DynamoDB. Setelah menerima data, DynamoDB mendekodekannya ke dalam array byte yang tidak ditandatangani dan menggunakannya sebagai panjang atribut.

Item

Ukuran item

Ukuran item maksimum di DynamoDB adalah 400 KB, yang mencakup panjang biner nama atribut (panjang UTF-8) dan panjang nilai atribut (sekali lagi panjang biner). Nama atribut dihitung terhadap batas ukuran.

Misalnya, pertimbangkan sebuah item dengan dua atribut: satu atribut bernama "warna baju" dengan nilai "R" dan atribut lainnya bernama "ukuran baju" dengan nilai "M". Ukuran total item tersebut adalah 23 byte.

Ukuran item untuk tabel dengan Indeks Sekunder Lokal

Untuk setiap indeks sekunder lokal pada tabel, terdapat batas 400 KB untuk total hal berikut:

- Ukuran data item dalam tabel.
- Ukuran entri terkait (termasuk nilai kunci dan atribut yang diproyeksikan) di semua indeks sekunder lokal.

Atribut

Pasangan atribut nama-nilai per item

Ukuran kumulatif atribut per item harus sesuai dengan ukuran item DynamoDB maksimum (400 KB).

Jumlah nilai dalam daftar, peta, atau set

Tidak ada batasan jumlah nilai dalam Daftar, Peta, atau Kumpulan, selama item yang berisi nilai tersebut sesuai dengan batas ukuran item 400 KB.

Nilai atribut

Nilai atribut String dan Binari kosong diperbolehkan, jika atribut tersebut tidak digunakan sebagai atribut kunci untuk tabel atau indeks. Nilai String dan Biner kosong diperbolehkan di dalam jenis Set, Daftar, dan Peta. Nilai atribut tidak boleh berupa Set kosong (Set String, Set Angka, atau Set Biner). Namun, Daftar dan Peta kosong diperbolehkan.

Kedalaman atribut bertingkat

DynamoDB mendukung atribut bersarang hingga sedalam 32 tingkat.

Parameter ekspresi

Parameter ekspresi termasuk `ProjectionExpression`, `ConditionExpression`, `UpdateExpression`, dan `FilterExpression`.

Panjang

Panjang maksimum string ekspresi adalah 4 KB. Misalnya, ukuran `ConditionExpression a=b` adalah 3 byte.

Panjang maksimum nama atribut ekspresi tunggal atau nilai atribut ekspresi adalah 255 byte. Misalnya, `#name` adalah 5 byte; `:val` adalah 4 byte.

Panjang maksimum semua variabel substitusi dalam sebuah ekspresi adalah 2 MB. Ini adalah jumlah panjang semua `ExpressionAttributeNames` dan `ExpressionAttributeValues`.

Operator dan operan

Jumlah maksimum operator atau fungsi yang diperbolehkan dalam `UpdateExpression` adalah 300. Misalnya, UpdateExpression`SET a = :val1 + :val2 + :val3`berisi dua "+" operator.

Jumlah maksimum operan untuk komparator `IN` adalah 100.

Kata yang dicadangkan

DynamoDB tidak mencegah Anda menggunakan nama yang bertentangan dengan kata khusus. (Untuk daftar lengkap, lihat Kata-kata penggunaan khusus di DynamoDB.)

Namun, jika Anda menggunakan kata khusus dalam parameter ekspresi, Anda juga harus menentukan `ExpressionAttributeNames`. Untuk informasi selengkapnya, lihat Nama atribut ekspresi (alias) di DynamoDB.

Transaksi DynamoDB

Operasi API transaksional DynamoDB memiliki batasan berikut:

- Transaksi tidak boleh berisi lebih dari 100 item unik.
- Sebuah transaksi tidak boleh berisi lebih dari 4 MB data.
- Tidak ada dua tindakan dalam suatu transaksi yang dapat bekerja terhadap item yang sama dalam tabel yang sama. Misalnya, Anda tidak dapat melakukan `ConditionCheck` dan `Update` item yang sama dalam satu transaksi.
- Transaksi tidak dapat beroperasi pada tabel di lebih dari satu AWS akun atau Wilayah.
- Operasi transaksional memberikan jaminan atomisitas, konsistensi, isolasi, dan daya tahan (ACID) hanya di AWS Wilayah tempat penulisan dibuat awalnya. Transaksi tidak didukung di seluruh

Wilayah dalam tabel global. Misalnya, Anda memiliki tabel global dengan replika di Wilayah AS Timur (Ohio) dan AS Barat (Oregon) dan Anda melakukan operasi `TransactWriteItems` di Wilayah AS Timur (Virginia Utara). Dalam hal ini, Anda mungkin mengamati transaksi yang diselesaikan sebagian di Wilayah AS Barat (Oregon) saat perubahan direplikasi. Perubahan direplikasi ke Wilayah lain hanya setelah diterapkan di Wilayah sumber.

DynamoDB Streams

Pembaca serpihan secara bersamaan di DynamoDB Streams

Untuk tabel Wilayah tunggal yang bukan tabel global, Anda dapat merancang hingga dua proses untuk membaca dari serpihan DynamoDB Streams yang sama secara bersamaan. Melebihi batas ini dapat mengakibatkan throttling permintaan. Untuk tabel global, kami menyarankan Anda membatasi jumlah pembaca simultan menjadi satu untuk menghindari throttling permintaan.

DynamoDB Accelerator (DAX)

AWS Ketersediaan wilayah

Untuk daftar AWS Wilayah di mana DAX tersedia, lihat DynamoDB Accelerator (DAX) di. Referensi Umum AWS

Simpul

Klaster DAX terdiri dari tepat satu simpul primer, dan antara nol hingga sepuluh simpul replika baca.

Jumlah total node (per AWS akun) tidak dapat melebihi 50 dalam satu AWS Wilayah.

Grup parameter

Anda dapat membuat hingga 20 grup parameter DAX per Wilayah.

Grup subnet

Anda dapat membuat hingga 50 grup subnet DAX per Wilayah.

Dalam grup subnet, Anda dapat menentukan hingga 20 subnet.

> **Important**
>
> Cluster DAX mendukung maksimal 500 tabel DynamoDB. Setelah Anda melampaui 500 tabel DynamoDB, klaster Anda mungkin mengalami penurunan ketersediaan dan kinerja.

Kendala khusus API

`CreateTable`/`UpdateTable`/`DeleteTable`/`PutResourcePolicy`/`DeleteResourcePolicy`

Secara umum, Anda dapat memiliki hingga 500 CreateTable,,, UpdateTableDeleteTablePutResourcePolicy, dan DeleteResourcePolicypermintaan yang berjalan secara bersamaan dalam kombinasi apa pun. Akibatnya, jumlah total tabel dalam status `CREATING`, `UPDATING`, atau `DELETING` tidak boleh melebihi 500.

Anda dapat mengirimkan hingga 2.500 permintaan per detik permintaan API bidang kontrol yang dapat diubah (`CreateTableDeleteTableUpdateTablePutResourcePolicy`,,, dan`DeleteResourcePolicy`) di seluruh grup tabel. Namun, `DeleteResourcePolicy` permintaan `PutResourcePolicy` dan memiliki batas individu yang lebih rendah. Untuk informasi selengkapnya, lihat detail kuota berikut untuk `PutResourcePolicy` dan`DeleteResourcePolicy`.

`CreateTable`dan `PutResourcePolicy` permintaan yang mencakup kebijakan berbasis sumber daya akan dihitung sebagai dua permintaan tambahan untuk setiap KB kebijakan. Misalnya, `PutResourcePolicy` permintaan `CreateTable` atau dengan kebijakan ukuran 5 KB akan dihitung sebagai 11 permintaan. 1 untuk `CreateTable` permintaan dan 10 untuk kebijakan berbasis sumber daya (2 x 5 KB). Demikian pula, kebijakan ukuran 20 KB akan dihitung sebagai 41 permintaan. 1 untuk `CreateTable` permintaan dan 40 untuk kebijakan berbasis sumber daya (2 x 20 KB).

`PutResourcePolicy`

Anda dapat mengirimkan hingga 25 permintaan `PutResourcePolicy` API per detik di seluruh grup tabel. Setelah permintaan yang berhasil untuk tabel individual, tidak ada `PutResourcePolicy` permintaan baru yang didukung selama 15 detik berikutnya.

Ukuran maksimum yang didukung untuk dokumen kebijakan berbasis sumber daya adalah 20 KB. DynamoDB menghitung spasi putih saat menghitung ukuran kebijakan terhadap batas ini.

DeleteResourcePolicy

Anda dapat mengirimkan hingga 50 permintaan `DeleteResourcePolicy` API per detik di seluruh grup tabel. Setelah `PutResourcePolicy` permintaan yang berhasil untuk tabel individual, tidak ada `DeleteResourcePolicy` permintaan yang didukung selama 15 detik berikutnya.

BatchGetItem

Satu operasi `BatchGetItem` dapat mengambil maksimal 100 item. Ukuran total semua item yang diambil tidak boleh melebihi 16 MB.

BatchWriteItem

Satu operasi `BatchWriteItem` dapat berisi hingga 25 permintaan `PutItem`atau `DeleteItem`. Ukuran total semua item yang ditulis tidak boleh melebihi 16 MB.

DescribeStream

Anda dapat menelepon dengan `DescribeStream` kecepatan maksimum 10 kali per detik.

DescribeTableReplicaAutoScaling

Metode `DescribeTableReplicaAutoScaling` hanya mendukung 10 permintaan per detik.

DescribeLimits

`DescribeLimits` harus dipanggil hanya secara berkala. Anda dapat mengalami kesalahan throttling jika Anda memanggilnya lebih dari sekali dalam satu menit.

DescribeContributorInsights/ListContributorInsights/UpdateContributorInsights

`DescribeContributorInsights`, `ListContributorInsights`, dan `UpdateContributorInsights` hanya boleh dipanggil secara berkala. DynamoDB mendukung hingga lima permintaan per detik untuk masing-masing permintaan ini. APIs

`DescribeTable`/`ListTables`/`GetResourcePolicy`

Anda dapat mengirimkan hingga 2.500 permintaan per detik dari kombinasi permintaan API bidang kontrol hanya-baca (`DescribeTable`,`ListTables`, dan`GetResourcePolicy`). `GetResourcePolicy`API memiliki batas individu yang lebih rendah yaitu 100 permintaan per detik.

`DescribeTimeToLive`

`DescribeTimeToLive`Operasi dibatasi pada 10 unit permintaan baca per detik. Jika Anda melebihi batas ini, DynamoDB mengembalikan kesalahan. `ThrottlingException`

`Query`

Hasil yang ditetapkan dari `Query` dibatasi hingga 1 MB per panggilan. Anda dapat menggunakan `LastEvaluatedKey` dari respons kueri untuk mengambil lebih banyak hasil.

`Scan`

Hasil yang ditetapkan dari `Scan` dibatasi hingga 1 MB per panggilan. Anda dapat menggunakan `LastEvaluatedKey` dari respons pemindaian untuk mengambil hasil lainnya.

`UpdateKinesisStreamingDestination`

Saat melakukan `UpdateKinesisStreamingDestination` operasi, Anda dapat mengatur `ApproximateCreationDateTimePrecision` ke nilai baru maksimal 3 kali dalam periode 24 jam.

`UpdateTableReplicaAutoScaling`

Metode `UpdateTableReplicaAutoScaling` hanya mendukung sepuluh permintaan per detik.

`UpdateTableTimeToLive`

Metode `UpdateTableTimeToLive` hanya mendukung satu permintaan untuk mengaktifkan atau menonaktifkan `Time to Live (TTL)` per tabel tertentu per jam. Perubahan ini memerlukan waktu hingga satu jam untuk diproses sepenuhnya. Setiap `UpdateTimeToLive` panggilan tambahan untuk tabel yang sama selama durasi satu jam ini menghasilkan a ValidationException.

Enkripsi DynamoDB saat diam

Anda dapat beralih antara kunci yang Kunci yang dikelola AWS dikelola pelanggan hingga empat kali, kapan saja per jendela 24 jam, berdasarkan per tabel, mulai dari saat tabel dibuat. Kunci milik AWS Jika tidak ada perubahan dalam enam jam terakhir, perubahan tambahan diperbolehkan. Hal ini secara efektif menjadikan jumlah maksimum perubahan dalam sehari menjadi delapan (empat perubahan dalam enam jam pertama, dan satu perubahan untuk masing-masing jendela enam jam berikutnya dalam sehari).

Anda dapat mengganti kunci enkripsi untuk menggunakan Kunci milik AWS sesering yang diperlukan, bahkan jika kuota di atas telah habis.

Ini adalah kuota kecuali jika Anda meminta jumlah yang lebih tinggi. Untuk meminta peningkatan kuota layanan, lihat https://aws.amazon.com/support.

Referensi API DynamoDB

Referensi API Amazon DynamoDB berisi daftar lengkap operasi yang didukung oleh:

- DynamoDB
- DynamoDB Streams
- DynamoDB Accelerator (DAX)

Memecahkan masalah Amazon DynamoDB

Topik berikut memberikan saran pemecahan masalah untuk kesalahan dan masalah yang mungkin Anda temui saat menggunakan Amazon DynamoDB. Jika Anda menemukan masalah yang tidak tercantum di sini, Anda dapat menggunakan tombol umpan balik di halaman ini untuk melaporkannya.

Untuk saran pemecahan masalah selengkapnya dan jawaban atas pertanyaan dukungan umum, kunjungi Pusat Pengetahuan AWS.

Topik

- Memecahkan masalah kesalahan server internal di Amazon DynamoDB
- Memecahkan masalah latensi di Amazon DynamoDB
- Masalah pelambatan untuk DynamoDB

Memecahkan masalah kesalahan server internal di Amazon DynamoDB

Di DynamoDB, kesalahan server internal (500 kesalahan) menunjukkan bahwa layanan tidak dapat melayani permintaan. Kesalahan ini dapat terjadi karena berbagai alasan, seperti masalah jaringan sementara di armada, masalah infrastruktur, masalah terkait node penyimpanan, dan banyak lagi.

Anda mungkin mengalami beberapa kesalahan server internal selama siklus hidup tabel DynamoDB Anda. Hal ini diharapkan karena sifat layanan yang didistribusikan dan biasanya tidak perlu dikhawatirkan. DynamoDB secara otomatis memperbaiki dan menyembuhkan masalah sementara dengan layanan secara real time, tanpa memerlukan intervensi dari Anda. Namun, jika Anda mengamati jumlah kesalahan server internal yang tinggi secara konsisten pada permintaan ke tabel Anda (seperti yang terlihat dalam the section called “SystemErrors” metrik), Anda harus menyelidiki lebih lanjut.

Topik

- Menyelidiki kesalahan server internal
- Meminimalkan dampak dari kesalahan server internal
- Meningkatkan kesadaran operasional

Menyelidiki kesalahan server internal

Jika Anda mengalami kesalahan server internal di tabel DynamoDB Anda, pertimbangkan opsi ini:

1. Periksa Dasbor AWS Kesehatan.

   Untuk mengidentifikasi masalah, langkah pertama adalah memeriksa AWS Service Health Dashboard dan Dashboard Kesehatan AWS Akun Anda. Dasbor ini memberikan informasi berharga tentang masalah di seluruh layanan, tabel yang terkena dampak, masalah yang sedang berlangsung, dan akar penyebab setelah masalah diselesaikan.

   Meninjau detail di dasbor ini akan memberi Anda pemahaman yang lebih baik tentang status saat ini yang Layanan AWS Anda gunakan dan potensi masalah apa pun yang memengaruhi akun Anda. Informasi ini dapat membantu Anda menentukan langkah selanjutnya untuk mengatasi masalah dan meminimalkan gangguan pada operasi Anda.

2. Jangkau Dukungan.

   Jika Anda mengamati kesalahan yang berkepanjangan dan berkelanjutan dalam permintaan Anda, itu mungkin menunjukkan masalah dengan layanan. Sebagai aturan umum, jika Anda melihat tingkat kegagalan keseluruhan 1% atau lebih selama 15 menit terakhir, ini adalah waktu yang tepat untuk meningkatkan masalah ke tim AWS Support. Lihat, DynamoDB Service Level Agreement untuk mempelajari lebih lanjut.

   Saat membuka kasus dengan tim AWS Support, berikan detail berikut untuk membantu mempercepat proses pemecahan masalah:

   - DDB yang terkena dampak; tabel atau indeks sekunder
   - Jendela waktu ketika kesalahan diamati
   - IDsPermintaan DynamoDB, `4KBNVRGD25RG1KE09UT4V3FQDJVV4KQNSO5AEMVJF66Q9ASUAAJG` seperti, yang dapat Anda temukan di log aplikasi Anda.

   Menyertakan detail ini dalam kasus dukungan akan membantu AWS tim memahami masalah dan memberikan resolusi yang lebih cepat. Jika Anda tidak memiliki permintaan IDs, Anda masih harus mencatat kasus ini dengan detail lain yang tersedia.

Meminimalkan dampak dari kesalahan server internal

Jika kesalahan server internal terjadi saat menggunakan DynamoDB, meminimalkan dampaknya pada aplikasi Anda, pertimbangkan praktik terbaik berikut:

- Gunakan backoff dan percobaan ulang — Perilaku SDK default DynamoDB dirancang untuk menemukan keseimbangan yang tepat untuk sebagian besar aplikasi dalam hal strategi back-off dan coba lagi. Namun, Anda dapat menyesuaikan pengaturan ini berdasarkan toleransi aplikasi Anda terhadap waktu henti dan persyaratan kinerja. Pelajari lebih lanjut tentang back-off dan percobaan ulang untuk memahami bagaimana Anda dapat menyempurnakan pengaturan coba ulang ini.
- Gunakan pembacaan yang konsisten pada akhirnya — Jika aplikasi Anda tidak memerlukan pembacaan yang sangat konsisten, pertimbangkan untuk menggunakan pembacaan yang konsisten pada akhirnya. Pembacaan ini berbiaya lebih rendah dan kecil kemungkinannya untuk mengalami masalah sementara karena kesalahan server internal karena akan disajikan dari salah satu Node Penyimpanan yang tersedia. Untuk informasi selengkapnya, lihat DynamoDB membaca konsistensi.

Meningkatkan kesadaran operasional

Mempertahankan ketersediaan tinggi dan keandalan aplikasi Anda sangat penting dalam lanskap digital saat ini. Salah satu aspek kunci dari ini adalah secara proaktif memantau kesalahan server internal (ISEs) di tabel DynamoDB Anda dan indeks sekunder global (). GSIs Dengan membuat CloudWatch alarm untuk memantau kesalahan ini, Anda dapat memperoleh kesadaran operasional yang lebih baik dan diberi tahu tentang potensi masalah sebelum berdampak pada pengguna akhir Anda. Pendekatan ini sejalan dengan pilar Operational Excellence dari AWS Well-Architected Framework, memastikan beban kerja DynamoDB Anda dioptimalkan untuk kinerja, keamanan, dan keandalan.

Membuat CloudWatch alarm

Anda harus mengatur CloudWatch alarm pada tabel DynamoDB Anda untuk menerima pemberitahuan untuk jumlah kesalahan server internal yang tinggi secara konsisten alih-alih mengamati metrik secara manual. Ini terkait dengan pilar keunggulan operasional kerangka kerja Well-Architected untuk setiap beban kerja. AWS Lihat Menggunakan DynamoDB Well-Architected Lens untuk mengoptimalkan beban kerja DynamoDB Anda untuk mempelajari lebih lanjut tentang Merancang dengan Baik tabel DynamoDB Anda.

Alarm ini menggunakan matematika metrik khusus untuk menghitung persentase permintaan yang gagal untuk jendela 5 menit. Praktik terbaik yang disarankan adalah mengonfigurasi alarm untuk memasuki `ALARM` status ketika 3 titik data berturut-turut melanggar ambang batas 1%, yang berarti bahwa keseluruhan 1% permintaan gagal dalam periode 15 menit.

Contoh di bawah ini adalah AWS CloudFormation template yang dapat membantu Anda membuat CloudWatch alarm di meja Anda dan GSI di atas meja.

```
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'
```

Memecahkan masalah latensi di Amazon DynamoDB

Jika beban kerja Anda tampaknya mengalami latensi tinggi, Anda dapat menganalisis CloudWatch `SuccessfulRequestLatency` metrik, dan memeriksa latensi rata-rata dan latensi median melalui metrik persentil (p50) untuk melihat apakah itu terkait dengan DynamoDB. Beberapa variabilitas dalam laporan `SuccessfulRequestLatency` adalah normal, dan lonjakan sesekali (terutama dalam `Maximum` statistik dan persentil tinggi) tidak boleh menjadi perhatian. Namun, jika `Average` statistik atau p50 (median) menunjukkan peningkatan tajam dan berlanjut, Anda harus memeriksa AWS Service Health Dashboard dan Personal Health Dashboard Anda untuk informasi lebih lanjut. Beberapa kemungkinan penyebab termasuk ukuran item di tabel Anda (item 1 KB dan item 400 KB akan bervariasi dalam latensi) atau ukuran kueri (10 item versus 100 item).

Metrik persentil (p99, p90, dll.) Dapat membantu Anda lebih memahami distribusi latensi Anda. Misalnya:

- p50 (median) menunjukkan latensi tipikal untuk beban kerja Anda.
- p90 menunjukkan bahwa 90 persen permintaan lebih cepat dari nilai ini.
- p99 membantu mengidentifikasi latensi kasus terburuk yang memengaruhi 1 persen permintaan.

Nilai p99 tinggi dengan nilai p50 normal mungkin menunjukkan masalah sporadis yang memengaruhi sebagian kecil permintaan, sementara nilai p50 yang meningkat secara konsisten mungkin menunjukkan beberapa penurunan kinerja.

> **Note**
>
> Untuk menganalisis nilai persentil kustom (seperti p99.9), Anda dapat memasukkan persentil yang diinginkan secara manual (misalnya, p99.9) di bidang statistik metrik. CloudWatch Ini memungkinkan Anda untuk mengevaluasi distribusi latensi di luar persentil default yang tercantum dalam dropdown.

Beberapa variasi dalam metrik latensi, terutama dalam persentil yang lebih tinggi, diharapkan dan dapat dilihat sebagai hasil dari operasi latar belakang berbasis DynamoDB yang membantu menjaga ketersediaan dan daya tahan tinggi untuk data Anda yang disimpan dalam tabel DynamoDB atau masalah infrastruktur sementara.

Jika perlu, pertimbangkan untuk membuka kasus dukungan dengan AWS Dukungan, dan terus menilai opsi mundur yang tersedia untuk aplikasi Anda (seperti evakuasi Wilayah jika Anda memiliki arsitektur Multi-wilayah) sesuai dengan runbook Anda. Anda harus mencatat permintaan IDs untuk permintaan lambat untuk menyediakan ini IDs AWS Dukungan ketika Anda membuka kasus dukungan.

`SuccessfulRequestLatency`Metrik hanya mengukur latensi yang internal ke layanan DynamoDB — aktivitas sisi klien dan waktu perjalanan jaringan tidak disertakan. Untuk mempelajari lebih lanjut tentang latensi keseluruhan untuk panggilan dari klien ke layanan DynamoDB, Anda dapat mengaktifkan pencatatan metrik latensi di SDK. AWS

> **Note**
>
> Untuk sebagian besar operasi tunggal (operasi yang berlaku pada satu item dengan menentukan sepenuhnya nilai kunci primer), DynamoDB memberikan `Average SuccessfulRequestLatency` milidetik satu digit. Nilai ini tidak termasuk overhead transport untuk kode pemanggil yang mengakses titik akhir DynamoDB. Untuk operasi data multi-item, latensi akan bervariasi berdasarkan faktor-faktor seperti ukuran set hasil, kompleksitas struktur data yang dikembalikan, dan ekspresi kondisi dan ekspresi filter apa pun yang diterapkan. Untuk operasi multi-item berulang pada kumpulan data yang sama dengan parameter yang sama, DynamoDB akan memberikan `Average SuccessfulRequestLatency` yang sangat konsisten.

Pertimbangkan satu atau beberapa strategi berikut untuk mengurangi latensi:

- Gunakan kembali koneksi: Permintaan DynamoDB dibuat melalui sesi yang diautentikasi melalui HTTPS secara default. Memulai koneksi memerlukan beberapa perjalanan pulang-pergi dan membutuhkan waktu sehingga latensi permintaan pertama lebih tinggi daripada permintaan berikut yang menggunakan kembali koneksi. Permintaan melalui koneksi yang sudah diinisialisasi menghasilkan latensi rendah DynamoDB yang konsisten. Untuk menghindari overhead latensi dalam membuat koneksi baru, Anda mungkin ingin menerapkan mekanisme “keep-alive” dengan mengirimkan `GetItem` permintaan setiap 30 detik jika tidak ada permintaan lain yang dibuat.
- Gunakan pembacaan yang pada akhirnya konsisten: Jika aplikasi Anda tidak memerlukan bacaan sangat konsisten, pertimbangkan untuk menggunakan bacaan akhir konsisten secara default. Akhirnya pembacaan yang konsisten memiliki biaya lebih rendah dan dapat berasal dari beberapa zona ketersediaan, memungkinkan pemilihan zona ketersediaan yang ditempatkan bersama ke pemohon yang mengurangi latensi. Untuk informasi selengkapnya, lihat DynamoDB membaca konsistensi.
- Terapkan lindung nilai permintaan: Untuk persyaratan latensi p99 yang sangat rendah, pertimbangkan untuk menerapkan lindung nilai permintaan. Dengan hedging permintaan, jika permintaan awal tidak menerima respons dengan cukup cepat, kirim permintaan setara kedua dan biarkan mereka berlomba, respons pertama menang. Ini meningkatkan latensi ekor dengan mengorbankan beberapa permintaan tambahan. Anda dapat memutuskan berapa lama menunggu sebelum mengirim permintaan kedua. Hedging lebih mudah dibaca. Untuk penulisan, gunakan pemesanan berbasis stempel waktu untuk memastikan permintaan lindung nilai diperlakukan sebagai terjadi pada saat upaya pertama, mencegah pembaruan. out-of-order Pendekatan ini telah dibahas di Timestamp menulis untuk lindung nilai tulis di Amazon DynamoDB.
- Sesuaikan batas waktu permintaan dan perilaku coba lagi: Jalur dari klien Anda ke DynamoDB melintasi banyak komponen, masing-masing dirancang dengan mempertimbangkan redundansi. Pertimbangkan aspek-aspek berikut:
  - Ketahanan jaringan
  - Batas waktu paket TCP
  - Arsitektur terdistribusi DynamoDB

  Perilaku SDK default dioptimalkan untuk sebagian besar aplikasi. Namun, Anda dapat menerapkan strategi cepat gagal dan menyesuaikan pengaturan batas waktu. Permintaan yang memakan waktu jauh lebih lama dari biasanya cenderung tidak berhasil pada akhirnya. Dengan gagal cepat dan mencoba lagi, Anda dapat dengan cepat berhasil melalui jalan yang berbeda. Ini mirip dengan permintaan lindung nilai tetapi mengakhiri permintaan pertama alih-alih mengizinkannya untuk melanjutkan.

Hindari pengaturan nilai batas waktu terlalu rendah. Batas waktu yang terlalu rendah dapat menyebabkan masalah ketersediaan yang disebabkan oleh klien. Misalnya, batas waktu soket 50 milidetik dapat menyebabkan kesalahan koneksi selama lonjakan latensi jaringan, seperti saat mendekati batas bandwidth EC2 instans Amazon untuk lalu lintas aliran tunggal. Pertimbangkan dengan cermat manfaat batas waktu yang lebih rendah terhadap potensi risiko ketersediaan aplikasi, dan lebih memilih lindung nilai daripada batas waktu singkat.

Untuk diskusi bermanfaat tentang topik ini, lihat Menyetel setelan permintaan HTTP SDK AWS Java untuk aplikasi Amazon DynamoDB yang sadar latensi.

- Kurangi jarak antara klien dan titik akhir DynamoDB: Jika Anda memiliki pengguna yang tersebar secara global, pertimbangkan untuk menggunakan Tabel global - Replikasi multi-Wilayah untuk DynamoDB. Dengan tabel global, Anda dapat mereplikasi tabel Anda ke AWS Wilayah tertentu di mana Anda ingin tabel tersedia. Anda dapat menempatkan salinan data lebih dekat ke pengguna akhir untuk mengurangi latensi jaringan selama operasi baca dan tulis. Untuk informasi selengkapnya tentang penggunaan tabel global DynamoDB secara efektif, lihat Menggunakan tabel global Amazon DynamoDB di Panduan Preskriptif. AWS
- Gunakan caching: Jika lalu lintas Anda banyak dibaca, pertimbangkan untuk menggunakan salah satu layanan caching ini:
  - DynamoDB Accelerator (DAX): Cache dalam memori yang dikelola sepenuhnya, sangat tersedia, untuk DynamoDB yang memberikan peningkatan kinerja hingga 10x, dari milidetik hingga mikrodetik, bahkan pada jutaan permintaan per detik. Untuk informasi selengkapnya tentang DAX, lihatAkselerasi dalam memori dengan DynamoDB Accelerator (DAX):
  - Amazon ElastiCache: Layanan caching dalam memori yang dikelola sepenuhnya yang dapat diintegrasikan dengan DynamoDB untuk meningkatkan kinerja baca menggunakan pola selain cache. Untuk informasi selengkapnya, lihat Mengintegrasikan Amazon DynamoDB ElastiCache dan Amazon menggunakan caching AWS baca di Panduan Preskriptif.

Masalah pelambatan untuk DynamoDB

Pembatasan mencegah aplikasi Anda menggunakan terlalu banyak unit kapasitas. Topik ini membahas cara memecahkan masalah pelambatan umum untuk mode kapasitas yang disediakan dan sesuai permintaan. Topik ini juga menjelaskan cara menggunakan CloudWatch untuk menyelidiki dari mana masalah mungkin berasal.

Topik

- Memecahkan masalah pembatasan untuk mode yang disediakan
- Memecahkan masalah pembatasan untuk mode on-demand
- Menggunakan CloudWatch metrik untuk menyelidiki masalah pelambatan

Memecahkan masalah pembatasan untuk mode yang disediakan

Jika aplikasi Anda melebihi kapasitas throughput yang disediakan pada tabel atau indeks, aplikasi tersebut akan dikenakan throttling permintaan. Pembatasan mencegah aplikasi Anda menggunakan terlalu banyak unit kapasitas. Ketika DynamoDB membatasi operasi baca atau tulis, ia mengembalikan a ke pemanggil. `ProvisionedThroughputExceededException` Aplikasi kemudian dapat mengambil tindakan yang sesuai, seperti menunggu beberapa saat sebelum mencoba kembali permintaan tersebut.

Untuk mengatasi masalah yang tampaknya terkait dengan pelambatan, langkah pertama yang penting adalah mengonfirmasi apakah pelambatan berasal dari DynamoDB atau dari aplikasi.

Topik ini membahas cara memecahkan masalah pelambatan umum untuk mode kapasitas yang disediakan. Berikut ini adalah beberapa skenario umum, dan langkah-langkah yang mungkin untuk membantu menyelesaikannya.

Tabel DynamoDB tampaknya memiliki kapasitas yang disediakan yang cukup, tetapi permintaan sedang dibatasi

Ini dapat terjadi ketika throughput di bawah rata-rata per menit, tetapi melebihi jumlah yang tersedia per detik. DynamoDB hanya melaporkan metrik tingkat menit CloudWatch ke, yang dihitung sebagai jumlah selama satu menit dan rata-rata. Namun DynamoDB sendiri menerapkan batasan kecepatan per detik. Jadi, jika terlalu banyak throughput yang terjadi dalam sebagian kecil menit tersebut, misalnya beberapa detik atau kurang, maka permintaan untuk sisa menit tersebut dapat dibatasi.

Misalnya, jika kita telah menyediakan 60 WCU di atas meja, maka dapat melakukan 3600 operasi tulis dalam satu menit. Namun jika seluruh 3600 permintaan WCU tercapai dalam detik yang sama, maka sisa menit tersebut akan di-throttle.

Salah satu cara untuk mengatasi skenario ini adalah dengan menambahkan beberapa kegugupan dan kemunduran eksponensial ke panggilan API. Untuk informasi lebih lanjut lihat posting ini tentang backoff dan jitter.

Penskalaan otomatis diaktifkan, tetapi tabel masih dibatasi

Hal ini dapat terjadi ketika terjadi lonjakan lalu lintas secara tiba-tiba. Penskalaan otomatis dapat dipicu ketika 2 titik data melanggar nilai pemanfaatan target yang dikonfigurasi dalam rentang satu menit. Oleh karena itu, penskalaan otomatis dapat terjadi karena kapasitas yang dikonsumsi berada di atas target pemanfaatan selama dua menit yang konsisten. Tetapi jika paku terpisah lebih dari satu menit, penskalaan otomatis mungkin tidak dipicu.

Demikian pula, peristiwa penurunan skala dapat dipicu ketika 15 titik data berturut-turut lebih rendah dari pemanfaatan target. Dalam kedua kasus tersebut, setelah penskalaan otomatis dipicu, operasi `UpdateTable` API dipanggil. Kemudian dapat memakan waktu beberapa menit untuk memperbarui kapasitas yang disediakan untuk tabel atau indeks. Selama periode ini, setiap permintaan yang melebihi kapasitas tabel yang disediakan sebelumnya akan dibatasi.

Singkatnya, penskalaan otomatis memerlukan titik data berturut-turut di mana nilai pemanfaatan target dilanggar untuk meningkatkan tabel DynamoDB. Untuk alasan ini, penskalaan otomatis tidak disarankan sebagai solusi untuk menangani beban kerja spikey. Silakan lihat dokumentasi pengoptimalan biaya penskalaan otomatis untuk informasi lebih lanjut.

Kunci pintas mungkin menyebabkan masalah pelambatan

Di DynamoDB, kunci partisi yang tidak memiliki kardinalitas tinggi dapat mengakibatkan banyak permintaan yang menargetkan beberapa partisi saja. Jika partisi panas yang dihasilkan melewati batas partisi 3000 RCU atau 1000 WCU per detik, ini dapat mengakibatkan pelambatan. Alat diagnostik CloudWatch Contributor Insights (CCI) dapat membantu men-debug ini dengan menyediakan grafik CCI untuk setiap pola akses item tabel. Anda dapat terus memantau kunci tabel DynamoDB yang paling sering diakses dan tren lalu lintas lainnya. Untuk informasi selengkapnya tentang CloudWatch Contributor Insights lihat Contributor Insights for CloudWatch DynamoDB. Untuk informasi selengkapnya, lihat Merancang kunci partisi untuk mendistribusikan beban kerja Anda di DynamoDB dan Memilih Kunci Partisi DynamoDB yang Tepat.

Lalu lintas Anda ke tabel melebihi kuota throughput tingkat tabel.

Kuota throughput baca tingkat tabel dan throughput tulis tingkat tabel berlaku di tingkat akun di Wilayah mana pun. Kuota ini berlaku untuk tabel dengan mode kapasitas yang disediakan dan mode kapasitas sesuai permintaan. Secara default, kuota throughput yang ditempatkan pada tabel Anda adalah 40.000 unit permintaan baca dan 40.000 unit permintaan tulis. Jika lalu lintas ke tabel Anda melebihi kuota ini, tabel tersebut mungkin dibatasi. Untuk informasi lebih lanjut tentang cara mencegah hal ini terjadi, lihat Memantau DynamoDB untuk kesadaran operasional.

Untuk mengatasi masalah ini, gunakan konsol Kuota Layanan untuk meningkatkan kuota throughput baca atau tulis tingkat tabel untuk akun Anda.

Memecahkan masalah pembatasan untuk mode on-demand

Tabel DynamoDB yang menggunakan mode kapasitas sesuai permintaan secara otomatis beradaptasi dengan volume lalu lintas aplikasi Anda. Namun, tabel yang menggunakan mode on-demand mungkin masih melambat. Topik ini membahas cara memecahkan masalah pembatasan umum untuk tabel sesuai permintaan.

Lalu lintas lebih dari dua kali lipat dari puncak sebelumnya

Jika Anda melebihi dua kali lipat puncak lalu lintas sebelumnya dalam waktu 30 menit, maka Anda mungkin mengalami pelambatan. Sebelum Anda melampaui puncak lalu lintas sebelumnya, kami sarankan Anda menyebarkan pertumbuhan lalu lintas Anda setidaknya selama 30 menit. Untuk memantau lalu lintas ke tabel, gunakan `ConsumedReadCapacityUnits` metrik di Amazon CloudWatch. Untuk informasi selengkapnya, lihat Dimensi dan Metrik DynamoDB.

Untuk tabel on-demand baru, Anda dapat langsung mengendarai hingga 4.000 unit permintaan tulis atau 12.000 unit permintaan baca per detik.

Untuk tabel yang sudah ada yang Anda alihkan ke mode kapasitas sesuai permintaan, puncak sebelumnya adalah salah satu nilai berikut:

- Setengah dari throughput yang disediakan sebelumnya untuk tabel
- Pengaturan untuk tabel yang baru dibuat dengan mode kapasitas sesuai permintaan

Untuk informasi selengkapnya, lihat Throughput awal untuk mode kapasitas sesuai permintaan.

Lalu lintas melebihi maksimum per partisi

Setiap partisi tabel atau GSI dapat melayani hingga 3.000 unit permintaan baca atau 1.000 unit permintaan tulis per detik. Jika lalu lintas ke partisi melebihi batas ini, partisi mungkin dibatasi. Untuk mengatasi masalah ini, lakukan tindakan berikut:

1. Gunakan CloudWatch Contributor Insights for DynamoDB untuk mengidentifikasi kunci yang paling sering diakses dan dibatasi dalam tabel Anda.
2. Acak permintaan ke tabel sehingga permintaan ke tombol partisi panas didistribusikan dari waktu ke waktu. Untuk informasi selengkapnya, lihat Menggunakan sharding tulis untuk mendistribusikan beban kerja secara merata di tabel DynamoDB Anda.

Kunci pintas mungkin menyebabkan masalah pelambatan

Di DynamoDB, kunci partisi yang tidak memiliki kardinalitas tinggi dapat menghasilkan banyak permintaan yang menargetkan hanya beberapa partisi. Jika partisi panas yang dihasilkan melewati batas partisi 3.000 RCU atau 1.000 WCU per detik, itu dapat mengakibatkan pelambatan.

Alat diagnostik CloudWatch Contributor Insights (CCI) dapat membantu Anda men-debug ini dengan menyediakan grafik CCI untuk setiap pola akses item tabel. Anda dapat terus memantau kunci tabel DynamoDB yang paling sering diakses dan tren lalu lintas lainnya. Untuk informasi selengkapnya tentang CloudWatch Contributor Insights lihat Contributor Insights for CloudWatchDynamoDB. Untuk informasi selengkapnya, lihat Merancang kunci partisi untuk mendistribusikan beban kerja Anda di DynamoDB dan Memilih Kunci Partisi DynamoDB yang Tepat.

Lalu lintas melebihi kuota akun per tabel

Untuk tabel sesuai permintaan, throughput baca tingkat tabel dan kuota throughput tulis tingkat tabel berlaku di tingkat akun. Secara default, throughput tabel memiliki maksimum 40.000 unit permintaan baca dan maksimum 40.000 unit permintaan tulis. Jika lalu lintas ke tabel melebihi kuota akun per tabel untuk throughput, maka tabel mungkin mengalami pelambatan. Untuk mengatasi masalah ini, gunakan konsol Service Quotas untuk meningkatkan throughput baca tingkat tabel dan kuota throughput tulis untuk akun Anda.

Indeks sekunder global tabel Anda dibatasi

Jika tabel DynamoDB Anda memiliki indeks global sekunder yang sedang dibatasi, maka throttling mungkin membuat throttle tekanan balik pada tabel dasar. Untuk informasi selengkapnya, lihat Bagaimana pembatasan pada indeks sekunder global memengaruhi tabel Amazon DynamoDB saya dan. Menggunakan Indeks Sekunder Global di DynamoDB

Lalu lintas melebihi throughput maksimum yang dikonfigurasi

Jika operasi baca atau tulis tabel sesuai permintaan melebihi batas throughput yang telah ditentukan, tabel akan dibatasi sementara, dan Anda akan menerima pesan ThrottlingExceptionkesalahan.

Selesaikan tindakan berikut berdasarkan kasus penggunaan Anda:

- Untuk menambah atau mematikan setelan throughput tabel maksimum, gunakan UpdateTableAPI.
- Tunggu dan coba lagi permintaannya. Lihat, the section called “Percobaan ulang kesalahan dan penundaan eksponensial”.
- Untuk memantau throughput maksimum yang dikonfigurasi untuk tabel atau indeks sekunder global, gunakan the section called “OnDemandMaxWriteRequestUnits” metrik the section called “OnDemandMaxReadRequestUnits” dan di CloudWatch konsol.

Menggunakan CloudWatch metrik untuk menyelidiki masalah pelambatan

Di bawah ini adalah beberapa metrik DynamoDB untuk dipantau selama peristiwa pelambatan. Gunakan ini untuk membantu menemukan operasi mana yang membuat permintaan terbatas dan mengidentifikasi masalah akar.

- `ThrottledRequests`
  - Satu permintaan yang dibatasi dapat berisi beberapa peristiwa yang dibatasi, sehingga peristiwa dapat lebih relevan dengan contoh dibandingkan dengan permintaan. Misalnya, saat Anda memperbarui item dalam tabel dengan GSIs, ada beberapa peristiwa: operasi tulis ke tabel dan operasi tulis untuk setiap indeks. Bahkan jika satu atau lebih dari peristiwa ini dibatasi, hanya akan ada satu. `ThrottledRequest`
- `ReadThrottleEvents`
  - Perhatikan permintaan yang melebihi RCU yang disediakan untuk tabel atau GSI.
- `WriteThrottleEvents`
  - Perhatikan permintaan yang melebihi WCU yang disediakan untuk sebuah tabel atau GSI.
- `OnlineIndexConsumedWriteCapacity`
  - Perhatikan jumlah WCU yang dikonsumsi saat menambahkan GSI baru ke tabel. Perhatikan bahwa `ConsumedWriteCapacityUnits` untuk GSI tidak menyertakan WCU yang digunakan selama pembuatan indeks.
  - Jika Anda telah menyetel WCU untuk GSI terlalu rendah, maka aktivitas penulisan yang masuk selama fase pengisian ulang mungkin dibatasi.
- `Provisioned Read/Write`
  - Lihat berapa banyak unit kapasitas baca atau tulis yang disediakan yang digunakan selama jangka waktu tertentu, untuk tabel atau indeks sekunder global tertentu.
  - Perhatikan bahwa dimensi `TableName` mengembalikan `ProvisionedReadCapacityUnits` untuk tabel hanya secara default. Untuk melihat jumlah unit kapasitas baca atau tulis yang disediakan untuk indeks sekunder global, Anda harus menentukan keduanya dan`TableName`. `GlobalSecondaryIndexName`
- `Consumed Read/Write`
  - Lihat berapa banyak unit kapasitas baca atau tulis yang dikonsumsi selama jangka waktu tertentu.

Untuk informasi selengkapnya tentang metrik CloudWatch DynamoDB, lihat. Dimensi dan Metrik DynamoDB

Lampiran DynamoDB

Topik

- Memecahkan masalah pembuatan koneksi SSL/TLS dengan DynamoDB
- Contoh tabel dan data untuk digunakan di DynamoDB
- Membuat tabel contoh dan mengunggah data di DynamoDB
- DynamoDB contoh aplikasi menggunakan: AWS SDK for Python (Boto) Tic-tac-toe
- Kata-kata penggunaan khusus di DynamoDB
- AWS Contoh SDK for Java 1.x
- AWS Contoh SDK for Go 1.x
- AWS Contoh SDK untuk Node.js 2.x

Memecahkan masalah pembuatan koneksi SSL/TLS dengan DynamoDB

Amazon DynamoDB sedang dalam proses memindahkan titik akhir ke sertifikat aman yang ditandatangani oleh Otoritas Sertifikat Amazon Trust Services (ATS), bukan Otoritas Sertifikat pihak ketiga. Pada bulan Desember 2017, kami meluncurkan Wilayah EU-WEST-3 (Paris) dengan sertifikat aman yang dikeluarkan oleh Amazon Trust Services. Semua wilayah baru yang diluncurkan setelah Desember 2017 memiliki titik akhir dengan sertifikat yang dikeluarkan oleh Amazon Trust Services. Panduan ini menunjukkan cara untuk memvalidasi dan memecahkan masalah koneksi SSL/TLS.

Menguji aplikasi atau layanan Anda

Sebagian besar AWS SDKs dan Antarmuka Baris Perintah (CLIs) mendukung Amazon Trust Services Certificate Authority. Jika Anda menggunakan versi AWS SDK untuk Python atau CLI yang dirilis sebelum 29 Oktober 2013, Anda harus meng-upgrade. .NET, Java, PHP, Go JavaScript, dan C ++ SDKs dan CLIs tidak menggabungkan sertifikat apa pun, sertifikat mereka berasal dari sistem operasi yang mendasarinya. Ruby SDK telah memasukkan setidaknya satu dari yang diperlukan CAs sejak 10 Juni 2015. Sebelum tanggal tersebut, Ruby V2 SDK tidak menyertakan sertifikat. Jika Anda menggunakan versi AWS SDK yang tidak didukung, kustom, atau modifikasi, atau jika Anda menggunakan toko kepercayaan khusus, Anda mungkin tidak memiliki dukungan yang diperlukan untuk Amazon Trust Services Certificate Authority.

Untuk memvalidasi akses ke titik akhir DynamoDB, Anda perlu mengembangkan pengujian yang mengakses API DynamoDB atau API DynamoDB Streams di wilayah EU-WEST-3 dan memvalidasi bahwa handshake TLS berhasil. Titik akhir tertentu yang akan Anda perlukan untuk mengakses pengujian tersebut adalah:

- DynamoDB: https://dynamodb.eu-west-3.amazonaws.com
- DynamoDB Streams: https://streams.dynamodb.eu-west-3.amazonaws.com

Jika aplikasi Anda tidak mendukung Otoritas Sertifikat Amazon Trust Services, Anda akan melihat salah satu kegagalan berikut:

- Kesalahan Negosiasi SSL/TLS
- Penundaan lama sebelum perangkat lunak Anda menerima kesalahan yang menunjukkan kegagalan negosiasi SSL/TLS. Waktu tunda bergantung pada strategi coba lagi dan konfigurasi waktu habis klien Anda.

Menguji browser klien Anda

Untuk memverifikasi bahwa browser Anda dapat terhubung ke Amazon DynamoDB, buka URL berikut:. https://dynamodb.eu-west-3.amazonaws.com Jika pengujian berhasil, Anda akan melihat pesan seperti ini:

```
healthy: dynamodb.eu-west-3.amazonaws.com
```

Jika tes tidak berhasil, itu akan menampilkan kesalahan yang mirip dengan ini: https://untrusted-root.badssl.com/.

Memperbarui klien aplikasi perangkat lunak Anda

Aplikasi yang mengakses titik akhir API DynamoDB atau DynamoDB Streams (baik melalui browser atau secara terprogram) perlu memperbarui daftar CA tepercaya pada mesin klien jika mereka belum mendukung salah satu dari berikut ini: CAs

- Amazon Root CA 1
- Starfield Services Root Certificate Authority - G2
- Starfield Class 2 Certification Authority

Jika klien sudah mempercayai APAPUN dari tiga di atas CAs maka ini akan mempercayai sertifikat yang digunakan oleh DynamoDB dan tidak ada tindakan yang diperlukan. Namun, jika klien Anda belum mempercayai salah satu hal di atas CAs, koneksi HTTPS ke DynamoDB atau DynamoDB Streams akan gagal. APIs Untuk informasi lebih lanjut, silakan kunjungi posting blog ini: https:// aws.amazon.com/blogs/keamanan/ how-to-prepare-for - aws-move-to-its -own-certificate-authority/.

Memperbarui browser klien Anda

Anda dapat memperbarui paketan sertifikat di browser cukup dengan memperbarui browser Anda. Petunjuk tentang browser yang paling umum dapat ditemukan di situs web browser tersebut:

- Chrome: https://support.google.com/chrome/jawaban/95414? hl=id
- 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/bantuan/17295/ windows-internet-explorer-which -versi #ie =lainnya

Memperbarui paketan sertifikat Anda secara manual

Jika Anda tidak dapat mengakses DynamoDB API atau DynamoDB Streams API maka Anda perlu memperbarui paket sertifikat Anda. Untuk melakukan ini, Anda perlu mengimpor setidaknya satu dari yang diperlukan CAs. Anda dapat menemukan ini di https://www.amazontrust.com/repository/.

Sistem operasi dan bahasa pemrograman berikut mendukung sertifikat Amazon Trust Services:

- Versi Microsoft Windows yang sudah menginstal pembaruan Januari 2005 atau yang lebih baru, Windows Vista, Windows 7, Windows Server 2008, dan versi yang lebih baru.
- MacOS X 10.4 dengan Java untuk MacOS X 10.4 Release 5, MacOS X 10.5 dan versi yang lebih baru.
- Red Hat Enterprise Linux 5 (Maret 2007), Linux 6, dan Linux 7 serta CentOS 5, CentOS 6, dan CentOS 7
- Ubuntu 8.10
- Debian 5.0
- Amazon Linux (semua versi)
- Java 1.4.2_12, Java 5 update 2, dan semua versi yang lebih baru, termasuk Java 6, Java 7, dan Java 8

Jika Anda masih tidak dapat terhubung, lihat dokumentasi perangkat lunak Anda, Vendor OS, atau hubungi AWS Support https://aws.amazon.com/support untuk bantuan lebih lanjut.

Contoh tabel dan data untuk digunakan di DynamoDB

Panduan Developer Amazon DynamoDB menggunakan tabel sampel untuk menggambarkan berbagai aspek DynamoDB.

Nama tabel	Kunci primer
ProductCatalog	Kunci primer sederhana: • `Id` (Angka)
Forum	Kunci primer sederhana: • `Name` (String)
Thread	Kunci primer komposit: • `ForumName` (String) • `Subject` (String)
Reply	Kunci primer komposit: • `Id` (String) • `ReplyDateTime` (String)

Tabel Balas memiliki indeks sekunder global bernama PostedBy-Message-Index. Indeks ini akan memfasilitasi kueri pada dua atribut non-kunci dari tabel Reply.

Nama indeks	Kunci primer
PostedBy-Indeks Pesan-	Kunci primer komposit: • `PostedBy` (String) • `Message` (String)

Untuk informasi selengkapnya tentang tabel ini, lihat Langkah 1: Buat tabel di DynamoDB dan Langkah 2: Menulis data ke tabel DynamoDB.

File data sampel

Topik

- ProductCatalogdata sampel
- Data sampel Forum
- Data sampel Thread
- Data sampel Reply

Bagian berikut menunjukkan contoh file data yang digunakan untuk memuat tabel ProductCatalog, Forum, Thread dan Reply.

Setiap file data berisi beberapa elemen `PutRequest`, masing-masing berisi satu item. `PutRequest`Elemen-elemen ini digunakan sebagai masukan untuk `BatchWriteItem` operasi, menggunakan AWS Command Line Interface (AWS CLI).

ProductCatalogdata sampel

```
{
    "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"
                    }
                }
            }
        }
    ]
}
```

Data sampel Forum

```
{
    "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"}
```

```
                }
            }
        }
    ]
}
```

Data sampel Thread

```
{
    "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"
                        }
```

```
                        ]
                    }
                }
            }
        }
    ]
}
```

Data sampel Reply

```
{
    "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"
                    }
                }
            }
        }
    ]
}
```

Membuat tabel contoh dan mengunggah data di DynamoDB

Lampiran ini menyediakan kode untuk membuat tabel dan menambahkan data secara terprogram.

Topik

- Membuat tabel contoh dan mengunggah data menggunakan AWS SDK untuk Java
- Membuat tabel contoh dan mengunggah data menggunakan AWS SDK untuk .NET

Membuat tabel contoh dan mengunggah data menggunakan AWS SDK untuk Java

Contoh kode Java berikut ini membuat tabel dan mengunggah data ke tabel. Untuk step-by-step instruksi untuk menjalankan kode ini menggunakan Eclipse, lihat. Contoh kode Java

```
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());

        }
    }

}
```

Membuat tabel contoh dan mengunggah data menggunakan AWS SDK untuk .NET

Contoh kode C# berikut ini membuat tabel dan mengunggah data ke tabel. Untuk step-by-step petunjuk untuk menjalankan kode ini di Visual Studio, lihatContoh kode .NET.

```
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 contoh aplikasi menggunakan: AWS SDK for Python (Boto) Tic-tac-toe

Tic-Tac-ToeGim ini adalah contoh aplikasi web yang dibangun di Amazon DynamoDB. Aplikasi ini menggunakan AWS SDK for Python (Boto) untuk membuat panggilan DynamoDB yang diperlukan untuk menyimpan data game dalam tabel DynamoDB, dan kerangka web Python Flask untuk end-to-end menggambarkan pengembangan aplikasi di DynamoDB, termasuk cara memodelkan data. Ini juga menunjukkan praktik terbaik dalam hal pemodelan data di DynamoDB, termasuk tabel yang Anda buat untuk aplikasi game, kunci primer yang Anda tentukan, indeks tambahan yang Anda perlukan berdasarkan persyaratan kueri Anda, dan penggunaan atribut nilai bersambung.

Anda memainkan Tic-Tac-Toe aplikasi di web sebagai berikut:

1. Masuk ke halaman beranda aplikasi.
2. Kemudian, undang pengguna lain untuk memainkan game sebagai lawan Anda.

Jika pengguna lain belum menerima undangan Anda, status game tetap `PENDING`. Setelah lawan menerima undangan, status game berubah menjadi `IN_PROGRESS`.

3. Game dimulai setelah lawan masuk dan menerima undangan.
4. Aplikasi menyimpan semua progres game dan informasi status dalam tabel DynamoDB.
5. Game berakhir dengan kemenangan atau seri, yang menetapkan status game menjadi `FINISHED`.

Latihan membangun end-to-end aplikasi dijelaskan dalam langkah-langkah:

- Langkah 1: Deploy dan uji secara lokal – Pada bagian ini, Anda mengunduh, men-deploy, dan menguji aplikasi pada komputer lokal. Anda akan membuat tabel yang diperlukan dalam versi DynamoDB yang dapat diunduh.
- Langkah 2: Periksa model data dan detail implementasi – Bagian ini pertama-tama menjelaskan model data secara mendetail, termasuk indeks dan penggunaan atribut nilai bersambung. Kemudian, bagian ini menjelaskan cara kerja aplikasi.
- Langkah 3: Deploy dalam produksi menggunakan layanan DynamoDB – Bagian ini fokus pada pertimbangan deployment dalam produksi. Pada langkah ini, Anda akan membuat tabel menggunakan layanan Amazon DynamoDB dan men-deploy aplikasi menggunakan AWS Elastic Beanstalk. Jika Anda memiliki aplikasi ini dalam produksi, Anda juga akan memberikan izin yang sesuai sehingga aplikasi dapat mengakses tabel DynamoDB. Instruksi di bagian ini memandu Anda melalui penyebaran end-to-end produksi.
- Langkah 4: Bersihkan Sumber Daya – Bagian ini menyoroti area yang tidak tercakup dalam contoh ini. Bagian ini juga menyediakan langkah-langkah bagi Anda untuk menghapus AWS sumber daya yang Anda buat pada langkah-langkah sebelumnya sehingga Anda menghindari biaya apa pun.

Langkah 1: Deploy dan uji secara lokal

Topik

- 1.1: Unduh dan instal paket yang diperlukan
- 1.2: Uji aplikasi game

Pada langkah ini Anda mengunduh, menyebarkan, dan menguji aplikasi Tic-Tac-Toe game di komputer lokal Anda. Alih-alih menggunakan layanan web Amazon DynamoDB, Anda akan mengunduh DynamoDB ke komputer Anda, dan membuat tabel yang dibutuhkan di sana.

1.1: Unduh dan instal paket yang diperlukan

Anda akan memerlukan hal-hal berikut untuk menguji aplikasi ini secara lokal:

- Python
- Flask (microframework untuk Python)
- AWS SDK for Python (Boto)
- DynamoDB yang berjalan di komputer Anda
- Git

Untuk mendapatkan alat-alat ini, lakukan tindakan berikut:

1. Instal Python. Untuk step-by-step petunjuk, lihat Mengunduh Python.

   Tic-Tac-ToeAplikasi ini telah diuji menggunakan Python versi 2.7.
2. Instal Flask dan AWS SDK for Python (Boto) gunakan Python Package Installer (PIP):
   - Instal PIP.

     Untuk petunjuk selengkapnya, lihat Menginstal PIP. Di halaman penginstalan, pilih tautan get-pip.py, lalu simpan file. Kemudian, buka terminal perintah sebagai administrator, dan masukkan berikut ini pada prompt perintah.

     ```
     python.exe get-pip.py
     ```

     Di Linux, Anda tidak menentukan ekstensi `.exe`. Anda hanya menentukan `python get-pip.py`.
   - Menggunakan PIP, instal paket Flask dan Boto menggunakan kode berikut.

     ```
     pip install Flask
     pip install boto
     pip install configparser
     ```
3. Unduh DynamoDB ke komputer Anda. Untuk petunjuk cara menjalankannya, lihat Menyiapkan DynamoDB lokal (versi yang dapat diunduh) .
4. Unduh Tic-Tac-Toe aplikasinya:
   a. Instal Git. Untuk petunjuknya, lihat Unduhan git.
   b. Jalankan kode berikut untuk mengunduh aplikasi.

```
git clone https://github.com/awslabs/dynamodb-tictactoe-example-app.git
```

1.2: Uji aplikasi game

Untuk menguji Tic-Tac-Toe aplikasi, Anda perlu menjalankan DynamoDB secara lokal di komputer Anda.

Untuk menjalankan tic-tac-toe aplikasi

1. Mulai DynamoDB.
2. Mulai server web untuk Tic-Tac-Toe aplikasi.

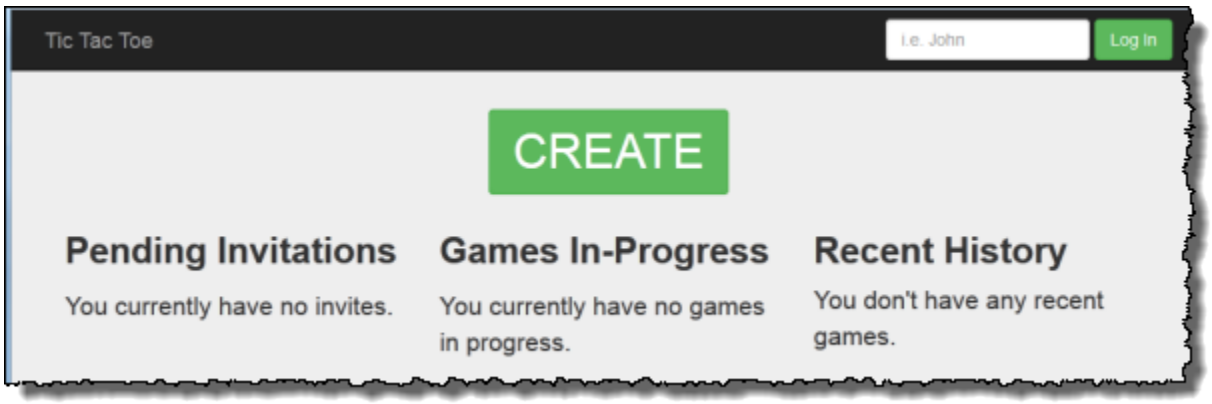
   Untuk melakukannya, buka terminal perintah, arahkan ke folder tempat Anda mengunduh Tic-Tac-Toe aplikasi, dan jalankan aplikasi secara lokal menggunakan kode berikut.

   ```
   python.exe application.py --mode local --serverPort 5000 --port 8000
   ```

   Di Linux, Anda tidak menentukan ekstensi `.exe`.
3. Buka browser web Anda, dan masukkan perintah berikut ini.

   ```
   http://localhost:5000/
   ```

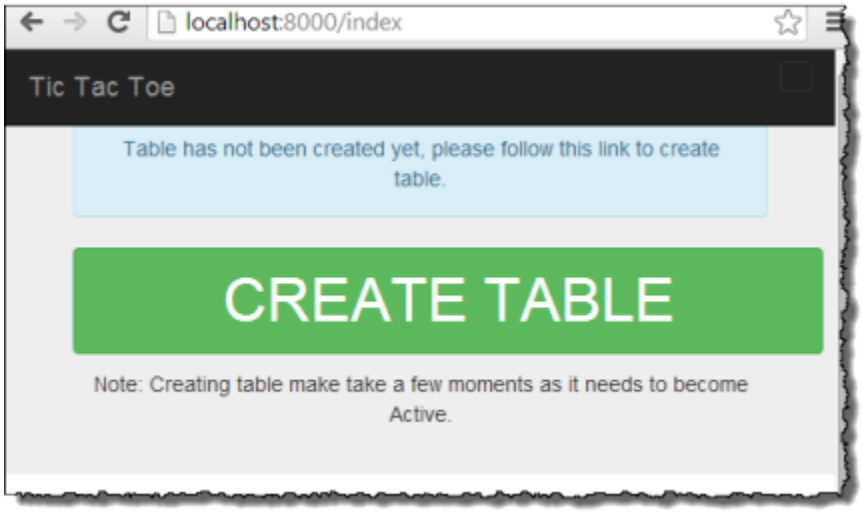
   Browser menampilkan halaman beranda.



4. Masukkan **`user1`** di kotak Masuk untuk masuk sebagai user1.

> **Note**
>
> Aplikasi contoh ini tidak melakukan autentikasi pengguna. ID pengguna hanya digunakan untuk mengidentifikasi pemain. Jika dua pemain masuk dengan alias yang sama, aplikasi berfungsi seolah-olah Anda bermain di dua browser yang berbeda.

5. Jika ini adalah pertama kalinya Anda memainkan game ini, sebuah halaman akan ditampilkan yang meminta Anda untuk membuat tabel yang diperlukan (`Games`) di DynamoDB. Pilih BUAT TABEL.



6. Pilih CREATE untuk membuat tic-tac-toe game pertama.
7. Masukkan **`user2`** di kotak Pilih Lawan, dan pilih Buat Game!

   Langkah ini akan membuat game dengan cara menambahkan item dalam tabel `Games`. Ini akan menetapkan status game menjadi `PENDING`.

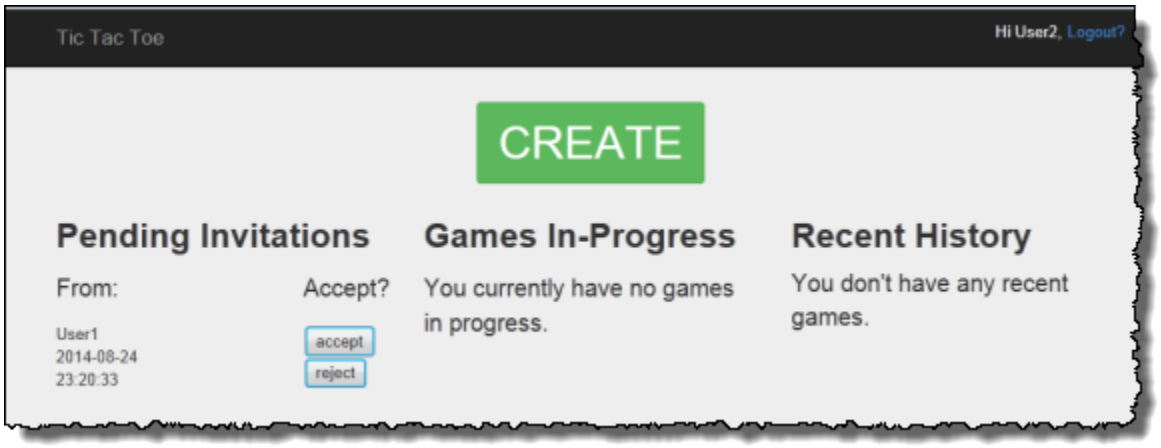
8. Buka jendela browser lain, dan masukkan berikut ini.

```
http://localhost:5000/
```

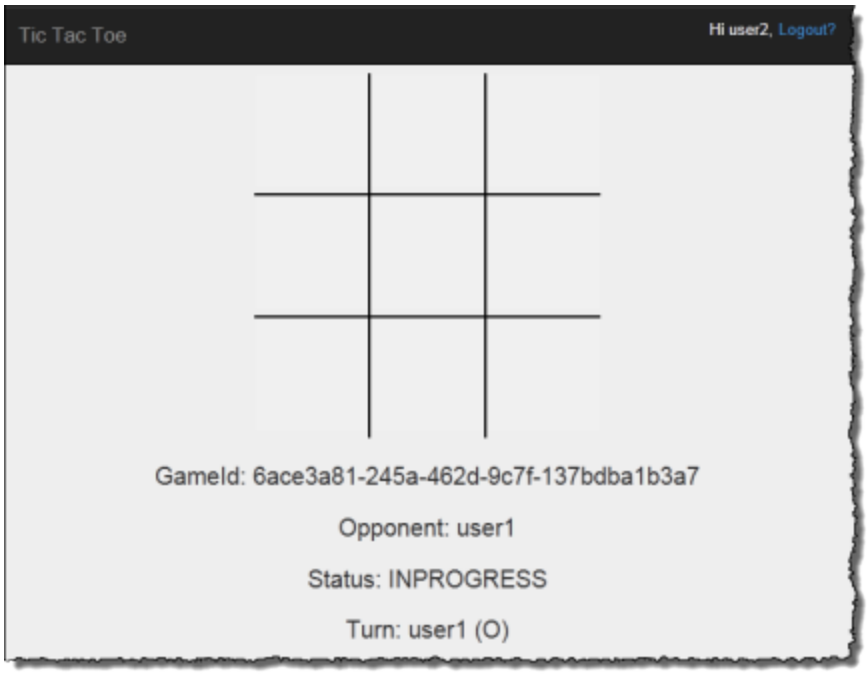
Browser meneruskan informasi melalui cookie, jadi Anda harus menggunakan mode penyamaran atau penjelajahan privat agar cookie Anda tidak terbawa.

9. Masuk sebagai user2.

   Sebuah halaman yang menampilkan undangan tertunda dari user1.



10. Pilih terima untuk menerima undangan.



   Halaman permainan muncul dengan tic-tac-toe kotak kosong. Halaman ini juga menunjukkan informasi game yang relevan seperti ID game, giliran pemain selanjutnya, dan status game.

11. Mainkan game.

Untuk setiap langkah yang dilakukan pengguna, layanan web mengirimkan permintaan ke DynamoDB untuk memperbarui item game secara kondisional di tabel `Games`. Sebagai contoh, kondisi memastikan bahwa langkah tersebut valid, kotak yang dipilih pengguna tersedia, dan itu adalah giliran pengguna yang melakukan langkah tersebut. Untuk langkah yang valid, operasi pembaruan menambahkan atribut baru yang sesuai dengan pilihan di papan. Operasi pembaruan juga menetapkan nilai atribut yang sudah ada untuk pengguna yang dapat melangkah berikutnya.

Pada halaman game, aplikasi membuat JavaScript panggilan asinkron setiap detik, hingga 5 menit, untuk memeriksa apakah status game di DynamoDB telah berubah. Jika berubah, aplikasi memperbarui halaman dengan informasi baru. Setelah 5 menit, aplikasi berhenti membuat permintaan, dan Anda perlu menyegarkan halaman untuk mendapatkan informasi terbaru.

Langkah 2: Periksa model data dan detail implementasi

Topik

- 2.1: Model data dasar
- 2.2: Aplikasi dalam tindakan (kode panduan)

2.1: Model data dasar

Aplikasi contoh ini menyoroti konsep model data DynamoDB berikut:

- Tabel – Di DynamoDB, tabel adalah kumpulan item (yaitu, catatan), dan setiap item adalah kumpulan pasangan nama-nilai yang disebut atribut.

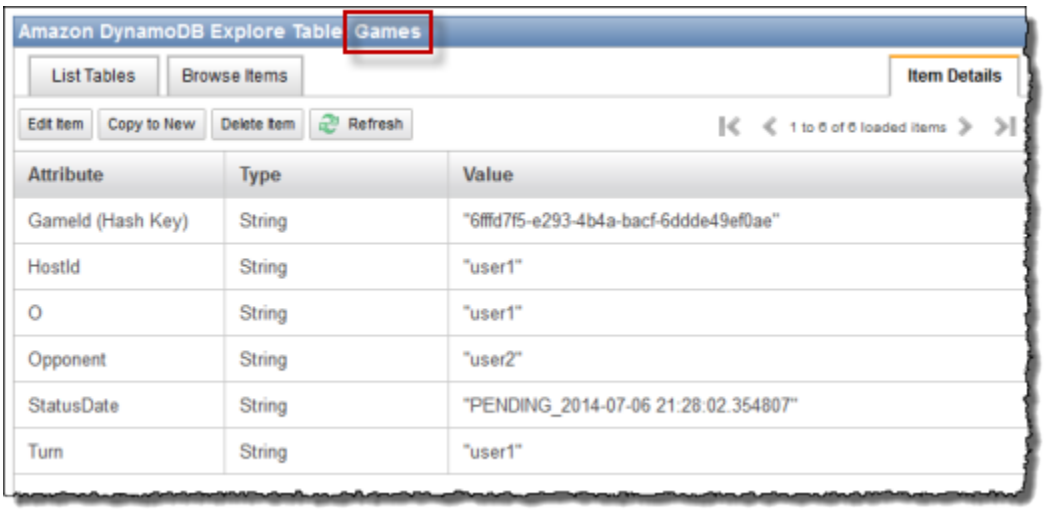
  Dalam Tic-Tac-Toe contoh ini, aplikasi menyimpan semua data game dalam sebuah tabel,`Games`. Aplikasi ini membuat satu item dalam tabel per game dan menyimpan semua data game sebagai atribut. Sebuah tic-tac-toe permainan dapat memiliki hingga sembilan gerakan. Karena tabel DynamoDB tidak memiliki skema jika hanya kunci primer yang merupakan atribut yang diperlukan, aplikasi dapat menyimpan sejumlah atribut yang bervariasi per item game.

  Tabel `Games` memiliki kunci primer sederhana yang dibuat dari satu atribut, `GameId`, jenis string. Aplikasi ini menetapkan ID unik untuk setiap game. Untuk informasi selengkapnya tentang kunci primer DynamoDB, lihat Kunci primer.

Saat pengguna memulai tic-tac-toe permainan dengan mengundang pengguna lain untuk bermain, aplikasi akan membuat item baru dalam `Games` tabel dengan atribut yang menyimpan metadata game, seperti berikut ini:

- `HostId`, pengguna yang memulai game.
- `Opponent`, pengguna yang diundang untuk ikut bermain.
- Pengguna yang mendapat giliran bermain. Pengguna yang memulai game akan bermain terlebih dahulu.
- Pengguna yang menggunakan simbol O di papan. Pengguna yang memulai game menggunakan simbol O.

Selain itu, aplikasi membuat atribut bersambung `StatusDate`, menandai status game awal sebagai `PENDING`. Tangkapan layar berikut menampilkan contoh item seperti yang muncul di konsol DynamoDB:

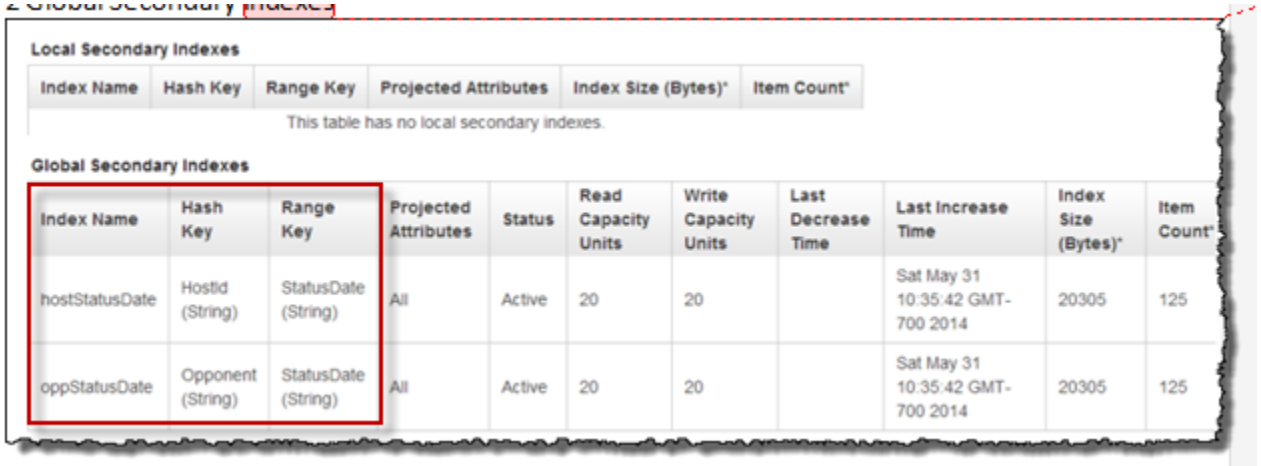


Seiring game berlangsung, aplikasi menambahkan satu atribut ke tabel untuk setiap langkah game. Nama atribut adalah posisi papan, misalnya `TopLeft` atau `BottomRight`. Misalnya, sebuah langkah mungkin memiliki atribut `TopLeft` dengan nilai `0`, atribut `TopRight` dengan nilai `0`, dan atribut `BottomRight` dengan nilai `X`. Nilai atribut adalah `0` atau `X`, tergantung pada pengguna mana yang melangkah. Contoh, pertimbangkan papan berikut ini.

- Atribut nilai bersambung – Atribut `StatusDate` mengilustrasikan atribut nilai bersambung. Dalam pendekatan ini, alih-alih membuat atribut terpisah untuk menyimpan status game (`PENDING`, `IN_PROGRESS`, dan `FINISHED`) dan tanggal (ketika langkah terakhir dilakukan), Anda menggabungkannya sebagai atribut tunggal, misalnya `IN_PROGRESS_2014-04-30 10:20:32`.

  Aplikasi kemudian menggunakan atribut `StatusDate` dalam membuat indeks sekunder dengan menentukan `StatusDate` sebagai kunci urutan untuk indeks. Manfaat penggunaan atribut nilai bersambung `StatusDate` diilustrasikan lebih lanjut dalam indeks yang akan dibahas selanjutnya.

- Indeks sekunder global – Anda dapat menggunakan kunci primer tabel, `GameId`, untuk mengkueri tabel secara efisien guna menemukan item game. Untuk mengkueri tabel pada atribut selain atribut kunci primer, DynamoDB mendukung pembuatan indeks sekunder. Dalam contoh aplikasi ini, Anda membuat dua indeks sekunder berikut:



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 -indeks. Indeks ini memiliki `HostId` sebagai kunci partisi dan `StatusDate` sebagai kunci urutan. Anda dapat menggunakan indeks ini untuk melakukan kueri pada `HostId`, misalnya untuk menemukan game yang di-hosting oleh pengguna tertentu.

- OpponentId- StatusDate -indeks. Indeks ini memiliki `OpponentId` sebagai kunci partisi dan `StatusDate` sebagai kunci urutan. Anda dapat menggunakan indeks ini untuk melakukan kueri pada `Opponent`, misalnya untuk menemukan game yang lawannya adalah pengguna tertentu.

Indeks ini disebut indeks sekunder global karena kunci partisi dalam indeks ini tidak sama dengan kunci partisi (`GameId`), yang digunakan dalam kunci primer tabel.

Perhatikan bahwa kedua indeks menentukan `StatusDate` sebagai kunci urutan. Ini akan memungkinkan hal berikut:

- Anda dapat mengkueri menggunakan operator perbandingan `BEGINS_WITH`. Misalnya, Anda dapat menemukan semua game dengan atribut `IN_PROGRESS` yang di-hosting oleh pengguna tertentu. Dalam hal ini, operator `BEGINS_WITH` memeriksa nilai `StatusDate` yang dimulai dengan `IN_PROGRESS`.
- DynamoDB menyimpan item dalam indeks dalam urutan yang diurutkan, berdasarkan nilai kunci urutan. Jadi, jika semua prefiks status sama (sebagai contoh, `IN_PROGRESS`), format ISO yang digunakan untuk bagian tanggal akan memiliki item yang diurutkan dari yang paling lama hingga yang paling baru. Pendekatan ini memungkinkan kueri tertentu untuk dilakukan secara efisien, misalnya berikut ini:
  - Ambil hingga 10 game `IN_PROGRESS` terbaru yang di-hosting oleh pengguna yang masuk. Untuk kueri ini, Anda menentukan indeks `HostId-StatusDate-index`.
  - Ambil hingga 10 game `IN_PROGRESS` terbaru yang lawannya adalah pengguna yang masuk. Untuk kueri ini, Anda menentukan indeks `OpponentId-StatusDate-index`.

Untuk informasi selengkapnya tentang indeks sekunder, lihat Meningkatkan akses data dengan indeks sekunder di DynamoDB.

2.2: Aplikasi dalam tindakan (kode panduan)

Aplikasi ini memiliki dua halaman utama:

- Halaman rumah — Halaman ini memberi pengguna login sederhana, tombol CREATE untuk membuat tic-tac-toe game baru, daftar game yang sedang berlangsung, riwayat game, dan undangan game aktif yang tertunda.

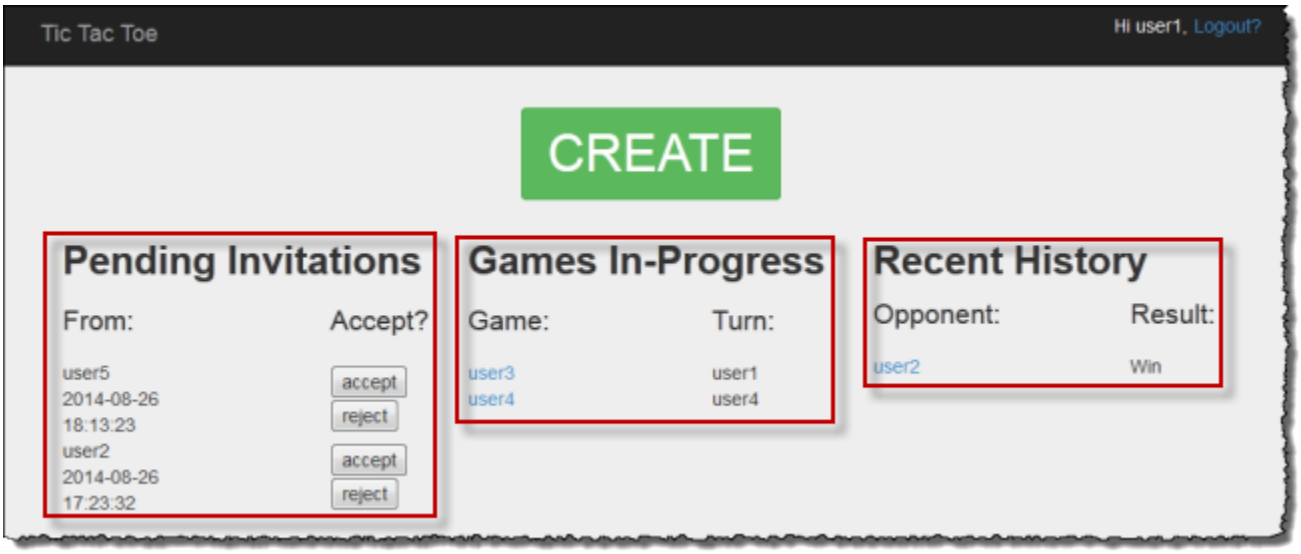
  Halaman beranda tidak disegarkan secara otomatis; Anda harus menyegarkan halaman untuk menyegarkan daftar.
- Halaman permainan — Halaman ini menunjukkan tic-tac-toe kisi tempat pengguna bermain.

Aplikasi memperbarui halaman game secara otomatis setiap detik. JavaScript Di browser Anda memanggil server web Python setiap detik untuk menanyakan tabel Game apakah item game dalam tabel telah berubah. Jika ada, JavaScript memicu penyegaran halaman sehingga pengguna melihat papan yang diperbarui.

Mari kita lihat secara detail cara kerja aplikasi.

Halaman beranda

Setelah pengguna masuk, aplikasi menampilkan tiga daftar informasi berikut.



- Undangan – Daftar ini menampilkan hingga 10 undangan terbaru dari pengguna lain yang menunggu penerimaan oleh pengguna yang masuk. Dalam tangkapan layar sebelumnya, user1 memiliki undangan tertunda dari user5 dan user2.
- Game sedang dimainkan – Daftar ini menampilkan hingga 10 game terbaru yang sedang dimainkan. Ini adalah game yang dimainkan pengguna secara aktif, yang berstatus `IN_PROGRESS`. Dalam tangkapan layar, user1 secara aktif memainkan tic-tac-toe game dengan user3 dan user4.
- Riwayat terbaru – Daftar ini menampilkan hingga 10 game terbaru yang telah diselesaikan pengguna, yang berstatus `FINISHED`. Dalam game yang ditampilkan di tangkapan layar, user1 sebelumnya pernah bermain dengan user2. Untuk setiap game yang telah selesai dimainkan, daftar ini menunjukkan hasil game.

Dalam kode, fungsi `index` (di `application.py`) membuat tiga panggilan berikut untuk mengambil informasi status game:

```
inviteGames     = controller.getGameInvites(session["username"])
inProgressGames = controller.getGamesWithStatus(session["username"], "IN_PROGRESS")
finishedGames   = controller.getGamesWithStatus(session["username"], "FINISHED")
```

Setiap panggilan ini mengembalikan daftar item dari DynamoDB yang dikemas oleh objek `Game`. Sangat mudah untuk mengekstrak data dari objek-objek ini dalam tampilan. Fungsi indeks meneruskan daftar objek ini ke tampilan untuk merender HTML.

```
return render_template("index.html",
                       user=session["username"],
                       invites=inviteGames,
                       inprogress=inProgressGames,
                       finished=finishedGames)
```

Tic-Tac-ToeAplikasi mendefinisikan `Game` kelas terutama untuk menyimpan data game diambil dari DynamoDB. Fungsi tersebut mengembalikan daftar objek `Game` yang memungkinkan Anda mengisolasi aplikasi lainnya dari kode yang terkait dengan item Amazon DynamoDB. Dengan demikian, fungsi tersebut membantu Anda memisahkan kode aplikasi Anda dari detail lapisan penyimpanan data.

Pola arsitektur yang dijelaskan di sini juga disebut sebagai pola UI model-view-controller (MVC). Dalam hal ini, instans objek `Game` (yang mewakili data) adalah model, dan halaman HTML adalah tampilan. Pengontrol dibagi menjadi dua file. File `application.py` memiliki logika pengontrol untuk kerangka kerja Flask, dan logika bisnis diisolasi dalam file `gameController.py`. Artinya, aplikasi menyimpan segala sesuatu yang ada hubungannya dengan DynamoDB SDK dalam file tersendiri di folder `dynamodb`.

Mari kita meninjau tiga fungsi dan cara fungsi tersebut mengkueri tabel Game menggunakan indeks sekunder global untuk mengambil data yang relevan.

Menggunakan getGameInvites untuk mendapatkan daftar undangan game yang tertunda

Fungsi `getGameInvites` mengambil daftar 10 undangan tertunda terbaru. Game ini telah dibuat oleh pengguna, tetapi lawan belum menerima undangan game. Untuk game tersebut, statusnya tetap `PENDING` sampai lawan menerima undangan. Jika lawan menolak undangan, aplikasi akan menghapus item yang sesuai dari tabel.

Fungsi menentukan kueri sebagai berikut:

- Fungsi menentukan indeks `OpponentId-StatusDate-index` untuk digunakan dengan nilai kunci indeks berikut dan operator perbandingan:
  - Kunci partisinya adalah `OpponentId` dan mengambil kunci indeks *`user ID`*.
  - Kunci urutannya adalah `StatusDate` dan mengambil operator perbandingan serta nilai kunci indeks `beginswith="PENDING_"`.

  Anda menggunakan indeks `OpponentId-StatusDate-index` untuk mengambil game yang mengundang pengguna yang masuk—yaitu, pengguna yang masuk adalah lawannya.
- Kueri membatasi hasil hingga 10 item.

```
gameInvitesIndex = self.cm.getGamesTable().query(
                                            Opponent__eq=user,
                                            StatusDate__beginswith="PENDING_",
                                            index="OpponentId-StatusDate-index",
                                            limit=10)
```

Dalam indeks, untuk setiap `OpponentId` (kunci partisi) DynamoDB menyimpan item yang diurutkan berdasarkan `StatusDate` (kunci urutan). Oleh karena itu, game yang dikembalikan oleh kueri akan berupa 10 game terbaru.

Menggunakan getGamesWith Status untuk mendapatkan daftar game dengan status tertentu

Setelah lawan menerima undangan game, status game berubah menjadi `IN_PROGRESS`. Setelah game selesai, status berubah menjadi `FINISHED`.

Kueri untuk menemukan game yang sedang dimainkan atau telah selesai sama, kecuali nilai statusnya berbeda. Oleh karena itu, aplikasi mendefinisikan fungsi `getGamesWithStatus`, yang mengambil nilai status sebagai parameter.

```
inProgressGames = controller.getGamesWithStatus(session["username"], "IN_PROGRESS")
finishedGames   = controller.getGamesWithStatus(session["username"], "FINISHED")
```

Bagian berikut membahas game yang sedang berlangsung, tetapi deskripsi yang sama juga berlaku untuk game yang sudah selesai.

Daftar game yang sedang berlangsung untuk pengguna tertentu meliputi hal berikut ini:

- Game yang sedang berlangsung yang di-hosting oleh pengguna

- Game yang sedang berlangsung yang lawannya adalah pengguna

Fungsi `getGamesWithStatus` menjalankan dua kueri berikut, setiap kali menggunakan indeks sekunder yang sesuai.

- Fungsi mengkueri tabel `Games` menggunakan indeks `HostId-StatusDate-index`. Untuk indeks, kueri menentukan nilai kunci primer—baik nilai kunci partisi (`HostId`) dan kunci urutan (`StatusDate`), bersama dengan operator perbandingan.

  ```
  hostGamesInProgress = self.cm.getGamesTable ().query(HostId__eq=user,
                                                       StatusDate__beginswith=status,
                                                       index="HostId-StatusDate-index",
                                                       limit=10)
  ```

  Perhatikan sintaks Python untuk operator perbandingan:

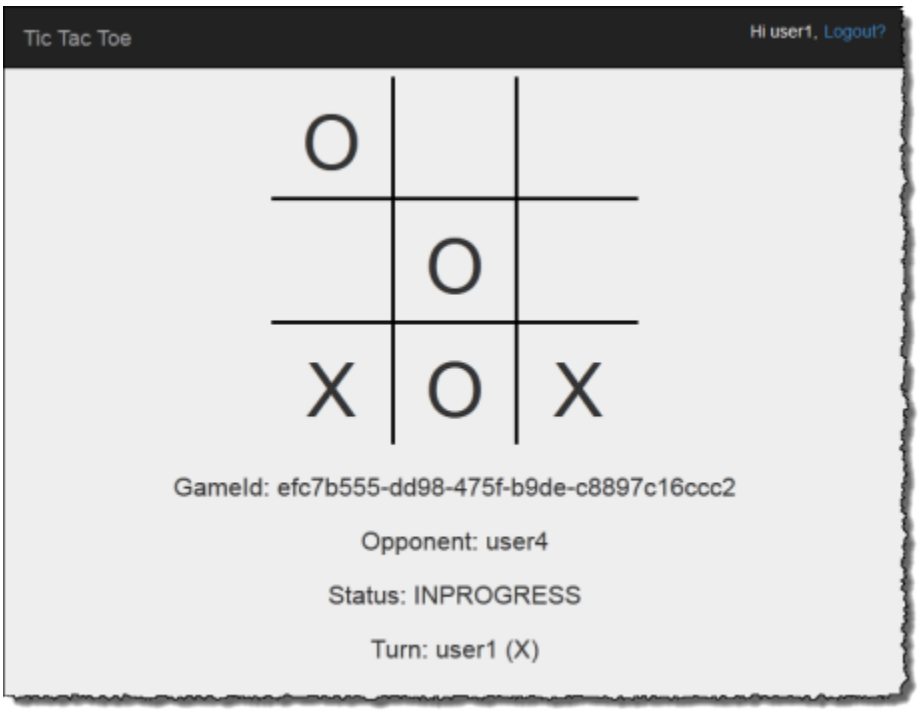
  - `HostId__eq=user` menentukan operator perbandingan kesetaraan.
  - `StatusDate__beginswith=status` menentukan operator perbandingan `BEGINS_WITH`.
- Fungsi mengkueri tabel `Games` menggunakan indeks `OpponentId-StatusDate-index`.

  ```
  oppGamesInProgress = self.cm.getGamesTable().query(Opponent__eq=user,
                                                     StatusDate__beginswith=status,
                                                     index="OpponentId-StatusDate-index",
                                                     limit=10)
  ```
- Fungsi tersebut kemudian menggabungkan dua daftar, mengurutkan, dan membuat daftar objek `Game` untuk 0 hingga 10 item pertama, serta mengembalikan daftar tersebut ke fungsi pemanggilan (yaitu, indeks).

  ```
  games = self.mergeQueries(hostGamesInProgress,
                            oppGamesInProgress)
  return games
  ```

Halaman game

Halaman game adalah tempat pengguna memainkan tic-tac-toe game. Halaman game menampilkan grid game bersama dengan informasi yang relevan terkait game. Tangkapan layar berikut menunjukkan contoh game yang sedang dimainkan:



Aplikasi menampilkan halaman game dalam situasi-situasi berikut:

- Pengguna membuat game yang mengundang pengguna lain untuk ikut bermain.

  Dalam hal ini, halaman menunjukkan pengguna sebagai host dan status game sebagai `PENDING`, menunggu lawan menerima undangan.
- Pengguna menerima salah satu undangan yang tertunda di halaman beranda.

  Dalam hal ini, halaman menunjukkan pengguna sebagai lawan dan status game sebagai `IN_PROGRESS`.

Pemilihan pengguna di papan menghasilkan permintaan `POST` formulir untuk aplikasi. Artinya, Flask memanggil fungsi `selectSquare` (di `application.py`) dengan data formulir HTML. Fungsi ini, pada gilirannya, memanggil fungsi `updateBoardAndTurn` (di `gameController.py`) untuk memperbarui item game sebagai berikut:

- Fungsi menambahkan atribut baru khusus untuk langkah.
- Fungsi memperbarui nilai atribut `Turn` untuk pengguna yang mendapat giliran berikutnya.

```
controller.updateBoardAndTurn(item, value, session["username"])
```

Fungsi mengembalikan true jika pembaruan item berhasil; jika tidak, fungsi akan mengembalikan false. Perhatikan hal berikut terkait fungsi `updateBoardAndTurn`:

- Fungsi memanggil fungsi `update_item` SDK untuk Python untuk membuat serangkaian pembaruan terbatas pada item yang sudah ada. Fungsi ini dipetakan ke operasi `UpdateItem` di DynamoDB. Untuk informasi selengkapnya, lihat UpdateItem.

> **Note**
>
> Perbedaan antara operasi `UpdateItem` dan `PutItem` adalah bahwa `PutItem` menggantikan seluruh item. Untuk informasi selengkapnya, lihat PutItem.

Untuk panggilan `update_item`, kode mengidentifikasi berikut ini:

- Kunci primer tabel `Games` (yaitu, `ItemId`).

```
key = { "GameId" : { "S" : gameId } }
```

- Atribut baru yang akan ditambahkan, khusus untuk langkah pengguna saat ini, dan nilainya (misalnya, `TopLeft="X"`).

```
attributeUpdates = {
    position : {
        "Action" : "PUT",
        "Value" : { "S" : representation }
    }
}
```

- Kondisi yang harus true agar pembaruan dapat diterapkan:
  - Game harus sedang dimainkan. Yaitu, nilai atribut `StatusDate` harus diawali dengan `IN_PROGRESS`.
  - Giliran saat ini harus merupakan giliran pengguna yang valid sebagaimana ditentukan oleh atribut `Turn`.
  - Kotak yang dipilih pengguna harus tersedia. Artinya, atribut yang sesuai dengan kotak tidak boleh ada.

```
expectations = {"StatusDate" : {"AttributeValueList": [{"S" : "IN_PROGRESS_"}],
    "ComparisonOperator": "BEGINS_WITH"},
```

```
    "Turn" : {"Value" : {"S" : current_player}},
    position : {"Exists" : False}}
```

Sekarang fungsi memanggil `update_item` untuk memperbarui item.

```
self.cm.db.update_item("Games", key=key,
    attribute_updates=attributeUpdates,
    expected=expectations)
```

Setelah fungsi kembali, fungsi `selectSquare` memanggil pengalihan, seperti yang ditunjukkan dalam contoh berikut.

```
redirect("/game="+gameId)
```

Panggilan ini menyebabkan browser disegarkan. Sebagai bagian dari penyegaran ini, aplikasi memeriksa apakah game berakhir dengan kemenangan atau seri. Jika game sudah selesai, aplikasi akan memperbarui item game.

Langkah 3: Deploy dalam produksi menggunakan layanan DynamoDB

Topik

- 3.1: Buat peran IAM untuk Amazon EC2
- 3.2: Buat tabel game di Amazon DynamoDB
- 3.3: Bundel dan gunakan kode tic-tac-toe aplikasi
- 3.4: Siapkan lingkungan AWS Elastic Beanstalk

Di bagian sebelumnya, Anda menerapkan dan menguji Tic-Tac-Toe aplikasi secara lokal di komputer Anda menggunakan DynamoDB lokal. Sekarang, Anda men-deploy aplikasi dalam produksi sebagai berikut:

- Menyebarkan aplikasi menggunakan AWS Elastic Beanstalk, easy-to-use layanan untuk menyebarkan dan menskalakan aplikasi dan layanan web. Untuk informasi selengkapnya, lihat Menerapkan aplikasi labu ke. AWS Elastic Beanstalk

  Elastic Beanstalk meluncurkan satu atau beberapa instans Amazon Elastic Compute Cloud (EC2Amazon), yang Anda konfigurasikan melalui Elastic Beanstalk, tempat aplikasi Anda akan berjalan. Tic-Tac-Toe

- Menggunakan layanan Amazon DynamoDB, buat tabel `Games` yang ada di AWS , bukan secara lokal di komputer Anda.

Selain itu, Anda juga harus mengonfigurasi izin. AWS Sumber daya apa pun yang Anda buat, seperti `Games` tabel di DynamoDB, bersifat pribadi secara default. Hanya pemilik sumber daya, yaitu akun AWS yang membuat tabel `Games`, yang dapat mengakses tabel ini. Dengan demikian, secara default Tic-Tac-Toe aplikasi Anda tidak dapat memperbarui `Games` tabel.

Untuk memberikan izin yang diperlukan, Anda membuat peran AWS Identity and Access Management (IAM) dan memberikan izin peran ini untuk mengakses tabel. `Games` EC2 Instans Amazon Anda pertama-tama mengasumsikan peran ini. Sebagai tanggapan, AWS mengembalikan kredensi keamanan sementara yang dapat digunakan EC2 instans Amazon untuk memperbarui `Games` tabel atas nama aplikasi. Tic-Tac-Toe Saat mengonfigurasi aplikasi Elastic Beanstalk, Anda menentukan peran IAM yang dapat diasumsikan oleh instans atau instans EC2 Amazon. Untuk informasi selengkapnya tentang peran IAM, lihat peran IAM untuk amazon EC2 di EC2 Panduan Pengguna Amazon.

> **Note**
>
> Sebelum Anda membuat EC2 instans Amazon untuk Tic-Tac-Toe aplikasi, Anda harus terlebih dahulu memutuskan AWS Wilayah di mana Anda ingin Elastic Beanstalk untuk membuat instance. Setelah Anda membuat aplikasi Elastic Beanstalk, berikan titik akhir dan nama wilayah yang sama dalam file konfigurasi. Tic-Tac-ToeAplikasi menggunakan informasi dalam file ini untuk membuat `Games` tabel dan mengirim permintaan berikutnya di AWS Wilayah tertentu. Baik tabel `Games` DynamoDB dan instance EC2 Amazon yang diluncurkan Elastic Beanstalk harus berada di Wilayah yang sama. Untuk daftar Wilayah yang tersedia, lihat Amazon DynamoDB di Referensi Umum Amazon Web.

Singkatnya, Anda melakukan hal berikut untuk menyebarkan Tic-Tac-Toe aplikasi dalam produksi:

1. Buat peran IAM menggunakan layanan IAM. Anda melampirkan kebijakan pada peran ini yang memberi tindakan DynamoDB izin untuk mengakses `Games` tabel.
2. Bundel kode Tic-Tac-Toe aplikasi dan file konfigurasi, dan buat `.zip` file. Anda menggunakan `.zip` file ini untuk memberikan kode Tic-Tac-Toe aplikasi ke Elastic Beanstalk untuk diletakkan di server Anda. Untuk informasi selengkapnya tentang membuat paket, lihat Membuat paket sumber aplikasi di Panduan Developer AWS Elastic Beanstalk .

Dalam file konfigurasi (`beanstalk.config`), berikan informasi titik akhir dan Wilayah AWS . Tic-Tac-ToeAplikasi ini menggunakan informasi ini untuk menentukan Wilayah DynamoDB mana yang akan diajak bicara.

3. Siapkan lingkungan Elastic Beanstalk. Elastic Beanstalk meluncurkan instans atau EC2 instance Tic-Tac-Toe Amazon dan menerapkan bundel aplikasi Anda di dalamnya. Setelah lingkungan Elastic Beanstalk siap, berikan nama file konfigurasi dengan menambahkan variabel lingkungan `CONFIG_FILE`.
4. Buat tabel DynamoDB. Menggunakan layanan Amazon DynamoDB, Anda membuat `Games` tabel, bukan secara lokal AWS di komputer Anda. Perlu diingat, tabel ini memiliki kunci primer sederhana yang dibuat dari kunci partisi `GameId` jenis string.
5. Uji game dalam produksi.

3.1: Buat peran IAM untuk Amazon EC2

Membuat peran IAM dari EC2 jenis Amazon memungkinkan EC2 instans Amazon yang menjalankan Tic-Tac-Toe aplikasi Anda untuk mengambil peran yang benar dan membuat permintaan aplikasi untuk mengakses `Games` tabel. Saat membuat peran, pilih opsi Kebijakan Kustom dan salin serta tempel kebijakan berikut.

```
{
   "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/*"
         ]
      }
```

```
    ]
}
```

Untuk petunjuk mendetail, lihat Membuat peran untuk layanan AWS (AWS Management Console) di Panduan Pengguna IAM.

3.2: Buat tabel game di Amazon DynamoDB

Tabel `Games` di DynamoDB menyimpan data game. Jika tabel tidak ada, aplikasi akan membuat tabel untuk Anda. Dalam hal ini, biarkan aplikasi membuat tabel `Games`.

3.3: Bundel dan gunakan kode tic-tac-toe aplikasi

Jika Anda mengikuti langkah-langkah contoh ini, maka Anda sudah mengunduh Tic-Tac-Toe aplikasi. Jika tidak, unduh aplikasi dan ekstrak semua file ke folder di komputer lokal Anda. Untuk petunjuk, lihat Langkah 1: Deploy dan uji secara lokal.

Setelah mengekstrak semua file, Anda akan memiliki folder `code`. Untuk menyerahkan folder ini ke Elastic Beanstalk, gabungkan konten folder ini sebagai file `.zip`. Pertama, tambahkan file konfigurasi ke folder tersebut. Aplikasi Anda menggunakan informasi titik akhir dan Wilayah untuk membuat tabel DynamoDB di Wilayah tertentu dan membuat permintaan operasi tabel berikutnya menggunakan titik akhir yang ditentukan.

1. Beralih ke folder tempat Anda mengunduh Tic-Tac-Toe aplikasi.
2. Di folder root aplikasi, buat file teks bernama `beanstalk.config` dengan konten berikut.

   ```
   [dynamodb]
   region=<AWS region>
   endpoint=<DynamoDB endpoint>
   ```

   Misalnya, Anda dapat menggunakan konten berikut.

   ```
   [dynamodb]
   region=us-west-2
   endpoint=dynamodb.us-west-2.amazonaws.com
   ```

   Untuk daftar Wilayah yang tersedia, lihat Amazon DynamoDB di Referensi Umum Amazon Web Services.

> **Important**
>
> Wilayah yang ditentukan dalam file konfigurasi adalah lokasi di mana Tic-Tac-Toe aplikasi membuat `Games` tabel di DynamoDB. Anda harus membuat aplikasi Elastic Beanstalk yang dibahas pada bagian berikutnya di Wilayah yang sama.

> **Note**
>
> Saat membuat aplikasi Elastic Beanstalk, Anda meminta untuk meluncurkan lingkungan tempat Anda dapat memilih jenis lingkungan. Untuk menguji aplikasi Tic-Tac-Toe contoh, Anda dapat memilih jenis lingkungan Single Instance, lewati yang berikut ini, dan pergi ke langkah berikutnya.
> Namun, jenis lingkungan penyeimbangan beban dan penskalaan otomatis menyediakan lingkungan dengan ketersediaan tinggi dan dapat diskalakan, sesuatu yang harus Anda pertimbangkan saat membuat dan men-deploy aplikasi lain. Jika memilih jenis lingkungan ini, Anda juga perlu untuk menghasilkan UUID dan menambahkannya ke file konfigurasi, seperti yang ditunjukkan berikut.
>
> ```
> [dynamodb]
> region=us-west-2
> endpoint=dynamodb.us-west-2.amazonaws.com
> [flask]
> secret_key= 284e784d-1a25-4a19-92bf-8eeb7a9example
> ```
>
> Dalam komunikasi klien-server, ketika server mengirimkan respons, demi keamanan, server mengirimkan cookie bertanda tangan yang dikirim kembali oleh klien ke server pada permintaan berikutnya. Jika hanya ada satu server, server dapat secara lokal menghasilkan kunci enkripsi saat server dimulai. Jika ada banyak server, semua server perlu mengetahui kunci enkripsi yang sama; jika tidak, server tersebut tidak akan dapat membaca cookie yang ditetapkan oleh server peer. Dengan menambahkan `secret_key` ke file konfigurasi, Anda memberi tahu semua server untuk menggunakan kunci enkripsi ini.

3. Buka konten folder root aplikasi (yang mencakup file `beanstalk.config`), misalnya, `TicTacToe.zip`.

4. Unggah file `.zip` ke bucket Amazon Simple Storage Service (Amazon S3). Di bagian berikutnya, Anda akan menyediakan file `.zip` ini ke Elastic Beanstalk untuk mengunggah di server atau di beberapa server.

   Untuk petunjuk tentang cara mengunggah bucket Amazon S3, lihat Membuat bucket dan Menambahkan objek ke bucket di Panduan Pengguna Amazon Simple Storage Service.

3.4: Siapkan lingkungan AWS Elastic Beanstalk

Pada langkah ini, Anda akan membuat aplikasi Elastic Beanstalk, yang merupakan kumpulan komponen termasuk lingkungan. Untuk contoh ini, Anda meluncurkan satu EC2 instans Amazon untuk menyebarkan dan menjalankan Tic-Tac-Toe aplikasi Anda.

1. Masukkan URL kustom berikut untuk mengatur konsol Elastic Beanstalk dalam mengatur lingkungan.

   ```
   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
   ```

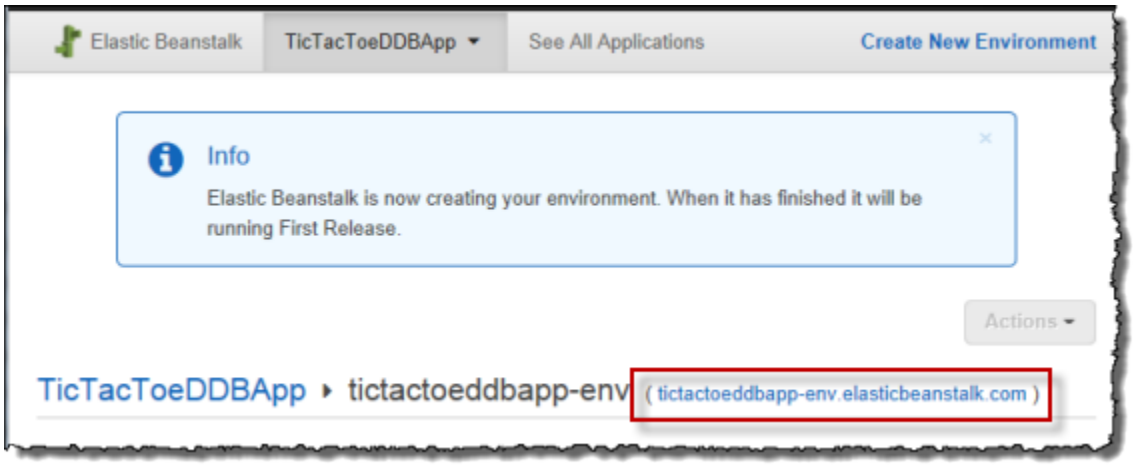
   Untuk informasi selengkapnya tentang kustom URLs, lihat Membuat URL Launch Now di Panduan AWS Elastic Beanstalk Pengembang. Untuk URL, perhatikan berikut ini:

   - Anda harus memberikan nama AWS Wilayah (sama dengan yang Anda berikan dalam file konfigurasi), nama bucket Amazon S3, dan nama objek.
   - Untuk pengujian, URL meminta jenis SingleInstancelingkungan, dan `t1.micro` sebagai tipe instance.
   - Nama aplikasi harus unik. Dengan demikian, di URL sebelumnya, kami sarankan agar Anda menambahkan nama Anda ke `applicationName`.

   Tindakan ini akan membuka konsol Elastic Beanstalk. Dalam beberapa kasus, Anda mungkin perlu masuk.

2. Di konsol Elastic Beanstalk, pilih Tinjau dan Luncurkan, lalu pilih Luncurkan.

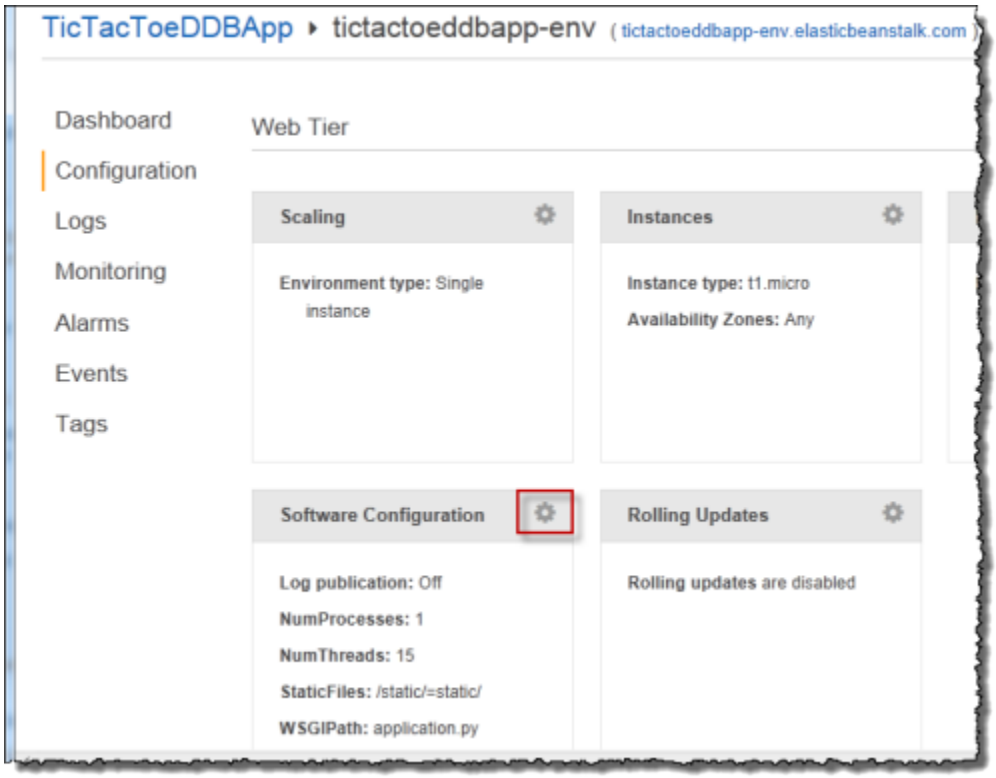
3. Catat URL untuk referensi di masa mendatang. URL ini membuka halaman beranda Tic-Tac-Toe aplikasi Anda.



4. Konfigurasikan Tic-Tac-Toe aplikasi sehingga mengetahui lokasi file konfigurasi.

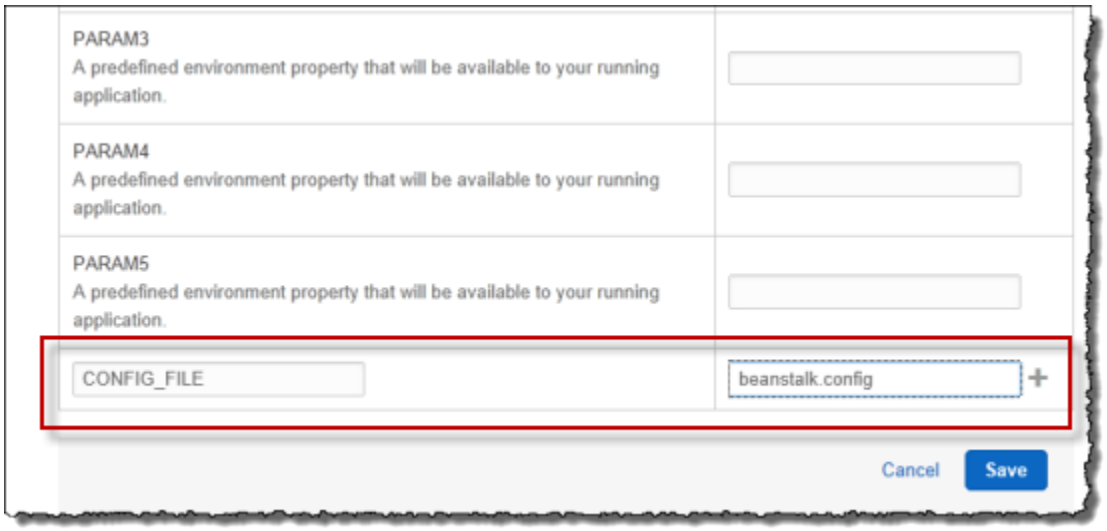
   Setelah Elastic Beanstalk membuat aplikasi, pilih Konfigurasi.

   a. Pilih ikon roda gigi di sebelah Konfigurasi Perangkat Lunak, seperti yang ditunjukkan dalam gambar berikut.



   b. Di akhir bagian Properti Lingkungan, masukkan **`CONFIG_FILE`** dan nilai **`beanstalk.config`**, lalu pilih Simpan.

Mungkin diperlukan waktu beberapa menit hingga pembaruan lingkungan ini selesai.

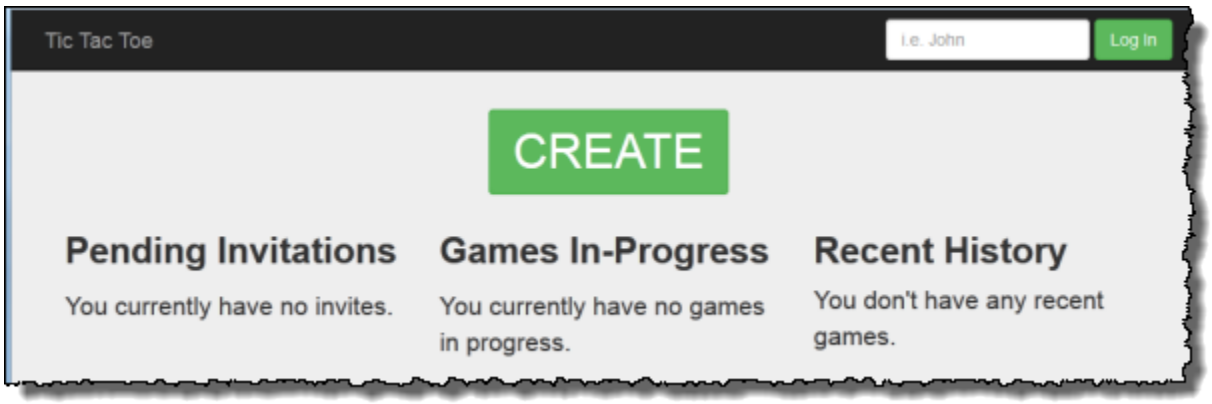


Setelah pembaruan selesai, Anda bisa memainkan game.

5. Di browser, masukkan URL yang Anda salin pada langkah sebelumnya, seperti yang ditunjukkan pada contoh berikut.

```
http://<pen-name>.elasticbeanstalk.com
```

Tindakan ini akan membuka halaman beranda aplikasi.



6. Masuk sebagai testuser1, dan pilih CREATE untuk memulai game baru tic-tac-toe.
7. Masukkan **`testuser2`** di kotak Pilih Lawan.

8. Buka jendela browser lain.

   Pastikan Anda menghapus semua cookie di jendela browser Anda sehingga Anda tidak akan login sebagai pengguna yang sama.

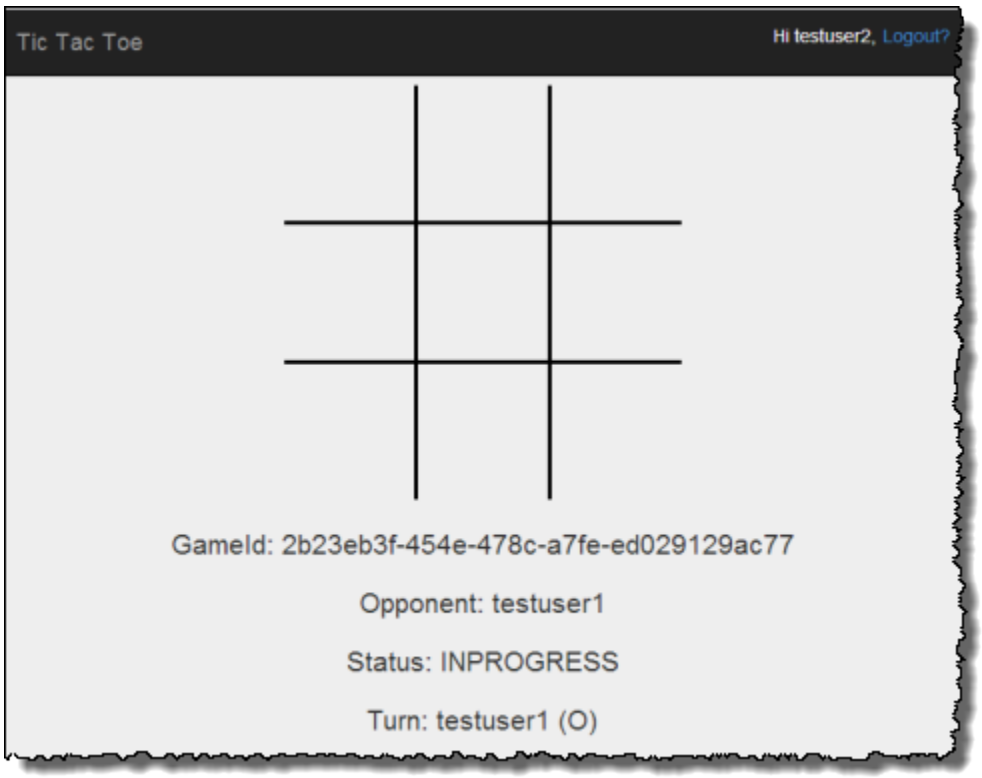
9. Masukkan URL yang sama untuk membuka halaman beranda aplikasi, seperti yang ditunjukkan dalam contoh berikut.

   ```
   http://<env-name>.elasticbeanstalk.com
   ```

10. Masuk sebagai testuser2.

11. Untuk undangan dari testuser1 dalam daftar undangan yang tertunda, pilih terima.

12. Sekarang halaman game muncul.



Testuser1 dan testuser2 dapat memainkan game. Untuk setiap langkah, aplikasi menyimpan langkah tersebut di item yang sesuai di tabel `Games`.

Langkah 4: Bersihkan Sumber Daya

Sekarang Anda telah menyelesaikan Tic-Tac-Toe penerapan dan pengujian aplikasi. Aplikasi ini mencakup pengembangan aplikasi end-to-end web di Amazon DynamoDB, kecuali untuk otentikasi pengguna. Aplikasi ini menggunakan informasi login di halaman beranda hanya untuk menambahkan nama pemain saat membuat game. Dalam aplikasi produksi, Anda akan menambahkan kode yang diperlukan untuk melakukan login dan autentikasi pengguna.

Jika Anda selesai menguji, Anda dapat menghapus sumber daya yang Anda buat untuk menguji Tic-Tac-Toe aplikasi agar tidak dikenakan biaya apa pun.

Untuk menghapus sumber daya yang Anda buat

1. Hapus tabel `Games` yang Anda buat di DynamoDB.
2. Hentikan lingkungan Elastic Beanstalk untuk membebaskan instans Amazon. EC2
3. Hapus peran IAM yang Anda buat.

4. Hapus objek yang Anda buat di Amazon S3.

Kata-kata penggunaan khusus di DynamoDB

Kata kunci berikut ini khusus untuk digunakan oleh DynamoDB. Jangan gunakan kata-kata ini sebagai nama atribut dalam ekspresi. Daftar ini tidak peka huruf besar/kecil.

Jika Anda harus menulis ekspresi berisi nama atribut yang bertentangan dengan kata penggunaan khusus di DynamoDB, Anda dapat menentukan nama atribut ekspresi untuk digunakan sebagai pengganti kata penggunaan khusus tersebut. Untuk informasi selengkapnya, lihat Nama atribut ekspresi (alias) di 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 Contoh SDK for Java 1.x

Bagian ini berisi kode contoh aplikasi DAX menggunakan SDK untuk Java 1.x.

Topik

- Menggunakan DAX dengan AWS SDK for Java 1.x
- Memodifikasi SDK untuk Java 1.x yang ada agar menggunakan DAX
- Mengkueri indeks sekunder global dengan SDK untuk Java 1.x

Menggunakan DAX dengan AWS SDK for Java 1.x

Ikuti prosedur ini untuk menjalankan sampel Java untuk Amazon DynamoDB Accelerator (DAX) di instans Amazon Anda. EC2

> **Note**
>
> Instruksi ini untuk aplikasi yang menggunakan AWS SDK for Java 1.x. Untuk aplikasi yang menggunakan AWS SDK untuk Java 2.x, lihat Java dan DAX.

Cara menjalankan sampel Java untuk DAX

1. Instal Java Development Kit (JDK).

```
sudo yum install -y java-devel
```

2. Unduh AWS SDK untuk Java (`.zip`file), lalu ekstrak.

```
wget http://sdk-for-java.amazonwebservices.com/latest/aws-java-sdk.zip

unzip aws-java-sdk.zip
```

3. Unduh versi terbaru dari klien DAX Java (file `.jar`).

```
wget http://dax-sdk.s3-website-us-west-2.amazonaws.com/java/DaxJavaClient-
latest.jar
```

> **Note**
>
> Klien untuk DAX SDK untuk Java tersedia di Apache Maven. Untuk informasi selengkapnya, lihat Menggunakan klien sebagai dependensi Apache Maven.

4. Atur variabel CLASSPATH Anda. Dalam contoh ini, ganti *`sdkVersion`* dengan nomor versi aktual dari AWS SDK untuk Java (misalnya,`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. Unduh sampel kode sumber program (file `.zip`).

```
wget http://docs.aws.amazon.com/amazondynamodb/latest/developerguide/samples/
TryDax.zip
```

Setelah unduhan selesai, ekstrak file sumber.

```
unzip TryDax.zip
```

6. Buka direktori kode Java dan kompilasi kode sebagai berikut.

```
cd TryDax/java/
javac TryDax*.java
```

7. Jalankan program.

```
java TryDax
```

Anda akan melihat output seperti yang berikut ini.

```
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.
```

Perhatikan informasi waktu, yaitu jumlah milidetik yang diperlukan untuk pengujian `GetItem`, `Query`, dan `Scan`.

8. Pada langkah sebelumnya, Anda menjalankan program terhadap titik akhir DynamoDB. Sekarang jalankan program lagi, tetapi kali ini, operasi `GetItem`, `Query` dan `Scan` diproses oleh klaster DAX Anda.

   Untuk menentukan titik akhir klaster DAX Anda, pilih salah satu dari berikut ini:

   - Menggunakan konsol DynamoDB — Pilih klaster DAX Anda. Titik akhir klaster ditampilkan pada konsol, seperti dalam contoh berikut.

     ```
     dax://my-cluster.l6fzcv.dax-clusters.us-east-1.amazonaws.com
     ```

   - Menggunakan AWS CLI — Masukkan perintah berikut.

     ```
     aws dax describe-clusters --query "Clusters[*].ClusterDiscoveryEndpoint"
     ```

     Titik akhir klaster ditampilkan pada output, seperti dalam contoh berikut.

     ```
     {
         "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"
     }
     ```

   Sekarang jalankan program lagi, tetapi kali ini tentukan titik akhir klaster sebagai parameter baris perintah.

```
java TryDax dax://my-cluster.l6fzcv.dax-clusters.us-east-1.amazonaws.com
```

Periksa sisa output dan perhatikan informasi waktu. Waktu berlalu untuk `GetItem`, `Query`, dan `Scan` harus jauh lebih rendah dengan DAX dibandingkan dengan DynamoDB.

Untuk informasi selengkapnya tentang program ini, lihat bagian berikut:

- TryDax.jawa
- TryDaxHelper.jawa
- TryDaxTests.jawa

Menggunakan klien sebagai dependensi Apache Maven

Ikuti langkah-langkah berikut untuk menggunakan klien untuk DAX SDK untuk Java dalam aplikasi Anda sebagai dependensi.

Cara menggunakan klien sebagai dependensi Maven

1. Unduh dan instal Apache Maven. Untuk informasi selengkapnya, lihat Mengunduh Apache Maven dan Menginstal Apache Maven.
2. Tambahkan dependensi Maven klien ke file Project Object Model (POM) aplikasi Anda. Dalam contoh ini, ganti `x.x.x.x` dengan nomor versi klien sebenarnya (misalnya, `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.jawa

File `TryDax.java` berisi metode `main`. Jika Anda menjalankan program tanpa parameter baris perintah, klien Amazon DynamoDB akan dibuat dan digunakan untuk semua operasi API. Jika Anda

menentukan titik akhir klaster DynamoDB Accelerator (DAX) pada baris perintah, program juga membuat klien DAX dan menggunakannya untuk operasi `GetItem`, `Query`, dan `Scan`.

Anda dapat memodifikasi program dengan beberapa cara:

- Menggunakan klien DAX, bukan klien DynamoDB. Untuk informasi selengkapnya, lihat Java dan DAX.
- Memilih nama yang berbeda untuk tabel pengujian.
- Memodifikasi jumlah item yang ditulis dengan mengubah parameter `helper.writeData`. Parameter kedua adalah angka kunci partisi dan parameter ketiga adalah kunci urutan. Secara default, program menggunakan 1-10 untuk nilai kunci partisi dan 1-10 untuk nilai kunci urutan, dengan total 100 item yang ditulis ke tabel. Untuk informasi selengkapnya, lihat TryDaxHelper.jawa.
- Memodifikasi jumlah pengujian `GetItem`, `Query`, dan `Scan`, serta memodifikasi parameternya.
- Mengomentari baris yang berisi `helper.createTable` dan `helper.deleteTable` (jika Anda tidak ingin membuat dan menghapus tabel setiap kali menjalankan program).

> **Note**
>
> Untuk menjalankan program ini, Anda dapat mengatur Maven untuk menggunakan klien untuk DAX SDK for Java dan dependensi as. AWS SDK untuk Java Untuk informasi selengkapnya, lihat Menggunakan klien sebagai dependensi Apache Maven.
> Selain itu, Anda juga dapat mengunduh dan menyertakan klien DAX Java dan AWS SDK untuk Java di classpath Anda. Lihat Java dan DAX untuk contoh pengaturan variabel `CLASSPATH`.

```
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.jawa

File `TryDaxHelper.java` berisi metode utilitas.

Metode `getDynamoDBClient` dan `getDaxClient` menyediakan klien untuk Amazon DynamoDB dan DynamoDB Accelerator (DAX). Untuk operasi bidang kontrol (`CreateTable`, `DeleteTable`) dan operasi penulisan, program menggunakan klien DynamoDB. Jika Anda menentukan titik akhir

klaster DAX, program utama membuat klien DAX untuk melakukan operasi baca (`GetItem`, `Query`, `Scan`).

Metode `TryDaxHelper` lainnya (`createTable`, `writeData`, `deleteTable`) digunakan untuk mengatur dan menghapus tabel DynamoDB dan datanya.

Anda dapat memodifikasi program dengan beberapa cara:

- Menggunakan pengaturan throughput lain yang disediakan untuk tabel.
- Memodifikasi ukuran setiap item yang ditulis (lihat variabel `stringSize` di metode `writeData`).
- Memodifikasi jumlah pengujian `GetItem`, `Query`, dan `Scan` serta parameternya.
- Mengomentari baris yang berisi `helper.CreateTable` dan `helper.DeleteTable` (jika Anda tidak ingin membuat dan menghapus tabel setiap kali menjalankan program).

> **Note**
>
> Untuk menjalankan program ini, Anda dapat mengatur Maven untuk menggunakan klien untuk DAX SDK for Java dan dependensi as. AWS SDK untuk Java Untuk informasi selengkapnya, lihat Menggunakan klien sebagai dependensi Apache Maven.
> Atau, Anda dapat mengunduh dan menyertakan klien DAX Java dan AWS SDK untuk Java di classpath Anda. Lihat Java dan DAX untuk contoh pengaturan variabel `CLASSPATH`.

```
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.jawa

File `TryDaxTests.java` berisi metode yang melakukan operasi pembacaan terhadap tabel pengujian di Amazon DynamoDB. Metode ini tidak peduli dengan cara data diakses (menggunakan klien DynamoDB atau klien DAX), sehingga tidak memerlukan modifikasi logika aplikasi.

Anda dapat memodifikasi program dengan beberapa cara:

- Memodifikasi metode `queryTest` agar menggunakan `KeyConditionExpression` yang berbeda.
- Menambahkan `ScanFilter` ke metode `scanTest` sehingga hanya beberapa item yang dikembalikan kepada Anda.

> **Note**
>
> Untuk menjalankan program ini, Anda dapat mengatur Maven untuk menggunakan klien untuk DAX SDK for Java dan dependensi as. AWS SDK untuk Java Untuk informasi selengkapnya, lihat Menggunakan klien sebagai dependensi Apache Maven.
> Atau, Anda dapat mengunduh dan menyertakan klien DAX Java dan AWS SDK untuk Java di classpath Anda. Lihat Java dan DAX untuk contoh pengaturan variabel `CLASSPATH`.

```
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));
    }
}
```

Memodifikasi SDK untuk Java 1.x yang ada agar menggunakan DAX

Jika sudah memiliki aplikasi Java yang menggunakan Amazon DynamoDB, Anda harus memodifikasinya agar dapat mengakses klaster DynamoDB Accelerator (DAX). Anda tidak harus menulis ulang seluruh aplikasi karena klien Java DAX mirip dengan klien tingkat rendah DynamoDB yang disertakan dalam AWS SDK untuk Java.

> **Note**
>
> Instruksi ini untuk aplikasi yang menggunakan AWS SDK for Java 1.x. Untuk aplikasi yang menggunakan AWS SDK untuk Java 2.x, lihat Memodifikasi aplikasi yang ada untuk menggunakan DAX.

Anggaplah Anda memiliki tabel DynamoDB bernama `Music`. Kunci partisi untuk tabel ini adalah `Artist`, dan kunci urutannya adalah `SongTitle`. Program berikut membaca item secara langsung dari tabel `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());
        }
    }
}
```

Untuk mengubah program, ganti klien DynamoDB dengan klien DAX.

```
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)
       ** ...
       */

    }
}
```

Menggunakan API dokumen DynamoDB

AWS SDK untuk Java Ini menyediakan antarmuka dokumen untuk DynamoDB. API dokumen bertindak sebagai pembungkus klien tingkat rendah DynamoDB. Untuk informasi selengkapnya, lihat Antarmuka dokumen.

Antarmuka dokumen juga dapat digunakan dengan klien DAX tingkat rendah, seperti yang ditunjukkan dalam contoh berikut.

```
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());
        }
```

```
    }
}
```

Klien asinkron DAX

`AmazonDaxClient` bersifat sinkron. Untuk operasi API DAX yang berjalan lama, seperti `Scan` dari tabel besar, operasi ini dapat memblokir eksekusi program hingga operasi selesai. Jika program perlu melakukan pekerjaan lain saat operasi API DAX sedang berlangsung, Anda dapat menggunakan `ClusterDaxAsyncClient` sebagai gantinya.

Program berikut menunjukkan cara menggunakan `ClusterDaxAsyncClient`, bersama dengan Java `Future`, untuk menerapkan solusi non-blocking.

```
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();

 }
}
```

Mengkueri indeks sekunder global dengan SDK untuk Java 1.x

Anda dapat menggunakan Amazon DynamoDB Accelerator (DAX) untuk mengkueri indeks sekunder global menggunakan antarmuka terprogram DynamoDB.

Contoh berikut menunjukkan cara menggunakan DAX untuk menanyakan indeks sekunder `CreateDateIndex` global yang dibuat di Contoh: Indeks sekunder global menggunakan API dokumen. AWS SDK untuk Java

Kelas `DAXClient` membuat instans objek klien yang diperlukan untuk berinteraksi dengan antarmuka pemrograman DynamoDB.

```
import com.amazon.dax.client.dynamodbv2.AmazonDaxClientBuilder;
import com.amazonaws.services.dynamodbv2.datamodeling.DynamoDBMapper;
import com.amazonaws.services.dynamodbv2.document.DynamoDB;
import com.amazonaws.util.EC2MetadataUtils;
import com.amazonaws.services.dynamodbv2.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);
 }
}
```

Anda dapat mengkueri indeks sekunder global dengan cara berikut:

- Menggunakan metode `queryIndex` pada kelas `QueryIndexDax` yang ditentukan dalam contoh berikut. `QueryIndexDax` dijadikan sebagai parameter objek klien yang dikembalikan oleh metode `getDaxDocClient` pada kelas `DaxClient`.
- Jika Anda menggunakan antarmuka persistensi objek, gunakan metode `queryIndexMapper` pada kelas `QueryIndexDax` yang ditentukan dalam contoh berikut. `queryIndexMapper` dijadikan sebagai parameter objek klien yang dikembalikan oleh metode `getDaxMapperClient` pada kelas `DaxClient`.

```
import java.util.Iterator;
import com.amazonaws.services.dynamodbv2.datamodeling.DynamoDBMapper;
import java.util.List;
import com.amazonaws.services.dynamodbv2.datamodeling.DynamoDBQueryExpression;
import com.amazonaws.services.dynamodbv2.model.AttributeValue;
import java.util.HashMap;
import com.amazonaws.services.dynamodbv2.document.Item;
import com.amazonaws.services.dynamodbv2.document.utils.ValueMap;
import com.amazonaws.services.dynamodbv2.document.spec.QuerySpec;
import com.amazonaws.services.dynamodbv2.document.QueryOutcome;
import com.amazonaws.services.dynamodbv2.document.ItemCollection;
import com.amazonaws.services.dynamodbv2.document.Index;
import com.amazonaws.services.dynamodbv2.document.Table;
import com.amazonaws.services.dynamodbv2.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());
```

```
  }
 }
}
```

Definisi kelas di bawah ini merupakan tabel Masalah dan digunakan dalam metode `queryIndexMapper`.

```
import com.amazonaws.services.dynamodbv2.datamodeling.DynamoDBTable;
import com.amazonaws.services.dynamodbv2.datamodeling.DynamoDBIndexHashKey;
import com.amazonaws.services.dynamodbv2.datamodeling.DynamoDBIndexRangeKey;
import com.amazonaws.services.dynamodbv2.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 Contoh SDK for Go 1.x

Bagian ini berisi contoh kode untuk aplikasi DAX menggunakan Go 1.x.

Topik

- DAX SDK untuk Go

DAX SDK untuk Go

Ikuti prosedur ini untuk menjalankan aplikasi sampel Amazon DynamoDB Accelerator (DAX) SDK for Go di instans Amazon Anda. EC2

Cara menjalankan sampel SDK untuk Go untuk DAX

1. Siapkan SDK for Go di instans EC2 Amazon Anda:

   a. Instal bahasa pemrograman Go (`Golang`).

   ```
   sudo yum install -y golang
   ```

   b. Pastikan bahwa Golang sudah diinstal dan berjalan dengan benar.

   ```
   go version
   ```

   Pesan seperti ini akan muncul.

   ```
   go version go1.15.5 linux/amd64
   ```

   Petunjuk yang tersisa bergantung pada dukungan modul, yang merupakan default pada Go versi 1.13.

2. Instal aplikasi Golang sampel.

   ```
   go get github.com/aws-samples/aws-dax-go-sample
   ```

3. Jalankan program Golang berikut. Program pertama membuat tabel DynamoDB bernama `TryDaxGoTable`. Program kedua menulis data ke tabel.

   ```
   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. Jalankan program Golang berikut.

```
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
```

   Perhatikan informasi waktu, yaitu jumlah milidetik yang diperlukan untuk pengujian `GetItem`, `Query`, dan `Scan`.

5. Pada langkah sebelumnya, Anda menjalankan program terhadap titik akhir DynamoDB. Sekarang, jalankan program lagi, tetapi kali ini operasi `GetItem`, `Query`, dan `Scan` diproses oleh klaster DAX Anda.

   Untuk menentukan titik akhir klaster DAX Anda, pilih salah satu dari berikut ini:

   - Menggunakan konsol DynamoDB — Pilih klaster DAX Anda. Titik akhir klaster ditampilkan pada konsol, seperti dalam contoh berikut.

     ```
     dax://my-cluster.l6fzcv.dax-clusters.us-east-1.amazonaws.com
     ```

   - Menggunakan AWS CLI — Masukkan perintah berikut.

     ```
     aws dax describe-clusters --query "Clusters[*].ClusterDiscoveryEndpoint"
     ```

     Titik akhir klaster ditampilkan pada output, seperti dalam contoh berikut.

     ```
     {
         "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"
     }
     ```

Sekarang jalankan program lagi, tetapi kali ini tentukan titik akhir klaster sebagai parameter baris perintah.

```
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
```

Periksa sisa output dan perhatikan informasi waktu. Waktu berlalu untuk `GetItem`, `Query`, dan `Scan` harus jauh lebih rendah dengan DAX dibandingkan dengan DynamoDB.

6. Jalankan program Golang berikut untuk menghapus `TryDaxGoTable`.

```
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 Contoh SDK untuk Node.js 2.x

Bagian ini berisi contoh kode untuk aplikasi DAX menggunakan AWS SDK untuk Node.js 2.x.

Topik

- Node.js dan DAX

Node.js dan DAX

Ikuti langkah-langkah ini untuk menjalankan contoh aplikasi Node.js pada EC2 instans Amazon Anda.

Cara menjalankan sampel Node.js untuk DAX

1. Siapkan Node.js di EC2 instans Amazon Anda, sebagai berikut:

   a. Instal manajer versi simpul (nvm).

   ```
   curl -o- https://raw.githubusercontent.com/nvm-sh/nvm/v0.35.3/install.sh | bash
   ```

   b. Gunakan nvm untuk menginstal Node.js.

   ```
   nvm install 12.16.3
   ```

   c. Pastikan bahwa Node.js sudah diinstal dan berjalan dengan benar.

   ```
   node -e "console.log('Running Node.js ' + process.version)"
   ```

   Tindakan ini akan menampilkan pesan berikut.

   ```
   Running Node.js v12.16.3
   ```

2. Instal klien Node.js DAX menggunakan manajer paket simpul (npm).

   ```
   npm install amazon-dax-client
   ```

3. Unduh sampel kode sumber program (file `.zip`).

   ```
   wget http://docs.aws.amazon.com/amazondynamodb/latest/developerguide/samples/
   TryDax.zip
   ```

   Setelah unduhan selesai, ekstrak file sumber.

   ```
   unzip TryDax.zip
   ```

4. Jalankan program Node.js berikut. Program pertama membuat tabel Amazon DynamoDB bernama `TryDaxTable`. Program kedua menulis data ke tabel.

   ```
   node 01-create-table.js
   node 02-write-data.js
   ```

5. Jalankan program Node.js berikut.

```
node 03-getitem-test.js
node 04-query-test.js
node 05-scan-test.js
```

Perhatikan informasi waktu, yaitu jumlah milidetik yang diperlukan untuk pengujian `GetItem`, `Query`, dan `Scan`.

6. Pada langkah sebelumnya, Anda menjalankan program terhadap titik akhir DynamoDB. Jalankan program lagi, tetapi kali ini, operasi `GetItem`, `Query` dan `Scan` diproses oleh klaster DAX Anda.

   Untuk menentukan titik akhir klaster DAX Anda, pilih salah satu dari berikut ini.

   - Menggunakan konsol DynamoDB—Pilih klaster DAX Anda. Titik akhir klaster ditampilkan pada konsol, seperti dalam contoh berikut.

     ```
     dax://my-cluster.l6fzcv.dax-clusters.us-east-1.amazonaws.com
     ```

   - Menggunakan AWS CLI —Masukkan perintah berikut.

     ```
     aws dax describe-clusters --query "Clusters[*].ClusterDiscoveryEndpoint"
     ```

     Titik akhir klaster ditampilkan pada output, seperti dalam contoh berikut.

     ```
     {
         "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"
     }
     ```

   Sekarang jalankan program lagi, tetapi kali ini tentukan titik akhir klaster sebagai parameter baris perintah.

   ```
   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
   ```

Periksa sisa output dan perhatikan informasi waktu. Waktu berlalu untuk `GetItem`, `Query`, dan `Scan` harus jauh lebih rendah dengan DAX dibandingkan dengan DynamoDB.

7. Jalankan program Node.js berikut untuk menghapus `TryDaxTable`.

```
node 06-delete-table
```

Untuk informasi selengkapnya tentang program tersebut, lihat bagian berikut:

- 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

Program `01-create-table.js` membuat tabel (`TryDaxTable`). Program Node.js yang tersisa di bagian ini bergantung pada tabel ini.

```
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

Program `02-write-data.js` menulis data pengujian ke `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

Program `03-getitem-test.js` melakukan operasi `GetItem` di `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 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

Program `04-query-test.js` melakukan operasi `Query` di `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 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

Program `05-scan-test.js` melakukan operasi `Scan` di `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 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

Program `06-delete-table.js` menghapus `TryDaxTable`. Jalankan program ini setelah Anda selesai melakukan pengujian.

```
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)
    );
  }
});
```

Riwayat dokumen untuk DynamoDB

Tabel berikut menjelaskan perubahan penting dalam setiap rilis Panduan Developer DynamoDB mulai tanggal 3 Juli 2018 dan seterusnya. Untuk notifikasi tentang pembaruan dokumentasi ini, Anda dapat berlangganan ke umpan RSS (di sudut kiri atas halaman ini).

Perubahan	Deskripsi	Tanggal
Amazon DynamoDB Streams Perpustakaan Klien Kinesis (KCL) 2.x adaptor	DynamoDB sekarang mendukung adaptor DynamoDB Streams yang ditingkatkan yang kompatibel dengan KCL 2.x dan SDK v2. AWS Untuk informasi selengkapnya, lihat Migrasi KCL1 dari.x ke KCL 3.x.	Juni 11, 2025
Ditambahkan dukungan untuk menjalankan DynamoDB lokal di AWS CloudShell	Anda sekarang dapat menjalankan DynamoDB lokal langsung di browser Anda melalui tanpa mengunduh atau AWS Management Console menginstal perangkat lunak. Untuk informasi selengkapnya, lihat Menerapkan DynamoDB secara lokal di komputer Anda.	19 Mei 2025
Menambahkan dukungan untuk tipe instans r7i di DAX	Opsi tipe instans yang diperluas untuk DAX untuk memberikan lebih banyak fleksibilitas dalam kinerja dan kapasitas. Untuk detail selengkapnya tentang jenis instans r7i baru dan kemampuannya, lihat enkripsi DAX saat istirahat.	April 29, 2025

Kebijakan akses penuh tercakup bawah baru menggantikan. `AmazonDynamoDBFullAccess_v2` `AmazonDynamoDBFullAccess`	Efektif 25 April 2025, DynamoDB telah menghentikan kebijakan tersebut dan menggantinya dengan kebijakan `AmazonDynamoDBFullAccess` cakupan bawah bernama. `AmazonDynamoDBFullAccess_v2` Untuk informasi selengkapnya, lihat kebijakan AWS terkelola : AmazonDynamo DBFull Access_v2.	April 28, 2025
DynamoDB PrivateLink memperkenalkan dukungan untuk DynamoDB Streams	AWS menambahkan PrivateLink dukungan untuk DynamoDB Streams, memungkinkan pelanggan untuk memanggil DynamoDB Streams APIs dari dalam Amazon Virtual Private Cloud (VPC) menggunakan konektivitas jaringan pribadi. Fitur ini memungkinkan akses jaringan pribadi yang disederhanakan tanpa melintasi internet publik, mendukung kepatuhan dan persyaratan keamanan untuk beban kerja DynamoDB. Untuk informasi selengkapnya, lihat AWS PrivateLink Amazon DynamoDB Streams.	Maret 26, 2025

DAX SDK Untuk JS 3.x dan Go 2.x sekarang tersedia	DynamoDB Accelerator (DAX) SDK untuk JS 3.x sekarang tersedia dan kompatibel dengan SDK for Java 3.x. AWS	Maret 17, 2025
NoSQL Workbench 3.13.5 mode kapasitas sesuai permintaan untuk rilis pengaturan tabel default	Saat Anda membuat tabel dengan pengaturan default, DynamoDB membuat tabel yang menggunakan mode kapasitas sesuai permintaan, bukan mode kapasitas yang disediakan.	Februari 24, 2025
Praktik terbaik baru untuk mengonfigurasi klien DAX Anda	Menerbitkan topik praktik terbaik panduan preskriptif DAX baru untuk mengonfig urasi klien DAX Anda. Untuk informasi selengkapnya, lihat Mengonfigurasi klien DAX Anda.	Februari 17, 2025
Praktik terbaik baru untuk operasi data massal	Menerbitkan topik praktik terbaik baru untuk bekerja dengan model data yang kompleks di DynamoDB. Untuk informasi selengkap nya, lihat Praktik terbaik untuk menggunakan operasi data massal di DynamoDB.	Januari 9, 2025

Amazon DynamoDB sekarang mendukung pemulihan yang dapat point-in-time dikonfigu rasi (PITR)	DynamoDB sekarang mendukung periode pemulihan yang dapat dikonfigurasi untuk PITR. Anda sekarang dapat mengatur periode PITR untuk setiap tabel menjadi antara 1 dan 35 hari. Untuk informasi selengkapnya, lihat Point-in-time backup untuk DynamoDB.	Januari 7, 2025
Amazon DynamoDB sekarang mendukung pemulihan yang dapat point-in-time dikonfigu rasi (PITR)	DynamoDB sekarang mendukung periode pemulihan yang dapat dikonfigurasi untuk PITR. Anda sekarang dapat mengatur periode PITR untuk setiap tabel menjadi antara 1 dan 35 hari. Untuk informasi selengkapnya, lihat Point-in-time backup untuk DynamoDB.	Januari 7, 2025
Menerbitkan topik baru tentang mengintegrasikan Amazon Managed Streaming untuk Apache Kafka Kafka dengan Amazon DynamoDB	Pelajari bagaimana Amazon Managed Streaming for Apache Kafka terintegrasi dengan Amazon DynamoDB dengan membaca data dari topik Apache Kafka dan menyimpannya di DynamoDB. Untuk informasi selengkap nya, lihat Mengintegrasikan DynamoDB dengan Amazon Managed Streaming for Apache Kafka.	Desember 26, 2024

Menerbitkan topik baru tentang mengintegrasikan Amazon Managed Streaming untuk Apache Kafka Kafka dengan Amazon DynamoDB	Pelajari bagaimana Amazon Managed Streaming for Apache Kafka terintegrasi dengan Amazon DynamoDB dengan membaca data dari topik Apache Kafka dan menyimpannya di DynamoDB. Untuk informasi selengkap nya, lihat Mengintegrasikan DynamoDB dengan Amazon Managed Streaming for Apache Kafka.	Desember 26, 2024
DynamoDB memperkenalkan dukungan untuk integrasi nol-ETL dengan AI Lakehouse SageMaker	DynamoDB memperken alkan integrasi nol-ETL yang mengotomatiskan pengekstr akan dan pemuatan data dari Amazon DynamoDB ke dalam data lake pelanggan. Untuk informasi selengkapnya, lihat Integrasi DynamoDB Zero-ETL dengan AI Lakehouse. SageMaker	Desember 3, 2024
DynamoDB memperkenalkan dukungan untuk integrasi nol-ETL dengan Amazon Lakehouse SageMaker	DynamoDB memperken alkan integrasi nol-ETL yang mengotomatiskan pengekstr akan dan pemuatan data dari Amazon DynamoDB ke dalam data lake pelanggan. Untuk informasi selengkapnya, lihat Integrasi DynamoDB Zero-ETL dengan Amazon Lakehouse. SageMaker	Desember 3, 2024

Tabel global DynamoDB sekarang mendukung konsistensi kuat Multi-region	Dengan konsistensi Multi-region yang kuat, Anda dapat membangun aplikasi Multi-region yang sangat tersedia dengan tujuan titik pemulihan (RPO) nol, mencapai tingkat ketahanan tertinggi. Untuk informasi selengkapnya, lihat Tabel global dengan konsistensi kuat Multi-wilayah.	Desember 3, 2024
Tabel global DynamoDB sekarang mendukung konsistensi kuat Multi-region	Dengan konsistensi Multi-region yang kuat, Anda dapat membangun aplikasi Multi-region yang sangat tersedia dengan tujuan titik pemulihan (RPO) nol, mencapai tingkat ketahanan tertinggi. Untuk informasi selengkapnya, lihat Tabel global dengan konsistensi kuat Multi-wilayah.	Desember 3, 2024

Pembaruan kebijakan terkelola DynamoDB	Menambahkan dua izin baru ke kebijakan `AmazonDynamoDBReadOnlyAccess` terkelola: `dynamodb:GetAbacStatus` dan `dynamodb:UpdateAbacStatus`. Izin ini memungkinkan Anda untuk melihat status ABAC dan mengaktifkan ABAC untuk Wilayah Anda Akun AWS saat ini. Untuk informasi selengkapnya, lihat kebijakan AWS terkelola: AmazonDynamo DBRead OnlyAccess.	November 18, 2024
DynamoDB memperkenalkan dukungan untuk kontrol akses berbasis atribut (ABAC)	ABAC adalah strategi otorisasi yang memungkinkan Anda menentukan izin akses berdasarkan tag yang dilampirkan ke pengguna, peran, dan sumber daya. AWS ABAC menggunakan kondisi berbasis tag dalam kebijakan AWS Identity and Access Management (IAM) Anda atau kebijakan lain untuk mengizinkan atau menolak tindakan tertentu pada tabel atau indeks Anda ketika tag prinsipal IAM cocok dengan tag untuk tabel. Untuk informasi selengkapnya, lihat Menggunakan kontrol akses berbasis atribut dengan DynamoDB.	November 18, 2024

DynamoDB memperkenalkan throughput hangat untuk tabel sesuai permintaan dan disediakan	DynamoDB sekarang mendukung throughput hangat. Hangat throughput memberikan visibilitas ke dalam jumlah operasi baca dan tulis tabel DynamoDB Anda dapat secara instan mendukung serta kemampuan untuk pra-pemanasan tabel DynamoDB Anda. Untuk informasi selengkapnya, lihat DynamoDB warm throughput.	November 13, 2024
Kemampuan untuk mengkonsumsi catatan Amazon DynamoDB Streams dengan Apache Flink	Anda sekarang dapat menggunakan data Amazon DynamoDB Streams dengan Apache Flink dan memanfaatkan Amazon Managed Service untuk Apache Flink untuk membuat dan mengelola aplikasi pemrosesan streaming dengan cepat. end-to-end Untuk informasi selengkapnya, lihat DynamoDB Streams dan Apache Flink.	November 12, 2024

Menerbitkan topik penagihan baru untuk tabel dan cadangan global	Menerbitkan dua topik baru mengenai penagihan untuk tabel global dan penagihan untuk cadangan. Untuk informasi selengkapnya, lihat Memahami penagihan Amazon DynamoDB untuk tabel global dan Memahami penagihan Amazon DynamoDB untuk pencadangan.	Oktober 16, 2024
Integrasi Amazon DynamoDB Zero-ETL dengan Amazon Redshift	Integrasi Amazon DynamoDB Zero-ETL dengan Amazon Redshift menyediakan pipeline ETL tanpa kode yang dikelola sepenuhnya dengan replikasi dari DynamoDB ke Amazon Redshift. Untuk informasi selengkapnya, lihat Amazon DynamoDB integrasi nol-ETL dengan Amazon Redshift.	Oktober 15, 2024
Pembaruan khusus dokumentasi untuk menambahkan topik tentang penggunaan AI generatif dengan DynamoDB	Menerbitkan topik baru yang memberikan informasi tentang penggunaan AI generatif dengan DynamoDB, termasuk contoh kasus penggunaan AI gen untuk DynamoDB. Untuk informasi selengkapnya, lihat Menggunakan AI generatif dengan DynamoDB.	Oktober 11, 2024

SDK sekarang mendukung titik akhir berbasis AWS akun	Menambahkan dokumentasi untuk endpoint berbasis akun dan `ACCOUNT_ID_ENDPOINT_MODE` pengaturan untuk klien SDK. Untuk informasi selengkapnya, lihat Dukungan SDK untuk endpoint AWS berbasis akun.	September 3, 2024
Mendesain ulang pengalaman memulai	Mendesain ulang pengalaman memulai untuk mengkonsolidasikan informasi dan membantu Anda memulai dengan orientasi yang lebih cepat. Untuk informasi selengkapnya, lihat Memulai DynamoDB.	Agustus 1, 2024
Ekspansi DAX ke Wilayah baru untuk Spanyol dan Swedia	DAX sekarang tersedia di Wilayah Spanyol dan Swedia. Untuk informasi selengkapnya, lihat komponen cluster DAX.	Juli 30, 2024
Merestrukturisasi dan mengkonsolidasikan dokumentasi backup dan restore DynamoDB	Panduan Pengembangan DynamoDB memiliki struktur baru untuk membuat cadangan dan pemulihan. Untuk informasi selengkapnya, lihat Backup dan restore untuk DynamoDB.	Juli 2, 2024
Apa itu Amazon DynamoDB? menulis ulang topik	Menerbitkan versi yang direvisi dan diperbarui dari Apa itu Amazon DynamoDB? topik. Untuk informasi selengkapnya, lihat Apa itu Amazon DynamoDB? .	Juni 21, 2024

Integrasikan DynamoDB Streams dengan EventBridge	Menerbitkan topik baru tentang mengintegrasikan DynamoDB Streams dengan. EventBridge Untuk informasi selengkapnya, lihat Menginteg rasikan dengan EventBridge.	Juni 21, 2024
Panduan preskriptif DAX	Menerbitkan topik praktik terbaik baru yang memberi Anda wawasan komprehen sif untuk menggunakan DynamoDB Accelerator secara efektif. Topik ini mencakup optimasi kinerja, manajemen biaya, dan praktik terbaik operasional. Untuk informasi selengkapnya, lihat panduan preskriptif DAX.	3 Juni 2024
Migrasi tabel DynamoDB dari satu akun ke akun lainnya	Menambahkan topik baru tentang memigrasi tabel DynamoDB dari satu akun ke akun lainnya. Untuk informasi selengkapnya, lihat Memigrasi tabel DynamoDB dari satu akun ke akun lainnya.	29 Mei 2024
Merestrukturisasi dan mengkonsolidasikan dokumentasi pemantauan dan pencatatan DynamoDB	Struktur baru untuk pemantauan dan pencatatan di DynamoDB mencakup tiga chapter singkat untuk metrik, operasi logging, dan wawasan kontributor.	3 Mei 2024

Merestrukturisasi dan mengkonsolidasikan dokumentasi mode kapasitas DynamoDB	Panduan DynamoDB sekarang mencakup babak baru yang berisi semua informasi tentang mode kapasitas DynamoDB - sesuai permintaan dan disediakan. Dengan pembaruan ini, topik Pertimbangan saat mengubah Mode read/write Kapasitas telah dipindahkan ke dalam chapter Praktik terbaik. Topik ini sekarang diubah namanya menjadi Pertimbangan saat mengganti mode kapasitas dan mencakup informasi rumit tentang praktik terbaik saat beralih di antara mode kapasitas. Selain itu, panduan ini sekarang menampilkan babak baru yang mencakup semua informasi tentang membaca dan menulis DynamoDB, dan konsumsi unit kapasitas untuk operasi baca dan tulis. Untuk informasi selengkapnya, lihat Kapasitas throughput DynamoDB, Pertimbangan saat beralih mode kapasitas, dan DynamoDB membaca dan menulis.	1 Mei 2024

Jumlah maksimum permintaan sesuai permintaan	Sekarang Anda dapat menentukan jumlah maksimum permintaan sesuai permintaan yang dapat dilakukan oleh tabel individual, indeks, atau keduanya. Menentukan throughput sesuai permintaan maksimum akan membantu menjaga penggunaan dan biaya tingkat tabel Anda tetap terbatas dan melindungi dari lonjakan sumber daya yang dikonsumsi secara tidak sengaja. Untuk informasi selengkapnya, lihat Throughput maksimum untuk tabel sesuai permintaan.	1 Mei 2024
Peningkatan pembuat operasi NoSQL Workbench	NoSQL Workbench sekarang menyertakan dukungan asli untuk mode gelap. Operasi tabel dan item yang ditingkatkan di pembuat operasi. Hasil item dan informasi permintaan pembuat operasi tersedia dalam format JSON. Untuk informasi selengkapnya, lihat Pembuat operasi NoSQL Workbench.	April 24, 2024

Kebijakan berbasis sumber daya untuk sumber daya Amazon DynamoDB	DynamoDB sekarang mendukung kebijakan berbasis sumber daya untuk tabel, indeks, dan aliran. Kebijakan berbasis sumber daya memungkinkan Anda menentukan izin akses dengan menentukan siapa yang memiliki akses ke setiap sumber daya, dan tindakan yang diizinkan untuk dilakukan pada setiap sumber daya. Untuk informasi selengkapnya, lihat Menggunakan kebijakan berbasis sumber daya untuk DynamoDB.	Maret 20, 2024
Pembaruan kebijakan terkelola DynamoDB	Menambahkan izin baru `dynamodb:GetResourcePolicy` ke kebijakan `AmazonDynamoDBReadOnlyAccess` terkelola. Izin ini menyediakan akses ke kebijakan berbasis sumber daya baca yang dilampirkan ke sumber daya DynamoDB. Untuk informasi selengkapnya, lihat kebijakan AWS terkelola : AmazonDynamo DBRead OnlyAccess.	Maret 20, 2024

AWS PrivateLink untuk Amazon DynamoDB	Amazon DynamoDB sekarang mendukung. AWS PrivateLink Dengan AWS PrivateLink, Anda dapat menyederhanakan konektivitas jaringan pribadi antara virtual private cloud (VPCs), DynamoDB, dan pusat data lokal Anda menggunakan titik akhir VPC antarmuka dan alamat IP pribadi. Untuk informasi selengkapnya, lihat AWS PrivateLink DynamoDB.	Maret 19, 2024
Pemrograman dengan JavaScript panduan	Amazon DynamoDB menyajikan panduan pemrograman untuk. AWS SDK untuk JavaScript Pelajari tentang, lapisan abstraksi AWS SDK untuk JavaScript, mengonfigurasi koneksi, menangani kesalahan, mendefinisikan kebijakan coba lagi, mengelola keep-alive, dan banyak lagi. Untuk informasi lebih lanjut, lihat Pemrograman dengan JavaScript.	Maret 6, 2024

Pemrograman dengan AWS SDK for Java 2.x panduan	Membuat panduan pemrogram an baru yang membahas secara mendalam tentang antarmuka tingkat tinggi, tingkat rendah, dan dokumen, klien HTTP dan konfigura sinya, penanganan kesalahan , dan alamat pengaturan konfigurasi paling umum yang harus Anda pertimban gkan saat menggunakan SDK for Java 2.x. Untuk informasi selengkapnya, lihat Memprogram Amazon AWS SDK for Java 2.x DynamoDB dengan.	Maret 5, 2024
Kloning tabel dengan NoSQL Workbench	Izinkan pengembang menggunakan NoSQL Workbench untuk menyalin atau mengkloning tabel antara lingkungan pengembangan dan wilayah (DynamoDB Local dan DynamoDB web). Untuk informasi selengkapnya, lihat Mengkloning tabel dengan NoSQL Workbench.	Februari 26, 2024

Pemrograman dengan panduan Python	Membuat panduan baru yang membahas secara mendalam tentang pustaka tingkat tinggi dan tingkat rendah dan membahas pengaturan konfigurasi paling umum yang harus dipertimbangkan saat menggunakan Python SDK. Untuk informasi selengkapnya, lihat Pemrograman dengan Python.	Januari 5, 2024
Tulis ulang topik Time to Live (TTL)	Sepenuhnya menulis ulang bagian TTL dari panduan ini. Panduan baru ini membantu Anda memulai TTL dengan memberikan cuplikan ready-to-use kode di sepanjang jalan. Cuplikan kode saat ini disediakan dalam Python dan Javascript. Untuk informasi lebih lanjut, lihat TTL.	20 Desember 2023
Praktik Terbaik untuk Memahami Laporan AWS Penagihan dan Penggunaan Anda	Menambahkan bagian baru yang menjelaskan berbagai jenis penggunaan dan biaya untuk jenis penggunaan tersebut di DynamoDB. Untuk informasi selengkapnya, lihat Laporan penagihan dan penggunaan.	15 Desember 2023

Integrasi Amazon DynamoDB Zero-ETL dengan Amazon Service OpenSearch	Amazon DynamoDB sekarang mendukung integrasi nol-ETL dengan OpenSearch Amazon Service, yang memungkinkan Anda melakukan pencarian pada data DynamoDB Anda dengan secara otomatis mereplikasi dan mengubahn ya tanpa kode atau infrastru ktur khusus. Untuk informasi selengkapnya, lihat Integrasi DynamoDB Zero-ETL dengan Amazon Service. OpenSearch	28 November 2023
Migrasi ke DynamoDB dari database relasional	Membuat panduan migrasi untuk membantu pengguna memahami dan cara bermigrasi ke DynamoDB dari database relasional.	27 November 2023
Menghasilkan data sampel dengan NoSQL Workbench	NoSQL Workbench untuk Amazon DynamoDB sekarang mendukung pembuatan model data langsung dari sampel templat model data untuk membantu Anda merancang skema data untuk beban kerja Anda. Anda dapat menggunak an fitur ini untuk membiasak an diri dengan praktik terbaik pemodelan data NoSQL saat membangun aplikasi di DynamoDB.	28 September 2023

Ekspor Tambahan ke S3	Anda sekarang dapat mengekspor data yang dimasukkan, diperbarui, atau dihapus, sedikit demi sedikit. Dengan ekspor tambahan, Anda dapat mengekspor data yang diubah mulai dari beberapa megabyte hingga terabyte dengan beberapa klik di AWS Management Console, panggilan API, atau Antarmuka Baris Perintah. AWS	26 September 2023
Pemodelan data untuk DynamoDB	Anda sekarang dapat mempelajari lebih lanjut tentang pemodelan data dengan contoh DynamoDB yang berfokus pada kasus penggunaan tertentu, pola aksesnya, step-by-step dan panduan dalam mewujudkan pola akses tersebut.	14 Juli 2023
Bagian pemecahan masalah	Sekarang Anda dapat menemukan konten pemecahan masalah untuk masalah latensi dan throttling yang mungkin terjadi di tabel DynamoDB Anda.	13 Maret 2023

Perlindungan penghapusan untuk Amazon DynamoDB	Perlindungan penghapusan sekarang tersedia untuk tabel Amazon DynamoDB di semua Wilayah AWS . DynamoDB kini memungkinkan Anda melindungi tabel Anda dari penghapusan yang tidak disengaja saat melakukan operasi manajemen tabel reguler.	8 Maret 2023
AWS CloudFormation dukungan untuk KDSD dalam tabel global	Amazon Kinesis Data Streams untuk DynamoDB AWS CloudFormation sekarang mendukung tabel global DynamoDB, yang berarti Anda dapat mengaktifkan streaming ke Amazon Kinesis Data Streams di tabel global DynamoDB Anda dengan templat. CloudFormation	15 Februari 2023
DynamoDB lokal mendukung 100 tindakan per transaksi	Anda sekarang dapat melakukan hingga 100 tindakan dalam satu transaksi di DynamoDB lokal.	9 Februari 2023
Menggunakan DynamoDB Well-Architected Lens untuk mengoptimalkan beban kerja DynamoDB Anda	Anda sekarang dapat menggunakan DynamoDB Well-Architected Lens, kumpulan prinsip dan panduan desain yang dapat Anda gunakan untuk merancang beban kerja DynamoDB yang dirancang dengan baik.	3 Februari 2023

Ketersediaan PartiQL GovGloud	PartiQL—Bahasa kueri yang kompatibel dengan SQL untuk Amazon DynamoDB sekarang didukung di (AS-Timur) dan (AS-Barat). AWS GovCloud AWS GovCloud	21 Desember 2022
Rangkaian instalasi tunggal untuk NoSQL Workbench dan DynamoDB lokal	NoSQL Workbench untuk DynamoDB sekarang menyertakan proses instalasi DynamoDB lokal terpandu untuk menyederhanakan pengaturan lingkungan pengembangan DynamoDB lokal Anda.	6 Desember 2022
Impor massal dari S3	Amazon DynamoDB sekarang memudahkan Anda untuk bermigrasi dan memuat data ke tabel DynamoDB baru dengan mendukung impor data massal dari Amazon S3.	18 Agustus 2022
Integrasi yang ditingkatkan dengan Kuota Layanan	Kuota Layanan sekarang memungkinkan Anda untuk mengelola kuota tabel dan akun Anda secara proaktif. Anda dapat melihat nilai saat ini, mengatur alarm ketika penggunaan kuota melebihi ambang batas yang dapat dikonfigurasi, dan banyak lagi.	15 Juni 2022

NoSQL Workbench menambahkan dukungan GSI dan tabel	Anda sekarang dapat menggunakan NoSQL Workbench untuk tabel dan operasi bidang kontrol indeks sekunder global (GSI) seperti CreateTable,, dan. UpdateTable DeleteTable	2 Juni 2022
Kelas tabel Standard-infrequent Access kini tersedia di Tiongkok	Kelas tabel Amazon DynamoDB Standard-Infrequent Access tersedia di Wilayah Tiongkok. Kurangi biaya DynamoDB Anda hingga 60 persen, dengan menggunakan kelas tabel baru ini untuk tabel yang menyimpan data yang jarang diakses.	18 April 2022
Peningkatan kuota layanan default dan operasi manajemen tabel	DynamoDB meningkatkan kuota default untuk jumlah tabel per akun dan Wilayah dari 256 menjadi 2.500 tabel, dan meningkatkan jumlah operasi manajemen tabel bersamaan dari 50 menjadi 500.	9 Maret 2022
Pembatasan item opsional dengan PartiQL untuk DynamoDB	DynamoDB dapat membatasi jumlah item yang diproses di PartiQL untuk operasi DynamoDB sebagai parameter opsional pada setiap permintaan.	8 Maret 2022

AWS Backup integrasi tersedia di Wilayah Tiongkok (Beijing dan Ningxia)	AWS Backup sekarang terintegrasi dengan DynamoDB di Wilayah Tiongkok (Beijing dan Ningxia). Anda dapat memenuhi persyaratan kepatuhan dan kelangsungan bisnis dengan lebih mudah melalui fitur pencadangan yang disempurnakan AWS Backup, seperti pencadangan lintas akun dan lintas wilayah.	26 Januari 2022
Informasi kapasitas throughput melalui panggilan API PartiQL	DynamoDB dapat mengembalikan kapasitas throughput yang digunakan oleh panggilan API PartiQL untuk membantu Anda mengoptimalkan kueri dan biaya throughput.	18 Januari 2022
AWS Backup integrasi	DynamoDB kini membantu Anda memenuhi persyaratan kepatuhan dan kelangsungan bisnis dengan lebih mudah melalui fitur pencadangan yang ditingkatkan di AWS Backup, seperti pencadangan lintas akun dan lintas Wilayah.	24 November 2021

Kumpulan data NoSQL Workbench import/export di CSV	NoSQL Workbench untuk Amazon DynamoDB sekarang memungkinkan Anda untuk mengimpor dan secara otomatis mengisi data sampel untuk membantu membangun dan memvisualisasikan model data Anda.	11 Oktober 2021
Filter dan ambil aktivitas bidang data Amazon DynamoDB Streams dengan AWS CloudTrail	Amazon DynamoDB kini memberi Anda kontrol pencatatan audit yang lebih mendetail dengan memungkinkan Anda memfilter aktivitas API bidang data Streams di AWS CloudTrail.	22 September 2021
Pembaruan konsol	Konsol DynamoDB kini menjadi konsol default Anda untuk membantu mengelola data dengan lebih mudah, menyederhanakan tugas umum, dan memberi Anda akses lebih cepat ke sumber daya dan fitur.	25 Agustus 2021
DAX SDK For Java 2.x sekarang tersedia	DynamoDB Accelerator (DAX) SDK for Java 2.x sekarang tersedia dan kompatibel dengan SDK for Java 2.x. AWS Anda bisa mendapatkan keuntungan dari fitur-fitur terbaru, termasuk non-blocking I/O.	29 Juli 2021

Pembaruan fitur NoSQL Workbench mencakup operasi bidang kontrol	NoSQL Workbench untuk Amazon DynamoDB sekarang membantu Anda menjalankan operasi yang sering dijalankan dengan lebih mudah untuk mengubah dan mengakses data tabel.	28 Juli 2021
Tabel global DynamoDB kini tersedia di Wilayah Asia Pasifik	Tabel global DynamoDB kini tersedia di Wilayah Asia Pasifik (Osaka). Mereplikasi tabel DynamoDB Anda secara otomatis di 22 Wilayah AWS pilihan Anda.	28 Juli 2021
DAX kini tersedia di Tiongkok	DynamoDB Accelerator (DAX) kini tersedia di Wilayah Tiongkok (Beijing), yang dioperasikan oleh Sinnet.	28 Juli 2021
Enkripsi bergerak DAX	DynamoDB Accelerator (DAX) sekarang mendukung enkripsi bergerak untuk data antara aplikasi Anda dan klaster DAX, dan di antara simpul dalam klaster DAX.	24 Juli 2021
CloudFormation dan CloudTrail integrasi	Integrasi dengan AWS CloudFormation dan peningkatan keamanan dengan CloudFormation data-plane logging.	18 Juni 2021

CloudFormation sekarang didukung untuk tabel global	Tabel global Amazon DynamoDB sekarang AWS CloudFormationmendukung , yang berarti Anda dapat membuat tabel global dan mengelola pengaturannya dengan templat. CloudFormation	14 Mei 2021
Dukungan Amazon DynamoDB lokal untuk Java 2.x	Anda sekarang dapat menggunakan AWS SDK untuk Java 2.x dengan DynamoDB lokal, versi Amazon DynamoDB yang dapat diunduh. Dengan DynamoDB lokal, Anda dapat mengembangkan dan menguji aplikasi menggunakan versi DynamoDB yang berjalan di lingkungan pengembangan lokal Anda tanpa menimbulkan biaya tambahan.	3 Mei 2021
NoSQL Workbench sekarang mendukung AWS CloudFormation	NoSQL Workbench untuk Amazon DynamoDB AWS CloudFormationsekarang mendukung, sehingga Anda dapat mengelola dan memodifikasi model data DynamoDB dengan template. CloudFormation Selain itu, Anda sekarang dapat mengonfigurasi pengaturan kapasitas tabel di NoSQL Workbench.	22 April 2021

DynamoDB dan sekarang integrasi fitur AWS Amplify	AWS Amplify sekarang mengorkestrasi beberapa pembaruan indeks sekunder global DynamoDB dalam satu deployment.	20 April 2021
AWS CloudTrail untuk mencatat bidang data Amazon DynamoDB Streams APIs	Anda sekarang dapat menggunakan AWS CloudTrail untuk mencatat aktivitas API bidang data Amazon DynamoDB Streams, dan memantau dan menyelidiki perubahan tingkat item di tabel DynamoDB Anda.	20 April 2021
Amazon Kinesis Data Streams untuk Amazon DynamoDB sekarang mendukung AWS CloudFormation	Amazon Kinesis Data Streams untuk Amazon DynamoDB AWS CloudFormation sekarang mendukung, yang berarti Anda dapat mengaktif kan streaming ke aliran data Amazon Kinesis di tabel DynamoDB Anda dengan templat. CloudFormation Dengan streaming perubahan data DynamoDB Anda ke aliran data Kinesis, Anda dapat membangun aplikasi streaming lanjutan dengan layanan Kinesis. AAmazon	12 April 2021

Amazon Keyspaces sekarang menawarkan titik akhir yang sesuai dengan FIPS 140-2	Amazon Keyspaces (untuk Apache Cassandra) sekarang menawarkan titik akhir yang sesuai dengan Federal Information Processing Standards (FIPS) 140-2 untuk membantu Anda menjalankan beban kerja yang teregulasi dengan lebih mudah. FIPS 140-2 adalah standar pemerintah AS dan Kanada yang menentukan persyaratan keamanan untuk modul kriptografi yang melindungi informasi sensitif.	8 April 2021
Instans Amazon EC2 T3 untuk DAX	DAX sekarang mendukung jenis instans Amazon EC2 T3, yang memberikan tingkat kinerja CPU dasar dengan kemampuan untuk meledak di atas garis dasar saat diperlukan.	15 Februari 2021
Dukungan NoSQL Workbench untuk Amazon DynamoDB untuk PartiQL	Anda sekarang dapat menggunakan NoSQL Workbench untuk DynamoDB untuk membuat pernyataan PartiQL untuk DynamoDB.	4 Desember 2020

PartiQL untuk DynamoDB	Sekarang Anda dapat menggunakan PartiQL untuk DynamoDB —bahasa kueri yang kompatibel dengan SQL—untuk berinteraksi dengan tabel DynamoDB dan menjalankan kueri ad hoc dengan menggunakan,, dan DynamoDB untuk PartiQL. AWS Management Console AWS Command Line Interface APIs	23 November 2020
Amazon Kinesis Data Streams untuk Amazon DynamoDB	Anda sekarang dapat menggunakan Amazon Kinesis Data Streams untuk Amazon DynamoDB dengan tabel DynamoDB Anda untuk menangkap perubahan tingkat item dan mereplikasinya ke aliran data Kinesis.	23 November 2020
Ekspor tabel DynamoDB	Sekarang Anda dapat mengekspor tabel DynamoDB Anda ke Amazon S3, memungkinkan Anda melakukan analitik dan kueri kompleks pada data Anda dengan layanan seperti Athena,, dan Lake Formation. AWS Glue	9 November 2020

Dukungan untuk nilai kosong	DynamoDB sekarang mendukung nilai kosong untuk String non kunci dan atribut Binari di tabel DynamoDB. Dukungan nilai kosong memberi Anda fleksibilitas lebih dalam menggunak an atribut untuk serangkai an kasus penggunaan yang lebih luas tanpa harus mengubah atribut tersebut sebelum mengirimkannya ke DynamoDB. Jenis Data set, Daftar, dan Peta juga mendukung String kosong dan nilai-nilai Biner.	18 Mei 2020
Dukungan NoSQL Workbench untuk Amazon DynamoDB untuk Linux	NoSQL Workbench untuk Amazon DynamoDB sekarang didukung di Linux- Ubuntu, Fedora, and Debian.	4 Mei 2020
CloudWatch Wawasan Kontributor untuk DynamoDB - GA	CloudWatchWawasan Kontributor untuk DynamoDB umumnya tersedia. CloudWatc h Contributor Insights for DynamoDB adalah alat diagnostik yang memberika n tampilan at-a-glance tren lalu lintas tabel DynamoDB Anda dan membantu Anda mengidentifikasi kunci tabel yang paling sering diakses (juga dikenal sebagai tombol pintas).	2 April 2020

Meningkatkan tabel global	Tabel global Anda sekarang dapat diperbarui dari versi 2017.11.29 ke tabel global versi terbaru (2019.11.21), dengan beberapa klik di Konsol DynamoDB. Dengan memutakhirkan versi tabel global Anda, Anda dapat meningkatkan ketersedi aan tabel DynamoDB Anda dengan mudah dengan memperluas tabel yang ada ke Wilayah AWS tambahan, tanpa perlu membangun kembali tabel.	16 Maret 2020
NoSQL Workbench untuk Amazon DynamoDB – GA	NoSQL Workbench untuk Amazon DynamoDB tersedia secara umum. Gunakan NoSQL Workbench untuk merancang, membuat, mengkueri, dan mengelola tabel DynamoDB.	2 Maret 2020
Metrik klaster cache DAX	Dukungan DAX untuk CloudWatch metrik baru, yang memungkinkan Anda untuk lebih memahami kinerja klaster DAX Anda.	6 Februari 2020

CloudWatch Wawasan Kontributor untuk DynamoDB - Pratinjau	CloudWatchContributor Insights for DynamoDB adalah alat diagnostik yang memberikan tampilan at-a-glance tren lalu lintas tabel DynamoDB Anda dan membantu Anda mengident ifikasi kunci tabel yang paling sering diakses (juga dikenal sebagai tombol pintas).	26 November 2019
Dukungan kapasitas adaptif untuk beban kerja yang tidak seimbang	Kapasitas adaptif Amazon DynamoDB kini menangani beban kerja yang tidak seimbang dengan lebih baik, dengan mengisolasi item yang sering diakses secara otomatis. Jika aplikasi Anda mendorong lalu lintas tinggi secara tidak proporsional ke satu atau beberapa item, DynamoDB akan menyeimba ngkan kembali partisi Anda sehingga item yang sering diakses tidak berada di partisi yang sama.	26 November 2019
Dukungan untuk kunci yang dikelola pelanggan	DynamoDB sekarang mendukung kunci yang dikelola pelanggan, yang berarti Anda dapat memiliki kontrol penuh atas cara Anda mengenkripsi dan mengelola keamanan data DynamoDB Anda.	25 November 2019

Dukungan NoSQL Workbench untuk DynamoDB lokal (Versi yang Dapat Diunduh)	NoSQL Workbench kini mendukung koneksi ke DynamoDB lokal (Versi yang Dapat Diunduh) untuk merancang, membuat, mengkueri, dan mengelola tabel DynamoDB.	8 November 2019
NoSQL Workbench - Pratinjau	Ini adalah rilis awal NoSQL Workbench untuk DynamoDB. Gunakan NoSQL Workbench untuk merancang, membuat, mengkueri, dan mengelola tabel DynamoDB. Untuk informasi selengkapnya, lihat NoSQL Workbench untuk Amazon DynamoDB (Pratinjau).	16 September 2019
DAX menambahkan dukungan untuk operasi transaksional menggunakan Python dan .NET	DAX mendukung `TransactWriteItems` dan `TransactGetItems` APIs untuk aplikasi yang ditulis dalam Go, Java, .NET, Node.js, dan Python. Untuk informasi selengkapnya, lihat Akselerasi Dalam Memori dengan DAX.	14 Februari 2019

Pembaruan Amazon DynamoDB lokal (Versi yang Dapat Diunduh)	DynamoDB lokal (Versi yang Dapat Diunduh) sekarang mendukung kapasitas APIs transaksional, read/write sesuai permintaan, pelaporan kapasitas untuk operasi baca dan tulis, dan 20 indeks sekunder global. Untuk informasi selengkapnya, lihat Perbedaan antara DynamoDB yang Dapat Diunduh dan Layanan Web DynamoDB.	4 Februari 2019
Amazon DynamoDB Sesuai Permintaan	DynamoDB sesuai permintaan adalah opsi penagihan fleksibel yang mampu melayani ribuan permintaan per detik tanpa perencanaan kapasitas. DynamoDB on-demand pay-per-request menawarkan harga untuk permintaan baca dan tulis sehingga Anda hanya membayar untuk apa yang Anda gunakan. Untuk informasi selengkapnya, lihat Kapasitas throughput DynamoDB.	28 November 2018

Amazon DynamoDB Transactions	Transaksi DynamoDB membuat terkoordinasi all-or-nothing, perubahan pada beberapa item baik di dalam maupun di seluruh tabel, memberikan atomisitas, konsistensi, isolasi, dan daya tahan (ACID) di DynamoDB. Untuk informasi selengkapnya, lihat Amazon DynamoDB Transactions.	27 November 2018
Amazon DynamoDB mengenkripsi semua data diam pelanggan	Enkripsi DynamoDB saat diam menyediakan lapisan tambahan perlindungan data dengan mengamankan data Anda dalam tabel terenkripsi, termasuk kunci primer, indeks sekunder lokal dan global, aliran, tabel global, cadangan, dan klaster DAX setiap kali data disimpan dalam media yang tahan lama. Untuk informasi selengkapnya, lihat Enkripsi Amazon DynamoDB saat Diam.	15 November 2018

Menggunakan Amazon DynamoDB Lokal Lebih Mudah dengan Citra Docker Baru	Sekarang, penggunaan DynamoDB lokal, versi DynamoDB yang dapat diunduh, menjadi lebih mudah untuk membantu Anda mengembangkan dan menguji aplikasi DynamoDB Anda dengan menggunakan citra Docker DynamoDB lokal baru. Untuk informasi selengkapnya, lihat DynamoDB (Versi yang Dapat Diunduh) dan Docker.	22 Agustus 2018
DynamoDB Accelerator (DAX) Menambahkan Dukungan untuk Enkripsi Diam	DynamoDB Accelerator (DAX) kini mendukung enkripsi diam untuk klaster DAX baru guna membantu Anda mempercepat pembacaan dari tabel Amazon DynamoDB dalam aplikasi yang sensitif terhadap keamanan yang tunduk pada persyaratan kepatuhan dan regulasi yang ketat. Untuk informasi selengkapnya, lihat Enkripsi Diam DAX.	9 Agustus 2018

point-in-timeDynamoDB recovery (PITR) menambahkan dukungan untuk memulihkan tabel yang dihapus	Jika Anda menghapus tabel dengan point-in-time pemulihan diaktifkan, cadangan sistem secara otomatis dibuat dan dipertahankan selama 35 hari (tanpa biaya tambahan). Untuk informasi selengkapnya, lihat Sebelum Anda Mulai Menggunakan Pemulihan Titik Waktu.	7 Agustus 2018
Pembaruan kini tersedia melalui RSS	Sekarang, Anda dapat berlangganan umpan RSS (di sudut kiri atas halaman ini) untuk menerima notifikasi tentang pembaruan Panduan Developer Amazon DynamoDB.	3 Juli 2018

Pembaruan lebih awal

Tabel berikut menjelaskan perubahan penting Panduan Developer DynamoDB sebelum 3 Juli 2018.

Perubahan	Deskripsi	Tanggal Diubah
Dukungan Go untuk DAX	Sekarang, Anda dapat mengaktifkan performa baca mikrodetik untuk tabel Amazon DynamoDB dalam aplikasi Anda yang ditulis dalam bahasa pemrograman Go dengan menggunakan DynamoDB Accelerator (DAX) SDK untuk Go baru. Untuk	26 Juni 2018

Perubahan	Deskripsi	Tanggal Diubah
	informasi selengkapnya, lihat DAX SDK untuk Go.	
DynamoDB mengumumkan SLA	DynamoDB telah merilis SLA ketersediaan publik. Untuk informasi selengkapnya, lihat Perjanjian Tingkat Layanan Amazon DynamoDB.	19 Juni 2018
Pencadangan dan Pemulihan berkelanjutan DynamoDB (Point-In-TimePITR)	Point-in-time pemulihan membantu melindungi tabel Amazon DynamoDB Anda dari operasi tulis atau hapus yang tidak disengaja. Dengan pemulihan titik waktu, Anda tidak perlu khawatir dengan membuat, mengelola, atau menjadwalkan pencadangan sesuai permintaan. Misalnya, skrip pengujian tidak sengaja menulis ke tabel DynamoDB produksi. Dengan point-in-time pemulihan, Anda dapat mengembalikan tabel itu ke titik waktu mana pun selama 35 hari terakhir. DynamoDB memelihara cadangan tambahan tabel Anda. Untuk informasi selengkapnya, lihat Point-in-time backup untuk DynamoDB.	25 April 2018

Perubahan	Deskripsi	Tanggal Diubah
Enkripsi diam untuk DynamoDB	Enkripsi DynamoDB saat diam, tersedia untuk tabel DynamoDB baru, membantu mengamankan data aplikasi Anda dalam tabel Amazon DynamoDB menggunakan kunci enkripsi yang dikelola AWS yang disimpan di AWS Key Management Service. Untuk informasi selengkapnya, lihat Enkripsi DynamoDB saat diam.	8 Februari 2018
Pencadangan DynamoDB dan pemulihan	Pencadangan Sesuai Permintaan memungkinkan Anda membuat cadangan lengkap data tabel DynamoDB untuk pengarsipan data, membantu Anda memenuhi persyaratan regulasi perusahaan dan pemerintah. Anda dapat mencadangkan tabel mulai dari beberapa megabyte hingga ratusan terabyte data, tanpa memengaruhi performa dan ketersediaan aplikasi produksi Anda. Untuk informasi selengkapnya, lihat Backup dan restore untuk DynamoDB.	29 November 2017

Perubahan	Deskripsi	Tanggal Diubah
Tabel Global DynamoDB	Tabel Global dibangun berdasarkan jejak global DynamoDB untuk memberi Anda basis data yang terkelola sepenuhnya, multi-wilayah, dan multi-aktif yang memberikan performa baca dan tulis yang cepat serta lokal, untuk aplikasi global berskala besar. Tabel Global mereplikasi tabel Amazon DynamoDB Anda secara otomatis di seluruh wilayah pilihan Anda. AWS Untuk informasi selengkapnya, lihat Tabel global - Replikasi multi-Wilayah untuk DynamoDB.	29 November 2017
Dukungan Node.js untuk DAX	Developer Node.js dapat memanfaatkan DynamoDB Accelerator (DAX), menggunakan klien DAX untuk Node.js. Untuk informasi selengkapnya, lihat Akselerasi dalam memori dengan DynamoDB Accelerator (DAX).	5 Oktober 2017

Perubahan	Deskripsi	Tanggal Diubah
Titik akhir VPC untuk DynamoDB	Titik akhir DynamoDB memungkinkan EC2 instans Amazon di VPC Amazon Anda mengakses DynamoDB, tanpa terpapar ke Internet publik. Lalu lintas jaringan antara VPC Anda dan DynamoDB tidak meninggal kan jaringan Amazon. Untuk informasi selengkapnya, lihat Menggunakan titik akhir Amazon VPC untuk mengakses DynamoDB.	16 Agustus 2017

Perubahan	Deskripsi	Tanggal Diubah
Penskalaan Otomatis untuk DynamoDB	Penskalaan otomatis DynamoDB menghilangkan kebutuhan untuk menetapkan atau menyesuaikan secara manual pengaturan throughput yang ditetapkan. Sebaliknya, penskalaan otomatis DynamoDB menyesuaikan kapasitas baca dan tulis secara dinamis dalam merespons pola lalu lintas aktual. Ini memungkinkan tabel atau indeks sekunder global meningkatkan kapasitas baca dan tulis yang disediakan untuk menangani peningkatan lalu lintas tiba-tiba, tanpa throttling. Ketika beban kerja menurun, penskalaan otomatis DynamoDB mengurangi kapasitas yang disediakan. Untuk informasi selengkapnya, lihat Mengelola kapasitas throughput secara otomatis dengan penskalaan otomatis DynamoDB.	14 Juni 2017

Perubahan	Deskripsi	Tanggal Diubah
DynamoDB Accelerator (DAX)	DynamoDB Accelerator (DAX) adalah cache dalam memori yang terkelola sepenuhnya dan sangat tersedia untuk DynamoDB yang memberikan peningkatan performa hingga 10x – dari milidetik hingga mikrodetik – bahkan pada jutaan permintaan per detik. Untuk informasi selengkapnya, lihat Akselerasi dalam memori dengan DynamoDB Accelerator (DAX).	19 April 2017
DynamoDB sekarang mendukung kedaluwarsa item otomatis dengan Waktu untuk Tayang (TTL)	Waktu untuk Tayang (TTL) Amazon DynamoDB memungkinkan Anda untuk menghapus item yang kedaluwarsa dari tabel Anda secara otomatis, tanpa biaya tambahan. Untuk informasi selengkapnya, lihat Menggunakan time to live (TTL) di DynamoDB.	27 Februari 2017

Perubahan	Deskripsi	Tanggal Diubah
DynamoDB sekarang mendukung Tanda Alokasi Biaya	Anda sekarang dapat menambahkan tanda ke tabel Amazon DynamoDB Anda untuk meningkatkan penggunaan kategorisasi dan pelaporan biaya yang lebih mendetail. Untuk informasi selengkapnya, lihat Menambahkan tag dan label ke sumber daya di DynamoDB.	19 Januari 2017

Perubahan	Deskripsi	Tanggal Diubah
API `DescribeLimits` DynamoDB baru	`DescribeLimits` API mengembalikan batas kapasitas yang disediakan saat ini untuk AWS akun Anda di suatu wilayah, baik untuk wilayah secara keseluruhan maupun untuk satu tabel DynamoDB yang Anda buat di sana. Ini memungkinkan Anda untuk menentukan batas tingkat akun saat ini sehingga Anda dapat membandingkannya dengan kapasitas yang disediakan yang saat ini Anda gunakan, dan memiliki banyak waktu untuk mengajukan peningkatan sebelum mencapai batas. Untuk informasi selengkapnya, lihat Kuota di Amazon DynamoDB dan DescribeLimitsdi Referensi Amazon DynamoDB API.	1 Maret 2016

Perubahan	Deskripsi	Tanggal Diubah
Pembaruan Konsol DynamoDB dan Terminologi Baru untuk Atribut Kunci Primer	Konsol manajemen DynamoDB telah didesain ulang agar lebih intuitif dan mudah digunakan. Sebagai bagian dari pembaruan ini, kami memperkenalkan terminologi baru untuk atribut kunci primer: • Kunci Partisi—juga dikenal sebagai atribut hash. • Kunci Urutan juga dikenal sebagai atribut rentang. Hanya nama yang berubah; fungsionalitasnya tetap sama. Ketika membuat tabel atau indeks sekunder, Anda dapat memilih kunci primer sederhana (hanya kunci partisi), atau kunci primer komposit (kunci partisi dan kunci urutan). Dokumentasi DynamoDB telah diperbarui untuk mencerminkan perubahan ini.	12 November 2015

Perubahan	Deskripsi	Tanggal Diubah
Backend Penyimpanan Amazon DynamoDB untuk Titan	Backend Penyimpanan DynamoDB untuk Titan adalah backend penyimpanan untuk basis data grafik Titan yang diimplementasikan di atas Amazon DynamoDB. Saat menggunakan Backend Penyimpanan DynamoDB untuk Titan, data Anda mendapat manfaat dari perlindungan DynamoDB, yang berjalan di seluruh pusat data Amazon dengan ketersediaan tinggi. Plugin ini tersedia untuk Titan versi 0.4.4 (terutama untuk kompatibilitas dengan aplikasi yang ada) dan Titan versi 0.5.4 (direkomendasikan untuk aplikasi baru). Seperti backend penyimpanan lainnya untuk Titan, plugin ini mendukung tumpukan Tinkerpop (versi 2.4 dan 2.5), termasuk API Blueprints dan shell Gremlin.	20 Agustus 2015

Perubahan	Deskripsi	Tanggal Diubah
DynamoDB Streams, Replikasi Lintas Wilayah, dan Memindai dengan Bacaan yang Sangat Konsisten	DynamoDB Streams menangkap urutan waktu modifikasi tingkat item di tabel DynamoDB mana pun, dan menyimpan informasi ini dalam log hingga 24 jam. Aplikasi dapat mengakses log ini dan melihat item data seperti yang ditampilkan sebelum dan setelah dimodifikasi, mendekati waktu nyata. Untuk informasi selengkapnya, lihat Tangkapan data perubahan DynamoDB Streams dan Referensi API DynamoDB Streams. Replikasi lintas wilayah DynamoDB adalah solusi sisi klien untuk mempertahankan salinan identik tabel DynamoDB di berbagai wilayah, dalam waktu dekat. AWS Anda dapat menggunakan replikasi lintas wilayah untuk mencadangkan tabel DynamoDB, atau untuk menyediakan akses latensi rendah ke data tempat pengguna didistribusikan secara geografis. Operasi `Scan` DynamoDB menggunakan bacaan akhir konsisten, secara default.	16 Juli 2015

Perubahan	Deskripsi	Tanggal Diubah
	Anda dapat menggunakan bacaan sangat konsisten dengan mengatur parameter `ConsistentRead` ke true. Untuk informasi selengkapnya, lihat Konsistensi baca untuk kueri dan Pindai di Referensi API Amazon DynamoDB.	
AWS CloudTrail dukungan untuk Amazon DynamoDB	DynamoDB sekarang terintegrasi dengan. CloudTrail CloudTrail menangkap panggilan API yang dibuat dari konsol DynamoDB atau dari DynamoDB API dan melacaknya dalam file log. Untuk informasi selengkapnya, lihat Pencatatan log operasi DynamoDB menggunakan AWS CloudTrail dan Panduan Pengguna AWS CloudTrail.	28 Mei 2015

Perubahan	Deskripsi	Tanggal Diubah
Peningkatan dukungan untuk ekspresi Kueri	Rilis ini menambahkan parameter `KeyConditionExpression` baru untuk API `Query`. `Query` membaca item dari tabel atau indeks menggunakan nilai-nilai kunci primer. Parameter `KeyConditionExpression` adalah string yang mengidentifikasi nama-nama kunci primer, dan ketentuan yang akan diterapkan ke nilai-nilai kunci; `Query` hanya mengambil item-item yang memenuhi ekspresi. Sintaks `KeyConditionExpression` mirip dengan parameter ekspresi lainnya di DynamoDB, dan memungkinkan Anda untuk menentukan variabel pengganti untuk nama dan nilai dalam ekspresi. Untuk informasi selengkapnya, lihat Menanyakan tabel di DynamoDB.	27 April 2015

Perubahan	Deskripsi	Tanggal Diubah
Fungsi perbandingan baru untuk penulisan bersyarat	Dalam DynamoDB, parameter `ConditionExpression` menentukan apakah `PutItem`, `UpdateItem`, atau `DeleteItem` berhasil: Item tersebut ditulis hanya jika ketentuan bernilai true. Rilis ini menambahkan dua fungsi baru, `attribute_type` dan `size`, untuk digunakan dengan `ConditionExpression`. Fungsi-fungsi ini memungkinkan Anda untuk melakukan penulisan bersyarat berdasarkan jenis data atau ukuran atribut dalam sebuah tabel. Untuk informasi selengkapnya, lihat Contoh CLI ekspresi kondisi DynamoDB.	27 April 2015

Perubahan	Deskripsi	Tanggal Diubah
Memindai API untuk indeks sekunder	Di DynamoDB, operasi `Scan` membaca semua item dalam sebuah tabel, menerapkan kriteria pemfilteran yang ditetapkan pengguna, dan mengembalikan item data yang dipilih ke aplikasi. Kemampuan yang sama ini sekarang tersedia untuk indeks sekunder juga. Untuk memindai indeks sekunder lokal atau indeks sekunder global, Anda menentukan nama indeks dan nama tabel induknya. Secara default, indeks `Scan` mengembalikan semua data dalam indeks; Anda dapat menggunakan ekspresi filter untuk mempersempit hasil yang dikembalikan ke aplikasi. Untuk informasi selengkapnya, lihat Memindai tabel di DynamoDB.	10 Februari 2015

Perubahan	Deskripsi	Tanggal Diubah
Operasi online untuk indeks sekunder global	Pengindeksan online memungkinkan Anda menambah atau menghapus indeks sekunder global pada tabel yang ada. Dengan pengindeksan online, Anda tidak perlu menentukan semua indeks tabel ketika Anda membuat tabel; sebaliknya, Anda dapat menambahkan indeks baru setiap saat. Demikian pula, jika memutuskan tidak memerlukan indeks lagi, Anda dapat menghapusnya kapan pun. Operasi pengindeksan online bersifat non-blocking, sehingga tabel tetap tersedia untuk aktivitas baca dan tulis saat indeks ditambahkan atau dihapus. Untuk informasi selengkapnya, lihat Mengelola Indeks Sekunder Global di DynamoDB.	27 Januari 2015

Perubahan	Deskripsi	Tanggal Diubah
Dukungan model dokumen dengan JSON	DynamoDB memungkinkan Anda untuk menyimpan dan mengambil dokumen dengan dukungan penuh untuk model dokumen. Jenis data baru sepenuhnya kompatibel dengan standar JSON dan memungkinkan Anda menyusun elemen dokumen satu sama lain. Anda dapat menggunakan operator dereferensi jalur dokumen untuk membaca dan menulis masing-masing elemen, tanpa harus mengambil seluruh dokumen. Rilis ini juga memperkenalkan parameter ekspresi baru untuk menentukan proyeksi, kondisi, dan tindakan pembaruan ketika membaca atau menulis item data. Untuk mempelajari selengkapnya tentang dukungan model dokumen dengan JSON, lihat Jenis Data dan Menggunakan ekspresi di DynamoDB.	7 Oktober 2014

Perubahan	Deskripsi	Tanggal Diubah
Penskalaan fleksibel	Untuk tabel dan indeks sekunder global, Anda dapat meningkatkan kapasitas throughput baca dan tulis yang disediakan dengan jumlah berapa pun, asalkan Anda tetap berada dalam batas per tabel dan per akun. Untuk informasi selengkapnya, lihat Kuota di Amazon DynamoDB.	7 Oktober 2014
Ukuran item yang lebih besar	Ukuran item maksimum di DynamoDB telah meningkat dari 64 KB menjadi 400 KB. Untuk informasi selengkap nya, lihat Kuota di Amazon DynamoDB.	7 Oktober 2014

Perubahan	Deskripsi	Tanggal Diubah
Ekspresi bersyarat yang ditingkatkan	DynamoDB memperluas operator yang tersedia untuk ekspresi bersyarat, sehingga memberikan fleksibilitas tambahan untuk penempatan, pembaruan, dan penghapusan bersyarat. Operator baru yang tersedia memungkinkan Anda memeriksa apakah atribut ada atau tidak ada, lebih besar dari atau kurang dari nilai tertentu, berada di antara dua nilai, dimulai dengan karakter tertentu, dan banyak lagi. DynamoDB juga menyediakan operator OR opsional untuk mengevaluasi beberapa kondisi. Secara default, beberapa kondisi dalam sebuah ekspresi di-AND bersama-sama, sehingga ekspresi hanya bernilai true jika semua kondisinya benar. Jika Anda menentukan OR, ekspresinya adalah true jika satu atau berberapa kondisi adalah true. Untuk informasi selengkapnya, lihat Bekerja dengan item dan atribut di DynamoDB.	24 April 2014

Perubahan	Deskripsi	Tanggal Diubah
Filter kueri	API `Query` DynamoDB mendukung opsi `QueryFilter` baru. Secara default, `Query` menemukan item yang cocok dengan nilai kunci partisi tertentu dan kondisi kunci urutan opsional. `Query` menerapkan ekspresi bersyarat ke atribut non kunci lain; jika filter `Query` ada, maka item yang tidak cocok dengan kondisi filter akan dibuang sebelum hasil `Query` dikembalikan ke aplikasi. Untuk informasi selengkap nya, lihat Menanyakan tabel di DynamoDB.	24 April 2014

Perubahan	Deskripsi	Tanggal Diubah
Ekspor dan impor data menggunakan AWS Management Console	Konsol DynamoDB telah ditingkatkan untuk menyederh anakan ekspor dan impor data dalam tabel DynamoDB. Hanya dengan beberapa klik, Anda dapat mengatur AWS Data Pipeline untuk mengatur alur kerja, dan MapReduce cluster Amazon Elastic untuk menyalin data dari tabel DynamoDB ke bucket Amazon S3, atau sebaliknya. Anda dapat melakukan ekspor atau impor satu kali saja, atau mengatur tugas ekspor harian. Anda bahkan dapat melakukan ekspor dan impor lintas wilayah, menyalin data DynamoDB dari tabel di satu AWS wilayah ke tabel di wilayah lain. AWS	6 Maret 2014

Perubahan	Deskripsi	Tanggal Diubah
Menata ulang dokumentasi API tingkat yang lebih tinggi	Informasi tentang hal berikut APIs sekarang lebih mudah ditemukan: • Jawa: Dinamo DBMappper • .NET: Model dokumen dan model persistensi objek Tingkat yang lebih tinggi ini sekarang APIs didokumen tasikan di sini:. Antarmuka pemrograman tingkat tinggi untuk DynamoDB	20 Januari 2014

Perubahan	Deskripsi	Tanggal Diubah
Indeks sekunder global	DynamoDB menambahkan dukungan untuk indeks sekunder global. Seperti pada indeks sekunder lokal, Anda menentukan indeks sekunder global dengan menggunakan kunci alternatif dari tabel lalu mengeluarkan permintaan Kueri pada indeks. Tidak seperti indeks sekunder lokal, kunci partisi untuk indeks sekunder global tidak harus sama dengan tabel; kunci partisi tersebut dapat berupa atribut skalar dari tabel. Kunci urutan bersifat opsional, dan juga dapat berupa atribut skalar apa pun. Indeks sekunder global juga memiliki pengaturan throughput tersendiri, yang terpisah dari tabel induk. Untuk informasi selengkapnya, lihat Meningkatkan akses data dengan indeks sekunder di DynamoDB dan Menggunakan Indeks Sekunder Global di DynamoDB.	12 Desember 2013

Perubahan	Deskripsi	Tanggal Diubah
Kontrol akses detail	DynamoDB menambahkan dukungan untuk kontrol akses detail. Fitur ini memungkinkan pelanggan menentukan (pengguna, grup, atau peran) utama yang dapat mengakses masing-masing item dan atribut dalam tabel DynamoDB atau indeks sekunder. Aplikasi juga dapat memanfaatkan federasi identitas web untuk memindahkan tugas autentikasi pengguna ke penyedia identitas pihak ketiga, seperti Facebook, Google, atau Login with Amazon. Dengan cara ini, aplikasi (termasuk aplikasi seluler) dapat menangani pengguna dalam jumlah yang sangat besar, sekaligus memastikan bahwa tidak ada seorang pun yang dapat mengakses item data DynamoDB kecuali mereka yang diberi wewenang untuk melakukannya. Untuk informasi selengkapnya, lihat Menggunakan ketentuan kebijakan IAM untuk kontrol akses terperinci.	29 Oktober 2013

Perubahan	Deskripsi	Tanggal Diubah
Ukuran unit kapasitas baca 4 KB	Ukuran unit kapasitas untuk baca telah meningkat dari 1 KB menjadi 4 KB. Peningkatan ini dapat mengurangi jumlah unit kapasitas baca yang disediakan yang diperlukan untuk banyak aplikasi. Sebagai contoh, sebelum rilis ini, pembacaan item 10 KB akan mengonsumsi 10 unit kapasitas baca; sekarang pembacaan 10 KB yang sama hanya akan mengonsumsi 3 unit (10 KB / 4 KB, dibulatkan ke batas 4 KB berikutnya). Untuk informasi selengkapnya, lihat Kapasitas throughput DynamoDB.	14 Mei 2013

Perubahan	Deskripsi	Tanggal Diubah
Pemindaian paralel	DynamoDB menambahkan dukungan untuk operasi Pindai paralel. Aplikasi sekarang dapat membagi tabel menjadi beberapa segmen logis dan memindai semua segmen secara bersamaan. Fitur ini mengurangi waktu yang diperlukan untuk menyelesaikan Pemindaian, dan sepenuhnya memanfaatkan kapasitas baca yang disediakan tabel. Untuk informasi selengkapnya, lihat Memindai tabel di DynamoDB.	14 Mei 2013
Indeks sekunder lokal	DynamoDB menambahkan dukungan untuk indeks sekunder lokal. Anda dapat menentukan indeks kunci urutan pada atribut non-kunci, lalu menggunakan indeks ini dalam permintaan Kueri. Dengan indeks sekunder lokal, aplikasi dapat mengambil item data di beberapa dimensi secara efisien. Untuk informasi selengkapnya, lihat Indeks Sekunder Lokal di DynamoDB.	18 April 2013

Perubahan	Deskripsi	Tanggal Diubah
Versi API baru	Dengan rilis ini, DynamoDB memperkenalkan versi API baru (2012-08-10). Versi API sebelumnya (2011-12-05) masih didukung untuk kompatibilitas mundur dengan aplikasi yang ada. Aplikasi baru harus menggunakan versi API terkini 2012-08-10. Sebaiknya Anda memigrasi kan aplikasi yang ada ke API versi 2012-08-10, karena fitur DynamoDB baru (seperti indeks sekunder lokal) tidak akan di-backport ke versi API sebelumnya. Untuk informasi selengkapnya tentang API versi 2012-08-1 0, lihat Referensi API Amazon DynamoDB.	18 April 2013

Perubahan	Deskripsi	Tanggal Diubah
Dukungan variabel kebijakan IAM	Bahasa kebijakan akses IAM kini mendukung variabel. Ketika suatu kebijakan dievaluasi, setiap variabel kebijakan diganti dengan nilai yang disediakan oleh informasi berbasis konteks dari sesi pengguna yang diautentikasi. Anda dapat menggunakan variabel kebijakan untuk menentuka n kebijakan tujuan umum tanpa mencantumkan semua komponen kebijakan secara eksplisit. Untuk informasi selengkapnya tentang variabel kebijakan, buka Variabel Kebijakan di panduan AWS Identity and Access Management Menggunakan IAM. Untuk contoh variabel kebijakan di DynamoDB, lihat Manajemen Identitas dan Akses untuk Amazon DynamoDB.	4 April 2013

Perubahan	Deskripsi	Tanggal Diubah
contoh kode PHP diperbarui untuk AWS SDK untuk PHP versi 2	Versi 2 AWS SDK untuk PHP sekarang tersedia. Contoh kode PHP di Panduan Developer Amazon DynamoDB telah diperbarui untuk menggunakan SDK baru ini. Untuk informasi selengkapnya tentang Versi 2 untuk SDK ini, lihat AWS SDK untuk PHP.	23 Januari 2013
Titik akhir baru	DynamoDB memperluas ke wilayah (AS-Barat) AWS GovCloud . Untuk daftar titik akhir dan protokol layanan saat ini, lihat Wilayah dan Titik Akhir.	3 Desember 2012
Titik akhir baru	Ekspansi DynamoDB ke wilayah Amerika Selatan (Sao Paulo). Untuk daftar titik akhir yang didukung saat ini, lihat Wilayah dan Titik Akhir.	3 Desember 2012
Titik akhir baru	Ekspansi DynamoDB ke wilayah Asia Pasifik (Sydney). Untuk daftar titik akhir yang didukung saat ini, lihat Wilayah dan Titik Akhir.	13 November 2012

Perubahan	Deskripsi	Tanggal Diubah
DynamoDB mengimple mentasikan dukungan CRC32 untuk checksum, mendukung batch get yang sangat konsisten, dan menghapus pembatasan pada pembaruan tabel bersamaan.	• DynamoDB menghitun g CRC32 checksum dari payload HTTP dan mengembalikan checksum ini di header baru,. `x-amz-crc32` Untuk informasi selengkapnya, lihat API tingkat rendah DynamoDB. • Secara default, operasi baca yang dilakukan oleh API `BatchGetItem` pada akhirnya konsisten . Parameter `ConsistentRead` baru di `BatchGetItem` memungkinkan Anda memilih konsistensi baca yang kuat, untuk tabel dalam permintaan. Untuk informasi selengkapnya, lihat Deskripsi. • Rilis ini menghilangkan beberapa batasan saat memperbarui banyak tabel secara bersamaan. Jumlah total tabel yang dapat diperbarui sekaligus masih 10; tetapi, tabel ini sekarang dapat berupa kombinasi dari status `CREATING`, `UPDATING`, atau `DELETING`. Selain itu, tidak ada lagi jumlah minimum untuk menambah atau mengurang i ReadCapacityUnitsatau	2 November 2012

Perubahan	Deskripsi	Tanggal Diubah
	WriteCapacityUnitsuntuk tabel. Untuk informasi selengkapnya, lihat Kuota di Amazon DynamoDB.	
Dokumentasi praktik terbaik	Panduan Developer Amazon DynamoDB mengidentifikasi praktik terbaik untuk bekerja dengan tabel dan item, serta rekomendasi untuk operasi kueri dan pindai.	28 September 2012

Perubahan	Deskripsi	Tanggal Diubah
Dukungan untuk jenis data biner	Selain jenis Angka dan String, DynamoDB kini mendukung jenis data Biner. Sebelum rilis ini, untuk menyimpan data biner, Anda mengonversi data biner ke dalam format string dan menyimpannya dalam DynamoDB. Selain pekerjaan konversi yang diperlukan di sisi klien, konversi sering kali meningkatkan ukuran item data yang memerlukan lebih banyak penyimpanan dan potensi kapasitas throughput yang disediakan tambahan. Dengan atribut jenis biner, Anda sekarang dapat menyimpan data biner apa pun, seperti data terkompresi, data terenkripsi, dan gambar. Untuk mengetahui informasi selengkapnya, lihat Jenis Data. Untuk contoh kerja penanganan data tipe biner menggunakan AWS SDKs, lihat bagian berikut: • Contoh: Menangani atribut tipe biner menggunakan API AWS SDK untuk Java dokumen	21 Agustus 2012

Perubahan	Deskripsi	Tanggal Diubah
	• Contoh: Menangani atribut tipe biner menggunakan API AWS SDK untuk .NET tingkat rendah Untuk dukungan tipe data biner yang ditambahkan di AWS SDKs, Anda perlu mengunduh yang terbaru SDKs dan Anda mungkin juga perlu memperbarui aplikasi yang ada. Untuk informasi tentang cara mengunduh AWS SDKs, lihat Contoh kode .NET.	
Item tabel DynamoDB dapat diperbarui dan disalin menggunakan konsol DynamoDB	Pengguna DynamoDB sekarang dapat memperbarui dan menyalin item tabel menggunakan konsol DynamoDB, selain dapat menambahkan dan menghapus item. Fungsi baru ini menyederhanakan pembuatan perubahan pada masing-masing item melalui Konsol tersebut.	14 Agustus 2012

Perubahan	Deskripsi	Tanggal Diubah
DynamoDB menurunkan persyaratan throughput tabel minimum	DynamoDB kini mendukung persyaratan throughput tabel minimum yang lebih rendah, yaitu 1 unit kapasitas tulis dan 1 unit kapasitas baca. Untuk informasi selengkap nya, lihat topik Kuota di Amazon DynamoDB di Panduan Developer Amazon DynamoDB.	9 Agustus 2012
Dukungan Signature Versi 4	DynamoDB kini mendukung Signature Versi 4 untuk mengautentikasi permintaan.	5 Juli 2012
Dukungan eksplorer tabel di Konsol DynamoDB	Konsol DynamoDB kini mendukung eksplorer tabel yang memungkinkan Anda untuk menelusuri dan mengkueri data dalam tabel Anda. Anda juga dapat memasukkan item baru atau menghapus item yang ada. Bagian `SampleData` dan Menggunakan konsol telah diperbarui untuk fitur ini.	22 Mei 2012

Perubahan	Deskripsi	Tanggal Diubah
Titik akhir baru	Ketersediaan DynamoDB diperluas dengan titik akhir baru di wilayah AS Barat (California Utara), wilayah AS Barat (Oregon), dan wilayah Asia Pasifik (Singapura). Untuk daftar titik akhir yang didukung saat ini, buka Wilayah dan Titik Akhir.	24 April 2012
BatchWriteItem Dukungan API	DynamoDB kini mendukung API penulisan batch yang memungkinkan Anda memasukkan dan menghapus beberapa item dari satu atau beberapa tabel dalam satu panggilan API. Untuk informasi selengkapnya tentang API penulisan batch DynamoDB, lihat BatchWriteItem. Untuk informasi tentang bekerja dengan item dan menggunakan fitur penulisan batch menggunakan AWS SDKs, lihat Bekerja dengan item dan atribut di DynamoDB danContoh kode .NET.	19 April 2012
Mendokumentasikan lebih banyak kode kesalahan	Untuk informasi selengkapnya, lihat Penanganan kesalahan dengan DynamoDB.	5 April 2012

Perubahan	Deskripsi	Tanggal Diubah
Titik akhir baru	Ekspansi DynamoDB ke wilayah Asia Pasifik (Tokyo). Untuk daftar titik akhir yang didukung saat ini, lihat Wilayah dan Titik Akhir.	29 Februari 2012
Metrik `ReturnedItemCount` ditambahkan	Metrik baru, `ReturnedItemCount`, memberikan jumlah item yang dikembalikan dalam respons operasi Query atau Scan untuk DynamoDB tersedia untuk dipantau. CloudWatch	24 Februari 2012
Menambahkan contoh untuk nilai tambahan	DynamoDB mendukung tambahan dan pengurangan nilai numerik yang ada. Contoh menunjukkan penambahan nilai yang ada di bagian “Memperbarui Item” di: Bekerja dengan item: Java. Bekerja dengan item: .NET.	25 Januari 2012
Rilis produk awal	DynamoDB diperkenalkan sebagai layanan baru dalam rilis Beta.	18 Januari 2012

Fitur lama DynamoDB

Topik berikut adalah fitur lama yang masih didukung DynamoDB. Tidak ada pengembangan aktif yang dilakukan pada fitur-fitur ini.

Topik

- Versi tabel global 2017.11.29 (Legacy)
- Versi API DynamoDB tingkat rendah sebelumnya (2011-12-05)
- Parameter bersyarat DynamoDB lama

Versi tabel global 2017.11.29 (Legacy)

> **Important**
>
> Dokumentasi ini ditujukan untuk versi 2017.11.29 (Lama) tabel global, yang tidak boleh digunakan untuk tabel global baru. Pelanggan harus menggunakan Tabel Global versi 2019.11.21 (Saat Ini) jika memungkinkan, karena memberikan fleksibilitas yang lebih besar, efisiensi yang lebih tinggi, dan mengkonsumsi kapasitas tulis yang lebih sedikit daripada 2017.11.29 (Legacy).
>
> Untuk menentukan versi mana yang sedang Anda gunakan, lihat Menentukan versi tabel global DynamoDB yang Anda gunakan. Untuk memperbarui tabel global yang ada dari versi 2017.11.29 (Lama) ke versi 2019.11.21 (Terbaru), lihat Meningkatkan tabel global.

Topik

- Tabel global: Cara kerjanya
- Praktik terbaik dan persyaratan untuk mengelola tabel global
- Membuat tabel global
- Memantau tabel global
- Menggunakan IAM dengan tabel global

Tabel global: Cara kerjanya

> **Important**
>
> Dokumentasi ini ditujukan untuk versi 2017.11.29 (Lama) tabel global, yang tidak boleh digunakan untuk tabel global baru. Pelanggan harus menggunakan Tabel Global versi 2019.11.21 (Saat Ini) jika memungkinkan, karena memberikan fleksibilitas yang lebih besar, efisiensi yang lebih tinggi, dan mengkonsumsi kapasitas tulis yang lebih sedikit daripada 2017.11.29 (Legacy).
> Untuk menentukan versi mana yang sedang Anda gunakan, lihat Menentukan versi tabel global DynamoDB yang Anda gunakan. Untuk memperbarui tabel global yang ada dari versi 2017.11.29 (Lama) ke versi 2019.11.21 (Terbaru), lihat Meningkatkan tabel global.

Bagian berikut membantu Anda memahami konsep dan perilaku tabel global di Amazon DynamoDB.

Konsep tabel global untuk Versi 2017.11.29 (Lama)

Tabel global adalah kumpulan dari satu atau lebih tabel replika, semuanya dimiliki oleh satu AWS akun.

Tabel replika (atau replika, untuk singkatnya) adalah tabel DynamoDB tunggal yang berfungsi sebagai bagian dari tabel global. Setiap replika menyimpan set item data yang sama. Setiap tabel global tertentu hanya dapat memiliki satu tabel replika per Wilayah AWS .

Berikut ini adalah gambaran umum konseptual tentang cara pembuatan tabel global.

1. Buat tabel DynamoDB biasa, dengan DynamoDB Streams diaktifkan, di Region. AWS
2. Ulangi langkah 1 untuk setiap Wilayah lain tempat Anda ingin mereplikasi data.
3. Definisikan tabel global DynamoDB berdasarkan tabel yang telah Anda buat.

AWS Management Console Mengotomatiskan tugas-tugas ini, sehingga Anda dapat membuat tabel global lebih cepat dan mudah. Untuk informasi selengkapnya, lihat Membuat tabel global.

Tabel global DynamoDB yang dihasilkan terdiri dari beberapa tabel replika, satu per Wilayah, yang diperlakukan DynamoDB sebagai satu unit. Setiap replika memiliki nama tabel yang sama dan skema kunci primer yang sama. Ketika aplikasi menulis data ke tabel replika di satu wilayah, DynamoDB otomatis menyebarkan aktivitas tulis tersebut ke tabel replika lain di AWS Wilayah lain.

> **Important**
>
> Agar data tabel Anda tetap sinkron, tabel global otomatis membuat atribut berikut untuk setiap item:
>
> - `aws:rep:deleting`
> - `aws:rep:updatetime`
> - `aws:rep:updateregion`
>
> Jangan mengubah atribut ini atau membuat atribut dengan nama yang sama.

Anda dapat menambahkan tabel replika ke tabel global sehingga dapat tersedia di Wilayah tambahan. (Untuk melakukannya, tabel global harus kosong. Dengan kata lain, tidak boleh ada tabel replika yang berisi data apa pun.)

Anda juga dapat menghapus tabel replika dari tabel global. Jika Anda melakukannya, tabel benar-benar terpisah dari tabel global. Tabel yang baru independen ini tidak lagi berinteraksi dengan tabel global, dan data tidak lagi disebarkan ke atau dari tabel global.

> **Warning**
>
> Ketahuilah bahwa menghapus replika bukanlah proses atom. Untuk memastikan perilaku yang konsisten dan status yang diketahui, Anda sebaiknya mempertimbangkan untuk mengalihkan lalu lintas tulis aplikasi dari replika untuk dihapus sebelumnya. Setelah menghapusnya, tunggu hingga semua titik akhir Wilayah replika menunjukkan replika sebagai terputus sebelum membuat penulisan lebih lanjut sebagai tabel regionalnya yang terisolasi.

Tugas umum

Tugas umum untuk tabel global berfungsi sebagai berikut.

Anda dapat menghapus tabel replika tabel global sama seperti tabel biasa. Ini akan menghentikan replikasi ke Wilayah itu dan menghapus salinan tabel yang disimpan di Wilayah itu. Anda tidak dapat memutuskan replikasi dan membuat salinan tabel ada sebagai entitas independen.

> **Note**
>
> Anda tidak akan dapat menghapus tabel sumber hingga setidaknya 24 jam setelah tabel tersebut digunakan untuk memulai Wilayah baru. Jika mencoba menghapusnya terlalu cepat, Anda akan menerima pesan kesalahan.

Konflik dapat terjadi jika aplikasi memperbarui item yang sama di Wilayah yang berbeda pada waktu yang sama. Untuk membantu memastikan konsistensi akhirnya, tabel global DynamoDB menggunakan metode “penulis terakhir menang” untuk merekonsiliasi antara pembaruan yang dilakukan secara bersamaan. Semua replika akan menyetujui pembaruan terkini dan berkumpul menuju status ketika semua replika memiliki data yang identik.

> **Note**
>
> Ada beberapa cara untuk menghindari konflik, antara lain:
>
> - Menggunakan kebijakan IAM untuk hanya mengizinkan penulisan ke tabel di satu Wilayah.
> - Menggunakan kebijakan IAM untuk mengarahkan pengguna ke hanya satu Wilayah dan menjaga yang lain sebagai siaga siaga, atau secara bergantian merutekan pengguna ganjil ke satu Wilayah dan bahkan pengguna ke Wilayah lain.
> - Menghindari penggunaan pembaruan non-idempoten seperti Bookmark = Bookmark + 1, dan mendukung pembaruan statis seperti Bookmark=25.

Memantau tabel global

Anda dapat menggunakan CloudWatch untuk mengamati metrik`ReplicationLatency`. Metrik ini melacak waktu yang telah berlalu antara saat item yang diperbarui muncul di aliran DynamoDB untuk satu tabel replika, dan kapan item itu muncul di replika lain di tabel global. `ReplicationLatency`dinyatakan dalam milidetik dan dipancarkan untuk setiap pasangan Source-region dan Destination-region. Ini adalah satu-satunya CloudWatch metrik yang disediakan oleh Global Tables v2.

Latensi yang akan Anda lihat bergantung pada jarak antara Wilayah yang Anda pilih, serta variabel lainnya. Latensi dalam kisaran 0,5 hingga 2,5 detik untuk Wilayah mungkin umum terjadi dalam wilayah geografis yang sama.

Waktu Untuk Tayang (TTL)

Anda dapat menggunakan Waktu Untuk Tayang (TTL) untuk menentukan nama atribut yang nilainya menunjukkan waktu kedaluwarsa item tersebut. Nilai ini ditentukan sebagai angka dalam detik sejak dimulainya zaman Unix.

Dengan versi warisan tabel global, penghapusan TTL tidak secara otomatis direplikasi di seluruh replika lainnya. Ketika item dihapus melalui aturan TTL, pekerjaan itu dilakukan tanpa menggunakan Unit Tulis.

Perlu diketahui jika tabel sumber dan target memiliki kapasitas tulis yang disediakan yang sangat rendah, hal ini dapat menyebabkan throttling karena penghapusan TTL memerlukan kapasitas tulis.

Aliran dan transaksi dengan tabel global

Setiap tabel global menghasilkan aliran independen berdasarkan semua tulisannya, terlepas dari titik asal untuk penulisan tersebut. Anda dapat memilih untuk menggunakan aliran DynamoDB ini di satu Wilayah atau di semua Wilayah secara independen.

Jika Anda ingin penulisan lokal yang diproses tetapi bukan penulisan yang direplikasi, Anda dapat menambahkan atribut wilayah Anda sendiri ke setiap item. Kemudian, Anda dapat menggunakan filter peristiwa Lambda untuk hanya menginvokasi Lambda untuk penulisan di Wilayah lokal.

Operasi transaksional memberikan jaminan ACID (Atomicity, Consistency, Isolation, and Durability) HANYA di Wilayah tempat penulisan awalnya dibuat. Transaksi tidak didukung di seluruh Wilayah dalam tabel global.

Misalnya, jika Anda memiliki tabel global dengan replika di Wilayah AS Timur (Ohio) dan AS Barat (Oregon) dan melakukan TransactWriteItems operasi di Wilayah AS Timur (Ohio), Anda dapat mengamati transaksi yang diselesaikan sebagian di Wilayah AS Barat (Oregon) saat perubahan direplikasi. Perubahan hanya akan direplikasi ke Wilayah lain setelah perubahan telah dilakukan di Wilayah sumber.

> **Note**
>
> - Tabel global “write-around” DynamoDB Accelerator dengan memperbarui DynamoDB secara langsung. Akibatnya, DAX tidak akan mengetahui jika sedang menyimpan data usang. Cache DAX hanya akan disegarkan ketika TTL cache kedaluwarsa.
> - Tanda pada tabel global tidak disebarkan secara otomatis.

Throughput baca dan tulis

Tabel global mengelola throughput baca dan tulis dengan cara berikut.

- Kapasitas tulis harus sama di semua instans tabel di seluruh Wilayah.
- Dengan Versi 2019.11.21 (Saat ini, jika tabel diatur untuk mendukung penskalaan otomatis atau dalam mode sesuai permintaan maka kapasitas tulis secara otomatis tetap sinkron. Jumlah kapasitas tulis saat ini yang disediakan di setiap Wilayah akan naik dan turun secara independen dalam pengaturan penskalaan otomatis yang disinkronkan tersebut. Jika tabel ditempatkan dalam mode sesuai permintaan, mode tersebut akan disinkronkan ke replika lainnya.
- Kapasitas baca dapat berbeda antarWilayah karena baca mungkin tidak sama. Saat menambahkan replika global ke tabel, kapasitas Wilayah sumber disebarkan. Setelah pembuatan, Anda dapat menyesuaikan kapasitas baca untuk satu replika, dan pengaturan baru ini tidak ditransfer ke sisi lain.

Konsistensi dan resolusi konflik

Setiap perubahan yang dibuat pada item apa pun di tabel replika mana pun akan direplikasi ke semua replika lain dalam tabel global yang sama. Dalam tabel global, item yang baru ditulis biasanya disebarkan ke semua tabel replika dalam hitungan detik.

Dengan tabel global, setiap tabel replika menyimpan set item data yang sama. DynamoDB tidak mendukung replikasi parsial hanya beberapa item.

Aplikasi dapat membaca dan menulis data ke tabel replika mana pun. DynamoDB mendukung bacaan akhir konsisten di seluruh Wilayah, tetapi tidak mendukung bacaan sangat konsisten di seluruh Wilayah. Jika aplikasi Anda hanya menggunakan pembacaan yang konsisten pada akhirnya dan hanya masalah yang dibaca terhadap satu AWS Wilayah, itu akan berfungsi tanpa modifikasi apa pun. Namun, jika aplikasi Anda membutuhkan bacaan sangat konsisten, aplikasi harus melakukan semua penulisan dan bacaan sangat konsisten di Wilayah yang sama. Jika tidak, jika Anda menulis ke satu Wilayah dan membaca dari Wilayah lain, maka respons baca mungkin menyertakan data usang yang tidak mencerminkan hasil penulisan yang baru saja diselesaikan di Wilayah lain.

Konflik dapat terjadi jika aplikasi memperbarui item yang sama di Wilayah yang berbeda pada waktu yang sama. Untuk membantu memastikan konsistensi akhir, tabel global DynamoDB menggunakan rekonsiliasi penulis terakhir menang antara pembaruan serentak, dengan DynamoDB melakukan upaya terbaik untuk menentukan penulis terakhir. Dengan mekanisme resolusi konflik ini, semua

replika akan menyetujui pembaruan terkini dan berkumpul menuju status ketika semua replika memiliki data yang identik.

Ketersediaan dan daya tahan

Jika satu AWS Wilayah menjadi terisolasi atau terdegradasi, aplikasi Anda dapat mengarahkan ulang ke Wilayah lain dan melakukan pembacaan dan penulisan terhadap tabel replika yang berbeda. Anda dapat menerapkan logika bisnis khusus untuk menentukan kapan harus mengalihkan permintaan ke Wilayah lain.

Jika suatu Wilayah menjadi terisolasi atau terdegradasi, DynamoDB melacak setiap penulisan yang telah dilakukan tetapi belum disebarkan ke semua tabel replika. Setelah Wilayah kembali online, DynamoDB melanjutkan penyebaran penulisan yang tertunda dari Wilayah tersebut ke tabel replika di Wilayah lain. DynamoDB juga melanjutkan penyebaran penulisan dari tabel replika lain ke Wilayah yang saat ini kembali online. Semua aktivitas tulis yang berhasil sebelumnya akan disebarkan pada akhirnya, terlepas dari berapa lama Wilayah tersebut terisolasi.

Praktik terbaik dan persyaratan untuk mengelola tabel global

> **Important**
>
> Dokumentasi ini ditujukan untuk versi 2017.11.29 (Lama) tabel global, yang tidak boleh digunakan untuk tabel global baru. Pelanggan harus menggunakan Tabel Global versi 2019.11.21 (Saat Ini) jika memungkinkan, karena memberikan fleksibilitas yang lebih besar, efisiensi yang lebih tinggi, dan mengkonsumsi kapasitas tulis yang lebih sedikit daripada 2017.11.29 (Legacy).
>
> Untuk menentukan versi mana yang sedang Anda gunakan, lihat Menentukan versi tabel global DynamoDB yang Anda gunakan. Untuk memperbarui tabel global yang ada dari versi 2017.11.29 (Lama) ke versi 2019.11.21 (Terbaru), lihat Meningkatkan tabel global.

Menggunakan tabel global Amazon DynamoDB, Anda dapat mereplikasi data tabel Anda di seluruh Wilayah. AWS Tabel replika dan indeks sekunder di tabel global Anda harus memiliki pengaturan kapasitas tulis yang identik untuk memastikan replikasi data yang tepat.

Topik

- Versi tabel global
- Persyaratan untuk menambahkan tabel replika baru

- Praktik terbaik dan persyaratan untuk mengelola kapasitas

Versi tabel global

Ada dua versi tabel global DynamoDB yang tersedia: Tabel Global versi 2019.11.21 (Saat Ini) dan. Versi tabel global 2017.11.29 (Legacy) Pelanggan harus menggunakan Tabel Global versi 2019.11.21 (Saat Ini) jika memungkinkan, karena memberikan fleksibilitas yang lebih besar, efisiensi yang lebih tinggi, dan mengkonsumsi kapasitas tulis yang lebih sedikit daripada 2017.11.29 (Legacy).

Untuk menentukan versi mana yang sedang Anda gunakan, lihat Menentukan versi tabel global DynamoDB yang Anda gunakan. Untuk memperbarui tabel global yang ada dari Versi 2017.11.29 (Lama) ke Versi 2019.11.21 (Terbaru), lihat Meningkatkan tabel global.

Persyaratan untuk menambahkan tabel replika baru

Jika Anda ingin menambahkan tabel replika baru ke tabel global, masing-masing syarat berikut harus benar:

- Tabel harus memiliki kunci partisi yang sama seperti semua replika lainnya.
- Tabel harus memiliki pengaturan manajemen kapasitas tulis yang sama yang ditentukan.
- Tabel harus memiliki nama yang sama seperti semua replika lainnya.
- Tabel harus memiliki DynamoDB Streams yang diaktifkan, dengan aliran yang berisi gambar item baru dan lama.
- Tabel replika baru atau yang sudah ada di tabel global tidak boleh berisi data.

Jika indeks sekunder global ditentukan, syarat berikut juga harus dipenuhi:

- Indeks sekunder global harus memiliki nama yang sama.
- Indeks sekunder global harus memiliki kunci partisi yang sama dan kunci urutan (jika ada).

> **Important**
>
> Pengaturan kapasitas tulis harus diatur secara konsisten di seluruh tabel replika tabel global Anda dan indeks sekunder yang cocok. Untuk memperbarui pengaturan kapasitas tulis untuk tabel global Anda, sebaiknya Anda menggunakan konsol DynamoDB atau operasi API `UpdateGlobalTableSettings`. `UpdateGlobalTableSettings` menerapkan

perubahan pada pengaturan kapasitas tulis ke semua tabel replika dan indeks sekunder yang cocok dalam tabel global secara otomatis. Jika menggunakan operasi `UpdateTable`, `RegisterScalableTarget`, atau `PutScalingPolicy`, Anda harus menerapkan perubahan pada setiap tabel replika dan indeks sekunder yang cocok satu per satu. Untuk informasi selengkapnya, lihat UpdateGlobalTableSettingsdi Referensi Amazon DynamoDB API.

Sebaiknya Anda mengaktifkan penskalaan otomatis untuk mengelola pengaturan kapasitas tulis yang disediakan. Jika lebih memilih untuk mengelola pengaturan kapasitas tulis secara manual, Anda harus menyediakan unit kapasitas tulis yang direplikasi yang setara untuk semua tabel replika Anda. Sediakan juga unit kapasitas tulis yang direplikasi yang setara untuk indeks sekunder yang cocok di seluruh tabel global Anda.

Anda juga harus memiliki izin AWS Identity and Access Management (IAM) yang sesuai. Untuk informasi selengkapnya, lihat Menggunakan IAM dengan tabel global.

Praktik terbaik dan persyaratan untuk mengelola kapasitas

Pertimbangkan hal berikut ketika mengelola pengaturan kapasitas untuk replika tabel di DynamoDB.

Menggunakan penskalaan otomatis DynamoDB

Menggunakan penskalaan otomatis DynamoDB adalah cara yang direkomendasikan untuk mengelola pengaturan kapasitas throughput untuk tabel replika yang menggunakan mode ditetapkan. Penskalaan otomatis DynamoDB secara otomatis menyesuaikan RCUs unit kapasitas baca () dan WCUs unit kapasitas tulis () untuk setiap tabel replika berdasarkan beban kerja aplikasi Anda yang sebenarnya. Untuk informasi selengkapnya, lihat Mengelola kapasitas throughput secara otomatis dengan penskalaan otomatis DynamoDB.

Jika Anda membuat tabel replika menggunakan AWS Management Console, penskalaan otomatis diaktifkan secara default untuk setiap tabel replika, dengan pengaturan penskalaan otomatis default untuk mengelola unit kapasitas baca dan unit kapasitas tulis.

Perubahan pada pengaturan penskalaan otomatis untuk tabel replika atau indeks sekunder yang dibuat melalui konsol DynamoDB atau menggunakan panggilan `UpdateGlobalTableSettings` diterapkan untuk semua tabel replika dan indeks sekunder yang cocok dalam tabel global secara otomatis. Perubahan ini menimpa pengaturan penskalaan otomatis yang sudah ada. Hal ini memastikan pengaturan kapasitas tulis yang disediakan konsisten di seluruh tabel replika dan indeks sekunder di tabel global Anda. Jika menggunakan operasi panggilan `UpdateTable`,

`RegisterScalableTarget`, atau `PutScalingPolicy`, Anda harus menerapkan perubahan pada setiap tabel replika dan indeks sekunder yang cocok satu per satu.

> **Note**
>
> Jika penskalaan otomatis tidak memenuhi perubahan kapasitas aplikasi Anda (beban kerja yang tidak dapat diprediksi), atau jika Anda tidak ingin mengonfigurasi pengaturannya (pengaturan target untuk ambang batas minimum, maksimum, atau pemanfaatan), Anda dapat menggunakan mode sesuai permintaan untuk mengelola kapasitas tabel global. Untuk informasi selengkapnya, lihat Mode sesuai permintaan.
>
> Jika Anda mengaktifkan mode sesuai permintaan pada tabel global, konsumsi unit permintaan tulis yang direplikasi (rWCUs) akan konsisten dengan cara r WCUs disediakan. Sebagai contoh, jika Anda melakukan 10 aktivitas tulis ke tabel lokal yang direplikasi di dua Wilayah tambahan, Anda akan menggunakan 60 unit permintaan tulis (10 + 10 + 10 = 30; 30 x 2 = 60). 60 unit permintaan tulis yang dikonsumsi termasuk penulisan tambahan yang dikonsumsi oleh tabel global Versi 2017.11.29 (Lama) untuk memperbarui atribut `aws:rep:deleting`, `aws:rep:updatetime`, dan `aws:rep:updateregion`.

Mengelola kapasitas secara manual

Jika memutuskan untuk tidak menggunakan penskalaan otomatis DynamoDB, Anda harus secara manual menetapkan pengaturan kapasitas baca dan kapasitas tulis pada setiap tabel replika dan indeks sekunder.

Unit kapasitas tulis direplikasi yang disediakan (rWCUs) pada setiap tabel replika harus disetel ke jumlah total r yang WCUs diperlukan untuk penulisan aplikasi di semua Wilayah dikalikan dua. Hal ini mengakomodasi aktivitas tulis aplikasi yang terjadi di Wilayah lokal dan aktivitas tulis aplikasi yang direplikasi yang berasal dari Wilayah lain. Misalnya, anggaplah Anda mengharapkan 5 aktivitas tulis per detik untuk tabel replika Anda di Ohio dan 5 aktivitas tulis per detik untuk tabel replika Anda di Virginia Utara. Dalam hal ini, Anda harus menyediakan 20 r WCUs untuk setiap tabel replika (5 + 5 = 10; 10 x 2 = 20).

Untuk memperbarui pengaturan kapasitas tulis untuk tabel global Anda, sebaiknya Anda menggunakan konsol DynamoDB atau operasi API `UpdateGlobalTableSettings`. `UpdateGlobalTableSettings` menerapkan perubahan pada pengaturan kapasitas tulis ke semua tabel replika dan indeks sekunder yang cocok dalam tabel global secara otomatis. Jika menggunakan operasi `UpdateTable`, `RegisterScalableTarget`, atau `PutScalingPolicy`, Anda harus

menerapkan perubahan pada setiap tabel replika dan indeks sekunder yang cocok satu per satu. Untuk informasi selengkapnya, lihat Referensi Amazon DynamoDB API.

> **Note**
>
> Untuk memperbarui pengaturan (`UpdateGlobalTableSettings`) tabel global di DynamoDB, Anda harus memiliki izin `dynamodb:UpdateGlobalTable`, `dynamodb:DescribeLimits`, `application-autoscaling:DeleteScalingPolicy`, dan `application-autoscaling:DeregisterScalableTarget`. Untuk informasi selengkapnya, lihat Menggunakan IAM dengan tabel global.

Membuat tabel global

> **Important**
>
> Dokumentasi ini ditujukan untuk versi 2017.11.29 (Lama) tabel global, yang tidak boleh digunakan untuk tabel global baru. Pelanggan harus menggunakan Tabel Global versi 2019.11.21 (Saat Ini) jika memungkinkan, karena memberikan fleksibilitas yang lebih besar, efisiensi yang lebih tinggi, dan mengkonsumsi kapasitas tulis yang lebih sedikit daripada 2017.11.29 (Legacy).
>
> Untuk menentukan versi mana yang sedang Anda gunakan, lihat Menentukan versi tabel global DynamoDB yang Anda gunakan. Untuk memperbarui tabel global yang ada dari versi 2017.11.29 (Lama) ke versi 2019.11.21 (Terbaru), lihat Meningkatkan tabel global.

Bagian ini menjelaskan cara membuat tabel global menggunakan konsol Amazon DynamoDB atau AWS Command Line Interface ().AWS CLI

Topik

- Membuat tabel global (konsol)
- Membuat tabel global (AWS CLI)

Membuat tabel global (konsol)

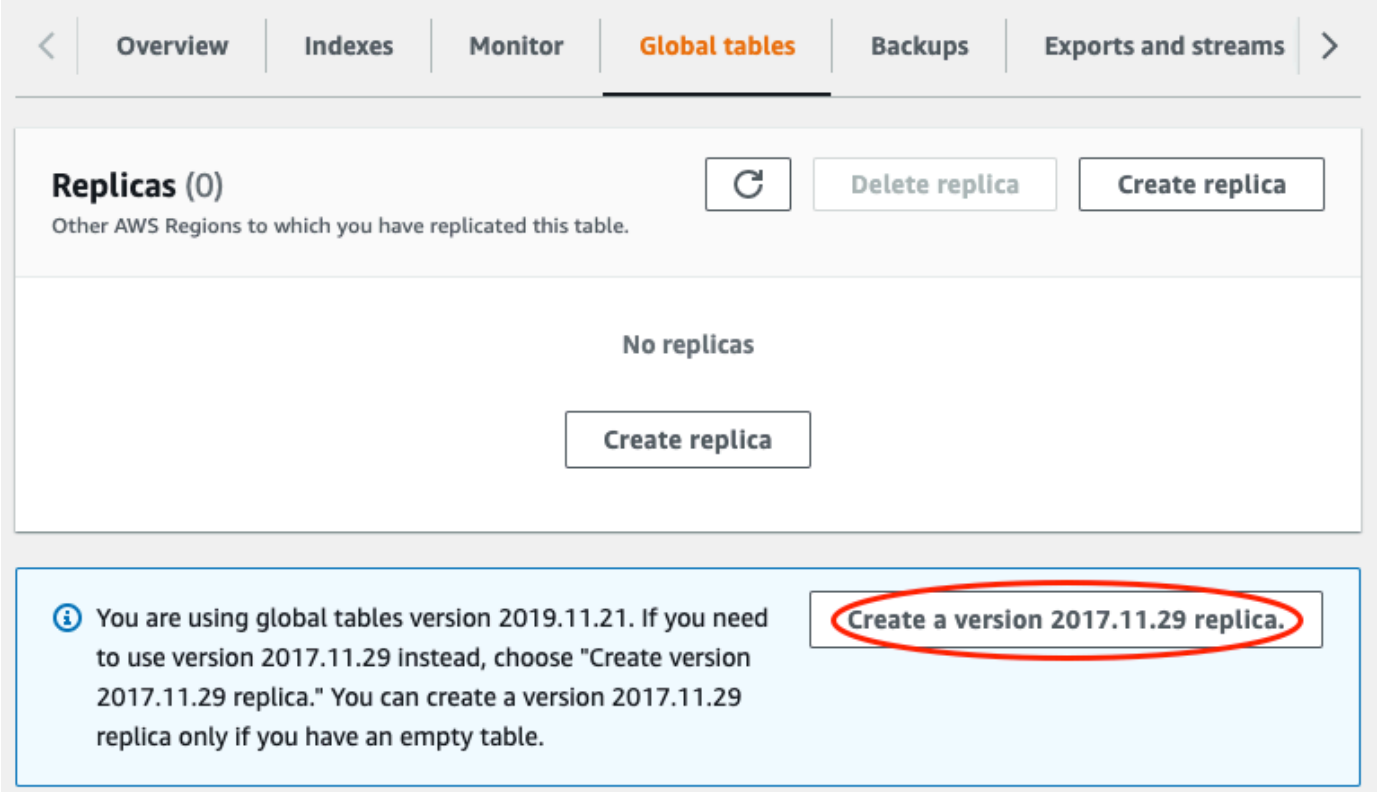
Ikuti langkah-langkah ini untuk membuat tabel global menggunakan konsol. Contoh berikut membuat tabel global dengan tabel replika di Amerika Serikat dan Eropa.

1. Buka konsol DynamoDB di rumah. https://console.aws.amazon.com/dynamodb/ Untuk contoh ini, pilih Wilayah us-east-2 (AS Timur Ohio) Region.
2. Di panel navigasi di sisi kiri konsol, pilih Tabel.
3. Pilih Buat Tabel.

   Untuk Nama tabel, masukkan **`Music`**.

   Untuk Kunci primer, masukkan **`Artist`**. Pilih Tambahkan kunci urutan, dan masukkan **`SongTitle`**. (**`Artist`** dan **`SongTitle`** harus berupa string.)

   Untuk membuat tabel, pilih Buat. Tabel ini berfungsi sebagai tabel replika pertama dalam tabel global baru. Ini adalah protojenis untuk tabel replika lain yang Anda tambahkan nantinya.
4. Pilih tab Tabel Global, lalu pilih Buat replika Versi 2017.11.29 (Lama).



5. Dari dropdown Wilayah replikasi yang Tersedia, pilih AS Barat (Oregon).

   Konsol memeriksa untuk memastikan bahwa tidak ada tabel dengan nama yang sama di Wilayah yang dipilih. Jika ada tabel dengan nama yang sama, Anda harus menghapus tabel yang ada sebelum dapat membuat tabel replika baru di Wilayah tersebut.
6. Pilih Buat Replika. Langkah ini memulai proses pembuatan tabel di AS Barat (Oregon);.

Tab Tabel Global untuk tabel yang dipilih (dan untuk setiap tabel replika lainnya) menunjukkan bahwa tabel telah direplikasi di beberapa Wilayah.

7. Sekarang tambahkan Wilayah lain sehingga tabel global Anda direplikasi dan disinkronkan di seluruh Amerika Serikat dan Eropa. Untuk melakukannya, ulangi langkah 5, tetapi kali ini tentukan Eropa (Frankfurt), bukan AS Barat (Oregon).
8. Anda masih harus menggunakan AWS Management Console di Wilayah Timur AS (Ohio). Pilih Item di menu navigasi kiri, pilih tabel Musik, lalu pilih Buat Item.

   a. Untuk Artist, masukkan **`item_1`**.
   b. Untuk SongTitle, masukkan **`Song Value 1`**.
   c. Untuk menulis item, pilih Buat item.
9. Setelah beberapa saat, item direplikasi di ketiga Wilayah tabel global Anda. Untuk memverifikasi ini, di pemilih Wilayah di sudut kanan atas di konsol, pilih Eropa (Frankfurt). Tabel `Music` di Eropa (Frankfurt) akan berisi item baru.
10. Ulangi langkah 9 dan pilih AS Barat (Oregon) untuk memverifikasi replikasi di wilayah itu.

Membuat tabel global (AWS CLI)

Ikuti langkah-langkah ini untuk membuat tabel global `Music` menggunakan AWS CLI. Contoh berikut membuat tabel global, dengan tabel replika di Amerika Serikat dan Eropa.

1. Buat tabel baru (`Music`) di AS Timur (Ohio), dengan DynamoDB Streams diaktifkan (`NEW_AND_OLD_IMAGES`).

```
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. Buat tabel `Music` identik di AS Timur (Virginia Utara).

```
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. Buat tabel global (`Music`) yang terdiri dari tabel replika di Wilayah `us-east-2` dan `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**
>
> Nama tabel global (`Music`) harus cocok dengan nama setiap tabel replika (`Music`). Untuk informasi selengkapnya, lihat Praktik terbaik dan persyaratan untuk mengelola tabel global.

4. Buat tabel lain di Eropa (Irlandia), dengan pengaturan yang sama seperti yang Anda buat pada langkah 1 dan langkah 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
```

Setelah melakukan langkah ini, tambahkan tabel baru untuk tabel global `Music`.

```
aws dynamodb update-global-table \
    --global-table-name Music \
    --replica-updates 'Create={RegionName=eu-west-1}' \
    --region us-east-2
```

5. Untuk memverifikasi bahwa replikasi berfungsi, tambahkan item baru ke tabel `Music` di AS Timur (Ohio).

```
aws dynamodb put-item \
    --table-name Music \
    --item '{"Artist": {"S":"item_1"},"SongTitle": {"S":"Song Value 1"}}' \
    --region us-east-2
```

6. Tunggu beberapa detik, lalu periksa untuk melihat apakah item tersebut telah berhasil direplikasi ke AS Timur (Virginia Utara) dan Eropa (Irlandia).

```
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
```

Memantau tabel global

> **Important**
>
> Dokumentasi ini ditujukan untuk versi 2017.11.29 (Lama) tabel global, yang tidak boleh digunakan untuk tabel global baru. Pelanggan harus menggunakan Tabel Global versi 2019.11.21 (Saat Ini) jika memungkinkan, karena memberikan fleksibilitas yang lebih besar, efisiensi yang lebih tinggi, dan mengkonsumsi kapasitas tulis yang lebih sedikit daripada 2017.11.29 (Legacy).

Untuk menentukan versi mana yang sedang Anda gunakan, lihat Menentukan versi tabel global DynamoDB yang Anda gunakan. Untuk memperbarui tabel global yang ada dari versi 2017.11.29 (Lama) ke versi 2019.11.21 (Terbaru), lihat Meningkatkan tabel global.

Anda dapat menggunakan Amazon CloudWatch untuk memantau perilaku dan kinerja tabel global. Amazon DynamoDB menerbitkan metrik `ReplicationLatency` dan `PendingReplicationCount` untuk setiap replika dalam tabel global.

- **`ReplicationLatency`**—Waktu yang berlalu antara saat item yang diperbarui muncul dalam aliran DynamoDB untuk satu tabel replika, dan saat item tersebut muncul di replika lain dalam tabel global. `ReplicationLatency` dinyatakan dalam milidetik dan dipancarkan untuk setiap pasangan Wilayah sumber dan tujuan.

  Selama operasi normal, `ReplicationLatency` akan cukup konstan. Nilai `ReplicationLatency` yang tinggi dapat menunjukkan bahwa pembaruan dari satu replika tidak disebarkan ke tabel replika lain pada waktu yang tepat. Seiring waktu, hal ini dapat mengakibatkan tabel replika lainnya tertinggal karena tidak lagi menerima pembaruan secara konsisten. Dalam hal ini, Anda harus memverifikasi bahwa unit kapasitas baca (RCUs) dan unit kapasitas tulis (WCUs) identik untuk masing-masing tabel replika. Selain itu, saat memilih pengaturan WCU, ikuti rekomendasi di Versi tabel global.

  `ReplicationLatency`dapat meningkat jika suatu AWS Wilayah menjadi terdegradasi dan Anda memiliki tabel replika di Wilayah itu. Dalam hal ini, Anda dapat mengalihkan sementara aktivitas baca dan tulis aplikasi Anda ke AWS Wilayah yang berbeda.
- **`PendingReplicationCount`**—Jumlah pembaruan item yang ditulis ke satu tabel replika, tetapi belum ditulis ke replika lain dalam tabel global. `PendingReplicationCount` dinyatakan dalam jumlah item dan dipancarkan untuk setiap pasangan Wilayah sumber dan tujuan.

  Selama operasi normal, `PendingReplicationCount` akan sangat rendah. Jika `PendingReplicationCount` meningkat untuk jangka waktu yang lama, selidiki apakah pengaturan kapasitas tulis yang disediakan tabel replika Anda mencukupi untuk beban kerja saat ini.

  `PendingReplicationCount`dapat meningkat jika suatu AWS Wilayah menjadi terdegradasi dan Anda memiliki tabel replika di Wilayah itu. Dalam kasus ini, Anda dapat mengalihkan aktivitas baca dan tulis aplikasi Anda ke Wilayah AWS lain untuk sementara waktu.

Untuk informasi selengkapnya, lihat Dimensi dan Metrik DynamoDB.

Menggunakan IAM dengan tabel global

> **Important**
>
> Dokumentasi ini ditujukan untuk versi 2017.11.29 (Lama) tabel global, yang tidak boleh digunakan untuk tabel global baru. Pelanggan harus menggunakan Tabel Global versi 2019.11.21 (Saat Ini) jika memungkinkan, karena memberikan fleksibilitas yang lebih besar, efisiensi yang lebih tinggi, dan mengkonsumsi kapasitas tulis yang lebih sedikit daripada 2017.11.29 (Legacy).
> Untuk menentukan versi mana yang sedang Anda gunakan, lihat Menentukan versi tabel global DynamoDB yang Anda gunakan. Untuk memperbarui tabel global yang ada dari versi 2017.11.29 (Lama) ke versi 2019.11.21 (Terbaru), lihat Meningkatkan tabel global.

Ketika Anda membuat tabel global untuk pertama kalinya, Amazon DynamoDB otomatis membuat peran tertaut layanan AWS Identity and Access Management (IAM) untuk Anda. Peran ini bernama `AWSServiceRoleForDynamoDBReplication`, dan memungkinkan DynamoDB untuk mengelola replikasi lintas-Wilayah untuk tabel global atas nama Anda. Jangan hapus peran tertaut layanan ini. Jika Anda melakukannya, semua tabel global Anda tidak akan berfungsi lagi.

Untuk informasi selengkapnya tentang peran tertaut layanan, lihat Menggunakan peran tertaut layanan di Panduan Pengguna IAM.

Untuk membuat dan memelihara tabel global di DynamoDB, Anda harus memiliki izin `dynamodb:CreateGlobalTable` untuk mengakses berikut ini:

- Tabel replika yang ingin Anda tambahkan.
- Setiap replika yang ada yang sudah menjadi bagian dari tabel global.
- Tabel global itu sendiri.

Untuk memperbarui pengaturan (`UpdateGlobalTableSettings`) tabel global di DynamoDB, Anda harus memiliki izin `dynamodb:UpdateGlobalTable`, `dynamodb:DescribeLimits`, `application-autoscaling:DeleteScalingPolicy`, dan `application-autoscaling:DeregisterScalableTarget`.

Izin `application-autoscaling:DeleteScalingPolicy` dan `application-autoscaling:DeregisterScalableTarget` diperlukan saat memperbarui kebijakan penskalaan yang ada. Hal ini agar layanan tabel global dapat menghapus kebijakan penskalaan lama sebelum melampirkan kebijakan baru ke tabel atau indeks sekunder.

Jika menggunakan kebijakan IAM untuk mengelola akses ke satu tabel replika, Anda harus menerapkan kebijakan yang identik pada semua replika lain dalam tabel global. Praktik ini membantu Anda mempertahankan model izin yang konsisten di semua tabel replika.

Menggunakan kebijakan IAM yang identik pada semua replika dalam tabel global, Anda juga dapat menghindari pemberian akses baca dan tulis yang tidak diinginkan ke data tabel global Anda. Misalnya, pertimbangkan pengguna yang hanya memiliki akses ke satu replika dalam tabel global. Jika pengguna tersebut dapat menulis ke replika ini, DynamoDB menyebarkan aktivitas tulis ini ke semua tabel replika lainnya. Akibatnya, pengguna tersebut (secara tidak langsung) dapat menulis ke semua replika lain dalam tabel global. Skenario ini dapat dihindari menggunakan kebijakan IAM yang konsisten pada semua tabel replika.

Contoh: Izinkan CreateGlobalTable tindakan

Sebelum dapat menambahkan replika ke tabel global, Anda harus memiliki izin `dynamodb:CreateGlobalTable` untuk tabel global dan untuk masing-masing tabel replikanya.

Kebijakan IAM berikut memberikan izin untuk memungkinkan tindakan `CreateGlobalTable` pada semua tabel.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": ["dynamodb:CreateGlobalTable"],
            "Resource": "*"
        }
    ]
}
```

Contoh: Izinkan UpdateGlobalTable,, application-autoscaling: DescribeLimits, dan application-autoscaling: actions DeleteScalingPolicy DeregisterScalableTarget

Untuk memperbarui pengaturan (`UpdateGlobalTableSettings`) tabel global di DynamoDB, Anda harus memiliki izin `dynamodb:UpdateGlobalTable`, `dynamodb:DescribeLimits`, `application-autoscaling:DeleteScalingPolicy`, dan `application-autoscaling:DeregisterScalableTarget`.

Kebijakan IAM berikut memberikan izin untuk memungkinkan tindakan `UpdateGlobalTableSettings` pada semua tabel.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "dynamodb:UpdateGlobalTable",
                "dynamodb:DescribeLimits",
                "application-autoscaling:DeleteScalingPolicy",
                "application-autoscaling:DeregisterScalableTarget"
            ],
            "Resource": "*"
        }
    ]
}
```

Contoh: Izinkan CreateGlobalTable tindakan untuk nama tabel global tertentu dengan replika yang diizinkan di wilayah tertentu saja

Kebijakan IAM berikut memberikan izin untuk memungkinkan tindakan `CreateGlobalTable` membuat tabel global bernama `Customers` dengan replika di dua Wilayah.

```
{
    "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"
            ]
        }
    ]
}
```

Versi API DynamoDB tingkat rendah sebelumnya (2011-12-05)

Bagian ini mendokumentasikan operasi yang tersedia dalam versi API tingkat rendah DynamoDB sebelumnya (2011-12-05). Versi API tingkat rendah ini dipertahankan untuk kompatibilitas mundur dengan aplikasi yang ada.

Aplikasi baru harus menggunakan versi API terkini (2012-08-10). Untuk informasi selengkapnya, lihat Referensi API DynamoDB.

> **Note**
>
> Sebaiknya Anda memigrasikan aplikasi Anda ke API versi terbaru (2012-08-10), karena fitur DynamoDB baru tidak akan di-backport ke versi API sebelumnya.

Topik

- BatchGetItem
- BatchWriteItem
- CreateTable
- DeleteItem
- DeleteTable
- DescribeTables
- GetItem
- ListTables
- PutItem
- Kueri
- Scan
- UpdateItem
- UpdateTable

BatchGetItem

> **Important**
>
> ***This section refers to API version 2011-12-05, which is deprecated and should not be used for new applications.***
>
> Untuk dokumentasi tentang API tingkat rendah saat ini, lihat Referensi API Amazon DynamoDB.

Deskripsi

Operasi `BatchGetItem` mengembalikan atribut untuk beberapa item dari sejumlah tabel menggunakan kunci primernya. Jumlah maksimum item yang dapat diambil untuk suatu operasi adalah 100. Selain itu, jumlah item yang diambil dibatasi pada ukuran 1 MB. Jika batas ukuran respons terlampaui atau hasil parsial dikembalikan karena throughput yang disediakan tabel terlampaui, atau karena kegagalan pemrosesan internal, DynamoDB mengembalikan nilai `UnprocessedKeys` agar Anda dapat mengulangi operasi, dimulai dengan item berikutnya untuk didapatkan. DynamoDB secara otomatis menyesuaikan jumlah item yang dikembalikan per halaman untuk memberlakukan batas ini. Misalnya, meskipun Anda meminta untuk mengambil 100 item, tetapi setiap item berukuran 50 KB, sistem akan mengembalikan 20 item dan nilai `UnprocessedKeys` yang sesuai agar Anda dapat memperoleh halaman hasil berikutnya. Jika diinginkan, aplikasi Anda dapat menyertakan logikanya sendiri untuk menyusun halaman hasil menjadi satu set.

Jika tidak ada item yang dapat diproses karena throughput yang disediakan tidak memadai pada setiap tabel yang terlibat dalam permintaan, DynamoDB akan mengembalikan kesalahan `ProvisionedThroughputExceededException`.

> **Note**
>
> Secara default, `BatchGetItem` melakukan bacaan akhir konsisten di setiap tabel dalam permintaan. Anda dapat mengatur parameter `ConsistentRead` menjadi `true`, sesuai tiap tabelnya, jika Anda ingin pembacaan yang konsisten.
> `BatchGetItem` mengambil item secara paralel untuk meminimalkan latensi respons.
> Saat merancang aplikasi Anda, ingatlah bahwa DynamoDB tidak menjamin bahwa pengurutan atribut dilakukan sesuai respons yang dikembalikan. Sertakan nilai kunci primer di `AttributesToGet` untuk item dalam permintaan Anda guna membantu menguraikan respons berdasarkan item.

Jika item yang diminta tidak ada, respons untuk item tersebut tidak mengembalikan apa pun. Permintaan untuk item yang tidak ada menggunakan unit kapasitas baca minimum sesuai dengan jenis pembacaan. Untuk informasi selengkapnya, lihat Ukuran dan format item DynamoDB.

Permintaan

Sintaks

```
// This header is abbreviated. For a sample of a complete header, see API tingkat
 rendah DynamoDB.
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"]
        }
    }
}
```

Nama	Deskripsi	Wajib
`RequestItems`	Sebuah kontainer nama tabel dan item yang sesuai untuk mendapatkan kunci primer. Saat meminta item, setiap	Ya

Nama	Deskripsi	Wajib
	nama tabel hanya dapat diinvokasi sekali per operasi. Jenis: String Default: Tidak Ada	
`Table`	Nama tabel yang berisi item untuk didapatkan. Entri hanyalah string yang menentukan tabel yang sudah ada tanpa label. Jenis: String Default: Tidak Ada	Ya
`Table:Keys`	Nilai kunci primer yang menentukan item dalam tabel yang ditentukan. Untuk informasi selengkapnya tentang kunci primer, lihat Kunci primer. Jenis: Kunci	Ya
`Table:AttributesToGet`	Array Nama atribut dalam tabel yang ditentukan. Jika nama atribut tidak ditentukan, semua atribut akan dikembali kan. Jika beberapa atribut tidak ditemukan, atribut tersebut tidak akan muncul dalam hasil. Jenis: Array	Tidak

Nama	Deskripsi	Wajib
`Table:ConsistentRead`	Jika diatur ke `true`, bacaan yang konsisten akan diterbitkan. Jika tidak, bacaan akhir konsisten akan digunakan. Jenis: Boolean	Tidak

Respons

Sintaks

```
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"]}
        }
}
```

Nama	Deskripsi
`Responses`	Nama tabel dan setiap atribut item dari tabel. Jenis: Peta
`Table`	Nama tabel yang berisi item. Entri hanyalah string yang menentukan tabel tanpa label. Jenis: String
`Items`	Kontainer untuk nama dan nilai atribut yang memenuhi parameter operasi. Jenis: Peta nama atribut serta nilai dan jenis data.
`ConsumedCapacityUnits`	Jumlah unit kapasitas baca yang dikonsumsi, untuk setiap tabel. Nilai ini menunjukkan jumlah yang diterapkan pada throughput yang disediakan. Permintaan untuk item yang tidak ada yang mengonsumsi unit kapasitas baca minimum, tergantung jenis baca. Untuk informasi selengkapnya, lihat Mode kapasitas yang disediakan DynamoDB.

Nama	Deskripsi
	Jenis: Angka
`UnprocessedKeys`	Berisi array tabel dan masing-masing kuncinya yang tidak diproses dengan respons terkini, kemungkinan karena mencapai batas ukuran respons. Nilai `UnprocessedKeys` memiliki format yang sama dengan parameter `RequestItems` (sehingga nilai dapat diberikan langsung ke operasi `BatchGetItem` berikutnya). Untuk informasi selengkapnya, lihat parameter `RequestItems` di atas. Jenis: Array
`UnprocessedKeys` : `Table`: `Keys`	Nilai atribut kunci primer yang menentukan item dan atribut yang terkait dengan item. Untuk informasi selengkapnya tentang kunci primer, lihat Kunci primer. Jenis: Array pasangan nama-nilai atribut.
`UnprocessedKeys` : `Table`: `AttributesToGet`	Nama atribut dalam tabel yang ditentukan. Jika nama atribut tidak ditentukan, semua atribut akan dikembalikan. Jika beberapa atribut tidak ditemukan, atribut tersebut tidak akan muncul dalam hasil. Jenis: Array nama atribut.
`UnprocessedKeys` : `Table`: `ConsistentRead`	Jika diatur menjadi `true`, bacaan konsisten digunakan untuk tabel tertentu. Jika tidak, bacaan akhir konsisten akan digunakan. Jenis: Boolean.

Kesalahan khusus

Kesalahan	Deskripsi
`ProvisionedThroughputExceededException`	Throughput tersedia maksimum milik Anda telah terlampaui.

Contoh

Contoh berikut menunjukkan permintaan HTTP POST dan respon menggunakan BatchGetItem operasi. Untuk contoh menggunakan AWS SDK, lihatBekerja dengan item dan atribut di DynamoDB.

Permintaan sampel

Contoh berikut meminta atribut dari dua tabel yang berbeda.

```
// This header is abbreviated.
// For a sample of a complete header, see API tingkat rendah DynamoDB.
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"]}
    }
}
```

Respons sampel

Sampel berikut ini adalah responsnya.

```
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**
>
> ***This section refers to API version 2011-12-05, which is deprecated and should not be used for new applications.***
>
> Untuk dokumentasi tentang API tingkat rendah saat ini, lihat Referensi API Amazon DynamoDB.

Deskripsi

Operasi ini memungkinkan Anda untuk menempatkan atau menghapus beberapa item di beberapa tabel dalam satu panggilan.

Untuk mengunggah satu item, Anda dapat menggunakan `PutItem`, dan untuk menghapus satu item, Anda dapat menggunakan `DeleteItem`. Namun, jika Anda ingin mengunggah atau menghapus sejumlah besar data, seperti mengunggah sejumlah besar data dari Amazon EMR (Amazon EMR)

atau memigrasi data dari basis data lain ke DynamoDB, `BatchWriteItem` menawarkan alternatif yang efisien.

Jika Anda menggunakan bahasa seperti Java, Anda dapat menggunakan thread untuk mengunggah item secara paralel. Hal ini menambah kompleksitas dalam aplikasi Anda untuk menangani thread. Bahasa lain tidak mendukung threading. Misalnya, jika Anda menggunakan PHP, Anda harus mengunggah atau menghapus item satu per satu. Dalam kedua situasi tersebut, `BatchWriteItem` menyediakan alternatif, yaitu pemrosesan operasi penempatan dan penghapusan yang ditentukan dilakukan secara paralel, sehingga bisa melakukan pendekatan thread pool tanpa harus menambah kompleksitas dalam aplikasi Anda.

Perhatikan bahwa setiap penempatan dan penghapusan individual yang ditentukan dalam operasi `BatchWriteItem` memiliki biaya yang sama dalam hal unit kapasitas yang dikonsumsi. Namun, karena `BatchWriteItem` melakukan operasi yang ditentukan secara paralel, Anda mendapatkan latensi yang lebih rendah. Operasi hapus pada item yang tidak ada mengonsumsi 1 unit kapasitas tulis. Untuk informasi selengkapnya tentang unit kapasitas yang dikonsumsi, lihat Bekerja dengan tabel dan data di DynamoDB.

Saat menggunakan `BatchWriteItem`, perhatikan batasan berikut:

- Operasi maksimum dalam satu permintaan—Anda dapat menentukan total hingga 25 operasi penempatan atau penghapusan; tetapi, ukuran total permintaan tidak boleh melebihi 1 MB (payload HTTP).
- Anda dapat menggunakan operasi `BatchWriteItem` hanya untuk menempatkan dan menghapus item. Anda tidak dapat menggunakannya untuk memperbarui item yang ada.
- Bukan operasi atomik—Operasi individual yang ditentukan dalam `BatchWriteItem` bersifat atomik; tetapi `BatchWriteItem` secara keseluruhan adalah operasi "upaya terbaik" dan bukan operasi atomik. Artinya, dalam permintaan `BatchWriteItem`, beberapa operasi mungkin berhasil dan yang lainnya mungkin gagal. Operasi yang gagal dikembalikan ke bidang `UnprocessedItems` dalam respons. Sejumlah kegagalan ini mungkin terjadi karena Anda melampaui throughput yang disediakan dan dikonfigurasi untuk tabel atau kegagalan sementara seperti kesalahan jaringan. Anda dapat menyelidiki dan mengirim ulang permintaan (opsional). Biasanya, Anda memanggil `BatchWriteItem` dalam satu loop serta di setiap pemeriksaan pengulangan untuk item yang belum diproses, dan mengirimkan permintaan `BatchWriteItem` baru dengan item yang belum diproses tersebut.
- Tidak mengembalikan item apa pun— `BatchWriteItem` dirancang untuk mengunggah data dalam jumlah besar secara efisien. Hal itu tidak menyediakan sejumlah kecanggihan yang

ditawarkan oleh `PutItem` dan `DeleteItem`. Sebagai contoh, `DeleteItem` mendukung bidang `ReturnValues` dalam isi permintaan Anda untuk meminta item yang dihapus dalam respons. Operasi `BatchWriteItem` tidak mengembalikan item apa pun dalam respons.

- Tidak seperti `PutItem` dan `DeleteItem`, `BatchWriteItem` tidak memungkinkan Anda untuk menentukan syarat pada permintaan tulis individu dalam operasi.
- Nilai atribut tidak boleh null; atribut jenis string dan biner harus memiliki panjang lebih dari nol; dan atribut jenis set tidak boleh kosong. Permintaan yang memiliki nilai kosong akan ditolak dengan `ValidationException`.

DynamoDB menolak keseluruhan batch operasi penulisan jika salah satu dari yang berikut ini true:

- Jika satu atau beberapa tabel yang ditentukan dalam permintaan `BatchWriteItem` tidak ada.
- Jika atribut kunci primer yang ditentukan pada item dalam permintaan tidak cocok dengan skema kunci primer dalam tabel yang sesuai.
- Jika Anda mencoba untuk melakukan beberapa operasi pada item yang sama dalam permintaan `BatchWriteItem` yang sama. Misalnya, Anda tidak dapat menempatkan dan menghapus item yang sama dalam permintaan `BatchWriteItem` yang sama.
- Jika total ukuran permintaan melampaui batas ukuran permintaan 1 MB (payload HTTP).
- Jika setiap item individu dalam suatu batch melampaui batas ukuran item 64 KB.

Permintaan

Sintaks

```
// This header is abbreviated. For a sample of a complete header, see API tingkat
 rendah DynamoDB.
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", ... ]
}
```

Dalam isi permintaan, objek JSON `RequestItems` menggambarkan operasi yang ingin Anda lakukan. Operasi dikelompokkan berdasarkan tabel. Anda dapat menggunakan `BatchWriteItem` untuk memperbarui atau menghapus beberapa item di beberapa tabel. Untuk setiap permintaan penulisan khusus, Anda harus mengidentifikasi jenis permintaan (`PutItem`, `DeleteItem`) diikuti dengan informasi terperinci tentang operasi tersebut.

- Untuk `PutRequest`, Anda memberikan item, yaitu daftar atribut dan nilai-nilainya.
- Untuk `DeleteRequest`, Anda memberikan nama dan nilai kunci primernya.

Respons

Sintaks

Berikut ini adalah sintaksis dari isi JSON yang dikembalikan dalam respons.

```
{
  "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.
```

Kesalahan khusus

Tidak ada kesalahan khusus untuk operasi ini.

Contoh

Contoh berikut menunjukkan permintaan HTTP POST dan respons operasi `BatchWriteItem`. Permintaan tersebut menentukan operasi berikut pada tabel Reply dan Thread:

- Menempatkan item dan menghapus item dari tabel Reply
- Menempatkan item ke dalam tabel Thread

Untuk contoh menggunakan AWS SDK, lihatBekerja dengan item dan atribut di DynamoDB.

Permintaan sampel

```
// This header is abbreviated. For a sample of a complete header, see API tingkat
 rendah DynamoDB.
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"
            }
          }
        }
      }
    ]
  }
}
```

Respons sampel

Respons contoh berikut menunjukkan operasi penempatan pada tabel Thread dan Reply yang berhasil dan operasi penghapusan pada tabel Reply yang gagal (karena alasan seperti throttling yang disebabkan terlampauinya throughput yang disediakan di tabel). Perhatikan hal berikut dalam respons JSON:

- Objek `Responses` menunjukkan satu unit kapasitas yang digunakan pada tabel `Thread` dan `Reply` sebagai hasil dari kesuksesan operasi penempatan pada setiap tabel ini.
- Objek `UnprocessedItems` menunjukkan kegagalan operasi penghapusan pada tabel `Reply`. Anda kemudian dapat mengeluarkan panggilan `BatchWriteItem` baru untuk mengatasi permintaan yang belum diproses ini.

```
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**
>
> ***This section refers to API version 2011-12-05, which is deprecated and should not be used for new applications.***
>
> Untuk dokumentasi tentang API tingkat rendah saat ini, lihat Referensi API Amazon DynamoDB.

Deskripsi

Operasi `CreateTable` menambahkan tabel baru ke akun Anda.

Nama tabel harus unik di antara yang terkait dengan AWS Akun yang mengeluarkan permintaan, dan AWS wilayah yang menerima permintaan (seperti dynamodb.us-west-2.amazonaws.com). Setiap titik akhir DynamoDB bersifat independen sepenuhnya. Misalnya, jika Anda memiliki dua tabel yang disebut "MyTable," satu di dynamodb.us-west-2.amazonaws.com dan satu di dynamodb.us-west-1.amazonaws.com, mereka sepenuhnya independen dan tidak berbagi data apa pun.

Operasi `CreateTable` memicu alur kerja asinkron untuk mulai membuat tabel. DynamoDB segera mengembalikan status tabel (`CREATING`) hingga tabel berada dalam status `ACTIVE`. Setelah tabel berada dalam status `ACTIVE`, Anda dapat melakukan operasi bidang data.

Gunakan operasi DescribeTables untuk memeriksa status tabel.

Permintaan

Sintaks

```
// This header is abbreviated.
// For a sample of a complete header, see API tingkat rendah DynamoDB.
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}
}
```

Nama	Deskripsi	Wajib
`TableName`	Nama tabel yang akan dibuat. Karakter yang diperbolehkan adalah a-z, A-Z, 0-9, '_' (garis bawah), '-' (tanda pisah), dan '.' (titik). Nama bisa berisi antara 3 dan 255 karakter. Jenis: String	Ya
`KeySchema`	Struktur kunci primer (sederhana atau komposit) untuk tabel. Pasangan nama-nilai untuk `HashKeyElement` diperlukan, dan pasangan nama-nilai untuk `RangeKeyElement` bersifat opsional (hanya diperlukan untuk kunci primer komposit). Untuk informasi selengkapnya tentang kunci primer, lihat Kunci primer. Nama elemen kunci primer dapat berisi antara 1 dan 255 karakter tanpa batasan karakter.	Ya

Nama	Deskripsi	Wajib
	Nilai yang mungkin untuk AttributeType adalah "S" (string), "N" (numerik), atau "B" (biner). Jenis: Peta `HashKeyElement` , atau `HashKeyElement` dan `RangeKeyElement` untuk kunci primer komposit.	
`ProvisionedThroughput`	Throughput baru untuk tabel yang ditentukan, yang terdiri dari nilai-nilai untuk `ReadCapacityUnits` dan `WriteCapacityUnits` . Untuk detailnya, lihat Mode kapasitas yang disediakan DynamoDB. **Note** Untuk nilai maksimum/minimum saat ini, lihat Kuota di Amazon DynamoDB. Jenis: Array	Ya

Nama	Deskripsi	Wajib
`ProvisionedThroughput : ReadCapacityUnits`	Menetapkan jumlah minimum `ReadCapacityUnits` yang konsisten digunakan per detik untuk tabel yang ditentukan sebelum DynamoDB menyeimbangkan beban dengan operasi lainnya. Operasi bacaan akhir konsisten memerlukan lebih sedikit usaha daripada operasi baca konsisten, jadi pengaturan 50 `ReadCapacityUnits` yang konsisten per detik akan menghasilkan 100 `ReadCapacityUnits` akhir konsisten per detik. Jenis: Angka	Ya
`ProvisionedThroughput : WriteCapacityUnits`	Menetapkan jumlah minimum `WriteCapacityUnits` yang digunakan per detik untuk tabel yang ditentukan sebelum DynamoDB menyeimbangkan beban dengan operasi lainnya. Jenis: Angka	Ya

Respons

Sintaks

```
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"
    }
}
```

Nama	Deskripsi
`TableDescription`	Kontainer untuk properti tabel.
`CreationDateTime`	Tanggal ketika tabel dibuat di jangka waktu UNIX. Jenis: Angka
`KeySchema`	Struktur kunci primer (sederhana atau komposit) untuk tabel. Pasangan nama-nilai untuk `HashKeyElement` diperlukan, dan pasangan nama-nilai untuk `RangeKeyElement` bersifat opsional (hanya diperlukan untuk kunci primer komposit). Untuk informasi selengkapnya tentang kunci primer, lihat Kunci primer. Jenis: Peta `HashKeyElement`, atau `HashKeyElement` dan `RangeKeyElement` untuk kunci primer komposit.
`ProvisionedThroughput`	Throughput untuk tabel yang ditentukan, yang terdiri dari nilai-nilai untuk `ReadCapac`

Nama	Deskripsi
	`ityUnits` dan `WriteCapacityUnits`. Lihat Mode kapasitas yang disediakan DynamoDB. Jenis: Array
`ProvisionedThroughput:ReadCapacityUnits`	Jumlah minimum `ReadCapacityUnits` yang digunakan per detik sebelum DynamoDB menyeimbangkan beban dengan operasi lainnya Jenis: Angka
`ProvisionedThroughput:WriteCapacityUnits`	Jumlah minimum `ReadCapacityUnits` yang digunakan per detik sebelum `WriteCapacityUnits` menyeimbangkan beban dengan operasi lainnya Jenis: Angka
`TableName`	Nama tabel yang dibuat. Jenis: String
`TableStatus`	Status tabel saat ini (`CREATING`). Setelah tabel berada dalam status `ACTIVE`, Anda dapat menempatkan data di dalamnya. Gunakan API DescribeTables untuk memeriksa status tabel. Jenis: String

Kesalahan khusus

Kesalahan	Deskripsi
`ResourceInUseException`	Mencoba untuk membuat kembali tabel yang sudah ada.
`LimitExceededException`	Jumlah permintaan tabel simultan (jumlah kumulatif tabel dalam status `CREATING`, `DELETING` atau `UPDATING`) melampaui jumlah maksimum yang diizinkan. **Note** Untuk nilai maksimum/minimum saat ini, lihat Kuota di Amazon DynamoDB. .

Contoh

Contoh berikut membuat tabel dengan kunci primer komposit yang berisi string dan angka. Untuk contoh menggunakan AWS SDK, lihatBekerja dengan tabel dan data di DynamoDB.

Permintaan sampel

```
// This header is abbreviated.
// For a sample of a complete header, see API tingkat rendah DynamoDB.
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}
```

```
}
```

Respons sampel

```
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"
    }
}
```

Tindakan terkait

- DescribeTables
- DeleteTable

DeleteItem

> **Important**
>
> ***This section refers to API version 2011-12-05, which is deprecated and should not be used for new applications.***
>
> Untuk dokumentasi tentang API tingkat rendah saat ini, lihat Referensi API Amazon DynamoDB.

Deskripsi

Menghapus satu item dalam tabel dengan kunci primer. Anda dapat melakukan operasi penghapusan bersyarat yang menghapus item jika ada, atau jika item memiliki nilai atribut yang diharapkan.

> **Note**
>
> Jika Anda menentukan `DeleteItem` tanpa atribut atau nilai, semua atribut untuk item tersebut akan dihapus.
> Kecuali jika Anda menentukan syarat, `DeleteItem` adalah operasi idempotensi; menjalankannya beberapa kali pada item atau atribut yang sama tidak menghasilkan respons kesalahan.
> Penghapusan bersyarat hanya berguna untuk menghapus item dan atribut jika syarat tertentu terpenuhi. Jika syarat terpenuhi, DynamoDB melakukan penghapusan. Jika tidak, item tersebut tidak akan dihapus.
> Anda dapat melakukan pemeriksaan bersyarat yang diharapkan pada satu atribut per operasi.

Permintaan

Sintaks

```
// This header is abbreviated.
// For a sample of a complete header, see API tingkat rendah DynamoDB.
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"}
}
```

Nama	Deskripsi	Wajib
`TableName`	Nama tabel yang berisi item untuk dihapus. Jenis: String	Ya
`Key`	Kunci primer yang menentukan item. Untuk informasi selengkapnya tentang kunci primer, lihat Kunci primer. Jenis: Peta `HashKeyElement` ke nilainya dan `RangeKeyElement` ke nilainya.	Ya
`Expected`	Menunjuk atribut untuk penghapusan bersyarat. Parameter `Expected` memungkinkan Anda untuk memberikan nama atribut, dan apakah DynamoDB harus memeriksa bahwa atribut memiliki nilai tertentu sebelum mengubahnya. Jenis: Peta nama atribut.	Tidak
`Expected:AttributeName`	Nama atribut untuk penempatan bersyarat. Jenis: String	Tidak
`Expected:AttributeName: ExpectedAttributeValue`	Gunakan parameter ini untuk menentukan apakah nilai sudah ada atau belum untuk pasangan nama-nilai atribut.	Tidak

Nama	Deskripsi	Wajib

	Notasi JSON berikut menghapus item jika atribut "Warna" tidak ada untuk item tersebut:
	`"Expected" : {"Color":{"Exists":false}}`
	Notasi JSON berikut memeriksa untuk melihat apakah atribut dengan nama "Warna" sudah memiliki nilai "Kuning" sebelum menghapus item tersebut:
	`"Expected" : {"Color":{"Exists":true},{"Value":{"S":"Yellow"}}}`
	Secara default, jika Anda menggunakan parameter `Expected` dan menyediakan `Value`, DynamoDB mengasumsikan bahwa atribut ada dan memiliki nilai terkini untuk diganti. Jadi Anda tidak perlu menentukan `{"Exists":true}`, karena hal tersebut sudah tersirat. Anda dapat mempersingkat permintaan untuk:
	`"Expected" :`	

Nama	Deskripsi	Wajib
	`{"Color":{"Value":{"S":"Yellow"}}}` **Note** Jika Anda menentukan `{"Exists":true}` tanpa nilai atribut untuk diperiksa, DynamoDB mengembalikan kesalahan.	
`ReturnValues`	Gunakan parameter ini jika Anda ingin mendapatkan pasangan nama-nilai atribut sebelum pasangan tersebut dihapus. Nilai parameter yang memungkinkan adalah `NONE` (default) atau `ALL_OLD`. Jika `ALL_OLD` ditentukan, isi item lama akan dikembalikan. Jika parameter ini tidak tersedia atau `NONE`, tidak ada yang dikembalikan. Jenis: String	Tidak

Respons

Sintaks

```
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
}
```

Nama	Deskripsi
`Attributes`	Jika parameter `ReturnValues` disediakan sebagai `ALL_OLD` dalam permintaan, DynamoDB mengembalikan array pasangan nama-nilai atribut (pada dasarnya, item tersebut dihapus). Jika tidak, respons berisi sebuah set kosong. Jenis: Array pasangan nama-nilai atribut.
`ConsumedCapacityUnits`	Jumlah unit kapasitas tulis yang digunakan dalam operasi. Nilai ini menunjukkan jumlah yang diterapkan pada throughput yang disediakan. Permintaan hapus pada item yang tidak ada mengonsumsi 1 unit kapasitas tulis. Untuk informasi selengkapnya, lihat Mode kapasitas yang disediakan DynamoDB. Jenis: Angka

Kesalahan khusus

Kesalahan	Deskripsi
`ConditionalCheckFailedException`	Pemeriksaan bersyarat gagal. Nilai atribut yang diharapkan tidak ditemukan.

Contoh

Permintaan sampel

```
// This header is abbreviated.
// For a sample of a complete header, see API tingkat rendah DynamoDB.
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"
}
```

Respons sampel

```
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
}
```

Tindakan terkait

- PutItem

DeleteTable

> **Important**
>
> ***This section refers to API version 2011-12-05, which is deprecated and should not be used for new applications.***
>
> Untuk dokumentasi tentang API tingkat rendah saat ini, lihat Referensi API Amazon DynamoDB.

Deskripsi

Operasi `DeleteTable` menghapus tabel beserta semua itemnya. Setelah permintaan `DeleteTable`, tabel yang ditentukan berada dalam status `DELETING` hingga DynamoDB menyelesaikan penghapusan. Jika tabel berada dalam status `ACTIVE`, Anda dapat menghapusnya. Jika tabel berada dalam status `CREATING` atau `UPDATING`, DynamoDB mengembalikan kesalahan `ResourceInUseException`. Jika tabel yang ditentukan tidak ada, DynamoDB mengembalikan `ResourceNotFoundException`. Jika tabel sudah berada dalam status `DELETING`, tidak ada kesalahan yang dikembalikan.

> **Note**
>
> DynamoDB dapat terus menerima permintaan operasi bidang data, seperti `GetItem` dan `PutItem`, pada tabel dalam status `DELETING` hingga penghapusan tabel selesai.

Tabel unik di antara yang terkait dengan AWS Akun yang mengeluarkan permintaan, dan AWS wilayah yang menerima permintaan (seperti dynamodb.us-west-1.amazonaws.com). Setiap titik akhir DynamoDB bersifat independen sepenuhnya. Misalnya, jika Anda memiliki dua tabel yang disebut "MyTable," satu di dynamodb.us-west-2.amazonaws.com dan satu di dynamodb.us-west-1.amazonaws.com, mereka sepenuhnya independen dan tidak berbagi data apa pun; menghapus satu tidak menghapus yang lain.

Gunakan operasi DescribeTables untuk memeriksa status tabel.

Permintaan

Sintaks

```
// This header is abbreviated.
// For a sample of a complete header, see API tingkat rendah DynamoDB.
POST / HTTP/1.1
x-amz-target: DynamoDB_20111205.DeleteTable
content-type: application/x-amz-json-1.0

{"TableName":"Table1"}
```

Nama	Deskripsi	Wajib
`TableName`	Nama tabel yang akan dihapus. Jenis: String	Ya

Respons

Sintaks

```
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"
    }
}
```

Nama	Deskripsi
`TableDescription`	Kontainer untuk properti tabel.
`CreationDateTime`	Tanggal ketika tabel dibuat. Jenis: Angka
`KeySchema`	Struktur kunci primer (sederhana atau komposit) untuk tabel. Pasangan nama-nilai untuk `HashKeyElement` diperlukan, dan pasangan nama-nilai untuk `RangeKeyElement` bersifat opsional (hanya diperlukan untuk kunci primer komposit). Untuk informasi selengkapnya tentang kunci primer, lihat Kunci primer. Jenis: Peta `HashKeyElement` , atau `HashKeyElement` dan `RangeKeyElement` untuk kunci primer komposit.
`ProvisionedThroughput`	Throughput untuk tabel yang ditentukan, yang terdiri dari nilai-nilai untuk `ReadCapacityUnits` dan `WriteCapacityUnits` . Lihat Mode kapasitas yang disediakan DynamoDB.
`ProvisionedThroughput` : `ReadCapacityUnits`	Jumlah minimum `ReadCapacityUnits` yang digunakan per detik untuk tabel yang ditentukan sebelum DynamoDB menyeimbangkan beban dengan operasi lainnya. Jenis: Angka
`ProvisionedThroughput` : `WriteCapacityUnits`	Jumlah minimum `WriteCapacityUnits` yang digunakan per detik untuk tabel yang ditentukan sebelum DynamoDB menyeimbangkan beban dengan operasi lainnya.

Nama	Deskripsi
	Jenis: Angka
`TableName`	Nama tabel yang dihapus. Jenis: String
`TableStatus`	Status tabel saat ini (`DELETING`). Setelah tabel dihapus, permintaan berikutnya untuk tabel mengembalikan `resource not found` . Gunakan operasi DescribeTables untuk memeriksa status tabel. Jenis: String

Kesalahan khusus

Kesalahan	Deskripsi
`ResourceInUseException`	Tabel berada dalam status `CREATING` atau `UPDATING` dan tidak dapat dihapus.

Contoh

Permintaan sampel

```
// This header is abbreviated. For a sample of a complete header, see API tingkat
 rendah DynamoDB.
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"}
```

Respons sampel

```
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"
}
```

Tindakan terkait

- CreateTable
- DescribeTables

DescribeTables

> **Important**
>
> ***This section refers to API version 2011-12-05, which is deprecated and should not be used for new applications.***
>
> Untuk dokumentasi tentang API tingkat rendah saat ini, lihat Referensi API Amazon DynamoDB.

Deskripsi

Mengembalikan informasi tentang tabel, termasuk status tabel saat ini, skema kunci utama dan kapan tabel dibuat. DescribeTable Hasilnya pada akhirnya konsisten. Jika Anda menggunakan DescribeTable terlalu dini dalam proses membuat tabel, DynamoDB mengembalikan sebuah. `ResourceNotFoundException` Jika Anda menggunakan DescribeTable terlalu dini dalam proses memperbarui tabel, nilai baru mungkin tidak segera tersedia.

Permintaan

Sintaks

```
// This header is abbreviated.
// For a sample of a complete header, see API tingkat rendah DynamoDB.
POST / HTTP/1.1
x-amz-target: DynamoDB_20111205.DescribeTable
content-type: application/x-amz-json-1.0

{"TableName":"Table1"}
```

Nama	Deskripsi	Wajib
`TableName`	Nama tabel yang akan dideskripsikan. Jenis: String	Ya

Respons

Sintaks

```
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"
    }
```

```
}
```

Nama	Deskripsi
`Table`	Kontainer untuk tabel yang sedang dideskripsikan. Jenis: String
`CreationDateTime`	Tanggal ketika tabel dibuat di jangka waktu UNIX.
`ItemCount`	Jumlah item dalam tabel yang ditentukan. DynamoDB memperbarui nilai ini kira-kira setiap enam jam. Perubahan terbaru mungkin tidak tercermin dalam nilai ini. Jenis: Angka
`KeySchema`	Struktur kunci primer (sederhana atau komposit) untuk tabel. Pasangan nama-nilai untuk `HashKeyElement` diperlukan, dan pasangan nama-nilai untuk `RangeKeyElement` bersifat opsional (hanya diperlukan untuk kunci primer komposit). Ukuran kunci hash maksimum adalah 2048 byte. Ukuran kunci rentang maksimum adalah 1024 byte. Kedua batas tersebut diberlakukan secara terpisah (yaitu Anda dapat memiliki kombinasi hash + rentang 2048 + 1024 kunci). Untuk informasi selengkapnya tentang kunci primer, lihat Kunci primer.
`ProvisionedThroughput`	Throughput untuk tabel tertentu, yang terdiri dari nilai-nilai untuk `LastIncreaseDateTime` (jika berlaku), `LastDecreaseDateTime` (jika berlaku), `ReadCapacityUnits`, dan `WriteCapacityUnits`. Jika throughpu

Nama	Deskripsi
	t untuk tabel tidak pernah meningkat atau berkurang, DynamoDB tidak mengembalikan nilai untuk elemen-elemen tersebut. Lihat Mode kapasitas yang disediakan DynamoDB. Jenis: Array
`TableName`	Nama tabel yang diminta. Jenis: String
`TableSizeBytes`	Ukuran total tabel yang ditentukan, dalam byte. DynamoDB memperbarui nilai ini kira-kira setiap enam jam. Perubahan terbaru mungkin tidak tercermin dalam nilai ini. Jenis: Angka
`TableStatus`	Status tabel saat ini (`CREATING`, `ACTIVE`, `DELETING`, atau `UPDATING`). Setelah tabel berada dalam status `ACTIVE`, Anda dapat menambahkan data.

Kesalahan khusus

Tidak ada kesalahan yang spesifik untuk operasi ini.

Contoh

Contoh berikut menunjukkan permintaan HTTP POST dan respon menggunakan DescribeTable operasi untuk tabel bernama “comp-table”. Tabel tersebut memiliki kunci primer komposit.

Permintaan Sampel

```
// This header is abbreviated.
// For a sample of a complete header, see API tingkat rendah DynamoDB.
POST / HTTP/1.1
x-amz-target: DynamoDB_20111205.DescribeTable
```

```
content-type: application/x-amz-json-1.0

{"TableName":"users"}
```

Respons sampel

```
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"
    }
}
```

Tindakan terkait

- CreateTable
- DeleteTable
- ListTables

GetItem

> **Important**
>
> ***This section refers to API version 2011-12-05, which is deprecated and should not be used for new applications.***
>
> Untuk dokumentasi tentang API tingkat rendah saat ini, lihat Referensi API Amazon DynamoDB.

Deskripsi

Operasi `GetItem` mengembalikan sekumpulan `Attributes` untuk item yang cocok dengan kunci primer. Jika tidak ada item yang cocok, `GetItem` tidak mengembalikan data apa pun.

Operasi `GetItem` menyediakan bacaan akhir konsisten secara default. Jika bacaan akhir konsisten tidak dapat diterima untuk aplikasi Anda, gunakan `ConsistentRead`. Meskipun operasi ini mungkin memakan waktu lebih lama dari baca standar, operasi ini selalu mengembalikan nilai yang terakhir diperbarui. Untuk informasi selengkapnya, lihat DynamoDB membaca konsistensi.

Permintaan

Sintaks

```
// This header is abbreviated.
// For a sample of a complete header, see API tingkat rendah DynamoDB.
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
}
```

Nama	Deskripsi	Wajib
`TableName`	Nama tabel yang berisi item yang diminta. Jenis: String	Ya
`Key`	Nilai kunci primer yang menentukan item. Untuk informasi selengkapnya tentang kunci primer, lihat Kunci primer.	Ya

Nama	Deskripsi	Wajib
	Jenis: Peta `HashKeyElement` ke nilainya dan `RangeKeyElement` ke nilainya.	
`AttributesToGet`	Array Nama atribut. Jika nama atribut tidak ditentukan, semua atribut akan dikembalikan. Jika beberapa atribut tidak ditemukan, atribut tersebut tidak akan muncul dalam hasil. Jenis: Array	Tidak
`ConsistentRead`	Jika diatur ke `true`, bacaan yang konsisten akan diterbitkan. Jika tidak, bacaan akhir konsisten akan digunakan. Jenis: Boolean	Tidak

Respons

Sintaks

```
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
}
```

Nama	Deskripsi
`Item`	Berisi atribut yang diminta. Jenis: Peta pasangan nama-nilai atribut.
`ConsumedCapacityUnits`	Jumlah unit kapasitas baca yang digunakan dalam operasi. Nilai ini menunjukkan jumlah yang diterapkan pada throughput yang disediakan. Permintaan untuk item yang tidak ada yang mengonsumsi unit kapasitas baca minimum, tergantung jenis baca. Untuk informasi selengkapnya, lihat Mode kapasitas yang disediakan DynamoDB. Jenis: Angka

Kesalahan khusus

Tidak ada kesalahan khusus untuk operasi ini.

Contoh

Untuk contoh menggunakan AWS SDK, lihatBekerja dengan item dan atribut di DynamoDB.

Permintaan sampel

```
// This header is abbreviated.
// For a sample of a complete header, see API tingkat rendah DynamoDB.
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
}
```

Respons sampel

Perhatikan ConsumedCapacityUnits nilainya adalah 1, karena parameter opsional `ConsistentRead` diatur `ketrue`. Jika `ConsistentRead` disetel ke `false` (atau tidak ditentukan) untuk permintaan yang sama, respons pada akhirnya konsisten dan ConsumedCapacityUnits nilainya akan menjadi 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**
>
> ***This section refers to API version 2011-12-05, which is deprecated and should not be used for new applications.***
> Untuk dokumentasi tentang API tingkat rendah saat ini, lihat Referensi API Amazon DynamoDB.

Deskripsi

Mengembalikan array dari semua tabel yang terkait dengan akun dan titik akhir saat ini.

Setiap titik akhir DynamoDB bersifat independen sepenuhnya. Misalnya, jika Anda memiliki dua tabel yang disebut "MyTable," satu di dynamodb.us-west-2.amazonaws.com dan satu di dynamodb.us-east-1.amazonaws.com, mereka sepenuhnya independen dan tidak berbagi data apa pun. ListTables Operasi mengembalikan semua nama tabel yang terkait dengan akun yang membuat permintaan, untuk titik akhir yang menerima permintaan.

Permintaan

Sintaks

```
// This header is abbreviated.
// For a sample of a complete header, see API tingkat rendah DynamoDB.
POST / HTTP/1.1
x-amz-target: DynamoDB_20111205.ListTables
content-type: application/x-amz-json-1.0

{"ExclusiveStartTableName":"Table1","Limit":3}
```

ListTables Operasi, secara default, meminta semua nama tabel yang terkait dengan akun yang membuat permintaan, untuk titik akhir yang menerima permintaan.

Nama	Deskripsi	Wajib
`Limit`	Jumlah maksimum nama tabel yang akan dikembalikan. Jenis: Integer	Tidak
`ExclusiveStartTableName`	Nama tabel yang memulai daftar. Jika Anda sudah menjalankan ListTables operasi dan menerima `LastEvaluatedTableName` nilai dalam respons, gunakan nilai itu di sini untuk melanjutkan daftar. Tipe: String	Tidak

Respons

Sintaks

```
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"}
```

Nama	Deskripsi
`TableNames`	Nama-nama tabel yang terkait dengan akun saat ini pada titik akhir terkini. Jenis: Array
`LastEvaluatedTableName`	Nama tabel terakhir dalam daftar saat ini, hanya jika beberapa tabel untuk akun dan titik akhir belum dikembalikan. Nilai ini tidak ada dalam respons jika semua nama tabel sudah dikembalikan. Gunakan nilai ini sebagai `ExclusiveStartTableName` dalam permintaan baru untuk melanjutkan daftar hingga semua nama tabel dikembalikan. Jenis: String

Kesalahan khusus

Tidak ada kesalahan yang spesifik untuk operasi ini.

Contoh

Contoh berikut menunjukkan permintaan HTTP POST dan respon menggunakan ListTables operasi.

Permintaan sampel

```
// This header is abbreviated.
// For a sample of a complete header, see API tingkat rendah DynamoDB.
POST / HTTP/1.1
x-amz-target: DynamoDB_20111205.ListTables
```

```
content-type: application/x-amz-json-1.0

{"ExclusiveStartTableName":"comp2","Limit":3}
```

Respons sampel

```
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"]}
```

Tindakan terkait

- DescribeTables
- CreateTable
- DeleteTable

PutItem

> **Important**
>
> ***This section refers to API version 2011-12-05, which is deprecated and should not be used for new applications.***
>
> Untuk dokumentasi tentang API tingkat rendah saat ini, lihat Referensi API Amazon DynamoDB.

Deskripsi

Membuat item baru, atau menggantikan item lama dengan item baru (termasuk semua atributnya). Jika item sudah ada dalam tabel tertentu dengan kunci primer yang sama, item baru sepenuhnya menggantikan item yang ada. Anda dapat melakukan penempatan bersyarat (masukkan item baru jika salah satu dengan kunci primer yang ditentukan tidak ada), atau mengganti item yang ada jika memiliki nilai atribut tertentu.

Nilai atribut tidak boleh null; atribut jenis string dan biner harus memiliki panjang lebih dari nol; dan atribut jenis set tidak boleh kosong. Permintaan dengan nilai kosong akan ditolak dengan `ValidationException`.

> **Note**
>
> Untuk memastikan bahwa item baru tidak menggantikan item yang ada, gunakan operasi penempatan bersyarat dengan mengatur `Exists` ke `false` untuk atribut kunci primer, atau atribut-atribut.

Untuk informasi selengkapnya tentang penggunaan `PutItem`, lihat Bekerja dengan item dan atribut di DynamoDB.

Permintaan

Sintaks

```
// This header is abbreviated.
// For a sample of a complete header, see API tingkat rendah DynamoDB.
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"}
```

Nama	Deskripsi	Wajib
`TableName`	Nama tabel untuk diisi item. Jenis: String	Ya
`Item`	Peta atribut untuk item, dan harus menyertakan nilai kunci	Ya

Nama	Deskripsi	Wajib
	primer yang menentukan item tersebut. Pasangan nama-nilai atribut lainnya dapat diberikan untuk item tersebut. Untuk informasi selengkap nya tentang kunci primer, lihat Kunci primer. Jenis: Peta nama atribut untuk nilai atribut.	
`Expected`	Menunjuk atribut untuk penempatan bersyarat . Parameter `Expected` memungkinkan Anda untuk memberikan nama atribut, dan apakah DynamoDB harus memeriksa keberadaan nilai atribut; atau apakah nilai atribut ada dan memiliki nilai tertentu sebelum mengubahn ya. Jenis: Peta nama atribut untuk nilai atribut, dan apakah nilai tersebut ada.	Tidak
`Expected:AttributeName`	Nama atribut untuk penempatan bersyarat. Jenis: String	Tidak

Nama	Deskripsi	Wajib

`Expected:AttributeName: ExpectedAttributeValue`	Gunakan parameter ini untuk menentukan apakah nilai sudah ada atau belum untuk pasangan nama-nilai atribut.
	
	Notasi JSON berikut mengganti item jika atribut "Warna" tidak ada untuk item tersebut:
	
	`"Expected" : {"Color":{"Exists":false}}`
	
	Notasi JSON berikut memeriksa untuk melihat apakah atribut dengan nama "Warna" sudah memiliki nilai "Kuning" sebelum mengganti item tersebut:
	
	`"Expected" : {"Color":{"Exists":true,{"Value":{"S":"Yellow"}}}`
	
	Secara default, jika Anda menggunakan parameter `Expected` dan menyediakan `Value`, DynamoDB mengasumsikan bahwa atribut ada dan memiliki nilai terkini untuk diganti. Jadi Anda tidak perlu menentukan `{"Exists":true}` , karena hal tersebut sudah tersirat. Anda dapat
		Tidak

Nama	Deskripsi	Wajib
	mempersingkat permintaan untuk: `"Expected" : {"Color":{"Value": {"S":"Yellow"}}}` **Note** Jika Anda menentukan `{"Exists" :true}` tanpa nilai atribut untuk diperiksa, DynamoDB mengembalikan kesalahan.	
`ReturnValues`	Gunakan parameter ini jika Anda ingin mendapatkan pasangan nama-nilai atribut sebelum pasangan tersebut diperbarui dengan permintaan `PutItem`. Nilai parameter yang memungkinkan adalah `NONE` (default) atau `ALL_OLD`. Jika `ALL_OLD` ditentukan, dan `PutItem` menimpa pasangan nama-nilai atribut, isi dari item lama akan dikembalikan. Jika parameter ini tidak tersedia atau `NONE`, tidak ada yang dikembalikan. Jenis: String	Tidak

Respons

Sintaks

Contoh sintaksis berikut mengasumsikan permintaan ditentukan dalam parameter `ReturnValues` dari `ALL_OLD`; jika tidak, respons hanya memiliki elemen `ConsumedCapacityUnits`.

```
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
}
```

Nama	Deskripsi
`Attributes`	Nilai atribut sebelum operasi penempatan, tetapi hanya jika parameter `ReturnValues` ditentukan sebagai dalam permintaan `ALL_OLD`. Jenis: Peta pasangan nama-nilai atribut.
`ConsumedCapacityUnits`	Jumlah unit kapasitas tulis yang digunakan dalam operasi. Nilai ini menunjukkan jumlah yang diterapkan pada throughput yang disediakan. Untuk informasi selengkapnya, lihat Mode kapasitas yang disediakan DynamoDB. Jenis: Angka

Kesalahan khusus

Kesalahan	Deskripsi
`ConditionalCheckFailedException`	Pemeriksaan bersyarat gagal. Nilai atribut yang diharapkan tidak ditemukan.
`ResourceNotFoundException`	Item atau atribut yang ditentukan tidak ditemukan.

Contoh

Untuk contoh menggunakan AWS SDK, lihatBekerja dengan item dan atribut di DynamoDB.

Permintaan sampel

```
// This header is abbreviated. For a sample of a complete header, see API tingkat
 rendah DynamoDB.
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"
}
```

Respons sampel

```
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
}
```

Tindakan terkait

- UpdateItem
- DeleteItem
- GetItem
- BatchGetItem

Kueri

> **Important**
>
> ***This section refers to API version 2011-12-05, which is deprecated and should not be used for new applications.***
>
> Untuk dokumentasi tentang API tingkat rendah saat ini, lihat Referensi API Amazon DynamoDB.

Deskripsi

Sebuah `Query` operasi mendapatkan nilai dari satu atau lebih item dan atributnya dengan kunci primer (hanya `Query` tersedia untuk tabel kunci hash-and-range primer). Anda harus memberikan `HashKeyValue` spesifik, dan dapat mempersempit ruang lingkup kueri menggunakan operator perbandingan pada `RangeKeyValue` kunci primer. Gunakan parameter `ScanIndexForward` untuk mendapatkan hasil dalam urutan maju atau mundur dengan kunci rentang.

Kueri yang tidak mengembalikan hasil menggunakan unit kapasitas baca minimum sesuai dengan jenis baca.

> **Note**
>
> Jika jumlah total item yang memenuhi parameter kueri melampaui batas 1 MB, kueri berhenti dan hasilnya dikembalikan ke pengguna dengan `LastEvaluatedKey` untuk melanjutkan

kueri dalam operasi berikutnya. Tidak seperti operasi Scan, operasi Kueri tidak pernah mengembalikan set hasil kosong dan sebuah `LastEvaluatedKey`. `LastEvaluatedKey` hanya diberikan jika hasil melampaui 1 MB, atau jika Anda telah menggunakan parameter `Limit`.
Hasilnya dapat diatur untuk bacaan yang konsisten menggunakan parameter `ConsistentRead`.

Permintaan

Sintaks

```
// This header is abbreviated.
// For a sample of a complete header, see API tingkat rendah DynamoDB.
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"]},
}
```

Nama	Deskripsi	Wajib
`TableName`	Nama tabel yang berisi item yang diminta. Jenis: String	Ya

Nama	Deskripsi	Wajib
`AttributesToGet`	Array Nama atribut. Jika nama atribut tidak ditentukan, semua atribut akan dikembalikan. Jika beberapa atribut tidak ditemukan, atribut tersebut tidak akan muncul dalam hasil. Jenis: Array	Tidak
`Limit`	Jumlah maksimum item untuk dikembalikan (tidak selalu merupakan jumlah item yang cocok). Jika DynamoDB memproses jumlah item hingga batasnya sekaligus mengajukan kueri tabel, layanan ini menghenti kan kueri dan mengembal ikan nilai-nilai yang cocok hingga titik tersebut, dan `LastEvaluatedKey` untuk diterapkan dalam operasi berikutnya guna melanjutk an kueri. Selain itu, jika ukuran set hasil melampaui 1 MB sebelum DynamoDB mencapai batas ini, layanan ini akan menghentikan kueri dan mengembalikan nilai-nilai yang cocok, serta `LastEvalu atedKey` untuk diterapkan dalam operasi berikutnya guna melanjutkan kueri. Jenis: Angka	Tidak

Nama	Deskripsi	Wajib
`ConsistentRead`	Jika diatur ke `true`, bacaan yang konsisten akan diterbitkan. Jika tidak, bacaan akhir konsisten akan digunakan. Jenis: Boolean	Tidak
`Count`	Jika diatur ke `true`, DynamoDB mengembalikan jumlah total item yang cocok dengan parameter kueri, bukan daftar item yang cocok dan atributnya. Anda dapat menerapkan parameter `Limit` pada kueri hanya-hitung. Jangan atur `Count` ke `true` saat menyediakan daftar `AttributesToGet` , karena ini membuat DynamoDB mengembalikan kesalahan validasi. Untuk informasi selengkapnya, lihat Menghitung item dalam hasil. Jenis: Boolean	Tidak
`HashKeyValue`	Nilai atribut komponen hash dari kunci primer komposit. Jenis: String, Angka, atau Biner	Ya

Nama	Deskripsi	Wajib
`RangeKeyCondition`	Sebuah kontainer untuk nilai atribut dan operator perbandingan untuk digunakan dalam kueri. Permintaan kueri tidak memerlukan `RangeKeyCondition`. Jika Anda hanya memberikan `HashKeyValue`, DynamoDB mengembalikan semua item dengan nilai elemen kunci hash yang ditentukan. Jenis: Peta	Tidak
`RangeKeyCondition : AttributeValueList`	Nilai atribut yang akan dievaluasi untuk parameter kueri. `AttributeValueList` berisi satu nilai atribut, kecuali jika perbandingan `BETWEEN` ditentukan. Untuk perbandingan `BETWEEN`, `AttributeValueList` berisi dua nilai atribut. Jenis: Peta `AttributeValue` ke `ComparisonOperator`.	Tidak

Nama	Deskripsi	Wajib
`RangeKeyCondition : ComparisonOperator`	Kriteria untuk mengevaluasi atribut yang disediakan, seperti sama dengan, lebih besar dari, dll. Berikut ini adalah operator perbandingan yang valid untuk operasi Kueri. **Note** Perbandingan nilai string untuk lebih besar dari, sama dengan, atau kurang dari didasarkan pada nilai-nilai kode karakter ASCII. Sebagai contoh, `a` lebih besar dari `A`, dan `aa` lebih besar dari `B`. Untuk daftar nilai kode, lihat http://en.wikipedia.org/wiki/ASCII#ASCII_printable_characters. Untuk Biner, DynamoDB memperlakukan setiap byte data biner sebagai tidak bertanda ketika membandingkan nilai-nilai biner, contohnya ketika mengevaluasi ekspresi kueri.	Tidak

Nama	Deskripsi	Wajib
	Jenis: String atau Biner	
	`EQ`: Sama dengan. Untuk `EQ`, `AttributeValueList` hanya dapat berisi satu `AttributeValue` berjenis String, Angka, atau Biner (bukan jenis set). Jika item berisi `AttributeValue` dengan jenis berbeda dari yang ditentukan dalam permintaan, nilainya tidak cocok. Sebagai contoh, `{"S":"6"}` tidak sama dengan `{"N":"6"}`. Selain itu, `{"N":"6"}` tidak sama dengan `{"NS":["6", "2", "1"]}`.	

Nama	Deskripsi	Wajib
	`LE`: Kurang dari atau sama dengan. Untuk `LE`, `Attribute ValueList` hanya dapat berisi satu `Attribute Value` berjenis String, Angka, atau Biner (bukan jenis set). Jika item berisi `AttributeValue` dengan jenis berbeda dari yang ditentukan dalam permintaan, nilainya tidak cocok. Sebagai contoh, `{"S":"6"}` tidak sama dengan `{"N":"6"}`. Selain itu, `{"N":"6"}` tidak sebanding dengan `{"NS": ["6", "2", "1"]}`.	

Nama	Deskripsi	Wajib
	`LT`: Kurang dari. Untuk `LT`, `AttributeValueList` hanya dapat berisi satu `AttributeValue` berjenis String, Angka, atau Biner (bukan jenis set). Jika item berisi `AttributeValue` dengan jenis berbeda dari yang ditentukan dalam permintaan, nilainya tidak cocok. Sebagai contoh, `{"S":"6"}` tidak sama dengan `{"N":"6"}`. Selain itu, `{"N":"6"}` tidak sebanding dengan `{"NS":["6", "2", "1"]}`.	

Nama	Deskripsi	Wajib
	`GE`: Lebih besar dari atau sama dengan. Untuk `GE`, `Attribute ValueList` hanya dapat berisi satu `Attribute Value` berjenis String, Angka, atau Biner (bukan jenis set). Jika item berisi `AttributeValue` dengan jenis berbeda dari yang ditentukan dalam permintaan, nilainya tidak cocok. Sebagai contoh, `{"S":"6"}` tidak sama dengan `{"N":"6"}`. Selain itu, `{"N":"6"}` tidak sebanding dengan `{"NS": ["6", "2", "1"]}`.	

Nama	Deskripsi	Wajib
	GT: Lebih besar dari. Untuk GT, AttributeValueList hanya dapat berisi satu AttributeValue berjenis String, Angka, atau Biner (bukan jenis set). Jika item berisi AttributeValue dengan jenis berbeda dari yang ditentukan dalam permintaan, nilainya tidak cocok. Sebagai contoh, {"S":"6"} tidak sama dengan {"N":"6"} . Selain itu, {"N":"6"} tidak sebanding dengan {"NS":["6", "2", "1"]} .	
	BEGINS_WITH : memeriksa prefiks. Untuk BEGINS_WITH , AttributeValueList hanya dapat berisi satu AttributeValue berjenis String atau Biner (bukan jenis Angka atau set). Atribut target perbandingan harus berupa String atau Biner (bukan Angka atau set).	

Nama	Deskripsi	Wajib
	`BETWEEN`: Lebih besar dari, atau sama dengan, nilai pertama dan kurang dari, atau sama dengan, nilai kedua. Untuk `BETWEEN`, `AttributeValueList` harus berisi dua elemen `AttributeValue` berjenis yang sama, baik String, Angka, atau Biner (bukan set). Suatu atribut target cocok jika nilai target lebih besar dari, atau sama dengan, elemen pertama dan kurang dari, atau sama dengan, elemen kedua. Jika item berisi `AttributeValue` dengan jenis berbeda dari yang ditentukan dalam permintaan, nilainya tidak cocok. Misalnya, `{"S":"6"}` tidak sebanding dengan `{"N":"6"}` . Selain itu, `{"N":"6"}` tidak sebanding dengan `{"NS":["6", "2", "1"]}`.	

Nama	Deskripsi	Wajib
`ScanIndexForward`	Menentukan traversal naik atau turun indeks. DynamoDB mengembalikan hasil yang mencerminkan urutan yang diminta, yang ditentukan oleh kunci rentang: Jika jenis data adalah Angka, hasilnya dikembalikan dalam urutan numerik; jika tidak, traversal didasarkan pada nilai kode karakter ASCII. Jenis: Boolean Default-nya adalah `true` (naik).	Tidak

Nama	Deskripsi	Wajib
`ExclusiveStartKey`	Kunci primer dari item untuk melanjutkan kueri sebelumnya. Kueri sebelumnya mungkin memberikan nilai ini sebagai `LastEvaluatedKey` jika operasi kueri terinterupsi sebelum menyelesaikan kueri; baik karena ukuran set hasil maupun parameter `Limit`. `LastEvaluatedKey` dapat diteruskan kembali dalam permintaan kueri baru untuk melanjutkan operasi dari titik tersebut. Jenis: `HashKeyElement`, atau `HashKeyElement` dan `RangeKeyElement` untuk kunci primer komposit.	Tidak

Respons

Sintaks

```
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
}
```

Nama	Deskripsi
`Items`	Atribut item yang memenuhi parameter kueri. Jenis: Peta nama atribut serta nilai dan jenis data.
`Count`	Jumlah item dalam respons. Untuk informasi selengkapnya, lihat Menghitung item dalam hasil. Jenis: Angka
`LastEvaluatedKey`	Kunci primer dari item di mana operasi kueri berhenti, termasuk set hasil sebelumnya. Gunakan nilai ini untuk memulai operasi baru dengan mengecualikan nilai ini dalam permintaan baru. `LastEvaluatedKey` adalah `null` ketika seluruh set hasil kueri selesai (yaitu operasi memproses “halaman terakhir”). Jenis: `HashKeyElement` , atau `HashKeyElement` dan `RangeKeyElement` untuk kunci primer komposit.
`ConsumedCapacityUnits`	Jumlah unit kapasitas baca yang digunakan dalam operasi. Nilai ini menunjukkan jumlah yang diterapkan pada throughput yang disediakan. Untuk informasi selengkapnya, lihat Mode kapasitas yang disediakan DynamoDB.

Nama	Deskripsi
	Jenis: Angka

Kesalahan khusus

Kesalahan	Deskripsi
`ResourceNotFoundException`	Tabel yang ditentukan tidak ditemukan.

Contoh

Untuk contoh menggunakan AWS SDK, lihatMenanyakan tabel di DynamoDB.

Permintaan sampel

```
// This header is abbreviated. For a sample of a complete header, see API tingkat
 rendah DynamoDB.
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"}
 }
}
```

Respons sampel

```
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
}
```

Permintaan sampel

```
// This header is abbreviated. For a sample of a complete header, see API tingkat
 rendah DynamoDB.
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}
```

Respons sampel

```
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
}
```

Tindakan terkait

- Scan

Scan

> **Important**
>
> ***This section refers to API version 2011-12-05, which is deprecated and should not be used for new applications.***
> Untuk dokumentasi tentang API tingkat rendah saat ini, lihat Referensi API Amazon DynamoDB.

Deskripsi

Operasi `Scan` mengembalikan satu atau beberapa item dan atributnya dengan melakukan pemindaian tabel secara menyeluruh. Berikan `ScanFilter` untuk mendapatkan hasil yang lebih spesifik.

> **Note**
>
> Jika jumlah total item yang dipindai melebihi batas 1 MB, pemindaian berhenti dan hasil dikembalikan ke pengguna dengan `LastEvaluatedKey` untuk melanjutkan pemindaian dalam operasi berikutnya. Hasilnya juga mencakup jumlah item yang melampaui batas. Pemindaian dapat mengakibatkan tidak adanya data tabel yang memenuhi kriteria filter. Set hasil pada akhirnya konsisten.

Permintaan

Sintaks

```
// This header is abbreviated.
// For a sample of a complete header, see API tingkat rendah DynamoDB.
```

```
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"]},
}
```

Nama	Deskripsi	Wajib
`TableName`	Nama tabel yang berisi item yang diminta. Jenis: String	Ya
`AttributesToGet`	Array Nama atribut. Jika nama atribut tidak ditentukan, semua atribut akan dikembalikan. Jika beberapa atribut tidak ditemukan, atribut tersebut tidak akan muncul dalam hasil. Jenis: Array	Tidak
`Limit`	Jumlah maksimum item untuk dievaluasi (tidak selalu merupakan jumlah item yang cocok). Jika DynamoDB memproses jumlah item hingga batasnya saat memproses hasilnya,	Tidak

Nama	Deskripsi	Wajib
	layanan ini berhenti dan mengembalikan nilai-nilai yang cocok hingga titik tersebut, dan `LastEvaluatedKey` untuk diterapkan dalam operasi berikutnya guna melanjutkan pengambilan item. Selain itu, jika ukuran set data yang dipindai melampaui 1 MB sebelum DynamoDB mencapai batas ini, layanan ini menghentikan pemindaian dan mengembalikan nilai-nilai yang cocok hingga batasnya, serta `LastEvaluatedKey` untuk diterapkan dalam operasi berikutnya guna melanjutkan pemindaian. Jenis: Angka	

Nama	Deskripsi	Wajib
`Count`	Jika diatur ke `true`, DynamoDB mengembalikan jumlah total item untuk operasi Scan, bahkan jika operasi tidak memiliki item yang cocok untuk filter yang ditetapkan. Anda dapat menerapkan parameter Limit untuk pemindaian hanya-hitung. Jangan atur `Count` ke `true` saat menyediakan daftar `AttributesToGet` , karena ini membuat DynamoDB mengembalikan kesalahan validasi. Untuk informasi selengkapnya, lihat Menghitung item dalam hasil. Jenis: Boolean	Tidak
`ScanFilter`	Mengevaluasi hasil pemindaian dan mengembalikan hanya nilai-nilai yang diinginkan. Beberapa syarat diperlakukan sebagai operasi "AND": semua syarat harus dipenuhi agar disertakan dalam hasil. Jenis: Peta nama atribut ke nilai dengan operator perbandingan.	Tidak

Nama	Deskripsi	Wajib
`ScanFilter` :`Attribute ValueList`	Nilai dan syarat untuk mengevaluasi hasil pemindaian untuk filter. Jenis: Peta `Attribute Value` ke `Condition` .	Tidak

Nama	Deskripsi	Wajib
`ScanFilter : ComparisonOperator`	Kriteria untuk mengevaluasi atribut yang disediakan, seperti sama dengan, lebih besar dari, dll. Berikut ini adalah operator perbandingan yang valid untuk operasi pemindaian. **Note** Perbandingan nilai string untuk lebih besar dari, sama dengan, atau kurang dari didasarkan pada nilai-nilai kode karakter ASCII. Sebagai contoh, `a` lebih besar dari `A`, dan `aa` lebih besar dari `B`. Untuk daftar nilai kode, lihat http://en.wikipedia.org/wiki/ASCII#ASCII_printable_characters. Untuk Biner, DynamoDB memperlakukan setiap byte data biner sebagai tidak bertanda ketika membandingkan nilai-nilai biner, contohnya ketika	Tidak

Nama	Deskripsi	Wajib
	mengevaluasi ekspresi kueri. Jenis: String atau Biner	
	EQ: Sama dengan. Untuk EQ, AttributeValueList hanya dapat berisi satu AttributeValue berjenis String, Angka, atau Biner (bukan jenis set). Jika item berisi AttributeValue dengan jenis berbeda dari yang ditentukan dalam permintaan, nilainya tidak cocok. Sebagai contoh, {"S":"6"} tidak sama dengan {"N":"6"} . Selain itu, {"N":"6"} tidak sama dengan {"NS":["6", "2", "1"]}.	

Nama	Deskripsi	Wajib
	`NE`: Tidak sama dengan. Untuk `NE`, `AttributeValueList` hanya dapat berisi satu `AttributeValue` berjenis String, Angka, atau Biner (bukan jenis set). Jika item berisi `AttributeValue` dengan jenis berbeda dari yang ditentukan dalam permintaan, nilainya tidak cocok. Sebagai contoh, `{"S":"6"}` tidak sama dengan `{"N":"6"}`. Selain itu, `{"N":"6"}` tidak sama dengan `{"NS":["6", "2", "1"]}`.	

Nama	Deskripsi	Wajib
	`LE`: Kurang dari atau sama dengan. Untuk `LE`, `AttributeValueList` hanya dapat berisi satu `AttributeValue` berjenis String, Angka, atau Biner (bukan jenis set). Jika item berisi `AttributeValue` dengan jenis berbeda dari yang ditentukan dalam permintaan, nilainya tidak cocok. Sebagai contoh, `{"S":"6"}` tidak sama dengan `{"N":"6"}`. Selain itu, `{"N":"6"}` tidak sebanding dengan `{"NS":["6", "2", "1"]}`.	

Nama	Deskripsi	Wajib
	`LT`: Kurang dari. Untuk `LT`, `Attribute ValueList` hanya dapat berisi satu `Attribute Value` berjenis String, Angka, atau Biner (bukan jenis set). Jika item berisi `AttributeValue` dengan jenis berbeda dari yang ditentukan dalam permintaan, nilainya tidak cocok. Sebagai contoh, `{"S":"6"}` tidak sama dengan `{"N":"6"}`. Selain itu, `{"N":"6"}` tidak sebanding dengan `{"NS": ["6", "2", "1"]}`.	

Nama	Deskripsi	Wajib
	`GE`: Lebih besar dari atau sama dengan. Untuk `GE`, `Attribute ValueList` hanya dapat berisi satu `Attribute Value` berjenis String, Angka, atau Biner (bukan jenis set). Jika item berisi `AttributeValue` dengan jenis berbeda dari yang ditentukan dalam permintaan, nilainya tidak cocok. Sebagai contoh, `{"S":"6"}` tidak sama dengan `{"N":"6"}`. Selain itu, `{"N":"6"}` tidak sebanding dengan `{"NS": ["6", "2", "1"]}`.	

Nama	Deskripsi	Wajib
	`GT`: Lebih besar dari. Untuk `GT`, `AttributeValueList` hanya dapat berisi satu `AttributeValue` berjenis String, Angka, atau Biner (bukan jenis set). Jika item berisi `AttributeValue` dengan jenis berbeda dari yang ditentukan dalam permintaan, nilainya tidak cocok. Sebagai contoh, `{"S":"6"}` tidak sama dengan `{"N":"6"}`. Selain itu, `{"N":"6"}` tidak sebanding dengan `{"NS":["6", "2", "1"]}`.	
	`NOT_NULL`: Atribut ada.	
	`NULL`: Atribut tidak ada.	

Nama	Deskripsi	Wajib
	`CONTAINS`: memeriksa urutan berikutnya, atau nilai dalam suatu set. Untuk `CONTAINS`, `AttributeValueList` hanya dapat berisi satu `AttributeValue` berjenis String, Angka, atau Biner (bukan jenis set). Jika atribut target perbandingan berupa String, operasi memeriksa kecocokan substring. Jika atribut target perbandingan berupa Biner, operasi mencari urutan berikutnya dari target yang cocok dengan input. Jika atribut target perbandingan adalah suatu set ("SS", "NS", atau "BS"), operasi memeriksa anggota set tersebut (bukan sebagai substring).	

Nama	Deskripsi	Wajib
	`NOT_CONTAINS` : memeriksa tidak adanya urutan berikutnya, atau tidak adanya nilai dalam suatu set. Untuk `NOT_CONTAINS` , `AttributeValueList` hanya dapat berisi satu `AttributeValue` berjenis String, Angka, atau Biner (bukan jenis set). Jika atribut target perbandingan adalah String, operasi memeriksa tidak adanya kecocokan substring. Jika atribut target perbandingan adalah Biner, operasi mencari tidak adanya urutan berikutnya dari target yang cocok dengan input. Jika atribut target perbandingan adalah suatu set ("SS", "NS", atau "BS"), operasi memeriksa tidak adanya anggota set tersebut (bukan sebagai substring).	

Nama	Deskripsi	Wajib
	`BEGINS_WITH` : memeriksa prefiks. Untuk `BEGINS_WITH` , `AttributeValueList` hanya dapat berisi satu `AttributeValue` berjenis String atau Biner (bukan jenis Angka atau set). Atribut target perbandingan harus berupa String atau Biner (bukan Angka atau set).	
	`IN`: memeriksa kecocokan yang tepat. Untuk `IN`, `AttributeValueList` dapat berisi lebih dari satu `AttributeValue` berjenis String, Angka, atau Biner (bukan jenis set). Atribut target perbandingan harus memiliki jenis yang sama dan nilai persis agar cocok. Sebuah String tidak pernah cocok dengan set String.	

Nama	Deskripsi	Wajib
	`BETWEEN`: Lebih besar dari, atau sama dengan, nilai pertama dan kurang dari, atau sama dengan, nilai kedua. Untuk `BETWEEN`, `AttributeValueList` harus berisi dua elemen `AttributeValue` berjenis yang sama, baik String, Angka, atau Biner (bukan set). Suatu atribut target cocok jika nilai target lebih besar dari, atau sama dengan, elemen pertama dan kurang dari, atau sama dengan, elemen kedua. Jika item berisi `AttributeValue` dengan jenis berbeda dari yang ditentukan dalam permintaan, nilainya tidak cocok. Misalnya, `{"S":"6"}` tidak sebanding dengan `{"N":"6"}` . Selain itu, `{"N":"6"}` tidak sebanding dengan `{"NS":["6", "2", "1"]}`.	

Nama	Deskripsi	Wajib
`ExclusiveStartKey`	Kunci primer dari item untuk melanjutkan pemindaian sebelumnya. Pemindaian sebelumnya mungkin memberikan nilai ini jika operasi pemindaian terganggu sebelum memindai seluruh tabel; baik karena ukuran set hasil atau parameter `Limit`. `LastEvaluatedKey` dapat diteruskan kembali dalam permintaan pemindaian baru untuk melanjutkan operasi dari titik tersebut. Jenis: `HashKeyElement`, atau `HashKeyElement` dan `RangeKeyElement` untuk kunci primer komposit.	Tidak

Respons

Sintaks

```
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}
}
```

Nama	Deskripsi
`Items`	Kontainer untuk atribut yang memenuhi parameter operasi. Jenis: Peta nama atribut serta nilai dan jenis data.
`Count`	Jumlah item dalam respons. Untuk informasi selengkapnya, lihat Menghitung item dalam hasil. Jenis: Angka
`ScannedCount`	Jumlah item dalam pemindaian lengkap sebelum filter diterapkan. Nilai `ScannedCount` yang tinggi dengan dengan sedikit, atau tanpa hasil, `Count` hasil menunjukkan operasi Scan yang tidak efisien. Untuk informasi selengkapnya, lihat Menghitung item dalam hasil. Jenis: Angka
`LastEvaluatedKey`	Kunci primer item tempat operasi pemindaian berhenti. Masukkan nilai ini dalam operasi pemindaian berikutnya untuk melanjutkan operasi dari titik tersebut.

Nama	Deskripsi
	`LastEvaluatedKey` adalah `null` ketika seluruh set hasil pemindaian selesai (yaitu operasi memproses “halaman terakhir”).
`ConsumedCapacityUnits`	Jumlah unit kapasitas baca yang digunakan dalam operasi. Nilai ini menunjukkan jumlah yang diterapkan pada throughput yang disediakan. Untuk informasi selengkapnya, lihat Mode kapasitas yang disediakan DynamoDB. Jenis: Angka

Kesalahan khusus

Kesalahan	Deskripsi
`ResourceNotFoundException`	Tabel yang ditentukan tidak ditemukan.

Contoh

Untuk contoh menggunakan AWS SDK, lihatMemindai tabel di DynamoDB.

Permintaan sampel

```
// This header is abbreviated. For a sample of a complete header, see API tingkat
 rendah DynamoDB.
POST / HTTP/1.1
x-amz-target: DynamoDB_20111205.Scan
content-type: application/x-amz-json-1.0

{"TableName":"1-hash-rangetable","ScanFilter":{}}
```

Respons sampel

```
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}
```

Permintaan sampel

```
// This header is abbreviated. For a sample of a complete header, see API tingkat
 rendah DynamoDB.
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"}
 }
}
```

Respons sampel

```
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
}
```

Permintaan sampel

```
// This header is abbreviated. For a sample of a complete header, see API tingkat
 rendah DynamoDB.
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"}}
}
```

Respons sampel

```
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
}
```

Tindakan terkait

- Kueri
- BatchGetItem

UpdateItem

> **Important**
>
> ***This section refers to API version 2011-12-05, which is deprecated and should not be used for new applications.***
>
> Untuk dokumentasi tentang API tingkat rendah saat ini, lihat Referensi API Amazon DynamoDB.

Deskripsi

Mengedit atribut item yang ada. Anda dapat melakukan pembaruan bersyarat (memasukkan pasangan nama-nilai atribut baru jika tidak ada, atau mengganti pasangan nama-nilai yang ada jika memiliki nilai atribut tertentu yang diharapkan).

> **Note**
>
> Anda tidak dapat memperbarui atribut kunci utama menggunakan UpdateItem. Sebagai gantinya, hapus item dan gunakan PutItem untuk membuat item baru dengan atribut baru.

UpdateItem Operasi mencakup `Action` parameter, yang mendefinisikan cara melakukan pembaruan. Anda dapat menempatkan, menghapus, atau menambahkan nilai atribut.

Nilai atribut tidak boleh null; atribut jenis string dan biner harus memiliki panjang lebih dari nol; dan atribut jenis set tidak boleh kosong. Permintaan dengan nilai kosong akan ditolak dengan `ValidationException`.

Jika item yang ada memiliki kunci primer yang ditentukan:

- PUT— Menambahkan atribut yang ditentukan. Jika sudah ada, atribut akan digantikan oleh nilai baru.
- DELETE— Jika tidak ada nilai yang ditentukan, ini akan menghapus atribut dan nilainya. Jika suatu set nilai ditentukan, nilai-nilai dalam set yang ditentukan dihapus dari set lama. Jadi, jika nilai atribut berisi [a,b,c] dan tindakan menghapus berisi [a,c], nilai atribut akhir adalah [b]. Jenis nilai yang ditentukan harus sesuai dengan jenis nilai yang ada. Menentukan suatu set kosong tidak valid.
- ADD— Hanya gunakan tindakan add untuk angka atau jika atribut target adalah suatu set (termasuk set string). ADD tidak berfungsi jika atribut target adalah nilai string tunggal atau nilai biner skalar. Nilai yang ditentukan ditambahkan ke nilai numerik (menambah atau mengurangi nilai numerik yang ada) atau ditambahkan sebagai nilai tambahan dalam suatu set string. Jika suatu set nilai ditentukan, nilai-nilai ditambahkan ke set yang ada. Misalnya jika set asli adalah [1,2] dan nilai yang disediakan adalah [3], maka setelah operasi add, set adalah [1,2,3], bukan [4,5]. Terjadi kesalahan jika tindakan Add ditentukan untuk atribut set dan jenis atribut yang ditentukan tidak cocok dengan jenis set yang ada.

  Jika Anda menggunakan ADD untuk atribut yang tidak ada, atribut dan nilai-nilainya ditambahkan ke item.

Jika tidak ada item yang cocok dengan kunci primer yang ditentukan:

- PUT – Membuat item baru dengan kunci primer yang ditentukan. Lalu menambahkan atribut yang ditentukan.
- DELETE— Tidak ada yang terjadi.

- ADD— Membuat item dengan kunci primer dan angka (atau set angka) yang disediakan untuk nilai atribut. Tidak berlaku untuk jenis string atau biner.

> **Note**
>
> Jika Anda menggunakan `ADD` untuk menambah atau mengurangi nilai angka untuk item yang tidak ada sebelum pembaruan, DynamoDB menggunakan `0` sebagai nilai awal. Selain itu, jika Anda memperbarui item menggunakan `ADD` guna menambah atau mengurangi nilai angka untuk atribut yang tidak ada sebelum pembaruan (tetapi item ada), DynamoDB menggunakan `0` sebagai nilai awal. Sebagai contoh, Anda menggunakan `ADD` untuk menambahkan `+3` ke atribut yang tidak ada sebelum pembaruan. DynamoDB menggunakan `0` sebagai nilai awal, dan nilai setelah pembaruan adalah `3`.

Untuk informasi lebih lanjut tentang penggunaan operasi ini, lihat Bekerja dengan item dan atribut di DynamoDB.

Permintaan

Sintaks

```
// This header is abbreviated.
// For a sample of a complete header, see API tingkat rendah DynamoDB.
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"
}
```

Nama	Deskripsi	Wajib
`TableName`	Nama tabel yang berisi item untuk diperbarui. Jenis: String	Ya
`Key`	Kunci primer yang menentukan item. Untuk informasi selengkapnya tentang kunci primer, lihat Kunci primer. Jenis: Peta `HashKeyElement` ke nilainya dan `RangeKeyElement` ke nilainya.	Ya
`AttributeUpdates`	Peta nama atribut ke nilai baru dan tindakan baru untuk diperbarui. Nama atribut menentukan atribut untuk dimodifikasi, dan tidak dapat berisi atribut kunci primer apa pun. Jenis: Peta nama atribut, nilai, dan tindakan untuk pembaruan atribut.	
`AttributeUpdates :Action`	Menentukan cara melakukan pembaruan. Kemungkinan nilai: `PUT` (default), `ADD` atau `DELETE`. Semantik dijelaskan dalam deskripsi. UpdateItem Tipe: String Default: `PUT`	Tidak

Nama	Deskripsi	Wajib
`Expected`	Menunjuk atribut untuk pembaruan bersyarat. Parameter `Expected` memungkinkan Anda untuk memberikan nama atribut, dan apakah DynamoDB harus memeriksa keberadaan nilai atribut; atau apakah nilai atribut ada dan memiliki nilai tertentu sebelum mengubahn ya. Jenis: Peta nama atribut.	Tidak
`Expected:AttributeName`	Nama atribut untuk penempatan bersyarat. Jenis: String	Tidak

Nama	Deskripsi	Wajib

`Expected:AttributeName: ExpectedAttributeValue`	Gunakan parameter ini untuk menentukan apakah nilai sudah ada atau belum untuk pasangan nama-nilai atribut.
	
	Notasi JSON berikut memperbarui item jika atribut "Warna" tidak ada untuk item tersebut:
	
	`"Expected" : {"Color":{"Exists":false}}`
	
	Notasi JSON berikut memeriksa untuk melihat apakah atribut dengan nama "Warna" sudah memiliki nilai "Kuning" sebelum memperbarui item tersebut:
	
	`"Expected" : {"Color":{"Exists":true},{"Value": {"S":"Yellow"}}}`
	
	Secara default, jika Anda menggunakan parameter `Expected` dan menyediakan `Value`, DynamoDB mengasumsikan bahwa atribut ada dan memiliki nilai terkini untuk diganti. Jadi Anda tidak perlu menentukan `{"Exists":true}` , karena hal tersebut sudah tersirat. Anda dapat
		Tidak

Nama	Deskripsi	Wajib
	mempersingkat permintaan untuk: `"Expected" : {"Color":{"Value": {"S":"Yellow"}}}` **Note** Jika Anda menentukan `{"Exists" :true}` tanpa nilai atribut untuk diperiksa, DynamoDB mengembalikan kesalahan.	

Nama	Deskripsi	Wajib
ReturnValues	Gunakan parameter ini jika Anda ingin mendapatkan pasangan nama-nilai atribut sebelum pasangan tersebut diperbarui dengan permintaan UpdateItem . Nilai parameter yang memungkinkan adalah NONE (default) atau ALL_OLD, UPDATED_OLD , ALL_NEW, atau UPDATED_NEW . Jika ALL_OLD ditentukan, dan UpdateItem menimpa pasangan nama-nilai atribut, isi dari item lama akan dikembalikan. Jika parameter ini tidak tersedia atau NONE, tidak ada yang dikembalikan. Jika ALL_NEW ditentukan, semua atribut versi baru dari item tersebut akan dikembalikan. Jika UPDATED_NEW ditentukan, versi baru dari hanya atribut yang diperbarui akan dikembalikan. Jenis: String	Tidak

Respons

Sintaks

Contoh sintaksis berikut mengasumsikan permintaan ditentukan dalam parameter ReturnValues dari ALL_OLD; jika tidak, respons hanya memiliki elemen ConsumedCapacityUnits.

```
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
}
```

Nama	Deskripsi
`Attributes`	Sebuah peta pasangan nama-nilai atribut, tetapi hanya jika parameter `ReturnValues` ditentukan sebagai sesuatu selain `NONE` dalam permintaan. Jenis: Peta pasangan nama-nilai atribut.
`ConsumedCapacityUnits`	Jumlah unit kapasitas tulis yang digunakan dalam operasi. Nilai ini menunjukkan jumlah yang diterapkan pada throughput yang disediakan. Untuk informasi selengkapnya, lihat Mode kapasitas yang disediakan DynamoDB. Jenis: Angka

Kesalahan khusus

Kesalahan	Deskripsi
`ConditionalCheckFailedException`	Pemeriksaan bersyarat gagal. Atribut nilai ("+ name +") adalah ("+ value +") tetapi diharapkan ("+ expValue +")

Kesalahan	Deskripsi
`ResourceNotFoundExceptions`	Item atau atribut yang ditentukan tidak ditemukan.

Contoh

Untuk contoh menggunakan AWS SDK, lihatBekerja dengan item dan atribut di DynamoDB.

Permintaan sampel

```
// This header is abbreviated. For a sample of a complete header, see API tingkat
 rendah DynamoDB.
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"
}
```

Respons sampel

```
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
```

```
}
```

Tindakan terkait

- PutItem
- DeleteItem

UpdateTable

> **Important**
>
> ***This section refers to API version 2011-12-05, which is deprecated and should not be used for new applications.***
>
> Untuk dokumentasi tentang API tingkat rendah saat ini, lihat Referensi API Amazon DynamoDB.

Deskripsi

Memperbarui throughput yang disediakan untuk tabel yang ditentukan. Mengatur throughput untuk tabel membantu Anda mengelola performa dan merupakan bagian dari fitur throughput yang disediakan dalam DynamoDB. Untuk informasi selengkapnya, lihat Mode kapasitas yang disediakan DynamoDB.

Nilai throughput yang disediakan dapat ditingkatkan atau diturunkan berdasarkan nilai maksimum dan minimum yang tercantum dalam Kuota di Amazon DynamoDB.

Tabel harus dalam `ACTIVE` keadaan agar operasi ini berhasil. UpdateTable adalah operasi asinkron; saat menjalankan operasi, tabel dalam keadaan. `UPDATING` Meskipun berada dalam status `UPDATING`, tabel masih memiliki throughput yang disediakan sejak sebelum panggilan. Pengaturan throughput baru yang disediakan hanya berlaku ketika tabel kembali ke `ACTIVE` status setelah operasi. UpdateTable

Permintaan

Sintaks

```
// This header is abbreviated.
// For a sample of a complete header, see API tingkat rendah DynamoDB.
```

```
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}
}
```

Nama	Deskripsi	Wajib
`TableName`	Nama tabel yang akan diperbarui. Jenis: String	Ya
`ProvisionedThroughput`	Throughput baru untuk tabel yang ditentukan, yang terdiri dari nilai-nilai untuk `ReadCapacityUnits` dan `WriteCapacityUnits` . Lihat Mode kapasitas yang disediakan DynamoDB. Jenis: Array	Ya
`ProvisionedThroughput :ReadCapacityUnits`	Menetapkan jumlah minimum `ReadCapacityUnits` yang konsisten digunakan per detik untuk tabel yang ditentukan sebelum DynamoDB menyeimbangkan beban dengan operasi lainnya. Operasi bacaan akhir konsisten memerlukan lebih sedikit usaha daripada operasi baca konsisten, jadi pengaturan 50 `ReadCapacityUnits`	Ya

Nama	Deskripsi	Wajib
	yang konsisten per detik akan menghasilkan 100 `ReadCapacityUnits` akhir konsisten per detik. Jenis: Angka	
`ProvisionedThroughput :WriteCapacityUnits`	Menetapkan jumlah minimum `WriteCapacityUnits` yang digunakan per detik untuk tabel yang ditentukan sebelum DynamoDB menyeimbangkan beban dengan operasi lainnya. Jenis: Angka	Ya

Respons

Sintaks

```
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"}}
```

Nama	Deskripsi
`CreationDateTime`	Tanggal ketika tabel dibuat. Jenis: Angka
`KeySchema`	Struktur kunci primer (sederhana atau komposit) untuk tabel. Pasangan nama-nilai untuk `HashKeyElement` diperlukan, dan pasangan nama-nilai untuk `RangeKeyElement` bersifat opsional (hanya diperlukan untuk kunci primer komposit). Ukuran kunci hash maksimum adalah 2048 byte. Ukuran kunci rentang maksimum adalah 1024 byte. Kedua batas tersebut diberlakukan secara terpisah (yaitu Anda dapat memiliki kombinasi hash + rentang 2048 + 1024 kunci). Untuk informasi selengkapnya tentang kunci primer, lihat Kunci primer. Jenis: Peta `HashKeyElement` , atau `HashKeyElement` dan `RangeKeyElement` untuk kunci primer komposit.
`ProvisionedThroughput`	Pengaturan throughput saat ini untuk tabel yang ditentukan, termasuk nilai untuk `LastIncreaseDateTime` (jika berlaku), `LastDecreaseDateTime` (jika berlaku), Jenis: Array
`TableName`	Nama tabel yang diperbarui. Jenis: String

Nama	Deskripsi
`TableStatus`	Status tabel saat ini (`CREATING`, `ACTIVE`, `DELETING`, atau `UPDATING`), yang seharusnya `UPDATING`. Gunakan operasi DescribeTables untuk memeriksa status tabel. Jenis: String

Kesalahan khusus

Kesalahan	Deskripsi
`ResourceNotFoundException`	Tabel yang ditentukan tidak ditemukan.
`ResourceInUseException`	Tabel ini tidak berada dalam status `ACTIVE`.

Contoh

Permintaan sampel

```
// This header is abbreviated.
// For a sample of a complete header, see API tingkat rendah DynamoDB.
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}
}
```

Respons sampel

```
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"}
}
```

Tindakan terkait

- CreateTable
- DescribeTables
- DeleteTable

Parameter bersyarat DynamoDB lama

Dokumen ini memberikan gambaran umum parameter bersyarat lama di DynamoDB dan merekomendasikan penggunaan parameter ekspresi baru sebagai gantinya. Ini mencakup detail tentang parameter seperti AttributesToGet, AttributeUpdates, ConditionalOperator, Diharapkan KeyConditions, QueryFilter,, dan ScanFilter, dan, dan memberikan contoh bagaimana menggunakan parameter ekspresi baru sebagai pengganti.

> **Important**
>
> Kami menyarankan agar Anda menggunakan parameter ekspresi yang baru, bukan parameter warisan ini, jika memungkinkan. Untuk informasi selengkapnya, lihat Menggunakan ekspresi di DynamoDB.
> Selain itu, DynamoDB tidak mengizinkan pencampuran parameter bersyarat warisan dan parameter ekspresi dalam satu panggilan. Sebagai contoh, memanggil operasi `Query` dengan `AttributesToGet` dan `ConditionExpression` akan memunculkan kesalahan.

Tabel berikut menunjukkan operasi API DynamoDB yang masih mendukung parameter warisan ini, dan parameter ekspresi mana yang akan digunakan sebagai gantinya. Tabel ini dapat membantu jika Anda akan memperbarui aplikasi Anda agar aplikasi tersebut menggunakan parameter ekspresi sebagai gantinya.

Jika Anda menggunakan operasi API ini...	Dengan parameter warisan ini...	Gunakan parameter ekspresi ini sebagai gantinya
`BatchGetItem`	`AttributesToGet`	`ProjectionExpression`
`DeleteItem`	`Expected`	`ConditionExpression`
`GetItem`	`AttributesToGet`	`ProjectionExpression`
`PutItem`	`Expected`	`ConditionExpression`
`Query`	`AttributesToGet`	`ProjectionExpression`
	`KeyConditions`	`KeyConditionExpression`
	`QueryFilter`	`FilterExpression`
`Scan`	`AttributesToGet`	`ProjectionExpression`
	`ScanFilter`	`FilterExpression`
`UpdateItem`	`AttributeUpdates`	`UpdateExpression`
	`Expected`	`ConditionExpression`

Bagian berikut menyediakan informasi lebih lanjut tentang parameter bersyarat warisan.

Topik

- AttributesToGet (warisan)
- AttributeUpdates (warisan)
- ConditionalOperator (warisan)
- Expected (warisan)

- KeyConditions (warisan)
- QueryFilter (warisan)
- ScanFilter (warisan)
- Syarat penulisan dengan parameter warisan

AttributesToGet (warisan)

> **Note**
>
> Kami menyarankan agar Anda menggunakan parameter ekspresi yang baru, bukan parameter warisan ini, jika memungkinkan. Untuk informasi selengkapnya, lihat Menggunakan ekspresi di DynamoDB. Untuk informasi spesifik tentang parameter baru sebagai pengganti untuk yang satu ini, Gunakan ProjectionExpressionsebagai gantinya.

Parameter bersyarat warisan `AttributesToGet` adalah susunan satu atau beberapa atribut untuk diambil dari DynamoDB. Jika tidak ada nama atribut yang disediakan, semua atribut akan dikembalikan. Jika salah satu atribut yang diminta tidak ditemukan, atribut tersebut tidak akan muncul dalam hasil.

`AttributesToGet` memungkinkan Anda untuk mengambil atribut dari jenis Daftar atau Peta; tetapi, parameter tersebut tidak dapat mengambil elemen individu dalam Daftar atau Peta.

Perhatikan bahwa `AttributesToGet` tidak berpengaruh pada konsumsi throughput yang disediakan. DynamoDB menentukan unit kapasitas yang digunakan berdasarkan ukuran item, bukan jumlah data yang dikembalikan ke aplikasi.

Gunakan ProjectionExpressionsebagai gantinya - Contoh

Misalkan Anda ingin mengambil item dari tabel Musik, tapi Anda hanya ingin mengembalikan beberapa atributnya. Anda dapat menggunakan `GetItem` permintaan dengan `AttributesToGet` parameter, seperti dalam AWS CLI contoh ini:

```
aws dynamodb get-item \
    --table-name Music \
    --attributes-to-get '["Artist", "Genre"]' \
    --key '{
        "Artist": {"S":"No One You Know"},
        "SongTitle": {"S":"Call Me Today"}
```

```
    }'
```

Anda dapat menggunakan `ProjectionExpression` sebagai gantinya:

```
aws dynamodb get-item \
    --table-name Music \
    --projection-expression "Artist, Genre" \
    --key '{
        "Artist": {"S":"No One You Know"},
        "SongTitle": {"S":"Call Me Today"}
    }'
```

AttributeUpdates (warisan)

> **Note**
>
> Kami menyarankan agar Anda menggunakan parameter ekspresi yang baru, bukan parameter warisan ini, jika memungkinkan. Untuk informasi selengkapnya, lihat Menggunakan ekspresi di DynamoDB. Untuk informasi spesifik tentang parameter baru sebagai pengganti untuk yang satu ini, gunakan UpdateExpressionsebagai gantinya..

Dalam operasi `UpdateItem`, parameter bersyarat warisan `AttributeUpdates` berisi nama-nama atribut yang akan dimodifikasi, tindakan yang akan dilakukan pada setiap atribut, dan nilai baru untuk setiap atribut. Jika Anda memperbarui atribut yang merupakan atribut kunci indeks untuk setiap indeks pada tabel tersebut, jenis atribut harus sesuai dengan jenis kunci indeks yang didefinisikan dalam `AttributesDefinition` pada deskripsi tabel. Anda dapat menggunakan `UpdateItem` untuk memperbarui setiap atribut non-kunci.

Nilai atribut tidak boleh nol. Atribut jenis Biner dan String harus memiliki panjang lebih dari nol. Atribut jenis set tidak boleh kosong. Permintaan dengan nilai kosong akan ditolak dengan pengecualian `ValidationException`.

Setiap elemen `AttributeUpdates` terdiri dari nama atribut untuk dimodifikasi, beserta hal berikut:

- `Value` - Nilai baru, jika berlaku, untuk atribut ini.
- `Action` - Sebuah nilai yang menentukan cara melakukan pembaruan. Tindakan ini hanya berlaku untuk atribut yang ada yang jenis datanya adalah Angka atau sebuah set; jangan gunakan `ADD` untuk jenis data lainnya.

Jika item dengan kunci primer tertentu ditemukan dalam tabel, nilai berikut melakukan tindakan berikut:

- `PUT` - Menambahkan atribut tertentu pada item. Jika sudah ada, atribut akan digantikan oleh nilai baru.
- `DELETE` - Menghapus atribut dan nilainya, jika tidak ada nilai yang ditentukan untuk `DELETE`. Jenis data dari nilai yang ditentukan harus sesuai dengan jenis data dari nilai yang ada.

  Jika suatu set nilai ditentukan, nilai-nilai tersebut dikurangi dari set lama. Sebagai contoh, jika nilai atribut adalah set `[a,b,c]` dan tindakan `DELETE` menentukan `[a,c]`, nilai atribut akhir adalah `[b]`. Menentukan satu set kosong adalah kesalahan.
- `ADD` - Menambahkan nilai yang ditentukan untuk item, jika atribut belum ada. Jika atribut tidak ada, perilaku `ADD` bergantung pada jenis data dari atribut:
  - Jika atribut yang ada berupa angka, dan jika `Value` juga berupa angka, `Value` secara matematis ditambahkan ke atribut yang ada. Jika `Value` adalah angka negatif, nilai tersebut dikurangi dari atribut yang ada.

    > **Note**
    >
    > Jika Anda menggunakan `ADD` untuk menambah atau mengurangi nilai angka untuk item yang tidak ada sebelum pembaruan, DynamoDB menggunakan 0 sebagai nilai awal.
    > Begitu pun halnya, jika Anda menggunakan `ADD` untuk item yang ada untuk menambah atau mengurangi nilai atribut yang tidak ada sebelum pembaruan, DynamoDB menggunakan `0` sebagai nilai awal. Misalnya, anggap item yang ingin Anda perbarui tidak memiliki atribut bernama itemcount, tetapi Anda memutuskan untuk tetap `ADD` angka `3` ke atribut ini. DynamoDB akan membuat atribut itemcount, mengatur nilai awalnya ke `0`, dan akhirnya menambahkan `3` ke nilai tersebut. Hasilnya akan menjadi atribut itemcount baru, dengan nilai `3`.
  - Jika jenis data yang ada merupakan sebuah set, dan jika `Value` juga merupakan sebuah set, `Value` ditambahkan ke set yang ada. Misalnya, jika nilai atribut adalah set `[1,2]`, dan tindakan `ADD` menentukan `[3]`, nilai atribut akhir adalah `[1,2,3]`. Terjadi kesalahan jika tindakan `ADD` ditentukan untuk atribut set dan jenis atribut yang ditentukan tidak cocok dengan jenis set yang ada.

Kedua set harus memiliki jenis data primitif yang sama. Sebagai contoh, jika jenis data yang ada adalah set string, `Value` juga harus berupa set string.

Jika item dengan kunci yang ditentukan ditemukan dalam tabel, nilai berikut melakukan tindakan berikut:

- `PUT` - Menyebabkan DynamoDB membuat item baru dengan kunci primer yang ditentukan, lalu menambahkan atribut.
- `DELETE` - Tidak terjadi apa-apa, karena atribut tidak dapat dihapus dari item yang tidak ada. Operasi berhasil, tapi DynamoDB tidak membuat item baru.
- `ADD` - Menyebabkan DynamoDB membuat item dengan kunci primer yang disediakan dan angka (atau set angka) untuk nilai atribut. Satu-satunya jenis data yang diperbolehkan adalah Angka dan Set Angka.

Jika Anda memberikan atribut apa pun yang merupakan bagian dari kunci indeks, set data untuk atribut tersebut harus sesuai dengan skema dalam definisi atribut tabel.

Gunakan UpdateExpressionsebagai gantinya - Contoh

Misalkan Anda ingin memodifikasi item dalam tabel Musik. Anda dapat menggunakan `UpdateItem` permintaan dengan `AttributeUpdates` parameter, seperti dalam AWS CLI contoh ini:

```
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"}
        }
    }'
```

Anda dapat menggunakan `UpdateExpression` sebagai gantinya:

```
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 (warisan)

> **Note**
>
> Kami menyarankan agar Anda menggunakan parameter ekspresi yang baru, bukan parameter warisan ini, jika memungkinkan. Untuk informasi selengkapnya, lihat Menggunakan ekspresi di DynamoDB.

Parameter bersyarat warisan `ConditionalOperator` adalah operator logis yang digunakan untuk menerapkan syarat dalam peta `Expected`, `QueryFilter`, atau `ScanFilter`:

- AND - Jika semua syarat bernilai true, seluruh peta bernilai true.
- OR - Jika setidaknya salah satu syarat bernilai true, seluruh peta bernilai true.

Jika Anda menghilangkan `ConditionalOperator`, `AND` menjadi pengaturan default.

Operasi akan berhasil hanya jika seluruh peta bernilai true.

> **Note**
>
> Parameter ini tidak mendukung atribut jenis Daftar atau Peta.

Expected (warisan)

> **Note**
>
> Kami menyarankan agar Anda menggunakan parameter ekspresi yang baru, bukan parameter warisan ini, jika memungkinkan. Untuk informasi selengkapnya, lihat

Menggunakan ekspresi di DynamoDB. Untuk informasi spesifik tentang parameter baru sebagai pengganti untuk yang satu ini, gunakan ConditionExpressionsebagai gantinya..

Parameter bersyarat warisan `Expected` adalah blok bersyarat untuk operasi `UpdateItem`. `Expected` adalah peta pasangan atribut/syarat. Setiap elemen peta terdiri dari nama atribut, operator perbandingan, dan satu atau beberapa nilai. DynamoDB membandingkan atribut dengan nilai(-nilai) yang Anda berikan, menggunakan operator perbandingan. Untuk setiap elemen `Expected`, hasil evaluasi adalah true atau false.

Jika Anda menentukan lebih dari satu elemen dalam peta `Expected`, secara default semua syarat harus bernilai true. Dengan kata lain, kondisi digabungkan menggunakan `AND` operator. (Anda dapat menggunakan parameter `ConditionalOperator` untuk syarat OR sebagai gantinya. Jika Anda melakukan ini, setidaknya salah satu syarat harus bernilai true, bukan semuanya.)

Jika peta `Expected` bernilai true, operasi bersyarat berhasil; jika tidak, gagal.

`Expected` berisi hal berikut ini:

- `AttributeValueList` - Satu atau beberapa nilai untuk mengevaluasi atribut yang disediakan. Jumlah nilai dalam daftar tergantung pada `ComparisonOperator` yang sedang digunakan.

  Untuk jenis Angka, perbandingan nilainya bersifat numerik.

  Perbandingan nilai string untuk lebih besar dari, sama dengan, atau kurang dari didasarkan pada Unicode dengan UTF-8 encoding biner. Sebagai contoh, `a` lebih besar dari `A`, dan `a` lebih besar dari `B`.

  Untuk jenis Biner, DynamoDB memperlakukan setiap byte data biner sebagai tidak bertanda ketika membandingkan nilai-nilai biner.
- `ComparisonOperator` - Sebuah pembanding untuk mengevaluasi atribut dalam `AttributeValueList`. Saat melakukan perbandingan, DynamoDB menggunakan bacaan sangat konsisten.

  Operator perbandingan berikut ini tersedia:

  `EQ | NE | LE | LT | GE | GT | NOT_NULL | NULL | CONTAINS | NOT_CONTAINS | BEGINS_WITH | IN | BETWEEN`

  Berikut adalah deskripsi setiap operator perbandingan.

- EQ: Sama dengan. EQ didukung untuk semua jenis data, termasuk daftar dan peta.

  AttributeValueList hanya dapat berisi satu elemen AttributeValue berjenis String, Angka, Biner, Set String, Set Angka, atau Set Biner. Jika item berisi elemen AttributeValue dengan jenis berbeda dari yang disediakan dalam permintaan, nilainya tidak cocok. Sebagai contoh, {"S":"6"} tidak sama dengan {"N":"6"}. Selain itu, {"N":"6"} tidak sama dengan {"NS":["6", "2", "1"]}.

- NE: Tidak sama dengan. NE didukung untuk semua set data, termasuk daftar dan peta.

  AttributeValueList hanya dapat berisi satu AttributeValue berjenis String, Angka, Biner, Set String, Set Angka, atau Set Biner. Jika item berisi AttributeValue dengan jenis berbeda dari yang disediakan dalam permintaan, nilainya tidak cocok. Sebagai contoh, {"S":"6"} tidak sama dengan {"N":"6"}. Selain itu, {"N":"6"} tidak sama dengan {"NS":["6", "2", "1"]}.

- LE: Kurang dari atau sama dengan.

  AttributeValueList hanya dapat berisi satu elemen AttributeValue berjenis String, Angka, atau Biner (bukan jenis set). Jika item berisi elemen AttributeValue dengan jenis berbeda dari yang disediakan dalam permintaan, nilainya tidak cocok. Sebagai contoh, {"S":"6"} tidak sama dengan {"N":"6"}. Selain itu, {"N":"6"} tidak sebanding dengan {"NS":["6", "2", "1"]}.

- LT: Kurang dari.

  AttributeValueList hanya dapat berisi satu AttributeValue berjenis String, Angka, atau Biner (bukan jenis set). Jika item berisi elemen AttributeValue dengan jenis berbeda dari yang disediakan dalam permintaan, nilainya tidak cocok. Sebagai contoh, {"S":"6"} tidak sama dengan {"N":"6"}. Selain itu, {"N":"6"} tidak sebanding dengan {"NS":["6", "2", "1"]}.

- GE: Lebih besar dari atau sama dengan.

  AttributeValueList hanya dapat berisi satu elemen AttributeValue berjenis String, Angka, atau Biner (bukan jenis set). Jika item berisi elemen AttributeValue dengan jenis berbeda dari yang disediakan dalam permintaan, nilainya tidak cocok. Sebagai contoh, {"S":"6"} tidak sama dengan {"N":"6"}. Selain itu, {"N":"6"} tidak sebanding dengan {"NS":["6", "2", "1"]}.

- GT: Lebih besar dari.

`AttributeValueList` hanya dapat berisi satu elemen `AttributeValue` berjenis String, Angka, atau Biner (bukan jenis set). Jika item berisi elemen `AttributeValue` dengan jenis berbeda dari yang disediakan dalam permintaan, nilainya tidak cocok. Sebagai contoh, `{"S":"6"}` tidak sama dengan `{"N":"6"}`. Selain itu, `{"N":"6"}` tidak sebanding dengan `{"NS":["6", "2", "1"]}`.

- `NOT_NULL`: Atribut ada. `NOT_NULL` didukung untuk semua jenis data, termasuk daftar dan peta.

  > **Note**
  >
  > Operator ini menguji keberadaan atribut, bukan jenis datanya. Jika jenis data atribut "a" adalah nol, dan Anda mengevaluasinya menggunakan `NOT_NULL`, hasilnya adalah Boolean `true`. Hasil ini karena atribut "a" ada; jenis data tidak relevan dengan operator perbandingan `NOT_NULL`.

- `NULL`: Atribut tidak ada. `NULL` didukung untuk semua jenis data, termasuk daftar dan peta.

  > **Note**
  >
  > Operator ini menguji tidak adanya atribut, bukan jenis datanya. Jika jenis data atribut "a" adalah nol, dan Anda mengevaluasinya menggunakan `NULL`, hasilnya adalah Boolean `false`. Hasil ini karena atribut "a" ada; jenis datanya tidak relevan dengan operator perbandingan `NULL`.

- `CONTAINS`: Memeriksa urutan berikutnya, atau nilai dalam suatu set.

  `AttributeValueList` hanya dapat berisi satu elemen `AttributeValue` berjenis String, Angka, atau Biner (bukan jenis set). Jika atribut target perbandingan berjenis String, operator memeriksa kecocokan substring. Jika atribut target perbandingan berjenis Biner, operator mencari urutan berikutnya dari target yang cocok dengan input. Jika atribut target perbandingan adalah suatu set ("SS", "NS", atau "BS"), operator bernilai true jika menemukan kecocokan sama persis dengan anggota set tersebut.

  CONTAINS didukung untuk daftar: Saat mengevaluasi "a `CONTAINS` b", "a" dapat berupa sebuah daftar; tetapi, "b" tidak dapat berupa set, peta, atau daftar.

- `NOT_CONTAINS`: Memeriksa tidak adanya urutan berikutnya, atau tidak adanya nilai dalam suatu set.

`AttributeValueList` hanya dapat berisi satu elemen `AttributeValue` berjenis String, Angka, atau Biner (bukan jenis set). Jika atribut target perbandingan adalah String, operator memeriksa tidak adanya kecocokan substring. Jika atribut target perbandingan adalah Biner, operator mencari tidak adanya urutan berikutnya dari target yang cocok dengan input. Jika atribut target perbandingan adalah suatu set ("SS", "NS", atau "BS"), operator bernilai true jika `does not` menemukan kecocokan sama persis dengan anggota set tersebut.

NOT_CONTAINS didukung untuk daftar: Saat mengevaluasi "`a NOT CONTAINS b`", "a" dapat berupa sebuah daftar; tetapi, "b" tidak dapat berupa set, peta, atau daftar.

- `BEGINS_WITH`: Memeriksa prefiks.

  `AttributeValueList` hanya dapat berisi satu `AttributeValue` berjenis String atau Biner (bukan jenis Angka atau set). Atribut target perbandingan harus berjenis String atau Biner (bukan jenis Angka atau set).

- `IN`: Memeriksa elemen yang cocok dalam dua set.

  `AttributeValueList` dapat berisi satu atau beberapa elemen `AttributeValue` berjenis String, Angka, atau Biner (bukan jenis set). Atribut ini dibandingkan dengan atribut jenis set yang ada dari suatu item. Jika elemen apa pun dari set input ada dalam atribut item, ekspresi bernilai true.

- `BETWEEN`: Lebih besar dari atau sama dengan nilai pertama, dan kurang dari atau sama dengan nilai kedua.

  `AttributeValueList` harus berisi dua elemen `AttributeValue` berjenis yang sama, baik String, Angka, atau Biner (bukan jenis set). Suatu atribut target cocok jika nilai target lebih besar dari, atau sama dengan, elemen pertama dan kurang dari, atau sama dengan, elemen kedua. Jika item berisi elemen `AttributeValue` dengan jenis berbeda dari yang disediakan dalam permintaan, nilainya tidak cocok. Misalnya, `{"S":"6"}` tidak sebanding dengan `{"N":"6"}`. Selain itu, `{"N":"6"}` tidak sebanding dengan `{"NS":["6", "2", "1"]}`

Parameter berikut dapat digunakan sebagai pengganti `AttributeValueList` dan `ComparisonOperator`:

- `Value` - Nilai untuk DynamoDB guna membandingkan dengan atribut.
- `Exists` - Nilai Boolean yang menyebabkan DynamoDB mengevaluasi nilai sebelum mencoba operasi bersyarat:

- Jika `Exists` adalah `true`, DynamoDB akan memeriksa untuk melihat apakah nilai atribut sudah ada dalam tabel. Jika ditemukan, syarat bernilai true; jika tidak, syarat akan bernilai false.
- Jika `Exists` adalah `false`, DynamoDB mengasumsikan bahwa nilai atribut `not` ada dalam tabel. Jika pada dasarnya nilai tidak ada, asumsi tersebut valid dan syarat bernilai true. Jika nilai ditemukan, terlepas dari asumsi bahwa nilai itu tidak ada, syaratnya bernilai false.

Perhatikan bahwa nilai default untuk `Exists` adalah `true`.

Parameter `Value` dan `Exists` tidak kompatibel dengan `AttributeValueList` dan `ComparisonOperator`. Perhatikan bahwa jika Anda menggunakan kedua set parameter sekaligus, DynamoDB akan mengembalikan pengecualian `ValidationException`.

> **Note**
>
> Parameter ini tidak mendukung atribut jenis Daftar atau Peta.

Gunakan ConditionExpressionsebagai gantinya - Contoh

Misalkan Anda ingin memodifikasi item dalam tabel Musik, tetapi hanya jika syarat tertentu bernilai true. Anda dapat menggunakan `UpdateItem` permintaan dengan `Expected` parameter, seperti dalam AWS CLI contoh ini:

```
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"} ]
        }
```

```
    }'
```

Anda dapat menggunakan `ConditionExpression` sebagai gantinya:

```
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 (warisan)

> **Note**
>
> Kami menyarankan agar Anda menggunakan parameter ekspresi yang baru, bukan parameter warisan ini, jika memungkinkan. Untuk informasi selengkapnya, lihat Menggunakan ekspresi di DynamoDB. Untuk informasi spesifik tentang parameter baru sebagai pengganti untuk yang satu ini, gunakan KeyConditionExpressionsebagai gantinya..

Parameter bersyarat warisan `KeyConditions` berisi kriteria pemilihan untuk operasi `Query`. Untuk kueri pada tabel, Anda dapat memiliki syarat pada tabel atribut kunci primer saja. Anda harus memberikan nama kunci partisi dan nilai sebagai syarat `EQ`. Anda dapat memberikan syarat kedua secara opsional, dengan merujuk pada kunci urutan.

> **Note**
>
> Jika Anda tidak memberikan syarat kunci urutan, semua item yang cocok dengan kunci partisi akan diambil. Jika ada, `FilterExpression` atau `QueryFilter` akan diterapkan setelah item diambil.

Untuk kueri pada indeks, Anda dapat memiliki syarat hanya pada atribut kunci indeks. Anda harus memberikan nama kunci partisi indeks dan nilai sebagai syarat EQ. Anda dapat memberikan syarat kedua secara opsional, dengan merujuk pada kunci urutan indeks.

Setiap elemen KeyConditions terdiri dari nama atribut untuk dimodifikasi, beserta hal berikut:

- AttributeValueList - Satu atau beberapa nilai untuk mengevaluasi atribut yang disediakan. Jumlah nilai dalam daftar tergantung pada ComparisonOperator yang sedang digunakan.

  Untuk jenis Angka, perbandingan nilainya bersifat numerik.

  Perbandingan nilai string untuk lebih besar dari, sama dengan, atau kurang dari didasarkan pada Unicode dengan UTF-8 encoding biner. Sebagai contoh, a lebih besar dari A, dan a lebih besar dari B.

  Untuk Biner, DynamoDB memperlakukan setiap byte data biner sebagai tidak bertanda ketika membandingkan nilai-nilai biner.
- ComparisonOperator - Sebuah pembanding untuk mengevaluasi atribut. Misalnya: sama dengan, lebih besar dari, dan kurang dari.

  Untuk KeyConditions, hanya operator perbandingan berikut yang didukung:

  ```
  EQ | LE | LT | GE | GT | BEGINS_WITH | BETWEEN
  ```

  Berikut adalah deskripsi setiap operator perbandingan.

  - EQ: Sama dengan.

    AttributeValueList hanya dapat berisi satu AttributeValue berjenis String, Angka, atau Biner (bukan jenis set). Jika item berisi elemen AttributeValue dengan jenis berbeda dari yang ditentukan dalam permintaan, nilainya tidak cocok. Sebagai contoh, {"S":"6"} tidak sama dengan {"N":"6"}. Selain itu, {"N":"6"} tidak sama dengan {"NS":["6", "2", "1"]}.
  - LE: Kurang dari atau sama dengan.

    AttributeValueList hanya dapat berisi satu elemen AttributeValue berjenis String, Angka, atau Biner (bukan jenis set). Jika item berisi elemen AttributeValue dengan jenis berbeda dari yang disediakan dalam permintaan, nilainya tidak cocok. Sebagai contoh, {"S":"6"} tidak sama dengan {"N":"6"}. Selain itu, {"N":"6"} tidak sebanding dengan {"NS":["6", "2", "1"]}.

- LT: Kurang dari.

  AttributeValueList hanya dapat berisi satu AttributeValue berjenis String, Angka, atau Biner (bukan jenis set). Jika item berisi elemen AttributeValue dengan jenis berbeda dari yang disediakan dalam permintaan, nilainya tidak cocok. Sebagai contoh, {"S":"6"} tidak sama dengan {"N":"6"}. Selain itu, {"N":"6"} tidak sebanding dengan {"NS":["6", "2", "1"]}.

- GE: Lebih besar dari atau sama dengan.

  AttributeValueList hanya dapat berisi satu elemen AttributeValue berjenis String, Angka, atau Biner (bukan jenis set). Jika item berisi elemen AttributeValue dengan jenis berbeda dari yang disediakan dalam permintaan, nilainya tidak cocok. Sebagai contoh, {"S":"6"} tidak sama dengan {"N":"6"}. Selain itu, {"N":"6"} tidak sebanding dengan {"NS":["6", "2", "1"]}.

- GT: Lebih besar dari.

  AttributeValueList hanya dapat berisi satu elemen AttributeValue berjenis String, Angka, atau Biner (bukan jenis set). Jika item berisi elemen AttributeValue dengan jenis berbeda dari yang disediakan dalam permintaan, nilainya tidak cocok. Sebagai contoh, {"S":"6"} tidak sama dengan {"N":"6"}. Selain itu, {"N":"6"} tidak sebanding dengan {"NS":["6", "2", "1"]}.

- BEGINS_WITH: Memeriksa prefiks.

  AttributeValueList hanya dapat berisi satu AttributeValue berjenis String atau Biner (bukan jenis Angka atau set). Atribut target perbandingan harus berjenis String atau Biner (bukan jenis Angka atau set).

- BETWEEN: Lebih besar dari atau sama dengan nilai pertama, dan kurang dari atau sama dengan nilai kedua.

  AttributeValueList harus berisi dua elemen AttributeValue berjenis yang sama, baik String, Angka, atau Biner (bukan jenis set). Suatu atribut target cocok jika nilai target lebih besar dari, atau sama dengan, elemen pertama dan kurang dari, atau sama dengan, elemen kedua. Jika item berisi elemen AttributeValue dengan jenis berbeda dari yang disediakan dalam permintaan, nilainya tidak cocok. Misalnya, {"S":"6"} tidak sebanding dengan {"N":"6"}. Selain itu, {"N":"6"} tidak sebanding dengan {"NS":["6", "2", "1"]}.

Gunakan KeyConditionExpressionsebagai gantinya - Contoh

Misalkan Anda ingin mengambil beberapa item dengan kunci partisi yang sama dari tabel Musik. Anda dapat menggunakan `Query` permintaan dengan `KeyConditions` parameter, seperti dalam AWS CLI contoh ini:

```
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"} ]
        }
    }'
```

Anda dapat menggunakan `KeyConditionExpression` sebagai gantinya:

```
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 (warisan)

> **Note**
>
> Kami menyarankan agar Anda menggunakan parameter ekspresi yang baru, bukan parameter warisan ini, jika memungkinkan. Untuk informasi selengkapnya, lihat Menggunakan ekspresi di DynamoDB. Untuk informasi spesifik tentang parameter baru sebagai pengganti untuk yang satu ini, gunakan FilterExpressionsebagai gantinya..

Dalam operasi `Query`, parameter bersyarat warisan `QueryFilter` adalah syarat yang mengevaluasi hasil kueri setelah item dibaca dan hanya mengembalikan nilai-nilai yang diinginkan.

Parameter ini tidak mendukung atribut jenis Daftar atau Peta.

> **Note**
>
> `QueryFilter` diterapkan setelah item dibaca; proses penyaringan tidak menggunakan unit kapasitas baca tambahan.

Jika Anda memberikan lebih dari satu syarat dalam peta `QueryFilter`, secara default semua syarat harus bernilai true. Dengan kata lain, kondisi digabungkan menggunakan `AND` operator. (Anda dapat menggunakan parameter ConditionalOperator (warisan) untuk syarat OR sebagai gantinya. Jika Anda melakukan ini, setidaknya salah satu syarat harus bernilai true, bukan semuanya.)

Perhatikan bahwa `QueryFilter` tidak mengizinkan atribut kunci. Anda tidak dapat menentukan syarat filter pada kunci partisi atau kunci urutan.

Setiap elemen `QueryFilter` terdiri dari nama atribut untuk dimodifikasi, beserta hal berikut:

- `AttributeValueList` - Satu atau beberapa nilai untuk mengevaluasi atribut yang disediakan. Jumlah nilai dalam daftar tergantung pada operator yang ditentukan dalam `ComparisonOperator`.

  Untuk jenis Angka, perbandingan nilainya bersifat numerik.

  Perbandingan nilai string untuk lebih besar dari, sama dengan, atau kurang dari didasarkan pada pengodean biner UTF-8. Sebagai contoh, `a` lebih besar dari `A`, dan `a` lebih besar dari `B`.

  Untuk jenis Biner, DynamoDB memperlakukan setiap byte data biner sebagai tidak bertanda ketika membandingkan nilai-nilai biner.

  Untuk informasi tentang menentukan jenis data di JSON, lihat API tingkat rendah DynamoDB.
- `ComparisonOperator` - Sebuah pembanding untuk mengevaluasi atribut. Misalnya: sama dengan, lebih besar dari, dan kurang dari.

  Operator perbandingan berikut ini tersedia:

  ```
  EQ | NE | LE | LT | GE | GT | NOT_NULL | NULL | CONTAINS | NOT_CONTAINS |
  BEGINS_WITH | IN | BETWEEN
  ```

Gunakan FilterExpressionsebagai gantinya - Contoh

Misalkan Anda ingin mengajukan kueri tabel Musik dan menerapkan syarat untuk item yang cocok. Anda dapat menggunakan permintaan `Query` dengan parameter `QueryFilter`, seperti dalam contoh AWS CLI ini:

```
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"} ]
        }
    }'
```

Anda dapat menggunakan `FilterExpression` sebagai gantinya:

```
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 (warisan)

> **Note**
>
> Kami menyarankan agar Anda menggunakan parameter ekspresi yang baru, bukan parameter warisan ini, jika memungkinkan. Untuk informasi selengkapnya, lihat Menggunakan ekspresi di DynamoDB. Untuk informasi spesifik tentang parameter baru sebagai pengganti untuk yang satu ini, gunakan FilterExpressionsebagai gantinya..

Dalam operasi `Scan`, parameter bersyarat warisan `ScanFilter` adalah syarat yang mengevaluasi hasil pemindaian dan hanya mengembalikan nilai-nilai yang diinginkan.

> **Note**
>
> Parameter ini tidak mendukung atribut jenis Daftar atau Peta.

Jika Anda menentukan lebih dari satu syarat di peta `ScanFilter`, secara default semua syarat harus bernilai true. Dengan kata lain, kondisinya ANDed bersama. (Anda dapat menggunakan parameter ConditionalOperator (warisan) untuk syarat OR sebagai gantinya. Jika Anda melakukan ini, setidaknya salah satu syarat harus bernilai true, bukan semuanya.)

Setiap elemen `ScanFilter` terdiri dari nama atribut untuk dimodifikasi, beserta hal berikut:

- `AttributeValueList` - Satu atau beberapa nilai untuk mengevaluasi atribut yang disediakan. Jumlah nilai dalam daftar tergantung pada operator yang ditentukan dalam `ComparisonOperator`.

  Untuk jenis Angka, perbandingan nilainya bersifat numerik.

  Perbandingan nilai string untuk lebih besar dari, sama dengan, atau kurang dari didasarkan pada pengodean biner UTF-8. Sebagai contoh, `a` lebih besar dari `A`, dan `a` lebih besar dari `B`.

  Untuk Biner, DynamoDB memperlakukan setiap byte data biner sebagai tidak bertanda ketika membandingkan nilai-nilai biner.

  Untuk informasi tentang menentukan jenis data di JSON, lihat API tingkat rendah DynamoDB.
- `ComparisonOperator` - Sebuah pembanding untuk mengevaluasi atribut. Misalnya: sama dengan, lebih besar dari, dan kurang dari.

  Operator perbandingan berikut ini tersedia:

  ```
  EQ | NE | LE | LT | GE | GT | NOT_NULL | NULL | CONTAINS | NOT_CONTAINS |
  BEGINS_WITH | IN | BETWEEN
  ```

Gunakan FilterExpressionsebagai gantinya - Contoh

Misalkan Anda ingin memindai tabel Musik dan menerapkan syarat untuk item yang cocok. Anda dapat menggunakan permintaan `Scan` dengan parameter `ScanFilter`, seperti dalam contoh AWS CLI ini:

```
aws dynamodb scan \
    --table-name Music \
    --scan-filter '{
        "Genre":{
            "AttributeValueList":[ {"S":"Rock"} ],
            "ComparisonOperator": "EQ"
        }
    }'
```

Anda dapat menggunakan `FilterExpression` sebagai gantinya:

```
aws dynamodb scan \
    --table-name Music \
    --filter-expression 'Genre = :g' \
    --expression-attribute-values '{
        ":g": {"S":"Rock"}
    }'
```

Syarat penulisan dengan parameter warisan

> **Note**
>
> Kami menyarankan agar Anda menggunakan parameter ekspresi yang baru, bukan parameter warisan ini, jika memungkinkan. Untuk informasi selengkapnya, lihat Menggunakan ekspresi di DynamoDB.

Bagian berikut menjelaskan cara menulis syarat untuk digunakan dengan parameter warisan, seperti `Expected`, `QueryFilter`, dan `ScanFilter`.

> **Note**
>
> Aplikasi baru harus menggunakan parameter ekspresi sebagai gantinya. Untuk informasi selengkapnya, lihat Menggunakan ekspresi di DynamoDB.

Syarat sederhana

Dengan nilai atribut, Anda dapat menulis syarat untuk perbandingan terhadap atribut tabel. Syarat selalu bernilai true atau false, dan terdiri dari:

- `ComparisonOperator` – lebih besar dari, kurang dari, sama dengan, dan sebagainya.
- `AttributeValueList` (opsional) – nilai atribut untuk dibandingkan. Tergantung pada `ComparisonOperator` yang sedang digunakan, `AttributeValueList` mungkin berisi satu, dua, atau lebih nilai; atau mungkin tidak ada sama sekali.

Bagian berikut menjelaskan berbagai operator perbandingan, beserta contoh cara menggunakannya dalam syarat.

Operator perbandingan tanpa nilai atribut

- `NOT_NULL` - true jika atribut ada.
- `NULL` - true jika atribut tidak ada.

Gunakan operator ini untuk memeriksa apakah atribut ada, atau tidak ada. Karena tidak ada nilai untuk dibandingkan, jangan tentukan `AttributeValueList`.

Contoh

Ekspresi berikut bernilai true jika ada atribut Dimensi.

```
...
    "Dimensions": {
         ComparisonOperator: "NOT_NULL"
    }
...
```

Operator perbandingan dengan satu nilai atribut

- `EQ` - true jika atribut sama dengan sebuah nilai.

  `AttributeValueList` hanya dapat berisi satu nilai berjenis String, Angka, Biner, Set String, Set Angka, atau Set Biner. Jika item berisi nilai dengan jenis berbeda dari jenis yang ditentukan dalam permintaan, nilainya tidak cocok. Misalnya, string `"3"` tidak sama dengan angka `3`. Selain itu, angka `3` tidak sama dengan set angka `[3, 2, 1]`.
- `NE` - true jika atribut tidak sama dengan sebuah nilai.

  `AttributeValueList` hanya dapat berisi satu nilai berjenis String, Angka, Biner, Set String, Set Angka, atau Set Biner. Jika item berisi nilai dengan jenis berbeda dari jenis yang ditentukan dalam permintaan, nilainya tidak cocok.
- `LE` - true jika atribut kurang dari atau sama dengan sebuah nilai.

  `AttributeValueList` hanya dapat berisi satu nilai berjenis String, Angka, atau Biner (bukan set). Jika item berisi `AttributeValue` dengan jenis berbeda dari yang ditentukan dalam permintaan, nilainya tidak cocok.
- `LT` - true jika atribut kurang dari sebuah nilai.

  `AttributeValueList` hanya dapat berisi satu nilai berjenis String, Angka, atau Biner (bukan set). Jika item berisi nilai dengan jenis berbeda dari jenis yang ditentukan dalam permintaan, nilainya tidak cocok.
- `GE` - true jika atribut lebih besar dari atau sama dengan suatu nilai.

  `AttributeValueList` hanya dapat berisi satu nilai berjenis String, Angka, atau Biner (bukan set). Jika item berisi nilai dengan jenis berbeda dari jenis yang ditentukan dalam permintaan, nilainya tidak cocok.
- `GT` - true jika atribut lebih besar dari sebuah nilai.

  `AttributeValueList` hanya dapat berisi satu nilai berjenis String, Angka, atau Biner (bukan set). Jika item berisi nilai dengan jenis berbeda dari jenis yang ditentukan dalam permintaan, nilainya tidak cocok.
- `CONTAINS` - true jika nilai ada dalam suatu set, atau jika satu nilai berisi nilai lain.

  `AttributeValueList` hanya dapat berisi satu nilai berjenis String, Angka, atau Biner (bukan set). Jika atribut target perbandingan berupa String, operator memeriksa kecocokan substring. Jika atribut target perbandingan berupa Biner, operator mencari urutan berikutnya dari target yang

cocok dengan input. Jika atribut target perbandingan adalah suatu set, operator bernilai true jika menemukan kecocokan sama persis dengan anggota set tersebut.

- `NOT_CONTAINS` - true jika nilai tidak ada dalam suatu set, atau jika satu nilai tidak berisi nilai lainnya.

  `AttributeValueList` hanya dapat berisi satu nilai berjenis String, Angka, atau Biner (bukan set). Jika atribut target perbandingan adalah String, operator memeriksa tidak adanya kecocokan substring. Jika atribut target perbandingan adalah Biner, operator mencari tidak adanya urutan berikutnya dari target yang cocok dengan input. Jika atribut target perbandingan adalah sebuah set, operator akan bernilai true jika tidak menemukan kecocokan yang sama persis dengan anggota set.

- `BEGINS_WITH` - true jika beberapa karakter pertama atribut cocok dengan nilai yang disediakan. Jangan gunakan operator ini untuk membandingkan angka.

  `AttributeValueList` hanya dapat berisi satu nilai berjenis String atau Biner (bukan Angka atau set). Atribut target perbandingan harus berupa String atau Biner (bukan Angka atau set).

Gunakan operator ini untuk membandingkan atribut dengan sebuah nilai. Anda harus menentukan `AttributeValueList` yang berisi nilai tunggal. Untuk sebagian besar operator, nilai ini harus berupa skalar; tetapi, operator `EQ` dan `NE` juga mendukung set.

Contoh

Ekspresi berikut bernilai true jika:

- Harga produk lebih besar dari 100.

```
...
    "Price": {
        ComparisonOperator: "GT",
        AttributeValueList: [ {"N":"100"} ]
    }
...
```

- Kategori produk dimulai dengan "Bo".

```
...
    "ProductCategory": {
        ComparisonOperator: "BEGINS_WITH",
```

```
        AttributeValueList: [ {"S":"Bo"} ]
    }
...
```

- Produk tersedia dalam warna merah, hijau, atau hitam:

```
...
    "Color": {
        ComparisonOperator: "EQ",
        AttributeValueList: [
            [ {"S":"Black"}, {"S":"Red"}, {"S":"Green"} ]
        ]
    }
...
```

> **Note**
>
> Saat membandingkan jenis data set, urutan elemen tidak menjadi masalah. DynamoDB hanya akan mengembalikan item dengan set nilai yang sama, terlepas dari urutan yang Anda tentukan dalam permintaan Anda.

Operator perbandingan dengan dua nilai atribut

- `BETWEEN` - true jika nilai adalah antara batas bawah dan batas atas, termasuk titik akhir.

  `AttributeValueList` harus berisi dua elemen dari jenis yang sama, baik String, Angka, atau Biner (bukan set). Suatu atribut target cocok jika nilai target lebih besar dari, atau sama dengan, elemen pertama dan kurang dari, atau sama dengan, elemen kedua. Jika item berisi nilai dengan jenis berbeda dari jenis yang ditentukan dalam permintaan, nilainya tidak cocok.

Gunakan operator ini untuk menentukan apakah nilai atribut termasuk dalam kisaran. `AttributeValueList` harus berisi dua elemen skalar berjenis yang sama - String, Angka, atau Biner.

Contoh

Ekspresi berikut bernilai true jika harga produk adalah antara 100 dan 200.

```
...
```

```
    "Price": {
        ComparisonOperator: "BETWEEN",
        AttributeValueList: [ {"N":"100"}, {"N":"200"} ]
    }
...
```

Operator perbandingan dengan nilai atribut n

- `IN` - true jika nilai sama dengan salah satu nilai dalam daftar yang dienumerasi. Hanya nilai skalar yang didukung dalam daftar, bukan set. Atribut target harus memiliki jenis yang sama dan nilai persis agar cocok.

  `AttributeValueList` dapat berisi satu atau beberapa elemen jenis String, Angka, atau Biner (bukan set). Atribut ini dibandingkan dengan atribut jenis non-set yang ada dari suatu item. Jika elemen apa pun dari set input ada dalam atribut item, ekspresi bernilai true.

  `AttributeValueList` dapat berisi satu atau beberapa nilai berjenis String, Angka, atau Biner (bukan set). Atribut target perbandingan harus memiliki jenis yang sama dan nilai persis agar cocok. Sebuah String tidak pernah cocok dengan set String.

Gunakan operator ini untuk menentukan apakah nilai yang disediakan berada dalam daftar yang dienumerasi. Anda dapat menentukan nilai skalar apa pun di `AttributeValueList`, tetapi semua angka tersebut harus dari jenis data yang sama.

Contoh

Ekspresi berikut bernilai true jika nilai untuk Id adalah 201, 203, atau 205.

```
...
    "Id": {
        ComparisonOperator: "IN",
        AttributeValueList: [ {"N":"201"}, {"N":"203"}, {"N":"205"} ]
    }
...
```

Menggunakan beberapa syarat

DynamoDB memungkinkan Anda menggabungkan beberapa syarat untuk membentuk ekspresi kompleks. Anda melakukan ini dengan menyediakan setidaknya dua ekspresi, dengan ConditionalOperator (warisan) opsional.

Secara default, ketika Anda menentukan lebih dari satu syarat, semua syarat harus bernilai true agar seluruh ekspresi bernilai true. Dengan kata lain, operasi AND sedang berlangsung secara implisit.

Contoh

Ekspresi berikut bernilai true jika produk adalah buku yang memiliki setidaknya 600 halaman. Kedua syarat harus bernilai true, karena keduanya menjalankan AND secara implisit dan bersama-sama.

```
...
    "ProductCategory": {
        ComparisonOperator: "EQ",
        AttributeValueList: [ {"S":"Book"} ]
    },
    "PageCount": {
        ComparisonOperator: "GE",
        AttributeValueList: [ {"N":600"} ]
    }
...
```

Anda dapat menggunakan ConditionalOperator (warisan) untuk memperjelas bahwa operasi AND akan berlangsung. Contoh berikut berperilaku dengan cara yang sama seperti sebelumnya.

```
...
    "ConditionalOperator" : "AND",
    "ProductCategory": {
        "ComparisonOperator": "EQ",
        "AttributeValueList": [ {"N":"Book"} ]
    },
    "PageCount": {
        "ComparisonOperator": "GE",
        "AttributeValueList": [ {"N":600"} ]
    }
...
```

Anda juga dapat mengatur `ConditionalOperator` menjadi OR, yang berarti bahwa setidaknya salah satu syarat harus bernilai true.

Contoh

Ekspresi berikut bernilai true jika produk adalah sepeda gunung, jika produk tersebut adalah nama merek tertentu, atau jika harganya lebih besar dari 100.

```
...
    ConditionalOperator : "OR",
    "BicycleType": {
        "ComparisonOperator": "EQ",
        "AttributeValueList": [ {"S":"Mountain" ]
    },
    "Brand": {
        "ComparisonOperator": "EQ",
        "AttributeValueList": [ {"S":"Brand-Company A" ]
    },
    "Price": {
        "ComparisonOperator": "GT",
        "AttributeValueList": [ {"N":"100"} ]
    }
...
```

> **Note**
>
> Dalam ekspresi yang kompleks, syarat diproses secara berurutan, mulai syarat pertama hingga terakhir.
> Anda tidak dapat menggunakan AND dan OR dalam satu ekspresi.

Operator bersyarat lainnya

Dalam rilis DynamoDB sebelumnya, parameter `Expected` berperilaku berbeda untuk penulisan bersyarat. Setiap item dalam peta `Expected` mewakili nama atribut untuk diperiksa DynamoDB, beserta hal berikut:

- `Value` – nilai untuk dibandingkan terhadap atribut.
- `Exists` – menentukan apakah nilai ada sebelum mencoba operasi.

Opsi `Value` dan `Exists` tetap didukung dalam DynamoDB; tetapi, keduanya hanya membiarkan Anda menguji syarat kesetaraan, atau apakah atribut ada. Kami menyarankan agar Anda menggunakan `ComparisonOperator` dan `AttributeValueList` sebagai gantinya, karena opsi ini memungkinkan Anda membangun berbagai syarat yang lebih luas.

Example

`DeleteItem` dapat memeriksa untuk melihat apakah buku tidak lagi dalam publikasi, dan hanya menghapusnya jika syarat ini bernilai true. Berikut adalah contoh AWS CLI menggunakan syarat warisan:

```
aws dynamodb delete-item \
    --table-name ProductCatalog \
    --key '{
        "Id": {"N":"600"}
    }' \
    --expected '{
        "InPublication": {
            "Exists": true,
            "Value": {"BOOL":false}
        }
    }'
```

Contoh berikut melakukan hal yang sama, tetapi tidak menggunakan syarat warisan:

```
aws dynamodb delete-item \
    --table-name ProductCatalog \
    --key '{
        "Id": {"N":"600"}
    }' \
    --expected '{
        "InPublication": {
            "ComparisonOperator": "EQ",
            "AttributeValueList": [ {"BOOL":false} ]
        }
    }'
```

Example

Operasi `PutItem` dapat melindungi terhadap penimpaan item yang ada dengan atribut kunci primer yang sama. Berikut adalah contoh penggunaan syarat warisan:

```
aws dynamodb put-item \
    --table-name ProductCatalog \
    --item '{
```

```
      "Id": {"N":"500"},
        "Title": {"S":"Book 500 Title"}
    }' \
    --expected '{
        "Id": { "Exists": false }
    }'
```

Contoh berikut melakukan hal yang sama, tetapi tidak menggunakan syarat warisan:

```
aws dynamodb put-item \
    --table-name ProductCatalog \
    --item '{
      "Id": {"N":"500"},
        "Title": {"S":"Book 500 Title"}
    }' \
    --expected '{
        "Id": { "ComparisonOperator": "NULL" }
    }'
```

> **Note**
>
> Untuk syarat di peta `Expected`, jangan gunakan opsi warisan `Value` dan `Exists` bersama dengan `ComparisonOperator` dan `AttributeValueList`. Jika Anda melakukan hal ini, penulisan bersyarat Anda akan gagal.

Fitur pratinjau

Topik berikut adalah fitur pratinjau untuk DynamoDB. Fitur pratinjau dapat berubah.

Topik

- Konsistensi kuat Multi-Region

Konsistensi kuat Multi-Region

> **Note**
>
> Konsistensi kuat Multi-Region (MRSC) tersedia dalam pratinjau dan dapat berubah sewaktu-waktu.

Multi-Region strong consistency (MRSC) adalah kemampuan tabel global DynamoDB baru yang tersedia dalam pratinjau. Tabel global yang dikonfigurasi untuk MRSC memberikan kemampuan untuk melakukan pembacaan yang sangat konsisten dengan cakupan Multi-wilayah. Melakukan pembacaan yang sangat konsisten pada tabel MRSC memastikan Anda selalu membaca versi terbaru dari suatu item, terlepas dari Wilayah tempat Anda melakukan pembacaan.

Anda dapat menggunakan tabel global Multi-region yang sangat konsisten untuk membangun aplikasi dengan tujuan titik pemulihan (RPO) nol. RPO nol memastikan aplikasi Anda selalu dapat membaca versi terbaru data DynamoDB, bahkan jika gangguan aplikasi menyebabkan Anda mengalihkan lalu lintas ke yang berbeda. Wilayah AWS

Pratinjau MRSC hanya didukung untuk tabel global versi 2019.11.21 (Saat ini).

Topik

- Mode konsistensi untuk tabel global
- Ketersediaan wilayah untuk pratinjau MRSC
- Pertimbangan pratinjau MRSC
- Mengelola tabel global MRSC

Mode konsistensi untuk tabel global

Saat Anda membuat tabel global, Anda dapat mengonfigurasi mode konsistensinya. Tabel global menawarkan mode konsistensi Multi-wilayah berikut: Konsistensi akhir dan Konsistensi yang kuat (pratinjau).

Jika Anda tidak menentukan mode konsistensi saat membuat tabel global, tabel global default ke Multi-region eventual consistency (MREC). Tabel global tidak dapat berisi replika yang dikonfigurasi dengan mode konsistensi yang berbeda. Anda tidak dapat mengubah mode konsistensi tabel global.

Konsistensi akhir Multi-Region (MREC)

Multi-Region akhirnya konsisten (MREC) adalah mode konsistensi default untuk tabel global. Perubahan yang Anda buat pada item dalam replika tabel global MREC direplikasi ke semua replika lainnya, biasanya dalam satu detik atau kurang. Ini berarti operasi baca yang dibuat dengan ConsistentReadparameter yang disetel ke `true` (pembacaan yang sangat konsisten) akan selalu mengembalikan versi terbaru dari suatu item jika item diperbarui di Wilayah tempat pembacaan terjadi, tetapi mungkin mengembalikan data basi jika item diperbarui di Wilayah yang berbeda.

Konflik yang terjadi karena item yang sama dimodifikasi di beberapa Wilayah secara bersamaan diselesaikan dengan pendekatan kemenangan penulis terakhir.

Tabel global MREC akan memiliki latensi baca tulis yang lebih rendah dan sangat konsisten dibandingkan dengan tabel global MRSC.

Anda harus menggunakan mode MREC ketika:

- Aplikasi Anda dapat mentolerir data basi yang dikembalikan dari operasi pembacaan yang sangat konsisten jika data tersebut diperbarui di Wilayah lain.
- Anda memprioritaskan penulisan yang lebih rendah dan latensi baca yang sangat konsisten daripada konsistensi baca Multi-wilayah.
- Strategi ketersediaan tinggi Multi-wilayah Anda dapat mentolerir RPO yang lebih besar dari nol.

Konsistensi kuat Multi-Region (pratinjau)

> **Note**
>
> Konsistensi kuat Multi-Region (MRSC) tersedia dalam pratinjau dan dapat berubah sewaktu-waktu.

Perubahan yang Anda buat pada item dalam replika tabel global MRSC dapat langsung dibaca di tabel replika lainnya di tabel global dengan pembacaan yang sangat konsisten. Ini berarti operasi baca yang dibuat dengan `ConsistentRead` parameter yang disetel ke `true` (pembacaan yang sangat konsisten) akan selalu mengembalikan versi terbaru item dari tabel replika apa pun.

Jika operasi tulis akan memodifikasi item yang sudah dimodifikasi di Wilayah lain, operasi penulisan itu akan gagal dengan file`ReplicatedWriteConflictException`. Menulis yang gagal dengan `ReplicatedWriteConflictException` dapat dicoba lagi, dan akan berhasil jika pembaruan yang bertentangan telah diselesaikan dan tidak ada pembaruan yang bertentangan lainnya yang sedang berlangsung.

Tabel global MRSC akan memiliki latensi baca tulis yang lebih tinggi dan sangat konsisten dibandingkan dengan tabel global MREC.

Anda harus menggunakan mode MRSC ketika:

- Anda memerlukan jaminan baca yang sangat konsisten dengan cakupan Multi-wilayah.
- Anda memprioritaskan konsistensi baca global daripada latensi tulis yang lebih rendah.
- Strategi ketersediaan tinggi Multi-wilayah Anda membutuhkan RPO nol.

Ketersediaan wilayah untuk pratinjau MRSC

Pratinjau MRSC tersedia sebagai berikut: Wilayah AWS

- US East (N. Virginia) – us-east-1
- US East (Ohio) – us-east-2
- US West (Oregon) – us-west-2

Pertimbangan pratinjau MRSC

Bila Anda menggunakan tabel global dengan MRSC, pertimbangan berikut berlaku untuk pratinjau:

Pertimbangan beban kerja

- Tabel global dengan MRSC hanya tersedia dalam pratinjau. Anda tidak boleh menggunakannya untuk beban kerja produksi.
- Karakteristik kinerja dan throughput tabel MRSC mungkin berubah di seluruh pratinjau.

Dukungan fitur

- Hanya kunci milik Amazon yang didukung untuk pratinjau.
- Kunci yang dikelola AWStidak didukung dalam pratinjau.
- Kunci terkelola pelanggan tidak didukung dalam pratinjau.
- Kebijakan berbasis sumber daya tidak dapat digunakan untuk mengganggu replikasi antar Wilayah.
- CloudWatch Informasi Wawasan Kontributor hanya dilaporkan untuk Wilayah tempat operasi terjadi untuk tabel global MRSC dalam pratinjau.
- Time to Live (TTL) tidak didukung untuk tabel global MRSC dalam pratinjau.
- Indeks sekunder lokal (LSIs) tidak didukung untuk tabel global MRSC dalam pratinjau.
- Transaksi APIs tidak didukung dalam pratinjau.

Perbedaan perilaku dari tabel global MREC

- Pratinjau MRSC tersedia dalam satu set Wilayah terbatas.
- Tabel global MRSC harus berisi tepat tiga tabel replika.
- Anda harus membuat tabel global MRSC dengan menambahkan dua tabel replika ke tabel Single-region yang ada yang tidak berisi data apa pun.
- Anda tidak dapat menghapus tabel replika tunggal dari tabel global MRSC. Untuk menghapus tabel global MRSC, Anda harus menghapus dua tabel replika dalam satu tindakan, menghasilkan tabel Single-region. Anda kemudian dapat menghapus tabel Single-region yang tersisa.
- Pelanggaran kunci indeks sekunder global mungkin terjadi di luar periode penimbunan awal.

Kuota

- Akun AWS Dapat memiliki maksimal tiga tabel global dengan MRSC.
- Throughput tulis dalam mode kapasitas yang disediakan dibatasi hingga 10.000 Unit Kapasitas Tulis yang direplikasi (r). WCUs
- Throughput baca dalam mode kapasitas yang disediakan dibatasi hingga 10.000 Unit Kapasitas Baca (). RCUs
- Throughput tulis dalam mode kapasitas sesuai permintaan dibatasi hingga 10.000 Unit Permintaan Tulis yang direplikasi (r). WRUs
- Throughput baca dalam mode kapasitas sesuai permintaan dibatasi hingga 10.000 Unit Permintaan Baca ()RRUs.

Mengelola tabel global MRSC

> **Note**
>
> Konsistensi kuat Multi-Region (MRSC) tersedia dalam pratinjau dan dapat berubah sewaktu-waktu.

Anda dapat mengelola tabel global Multi-region yang sangat konsisten di Wilayah yang didukung menggunakan salah satu cara berikut:

- AWS Management Console
- AWS APIs untuk DynamoDB
- AWS Command Line Interface (AWS CLI)
- AWS SDK

Tutorial berikut menjelaskan cara membuat dan menghapus tabel global MRSC menggunakan AWS Management Console dan AWS CLI.

Topik

- Tutorial: Membuat tabel global MRSC di DynamoDB
- Tutorial: Menghapus tabel global MRSC di DynamoDB

Tutorial: Membuat tabel global MRSC di DynamoDB

> **Note**
>
> Konsistensi kuat Multi-Region (MRSC) tersedia dalam pratinjau dan dapat berubah sewaktu-waktu.

Dalam pratinjau, tabel global dengan MRSC harus berisi tepat tiga replika di Wilayah yang didukung. Anda membuat tabel global MRSC dengan menambahkan dua tabel replika ke tabel DynamoDB wilayah tunggal yang tidak berisi data apa pun, dan tidak memiliki konfigurasi apa pun. unsupported features

Using the AWS Management Console

Prosedur konsol ini membuat tabel global MRSC dengan membuat tabel Single-region baru. Prosedur ini juga menambahkan dua tabel replika di Wilayah pratinjau yang didukung yang tersisa.

1. Masuk ke AWS Management Console dan buka konsol DynamoDB di. https://console.aws.amazon.com/dynamodb/
2. Dari panel navigasi atas, pilih Wilayah tempat tabel global dengan MRSC didukung. Misalnya, pilih **`us-east-2`**.
3. Buat tabel wilayah tunggal sesuai permintaan baru. Untuk informasi tentang membuat tabel, lihat AWS Management Console diLangkah 1: Buat tabel di DynamoDB.

   > **Note**
   >
   > Mungkin perlu beberapa menit untuk tabel yang baru dibuat untuk berubah ke status ACTIVE.

4. Pada halaman Tabel, pilih tabel yang baru dibuat.
5. Pilih tab tabel Global, lalu pilih Buat replika.
6. Pada halaman Create replica, lakukan hal berikut:

   a. Di bawah Konsistensi Multi-Region, pilih Konsistensi yang kuat.

   b. Pilih Buat replika.

> **Note**
>
> Mungkin perlu beberapa menit agar tabel replika baru muncul dan berubah ke status ACTIVE.

Using the AWS CLI

AWS CLI Prosedur ini membuat tabel global MRSC dengan membuat tabel Single-region baru, dan kemudian menambahkan dua tabel replika.

1. Buat tabel wilayah Tunggal sesuai permintaan baru yang diberi nama `MusicTable` di Wilayah us-east-2.

```
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
```

2. Verifikasi bahwa tabel baru telah dibuat dan berada dalam status AKTIF.

> **Note**
>
> Mungkin perlu beberapa menit agar tabel berubah ke status AKTIF.

```
aws dynamodb describe-table \
    --table-name MusicTable \
    --region us-east-2

    {
    "Table": {
        ...
```

```
        "TableStatus": "ACTIVE",
       ...
}
```

3. Tambahkan dua tabel replika baru ke tabel Single-region di Wilayah yang didukung tersisa untuk pratinjau dengan menentukan parameternya`multi-region-consistency`. `STRONG`

```
aws dynamodb update-table \
    --table-name MusicTable \
    --replica-updates '[{"Create": {"RegionName": "us-east-1"}}, {"Create":
 {"RegionName": "us-west-2"}}]' \
    --multi-region-consistency STRONG \
    --region us-east-2
```

4. Gunakan perintah deskripsi-tabel untuk memverifikasi bahwa dua replika baru telah dibuat dan berada dalam status ACTIVE, dan bahwa tabel global dikonfigurasi untuk konsistensi kuat Multi-region.

```
aws dynamodb describe-table \
    --table-name MusicTable \
    --region us-east-1

    {
    "Table": {
        ...
        "Replicas": [
            {
                "RegionName": "us-east-1",
                "ReplicaStatus": "ACTIVE"
            },
            {
                "RegionName": "us-west-2",
                "ReplicaStatus": "ACTIVE"
            }
        ],
        "MultiRegionConsistency": "STRONG"
       ...
}
```

Tutorial: Menghapus tabel global MRSC di DynamoDB

> **Note**
>
> Konsistensi kuat Multi-Region (MRSC) tersedia dalam pratinjau dan dapat berubah sewaktu-waktu.

Dalam pratinjau, untuk menghapus tabel global MRSC, Anda harus menghapus dua tabel replika dalam satu tindakan, meninggalkan tabel Single-region. Kemudian, Anda dapat menghapus tabel Single-region yang tersisa secara opsional. Anda tidak dapat menghapus hanya satu tabel replika dari tabel global MRSC dan Anda tidak dapat menghapus ketiga tabel replika dari tabel global MRSC dalam satu tindakan.

Using the AWS Management Console

Prosedur konsol ini menghapus tabel global MRSC dengan menghapus dua tabel replika, menghasilkan tabel Single-region.

1. Masuk ke AWS Management Console dan buka konsol DynamoDB di. https://console.aws.amazon.com/dynamodb/
2. Dari panel navigasi atas, pilih Region yang berisi tabel global MRSC. Misalnya, pilih **`us-east-2`**.
3. Pada halaman Tabel, pilih tabel global MRSC.
4. Pilih tab tabel Global, lalu pilih Hapus replika.
5. Pada dialog konfirmasi yang muncul, ketik**`confirm`**.

   > **Note**
   >
   > Wilayah yang Anda pilih di konsol akan berisi tabel Single-region yang tersisa setelah dua replika dihapus dari tabel global MRSC.

6. Pilih Hapus.

Using the AWS CLI

AWS CLI Prosedur ini menghapus tabel global MRSC dengan menghapus dua tabel replika, menghasilkan tabel Single-region.

1. Hapus dua tabel replika dari tabel global MRSC.

```
aws dynamodb update-table \
    --table-name MusicTable \
    --replica-updates '[{"Delete": {"RegionName": "us-east-1"}}, {"Delete":
 {"RegionName": "us-west-2"}}]' \
    --region us-east-2
```

2. Verifikasi bahwa tabel Single-region yang tersisa berada dalam status ACTIVE, dan tidak memiliki tabel replika terkait.

```
aws dynamodb describe-table \
    --table-name MusicTable \
    --region us-east-2

    {
    "Table": {
        "AttributeDefinitions": [
            {
                "AttributeName": "Artist",
                "AttributeType": "S"
            },
            {
                "AttributeName": "SongTitle",
                "AttributeType": "S"
            }
        ],
        "TableName": "MusicTable",
        "TableStatus": "ACTIVE",
        ...
    }
}
```

Terjemahan disediakan oleh mesin penerjemah. Jika konten terjemahan yang diberikan bertentangan dengan versi bahasa Inggris aslinya, utamakan versi bahasa Inggris.